Read the Docs Template Documentation

Read the Docs

Jul 05, 2022

Contents

Quick definitions and intuitive explanations around machine learning.

CHAPTER 1

Contents

1.1 About

ML Compiled is an encyclopedia for machine learning and related subjects with an emphasis on explanations that are concise and intuitive.

In addition it offers the reader:

- Consistent notation throughout.
- Relevance. Only information that is relevant to machine learning is included. For general topics there is a special emphasis on how they can be used for ML.
- Clearly labelled links to the first paper to introduce a technique, where applicable.
- No long paragraphs of prose. Sections are short and clearly labeled.
- Links to important papers and useful blog posts for further reading.

ML Compiled is a work in progress and is being continually updated.

1.2 How to contribute

Please contribute by opening a pull request at https://github.com/cjratcliff/ml-compiled.

List of sections that need expanding in rough priority order (high at the top):

- Transformer
- Deconvolution layer
- Lambda loss
- Wasserstein distance
- Rademacher complexity
- VC dimension

- Evidence lower-bound
- Normalizing flows
- Change of variables
- ZCA
- Gradient boosting
- Softmax bottleneck
- CRFs
- Codomain
- Continuous
- Image
- SVMs - dual form, primal form and training
- Matrix differentiation
- Weibull distribution
- Vanishing/exploding gradient problem
- Feature scaling
- Object tracking
- Positive-unlabeled learning
- Logit
- Question answering
- Translation
- Manifold hypothesis
- Variational Bayes
- Parametric and non-parametric models
- Discriminative model
- Energy-based model
- Expectation-maximisation (EM)
- Inductive bias
- Metric learning
- Overcomplete representation
- Sparsity
- Advantage function
- Counterfactual regret minimization
- Direct policy search
- Information set
- Regret matching
- Reward clipping

- Reward sparsity
- Value iteration
- Mean field approach
- Partition function

List of sections to be added:

- Imitation learning
- Catastrophic forgetting
- Trust region policy optimization
- Self-attention
- Lipschitz smoothness
- Lipschitz constant
- Canonical Correlation Analysis (CCA)
- Compositionality
- Bayesian neural networks
- Bayesian optimisation
- Deep Belief Networks/Machines
- Restricted Boltzmann Machines
- Regression - p-values
- Empirical risk

1.3 Notation

1.3.1 Functions

Symbol	Meaning
Σ	Sum
Π	Product
σ	Sigmoid function
$\mathbb{E}$	Expectation
$\log$	Natural logarithm
$\lvert x \rvert$	Absolute value of x

1.3.2 Sets

Symbol	Meaning
$\mathbb{R}$	The set of real numbers
$\mathbb{R}^n$	The set of vectors of real numbers of length n
$\mathbb{R}^{m \times n}$	The set of matrices of real numbers of size $m \times n$

1.3.3 Calculus

Symbol	Meaning
f'	Derivative of f, shorthand for $\frac{df(x)}{dx}$
f''	Second derivative of f
$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{d^2y}{dx^2}$	Second derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\frac{\partial^2 y}{\partial x^2}$	Second partial derivative of y with respect to x

1.3.4 Information theory

Symbol	Meaning
$D_{KL}(P\|\|Q)$	KL-divergence between two distributions, P and Q

1.3.5 Linear algebra

Symbol	Meaning
x	A vector
X	A matrix
X^T	Transpose of X
X^*	Conjugate transpose of X
X^{-1}	Inverse of X
$\|\|x\|\|_2$	Euclidean norm of x
I	Identity matrix
$X * Y$	Element-wise product of X and Y
$X \otimes Y$	Kronecker product of X and Y
$x \cdot y$	Dot product of x and y
$tr(X)$	Trace of X
$det(X)$	Determinant of X

1.3.6 Probability

Symbol	Meaning
X	A random variable
$P(X)$	Probability of a particular value of X. Shorthand for $P(X = x_i)$
$U(a, b)$	Uniform distribution
$N(\mu, \sigma^2)$	Normal distribution

1.3.7 Statistics

Symbol	Meaning
μ	Mean
σ	Standard deviation
$V(X)$	Variance of X
$Cov(X, Y)$	Covariance of X and Y

1.3.8 Machine learning

Symbol	Meaning
θ	Parameters of the model
O	Observations or data
x	Feature vector
$L(...)$	Loss function
y	Label
$\hat{y}$	Prediction
g_t	Gradient at time t
u_t	Parameter update at time t
η	Learning rate

1.3.9 Reinforcement learning

Symbol	Meaning
$\pi(s_t)$	Policy
a_t	Action at time t
s_t	State at time t
r_t	Reward at time t
$V(s_t)$	Value function
A	Action set
γ	Discount factor

1.4 Calculus

1.4.1 Euler's method

An iterative method for solving differential equations (ie integration).

1.4.2 Hessian matrix

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function mapping vectors onto real numbers. Then the Hessian is defined as the matrix of second order partial derivatives:

$$H_{ij} = \frac{\partial^2 f}{\partial x_i x_j}$$

Applied to neural networks

In the context of neural networks, f is usually the loss function and x is the parameter vector so we have:

$$H_{ij} = \frac{\partial^2 L}{\partial \theta_i \theta_j}$$

The size and therefore cost to compute of the Hessian is quadratic in the number of parameters. This makes it infeasible to compute for most problems.

However, it is of theoretical interest as its properties can tell us a lot about the nature of the loss function we are trying to optimize.

If the Hessian at a point on the loss surface has no negative eigenvalues the point is a local minimum.

Condition number of the Hessian

If the Hessian is ill-conditioned, the loss function may be hard to optimize with gradient descent.

Recall that the condition number of a matrix is the ratio of the highest and lowest singular values and that in an ill-conditioned matrix this ratio is high. Large singular values of the Hessian indicate a large change in the gradient in some direction but small ones indicate very little change. Having both of these means the loss function may have 'ravines' which cause many first-order gradient descent methods to zigzag, resulting in slow convergence.

Relationship to generalization

Keskar et al. (2016) argue that when the Hessian evaluated at the solution has many large eigenvalues the corresponding network is likely to generalize less well. Large eigenvalues in the Hessian make the minimum likely to be sharp, which in turn generalize less well since those optima are more sensitive to small changes in the parameters.

Further reading

Empirical Analysis of the Hessian of Over-Parametrized Neural Networks, Sagun et al. (2017)
Sharp Minima Can Generalize For Deep Nets, Dinh et al. (2017)
On Large Batch Training for Deep Learning: Generalization Gap and Sharp Minima, Keskar et al. (2016)

1.4.3 Jacobian matrix

Let $f : \mathbb{R}^n \to \mathbb{R}^m$ be a function. Then the Jacobian of f can be defined as the matrix of partial derivatives:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

Applied to neural networks

It is common in machine learning to compute the Jacobian of the loss function of a network with respect to its parameters. Then m in $f : \mathbb{R}^n \to \mathbb{R}^m$ is 1 and the Jacobian is a vector representing the gradients of the network:

$$J_i = \frac{\partial L}{\partial \theta_i}$$

1.4.4 Partial derivative

The derivative of a function of many variables with respect to one of those variables.

The notation for the partial derivative of y with respect to x is $\frac{\partial y}{\partial x}$

1.4.5 Rules of differentiation

Sum rule

$$(f+g)' = f' + g'$$

Product rule

$$(fg)' = fg' + f'g$$

Quotient rule

$$(f/g)' = (f'g + fg')/g^2$$

Reciprocal rule

$$(1/f)' = -f'/f^2$$

Power rule

$$(x^a)' = ax^{a-1}$$

Exponentials

$$(a^{bx})' = a^{bx} \cdot b \log(a)$$

Logarithms

$$(\log_a x)' = 1/(x \ln a)$$

$$(\ln(x))' = 1/x$$

Chain rule

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}$$

Multivariate chain rule

Used to calculate total derivatives.

$$\frac{dy}{dx} = \frac{dy}{da} \cdot \frac{da}{dx} + \frac{dy}{db} \cdot \frac{db}{dx}$$

The derivative of a function wrt a function

Can be done using the chain rule. For example, $\partial x^6 / \partial x^2$ can be found by setting $y = x^6$ and $z = x^2$.

Then do $\partial y / \partial z$

$$= \partial y / \partial x \cdot \partial x / \partial z$$

$$= 6x^5 \cdot 1/2x$$

$$= 3x^4$$

Inverse relationship

In general dy/dx is the inverse of dx/dy.

Matrix differentiation

$$\frac{dX}{dX} = I$$
$$\frac{dX^T Y}{dX} = Y$$
$$\frac{dYX}{dX} = Y^T$$

1.4.6 Total derivative

The derivative of a function of many arguments with respect to one of those arguments, taking into account any indirect effects via the other arguments.

The total derivative of $z(x, y)$ with respect to x is:

$$\frac{dz}{dx} = \frac{\partial z}{\partial x} + \frac{\partial z}{\partial y}\frac{dy}{dx}$$

1.5 Information theory and complexity

1.5.1 Akaike Information Criterion (AIC)

A measure of the quality of a model that combines accuracy with the number of parameters. Smaller AIC values mean the model is better. The formula is:

$$\mathrm{AIC}(x, \theta) = 2|\theta| - 2\ln L(\theta, x)$$

Where x is the data and L is the likelihood function.

1.5.2 Capacity

The capacity of a machine learning model describes the complexity of the functions it can learn. If the model can learn highly complex functions it is said to have a high capacity. If it can only learn simple functions it has a low capacity.

1.5.3 Entropy

The entropy of a discrete probability distribution $p(x)$ is:

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

Joint entropy

Measures the entropy of a joint distribution $p(x, y)$.

The joint entropy for a discrete distribution over two variables, X and Y is:

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y)$$

Conditional entropy

Measures the entropy of the distribution of one random variable given that the value of another is known.

The conditional entropy for a discrete distribution is given by:

$$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(y|x)$$

1.5.4 Finite-sample expressivity

The ability of a model to memorize the training set.

1.5.5 Fisher Information Matrix

An $N \times N$ matrix of second-order partial derivatives where N is the number of parameters in a model.

The matrix is defined as:

$$I(\theta)_{ij} = E[\frac{\partial \log f(X;\theta)}{\partial \theta_i} \frac{\partial \log f(X;\theta)}{\partial \theta_j} | \theta]$$

The Fisher Information Matrix is equal to the negative expected Hessian of the log likelihood.

1.5.6 Information bottleneck

An objective for training compressed representations that have predictive power.

$$\min I(X, T) - \beta I(T, Y)$$

Where $I(X,T)$ and $I(T,Y)$ represent the mutual information between their respective arguments. X is the input features, Y is the labels and T is a representation of the input such as the activations of a hidden layer in a neural network. β is a hyperparameter controlling the trade-off between compression and predictive power.

When the expression is minimised there is very little mutual information between the compressed representation and the input. At the same time, there is a lot of mutual information between the representation and the output, meaning the representation is useful for prediction.

Proposed in

The information bottleneck method, Tishby et al. (2000)

1.5.7 Jensen-Shannon divergence

A symmetric version of the KL-divergence. This means that $JS(P,Q) = JS(Q,P)$, which is not true for the KL-divergence.

$$JS(P,Q) = \frac{1}{2}(D_{KL}(P||M) + D_{KL}(M||Q))$$

where M is a mixture distribution equal to $\frac{1}{2}(P+Q)$

Alternatives

KL divergence

Total variation distance

Wasserstein distance

1.5.8 Kullback-Leibler divergence

A measure of the difference between two probability distributions. Also known as the KL-divergence and the relative entropy. In the usual use case one distribution is the true distribution of the data and the other is an approximation of it.

For discrete distributions it is given as:

$$D_{KL}(P||Q) = -\sum_i P_i \log \frac{Q_i}{P_i}$$

Disadvantages

If a point is outside the support of Q ($Q_i = 0$), the KL-divergence will explode since $\log(0)$ is undefined. This can be dealt with by adding some random noise to Q. However, this introduces a degree of error and a lot of noise is often needed for convergence when using the KL-divergence for MLE. The Wasserstein distance, which also measures the distance between two distributions, does not have this problem.

Properties

- The KL-divergence is not symmetric.
- A KL-Divergence of 0 means the distributions are identical. As the distributions become more different the divergence becomes more negative.

Advantage
Jensen-Shannon divergence
Total variation distance
Wasserstein distance

1.5.9 Mutual information

Measures the dependence between two random variables.

$$I(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

If the variables are independent $I(X,Y) = 0$. If they are completely dependent $I(X,Y) = H(X) = H(Y)$.

1.5.10 Rademacher complexity

TODO

1.5.11 Total variation distance

Like the Kullback-Leibler divergence, it is also a way of measuring the difference between two different probability distributions.

For two discrete distributions the total variation distance is given by:

TODO

Alternatives
Jensen-Shannon divergence
KL divergence
Wasserstein distance

1.5.12 VC dimension

Vapnik–Chervonenkis dimension is a measure of the capacity of a model.

1.6 Functions

1.6.1 Bijective

A function that is both surjective and injective.

1.6.2 Codomain

1.6.3 Concave

A function is concave if:

$$f(\alpha x + (1 - \alpha)y) \geq \alpha f(x) + (1 - \alpha)f(y)$$

Contrast with convex.

1.6.4 Continuous

1.6.5 Convex

A function is convex if:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

x^2 is an example of a convex function.

Contrast with concave.

1.6.6 Domain

The set of points the function is defined for.

1.6.7 Image

1.6.8 Injective

A function is injective if it never maps two different inputs to the same output.

See also: surjective, bijective

1.6.9 Monotonic

A function is monotonic if it is non-decreasing or non-increasing.

1.6.10 Surjective

A function is surjective if, for every possible output, there is one input that produces that output.

See also: injective, bijective

1.7 Geometry

1.7.1 Affine transformation

A linear mapping that preserves points, lines and planes. Examples include translation, scale, rotation or shear.

1.7.2 Cosine similarity

Measures the similarity of two vectors by calculating the cosine of the angle between them. The similarity is 1 if the vectors are pointing in the same direction, 0 if they are orthogonal, and -1 if they are pointing in exactly opposite directions.

$$c(x, y) = x \cdot y / (||x||_2 \cdot ||y||_2)$$

Where $x \cdot y$ is the dot product.

Relationship with the Euclidean distance

The major differences between the Euclidean distance and cosine similarity are as follows:

- The Euclidean distance takes magnitude of the two vectors into account. The cosine similarity ignores it.
- Unlike the Euclidean distance, the cosine distance does not suffer from the curse of dimensionality, making it useful for comparing high-dimensional feature vectors.
- The cosine similarity is not a metric in the mathematical sense.
- The cosine similarity is bounded between -1 and 1, whereas the Euclidean distance must be between 0 and infinity.
- The Euclidean distance for two identical vectors is zero. The cosine similarity in this case is 1.
- The cosine similarity does not satisfy the triangle inequality.
- The cosine similarity is undefined if one of the vectors is all zeros.

There is a linear relationship between the cosine similarity of two vectors and the squared Euclidean distance if the vectors first undergo L2 normalization.

The proof is as follows:

Let x and y be two vectors that have been normalized such that $||x||_2 = ||y||_2 = 1$. Then the expression for the squared Euclidean distance is:

$$\begin{aligned} & ||x - y||_2^2 \\ &= (x - y)^T (x - y) \\ &= x^T x - 2x^T y + y^T y \\ &= ||x||_2 - 2x^T y + ||y||_2 \\ &= 2 - 2x^T y \\ &= 2 - 2c(x, y) \end{aligned}$$

1.7.3 Euclidean distance

Measures the distance between two vectors.

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

Disadvantages

The Euclidean distance can have poor performance under high dimensionality. For points randomly distributed in space, the distribution of distances between random pairs of points falls tightly around the mean. This is because the Euclidean distance is the nth root of the sum of distances along each dimension. So this becomes close to the mean, just as for any sufficiently large sample.

The ratio between the distance between the two furthest points and the distance between the two closest approaches 1 as the dimensionality increases.

1.7.4 High dimensionality

This section describes the properties of some mathematical concepts in high dimensions.

Euclidean distance

- For points randomly distributed in space, the distribution of the distances between them falls tightly around the mean.
- For this reason the usefulness of the Euclidean distance is limited in high dimensions.
- This also means the ratio between the distance between the two furthest points and the distance between the two closest approaches 1 for high dimensions.

Gaussian distribution

- Although the value remains highest at the origin, there is very little volume there. Most points are in the 'tails'. This reflects the intuition that over many dimensions, any given point is likely to be anomalous in at least one aspect.

Sphere

- There is almost no interior volume. This follows the same intuition as for the Gaussian distribution - a random point is likely to be near the edge in at least one dimension, which is sufficient to call it exterior.
- The volume is mostly contained in a thin ring around the equator at the surface.
- The surface area is almost all at the equator.

Interpolation

- Linearly interpolating between two high-dimensional vectors will produce something that doesn't look much like either. The entries will tend to be atypically close to the mean. Polar interpolation should be used instead.

Inner product of random samples

- Two random high-dimensional vectors are likely to be close to orthogonal. This is because orthogonality is measured by the inner product, which is the sum of the elementwise products. Over a large number of dimensions, this will tend towards the mean of the products which will be zero, so long as the mean of the sampling distribution is also zero.

Further reading

High-Dimensional Space, Hopcroft and Kannan
Gaussian Distributions are Soap Bubbles, Huszár (2017)

1.7.5 Lebesgue measure

The concept of volume, generalised to an arbitrary number of dimensions. In one dimension it is the same as length and in two it is the same as area.

1.7.6 Manifold

Type of topological space. Includes lines, circles, planes, spheres and tori.

1.7.7 Metric

A metric $d(x, y)$ must have the following properties:

$$\begin{aligned} d(x, y) &\geq 0 \\ d(x, y) = 0 &\Leftrightarrow x = y \\ d(x, y) &= d(y, x) \\ d(x, z) &\leq d(x, y) + d(y, z) \end{aligned}$$

1.7.8 Polar interpolation

The polar interpolation of two vectors x and y is:

$$\sqrt{p}x + \sqrt{1 - p}y$$

Contrast this with linear interpolation which is $px + (1 - p)y$, where $0 \leq p \leq 1$.

Unlike linear interpolation, the sum of the coefficients may exceed 1.

1.7.9 Wasserstein distance

Also known as the earth mover distance. Like the Kullback-Leibler divergence, it is a way of measuring the difference between two different probability distributions.

Intuition

If the two probability distributions are visualised as mounds of earth, the Wasserstein distance is the least possible amount of effort required to turn one mound into the other. That is, the amount of earth mutliplied by the distance it has to be moved.

Defining the Wasserstein distance

There are many different ways to move the earth so calculating the Wasserstein distance requires solving an optimisation problem, in general. However, an exact solution exists if both distributions are normal.

Advantages

Unlike the Kullback-Leibler divergence, Jensen-Shannon divergence and total variation distance, this metric does not have zero gradients when the supports of P and Q are disjoint (the probability distributions have no overlap).

Disadvantages

Exact computation of the Wasserstein distance is intractable except in some special cases.

Used by

Wasserstein GAN, Arjovsky et al. (2017)

Further reading

Wasserstein GAN and the Kantorovich-Rubinstein Duality, Herrmann

1.8 Linear algebra

1.8.1 Adjoint

Another term for the conjugate transpose. Identical to the transpose if the matrix is real.

1.8.2 Affine combination

A linear combination of vectors where the weights sum to 1. Unlike a convex combination, the weights can be negative.

1.8.3 Condition number

The condition number of a matrix A is defined as:

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ are the largest and smallest singular values of A respectively.

If $\kappa(A)$ is high, the matrix A is said to be **ill-conditioned**. Conversely, if the condition number is very low (ie close to 0) we say A is **well-conditioned**.

Since singular values are always non-negative, condition numbers are also always non-negative.

1.8.4 Conjugate transpose

The matrix obtained by taking the transpose followed by the complex conjugate of each entry.

1.8.5 Dot product

The dot product for two vectors a and b is:

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

1.8.6 Eigenvalues and eigenvectors

Let A be a square matrix. Then the eigenvalues and eigenvectors of the matrix are the vectors v and scalars λ respectively that satisfy the equation:

$$Av = \lambda v$$

Properties

The trace of A is the sum of its eigenvalues:

$$\text{tr}(A) = \sum_{i} \lambda_i$$

The determinant of A is the product of its eigenvalues.

$$\det(A) = \prod_{i} \lambda_i$$

1.8.7 Gaussian elimination

An algorithm for solving SLEs that iteratively transforms the matrix into an upper triangular one in row echelon form.

1.8.8 Hadamard product

Synonymous with elementwise-multiplication.

1.8.9 Inverse

The inverse of a matrix A is written as A^{-1}.

A matrix A is invertible if and only if there exists a matrix B such that $AB = BA = I$.

The inverse can be found using:

- Gaussian elimination
- LU decomposition
- Gauss-Jordan elimination

1.8.10 Matrix decomposition

Also known as matrix factorization.

Cholesky decomposition

$$A = LL^*$$

where A is Hermitian and positive-definite, L is lower-triangular and L^* is its conjugate transpose. Can be used for solving SLEs.

Eigendecomposition

$$A = Q\Lambda Q^*$$

Where the columns of Q are the eigenvectors of A. Λ is a diagonal matrix in which Λ_{ii} is the i'th eigenvalue of A.

LU decomposition

A = LU, where L is lower triangular and U is upper triangular. Can be used to solve SLEs.

Polar decomposition

$$A = UP$$

where U is unitary and P is positive semi-definite and Hermitian.

QR decomposition

Decomposes a real square matrix A such that $A = QR$. Q is an orthogonal matrix and R is upper triangular.

Singular value decomposition (SVD)

Let $A \in \mathbb{R}^{m \times n}$ be the matrix to be decomposed. SVD is:

$$A = U\Sigma V^*$$

where $U \in \mathbb{R}^{m \times m}$ is a unitary matrix, $\Sigma \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix containing the singular values and $V \in \mathbb{R}^{n \times n}$ is a unitary matrix.

Can be used for computing the sum of squares or the pseudoinverse.

1.8.11 Orthonormal vectors

Two vectors are orthonormal if they are orthogonal and both unit vectors.

1.8.12 Outer product

The outer product of two column vectors x and y is:

$$A = xy^T$$

1.8.13 Rank

Matrix rank

The number of linearly independent columns.

Tensor rank

When the term is applied to tensors, the rank refers to the dimensionality: * Rank 0 is a scalar * Rank 1 is a vector * Rank 2 is a matrix etc.

1.8.14 Singular values

For a matrix A the singular values are the set of numbers:

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$$

where $\sigma_i = \sqrt{\lambda_i}$ and λ_i is an eigenvalue of the matrix $A^T A$.

1.8.15 Span

The span of a matrix is the set of all points that can be obtained as a linear combination of the vectors in the matrix.

1.8.16 Spectral norm

The maximum singular value of a matrix.

1.8.17 Spectral radius

The maximum of the magnitudes of the eigenvalues.

1.8.18 Spectrum

The set of eigenvalues of a matrix.

1.8.19 System of Linear Equations (SLE)

A set of n linear equations using a common set of m variables. For example:

$$3x_0 + 4x_1 = 5$$

$$-2x_0 + x_1 = 11$$

In matrix form an SLE can be written as:

$$Ax = b$$

Where x is the vector of unknowns to be determined, A is a matrix of the coefficients from the left-hand side and the vector b contains the numbers from the right-hand side of the equations.

Systems of linear equations can be solved in many ways. Gaussian elimination is one.

Underdetermined and overdetermined systems

- If the number of variables exceeds the number of equations the system is **underdetermined**.
- If the number of variables is less than the number of equations the system is **overdetermined**.

1.8.20 Trace

The sum of the elements along the main diagonal of a square matrix.

$$\text{tr}(A) = \sum_{i=1}^{n} A_{ii}$$

Satisfies the following properties:

$$\begin{aligned} \text{tr}(A) &= \text{tr}(A^T) \\ \text{tr}(A + B) &= \text{tr}(A) + \text{tr}(B) \\ \text{tr}(cA) &= c\text{tr}(A) \end{aligned}$$

1.8.21 Transpose

$$(A^T)_{ij} = A_{ji}$$

Satisfies the following properties:

$$\begin{aligned} (A + B)^T &= A^T + B^T \\ (AB)^T &= B^T A^T \\ (A^T)^{-1} &= (A^{-1})^T \end{aligned}$$

1.8.22 Types of matrix

This table summarises the relationship between types of real and complex matrices. The concept in the complex column is the same as the concept in the same row of the real column if the matrix is real-valued.

Real	Complex
Symettric	Hermitian
Orthogonal	Unitary
Transpose	Conjugate transpose

Degenerate

A matrix that is not invertible.

Diagonal matrix

A matrix where $A_{ij} = 0$ if $i \neq j$.

Can be written as diag(a) where a is a vector of values specifying the diagonal entries.

Diagonal matrices have the following properties:

$$\begin{aligned} \text{diag}(a) + \text{diag}(b) &= \text{diag}(a + b) \\ \text{diag}(a) \cdot \text{diag}(b) &= \text{diag}(a * b) \\ \text{diag}(a)^{-1} &= \text{diag}(a_1^{-1}, ..., a_n^{-1}) \\ \det(\text{diag}(a)) &= \prod_i a_i \end{aligned}$$

The eigenvalues of a diagonal matrix are the set of its values on the diagonal.

Hermitian matrix

The complex equivalent of a symmetric matrix. $A = A^*$, where * represents the conjugate transpose.

Also known as a self-adjoint matrix.

Normal matrix

$A^*A = AA^*$ where A^* is the conjugate transpose of A.

Orthogonal matrix

$$A^T A = AA^T = I$$

Positive and negative (semi-)definite

A matrix $A \in \mathbb{R}^{n \times n}$ is positive definite if:

$$z^T Az > 0, \forall z \in \mathbb{R}^n, z \neq 0$$

Positive semi-definite matrices are defined analogously, except with $z^T Az \geq 0$

Negative definite and negative semi-definite matrices are the same but with the inequality round the other way.

Singular matrix

A square matrix which is not invertible. A matrix is singular if and only if the determinant is zero.

Symmetric matrix

A square matrix A where $A = A^T$.

Some properties of symmetric matrices are:

- All the eigenvalues of the matrix are real.

Triangular matrix

Either a lower triangular or an upper triangular matrix.

Lower triangular matrix

A square matrix where only the lower triangle is not composed of zeros. Formally:

$$A_{ij} = 0, \text{if} i < j$$

Upper triangular matrix

A square matrix where only the upper triangle is not composed of zeros. Formally:

$$A_{ij} = 0, \text{if} i \geq j$$

Unitary matrix

A matrix where its inverse is the same as its complex conjugate. The complex version of an orthogonal matrix.

$$A^*A = AA^* = I$$

1.9 Monte Carlo methods

1.9.1 Gibbs sampling

A simple MCMC algorithm, used for sampling from the joint distribution when it cannot be calculated directly but the conditional can be.

An example use case is in generative image models. The joint distribution over all the pixels is intractable but the conditional distribution for one pixel given the rest is not.

Pseudocode:

```
1. Randomly initialise x.
2. For  i = 1,...,d
3.   Sample the ith dimension of x given the values in all the other dimensions.
```

1.9.2 Importance sampling

Monte Carlo method that attempts to estimate the mean of a distribution with zero density almost everywhere that would make simple Monte Carlo methods ineffective. Does this by sampling from a distribution that does not have this property then adjusting to compensate.

Can be used to deal with the computational problems of very large vocabularies in NLP but suffers from stability problems.

Quick Training of Probabilistic Neural Nets by Importance Sampling, Bengio and Senecal (2003)

Deep Learning, Section 17.2

1.9.3 MCMC (Markov Chain Monte Carlo)

A class of algorithm which is useful for sampling from and computing expectations of highly complex and high-dimensional probability distributions. For example, the distribution for images which contain dogs, as described by their pixel values.

High probability areas are very low proportion of the total space due to the high dimensionality, meaning that rejection sampling won't work. Instead, MCMC uses a random walk (specifically a Markov chain) that attempts to stay close to areas of high probability in the space.

MCMC algorithms do not generate independent samples.

1.9.4 Metropolis-Hastings algorithm

A simple MCMC algorithm.

Pseudocode:

```
1. Randomly initialise x
2. For t = 1,...,T_max
3. Generate a candidate for the next sample from a normal distribution
   centered on the current point.
4. Calculate the acceptance ratio, the probability that the new candidate
   will be retained. This is equal to the density at the current point,
   divided by the density at the candidate point.
5. Either accept or reject the candidate, based on a random sample from
   the distribution (a, 1-a).
```

The proposal distribution is the distribution over the possible points to sample next.

1.9.5 Mixing

The samples from an MCMC algorithm are said to be well mixed if they are independent of each other.

Poor mixing can be caused by getting stuck in local minima of the density function.

1.10 Probability

1.10.1 "Admits a density/distribution"

If a variable 'admits a distribution', that means it can be described by a probability density function. Contrast with

$$P(X = a) = \begin{cases} 1, & \text{if } a = 0 \\ 0, & \text{otherwise} \end{cases}$$

which cannot be described by a pdf, so we would say that X does not admit a distribution.

1.10.2 Bayes' rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

1.10.3 Bayesian inference

The use of Bayes' rule to update a probability distribution as the amount of evidence changes.

1.10.4 Chain rule of probability

Gives the joint probability for a set of variables as the product of conditionals and a prior.

$$P(A_n, ..., A_1) = \prod_{i=1}^{n} P(A_i | A_1, ..., A_{i-1})$$

For three variables this looks like:

$$P(A_3, A_2, A_1) = P(A_3 | A_2, A_1) \cdot P(A_2 | A_1) \cdot P(A_1)$$

1.10.5 Change of variables

In the context of probability densities the change of variables formula describes how one distribution $p(y)$ can be given in terms of another, $p(x)$:

$$p(y) = |\frac{\partial f(x)}{\partial x}|^{-1} p(x)$$

Where f is an invertible function.

1.10.6 Conjugate prior

If the prior $p(\theta)$ and the posterior $p(X|\theta)$ are both from the same family of distributions (eg Beta) the likelihood $p(X|\theta)$ is distributed according to the table below:

Likelihood	Conjugate prior
Bernoulli	Beta
Binomial	Beta
Negative binomial	Beta
Categorical	Dirichlet
Multinomial	Dirichlet
Poisson	Gamma

1.10.7 Distributions

Bernoulli

Distribution for a random variable which is 1 with probability p and zero with probability $1 - p$.

Special case of the Binomial distribution, which generalizes the Bernoulli to multiple trials.

$$P(x = k; p) = \begin{cases} p, & \text{if } k = 1 \\ 1 - p, & \text{if } k = 0 \end{cases}$$

Beta

Family of distributions defined over $[0, 1]$. This makes them particularly useful for defining the distribution of probabilities.

Binomial

Distribution for the number of successes in n trials, each with probability p of success and 1-p of failure. The probability density function is:

$$P(x = k; n, p) = \binom{n}{k} p^k (1-p)^{n-k}$$

Is closely approximated by the Poisson distribution when n is large and p is small.

Boltzmann

$$P(x_i; T, \epsilon) = \frac{1}{Q} e^{-\epsilon_i / T}$$

where ϵ_i is the energy of x_i, T is the temperature of the system and Q is a normalising constant.

Categorical

Generalizes the Bernoulli distribution to more than two categories.

$$P(x = k; p) = p_k$$

Dirichlet

Multivariate version of the Beta distribution.

Conjugate prior of the categorical and multinomial distributions.

Gamma

Can be used to model the amount of something a particular period, area or volume. For example, the amount of rainfall in an area in a month. This is as opposed to the Poisson which models the distribution for the number of discrete events.

Geometric

Special case of the Negative Binomial distribution.

Gibbs

See Boltzmann Distribution.

Gumbel

Used to model the distribution of the maximum (or the minimum) of a number of samples of various distributions.

Used by
Categorical Reparameterization with Gumbel-Softmax, Jang et al. (2016)

Hypergeometric

Models the probability of k successes in n draws without replacement from a population of size N, where K of the objects in the population have the desired characteristic. Similar to the Binomial, except that the draws are made without replacement which means they are no longer independent.

Multinomial

The distribution for n trials, each with k possible outcomes.

When n and k take on specific values or ranges the Multinomial distribution has specific names.

	$k = 2$	$k \geq 2$
$n = 1$	Bernoulli	Categorical
$n \geq 1$	Binomial	Multinomial

Multivariate

This section summarises some univariate distributions and their multivariate versions:

Univariate	Multivariate
Bernoulli	Binomial
Categorical	Multinomial
Beta	Dirichlet
Gamma	Wishart

Negative Binomial

Distribution of the number of successes before a given number of failures occur.

Poisson

Used to model the number of events which occur within a particular period, area or volume.

Zipf

A distribution that has been observed to be a good model for things like the frequency of words in a language, where there are a few very popular words and a long tail of lesser known ones.

For a population of size n, the frequency of the kth most frequent item is:

$$\frac{1/k^s}{\sum_{i=1}^{n} 1/i^s}$$

where $s \geq 0$ is a hyperparameter

1.10.8 Inference

Probabilistic inference is the task of determining the probability of a particular outcome.

1.10.9 Law of total probability

$$P(X) = \sum_i P(X|Y = y_i)P(Y = y_i)$$

1.10.10 Likelihood

The likelihood of the parameters θ given the observation data X is equal to the probability of the data given the parameters.

$$L(\theta|X) = P(X|\theta)$$

1.10.11 Marginal distribution

The most basic sort of probability, $P(x)$. Contrast with the conditional distribution $P(x|y)$ or the joint $P(x, y)$.

1.10.12 Marginal likelihood

A likelihood function in which some variable has been marginalised out (removed by summation).

1.10.13 MAP estimation

Maximum a posteriori estimation. A point estimate for the parameters θ, given the observations X. Can be seen as a regularization of MLE since it also incorporates a prior distribution. Uses Bayes' rule to incorporate a prior over the parameters and find the parameters that are most likely given the data (rather than the other way around). Unlike with MLE (which is a bit of a simplification), the most likely parameters given the data are exactly what we want to find.

$$\hat{\theta}_{MAP}(X) = \arg\max_{\theta} p(\theta|X) = \arg\max_{\theta} \frac{p(X|\theta)q(\theta)}{\int_{\theta'} p(O|\theta')q(\theta')d\theta'} = \arg\max_{\theta} p(X|\theta)q(\theta)$$

Where $q(\theta)$ is the prior for the parameters.

Note that in the equation above the denominator vanishes since it does not depend on θ.

1.10.14 Maximum likelihood estimation (MLE)

Finds the set of parameters θ that are most likely, given the data X. Since priors over parameters are not taken into account unless MAP estimation is taking place, this is equivalent to finding the parameters that maximize the probability of the data given the parameters:

$$\hat{\theta}_{MLE}(X) = \arg\max_{\theta} p(X|\theta)$$

1.10.15 Normalizing flow

A function that can be used to transform one random variable into another. The function must be invertible and have a tractable Jacobian.

Extensively used for density estimation.

1.10.16 Prior

A probability distribution before any evidence is taken into account. For example the probability that it will rain where there are no observations such as cloud cover.

Improper prior

A prior whose probability distribution has infinitesimal density over an infinitely large range. For example, the distribution for picking an integer at random.

Informative and uninformative priors

Below are some examples for each:

Informative

- The temperature is normally distributed with mean 20 and variance 3.

Uninformative

- The temperature is positive.
- The temperature is less than 200.
- All temperatures are equally likely.

'Uninformative' can be a misnomer. 'Not very informative' would be more accurate.

1.10.17 Posterior

A conditional probability distribution that takes evidence into account. For example, the probability that it will rain, given that it is cloudy.

1.11 Statistics

1.11.1 Arithmetic mean

The arithmetic mean of a set of inputs $\{x_1, x_2, ..., x_n\}$ is:

$$A(x_1, x_2, ..., x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

1.11.2 Correlation

The correlation between two random variables X and Y is:

$$\text{Corr}(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{V(X)V(Y)}}$$

1.11.3 Covariance

The covariance between two random variables X and Y is defined as:

$$\text{Cov}(X, Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)$$

Covariance matrix

A square matrix Σ where $\Sigma_{ij} = Cov(X_i, X_j)$ and X_i and X_j are two variables.

There are three types of covariance matrix:

- Full - All entries are specified. Has $O(n^2)$ parameters for n variables.
- Diagonal - The matrix is diagonal, meaning all off-diagonal entries are zero. Variances can differ across dimensions but there is no interplay between the dimensions. Has $O(n)$ parameters.
- Spherical - The matrix is equal to the identity matrix multiplied by a constant. This means the variance is the same in all dimensions. Has $O(1)$ parameters.

A valid covariance matrix is always symmetric and positive semi-definite.

1.11.4 Geometric mean

The geometric mean of a set of inputs $\{x_1, x_2, ..., x_n\}$ is:

$$G(x_1, x_2, ..., x_n) = \sqrt[n]{x_1 x_2 ... x_n}$$

Only applicable to positive numbers since otherwise it may involve taking the root of a negative number.

1.11.5 Harmonic mean

The harmonic mean for a set of inputs $\{x_1, x_2, ..., x_n\}$ is:

$$H(x_1, x_2, ..., x_n) = n/\sum_{i=1}^{n}\frac{1}{x_i}$$

Cannot be computed if one of the numbers is zero since that would necessitate dividing by zero.

Used for the F1-score, which is the Harmonic mean of the precision and recall.

1.11.6 Heteroscedasticity

When the error of a model is correlated with one or more of the features.

1.11.7 Moments

- 1st moment - Arithmetic mean
- 2nd moment - Variance
- 3rd moment - Skewness
- 4th moment - Kurtosis

1.11.8 Moving average

A moving average smooths a sequence of observations.

Exponential moving average (EMA)

A type of moving average in which the influence of past observations on the current average diminishes exponentially with time.

$$m_t = \alpha m_{t-1} + (1 - \alpha)x_t$$

m_t is the moving average at time t, x_t is the input at time t and $0 < \alpha < 1$ is a hyperparameter. As α decreases, the moving average weights recent observations more strongly.

Bias correction

If we initialise the EMA to equal zero ($m_0 = 0$) it will be very biased towards zero around the start. To correct this we can start with α being close to 0 and gradually increase it. This effect can be achieved by rewriting the formula as:

$$m_t = \frac{1}{1 - \alpha^t}(\alpha m_{t-1} + (1 - \alpha)x_t)$$

See Adam: A Method for Stochastic Optimization, Kingma et al. (2015) for an example of this bias correction being used in practice.

1.11.9 Point estimate

An estimate for a parameter, such as the mean of a population for example. It describes the belief about this quantity with a single number, in contrast with a distribution which could be used to describe the belief for the parameter with multiple numbers.

1.11.10 Skewness

Measures the asymmetry of a probability distribution.

$$= E\left[\left(\frac{X - \mu}{\sigma}\right)^3\right]$$

1.11.11 Standard deviation

The square root of the variance. The formula is:

$$\sigma = \sqrt{E[(X - \mu)^2]}$$

where μ is the mean of X.

Sample standard deviation

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2}$$

Note that the above is the biased estimator for the sample standard deviation. Estimators which are unbiased exist but they each only apply to some population distributions.

1.11.12 Variance

The variance of $X = \{x_1, ..., x_n\}$ is:

$$V(X) = E[(X - \mu)^2]$$

where μ is the mean of X.

The formula can also be written as:

$$V(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Sample variance

When it is impractical to compute the variance over the entire population, we can take a sample instead and compute the sample variance. The formula for the unbiased sample variance is:

$$V(X) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2$$

1.12 Activation functions

1.12.1 CReLU

Concatenated ReLU.

$$f(x) = \text{concat}(\text{ReLU}(x), \text{ReLU}(-x))$$

Using the CReLU doubles the size of the input to the next layer, increasing the number of parameters. However, Shang et al. showed that CReLU can improve accuracy on image recoginition tasks when used for the lower convolutional layers, even when halving the number of filters in those layers at the same time.

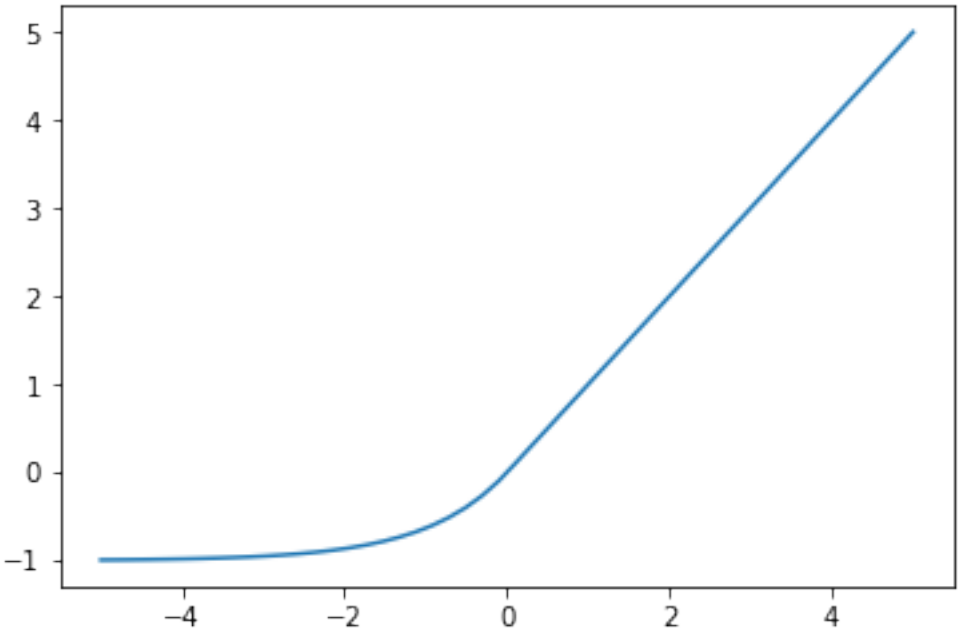
Proposed in

Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units, Shang et al. (2016)

1.12.2 ELU

Exponential Linear Unit.

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$



In practice the hyperparameter α is always set to 1.

Compared to ReLUs, ELUs have a mean activation closer to zero which is helpful. However, this advantage is probably nullified by batch normalization.

The more gradual decrease of the gradient should also make them less susceptible to the dying ReLU problem, although they will suffer from the vanishing gradients problem instead.

Proposed in

Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), Clevert et al. (2015)

1.12.3 GELU

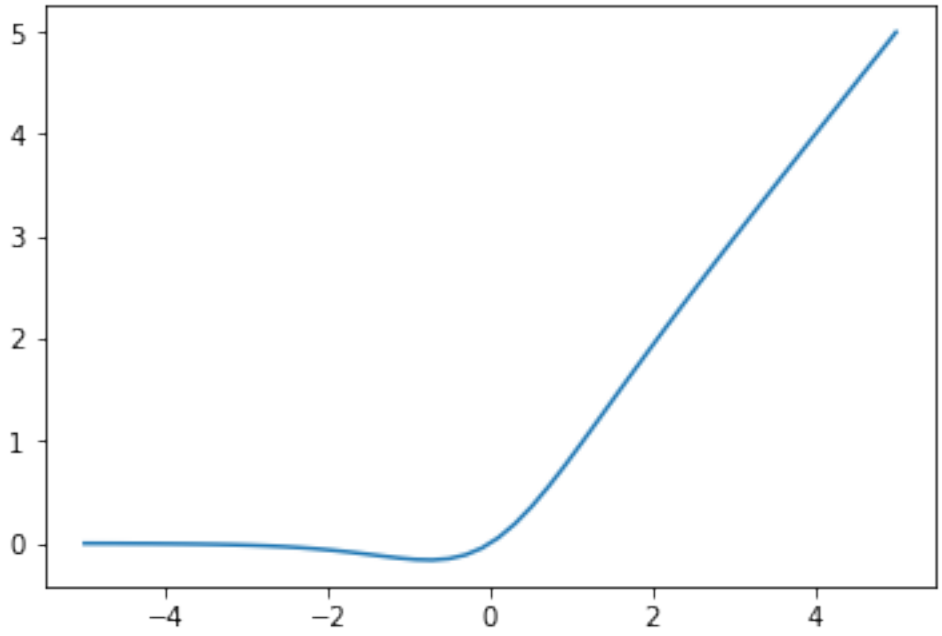
Gaussian Error Linear Unit. The name comes from the use of the Gaussian error function in the definition:

$$f(x) = x\Phi(x)$$

where $\Phi(x)$ is the CDF of the normal distribution.

It can be approximated as:

$$f(x) = x\sigma(1.702x)$$



This can be seen as a smoothed version of the ReLU.

Was found to improve performance on a variety of tasks compared to ReLU and ELU (Hendrycks and Gimpel (2016)). The authors speculate that the activation's curvature and non-monotonicity may help it to model more complex functions.

Proposed in

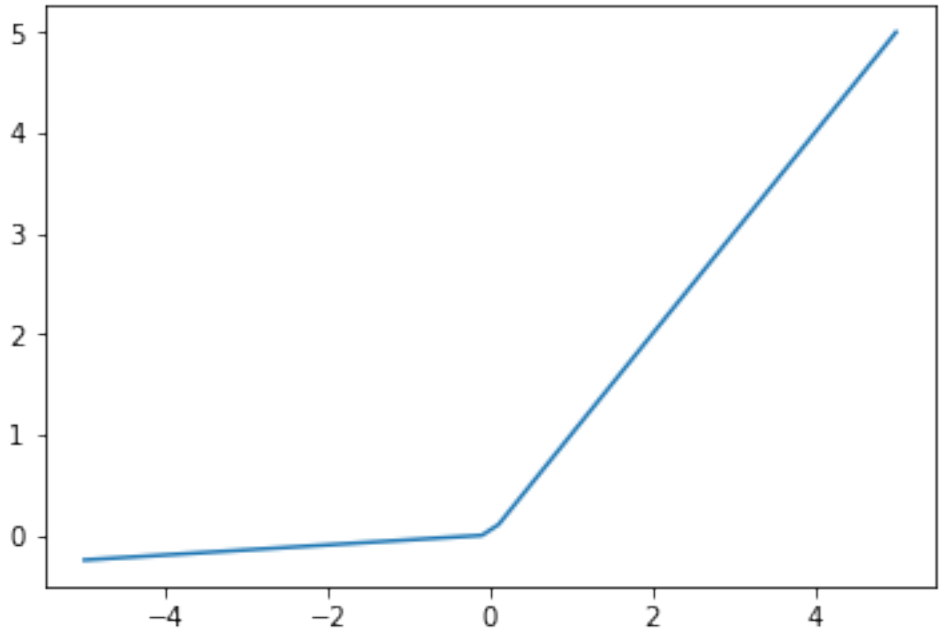
Gaussian Error Linear Units (GELUs), Hendrycks and Gimpel (2016)

1.12.4 LReLU

Leaky ReLU. Motivated by the desire to have gradients where the ReLU would have none but the gradients are very small and therefore vulnerable to the vanishing gradients problem in deep networks. The improvement in accuracy from using LReLU instead of ReLU has been shown to be very small (Maas et al. (2013)).

$$f(x) = \max\{ax, x\}$$

a is a fixed hyperparameter, unlike the PReLU. A common setting is 0.01.



Proposed in
Rectifier Nonlinearities Improve Neural Network Acoustic Models, Maas et al. (2013)

1.12.5 Maxout

An activation function designed to be used with dropout.

$$f(x) = \max_{j \in [1,k]} x^T W_j + b_j$$

where k is a hyperparameter.

Maxout can be a piecewise linear approximation for arbitrary convex activation functions. This means it can approximate ReLU, LReLU, ELU and linear activations but not tanh or sigmoid.

Was used to get state of the art performance on MNIST, SVHN, CIFAR-10 and CIFAR-100.

Proposed in
Maxout Networks, Goodfellow et al. (2013)

1.12.6 PReLU

Parametric ReLU.

$$f(x) = \max\{ax, x\}$$

Where a is a learned parameter, unlike in the Leaky ReLU where it is fixed.

Was used to achieve state of the art performance on ImageNet (He et al. (2015)).

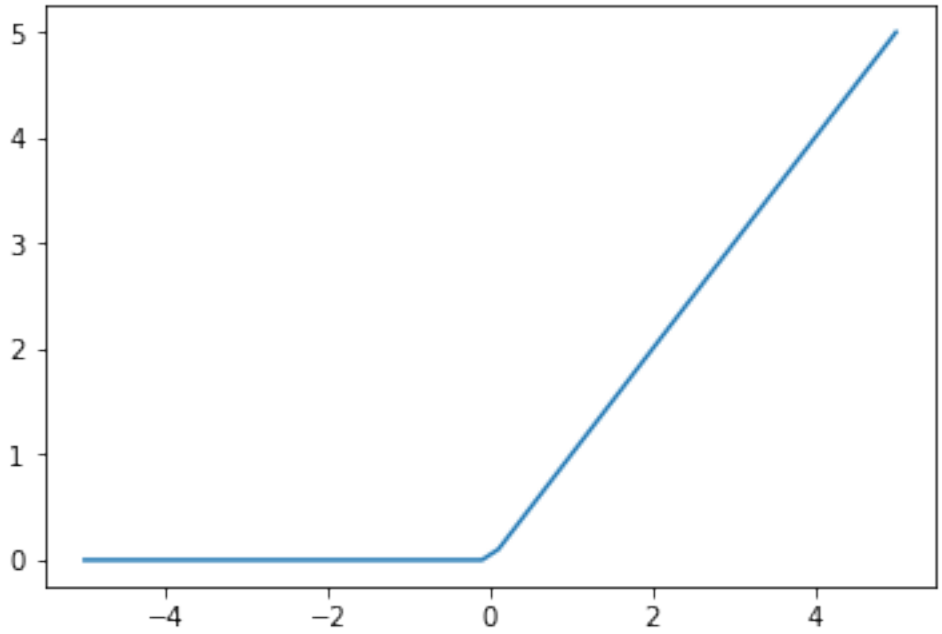
Proposed in

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification , He et al. (2015)

1.12.7 ReLU

Rectified Linear Unit. Unlike the sigmoid or tanh activations the ReLU does not saturate which has led to it being widely used in deep networks.

$$f(x) = \max\{0, x\}$$



The fact that the gradient is 1 when the input is positive means it does not suffer from vanishing and exploding gradients. However, it suffers from its own 'dying ReLU problem' instead.

The Dying ReLU Problem

When the input to a neuron is negative, the gradient will be zero. This means that gradient descent will not update the weights so long as the input remains negative. A smaller learning rate helps solve this problem.

The Leaky ReLU and the Parametric ReLU (PReLU) attempt to solve this problem by using $f(x) = \max\{ax, x\}$ where a is a small constant like 0.1. However, this small gradient when the input in negative means vanishing gradients are once again a problem.

Proposed in

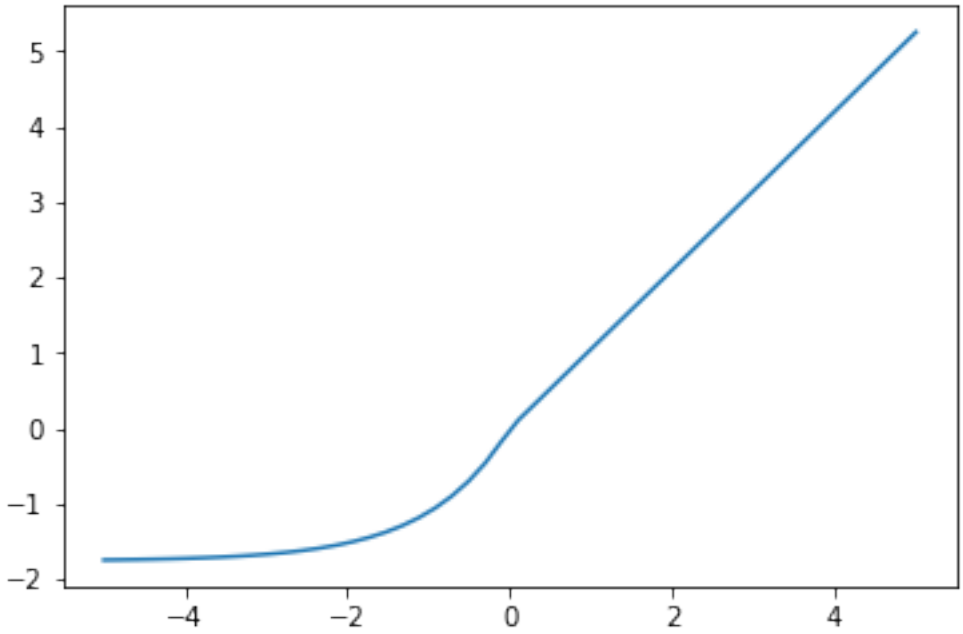
Rectified Linear Units Improve Restricted Boltzmann Machines, Nair and Hinton (2010)

1.12.8 SELU

Scaled Exponential Linear Unit.

$$f(x) = \lambda \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$

Where λ and α are hyperparameters, set to $\lambda = 1.0507$ and $\alpha = 1.6733$.



The SELU is designed to be used in networks composed of many fully-connected layers, as opposed to CNNs or RNNs, the principal difference being that CNNs and RNNs stabilize their learning via weight sharing. As with batch normalization, SELU activations give rise to activations with zero mean and unit variance but without having to explicitly normalize.

The ELU is a very similar activation. The only difference is that it has $\lambda = 1$ and $\alpha = 1$.

Initialisation

Klambauer et al. (2017) recommend initialising layers with SELU activations according to $\theta^{(i)} \sim N(0, \sqrt{1/n_i})$ where $\theta^{(i)}$ are the parameters for layer i of the network and n_i is the size of layer i of the network.

Dropout

Instead of randomly setting units to zero as in conventional dropout, the authors propose setting units to $\alpha' = -\lambda\alpha$ where λ and α are the hyperparameters given previously. They refer to this as **alpha dropout**.

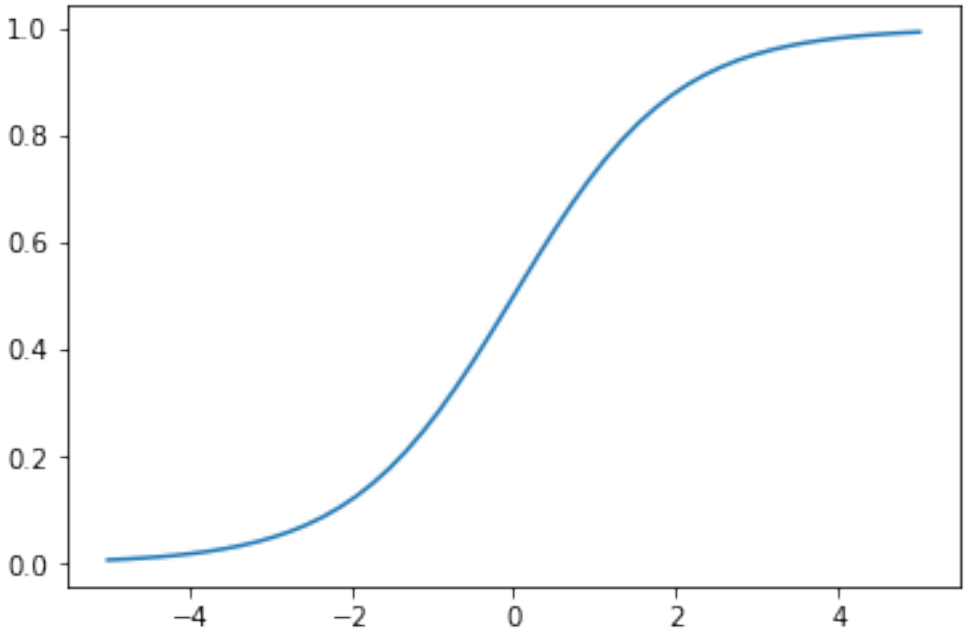
Proposed in

Self-Normalizing Neural Networks, Klambauer et al. (2017)

1.12.9 Sigmoid

Activation function that maps outputs to be between 0 and 1.

$$f(x) = \frac{e^x}{e^x + 1}$$



Has problems with saturation. This makes vanishing and exploding gradients a problem and initialization extremely important.

1.12.10 Softmax

All entries in the output vector are in the range (0,1) and sum to 1, making the result a valid probability distribution.

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, j \in 1, ..., K$$

Where x is a vector of length K. This vector is often referred to as the **logit**.

Unlike most other activation functions, the softmax does not apply the same function to each item in the input independently. The requirement that the output vector sums to 1 means that if one of the inputs is increased the others must decrease in the output.

The Softmax Bottleneck

A theorised problem that occurs when using the softmax to predict the next token in language modeling. It views language modeling as a matrix factorization problem:

$$HW^T = A$$

Where H is the contexts, W are the word vectors and A are the conditional probabilities for words given contexts. The vast number of contexts in language means that the matrix A is almost certainly high rank and so the dimensionality of the word embeddings is probably not sufficient to solve the matrix factorization problem adequately.

Mixture of Softmaxes

Mixture model intended to avoid the Softmax Bottleneck. The probability of a word x given some context c is the weighted average of k softmax distributions:

$$P(x|c) = \sum_{k=1}^{K} \pi_{ck} \frac{\exp h_{ck}^T w_x}{\sum_{x'} \exp h_{ck}^T w_{x'}} s.t. \sum_{k=1}^{K} \pi_{ck} = 1$$

where π_{ck} is the weight of component k.

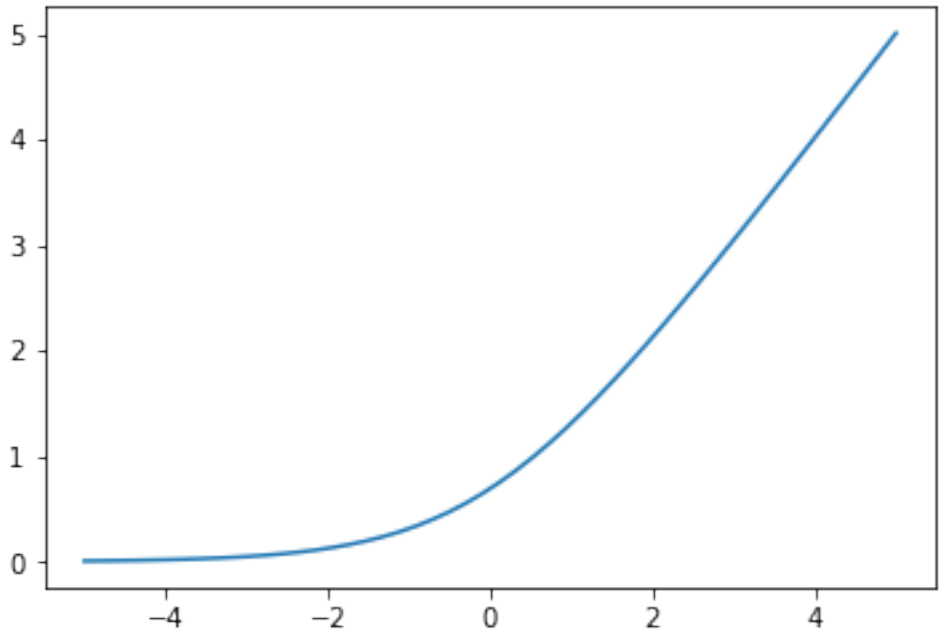
Proposed by

Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, Yang et al. (2017)

1.12.11 Softplus

Activation whose output is bounded between 0 and infinity, making it useful for modeling quantities that should never be negative such as the variance of a distribution.

$$f(x) = \log(1 + e^x)$$

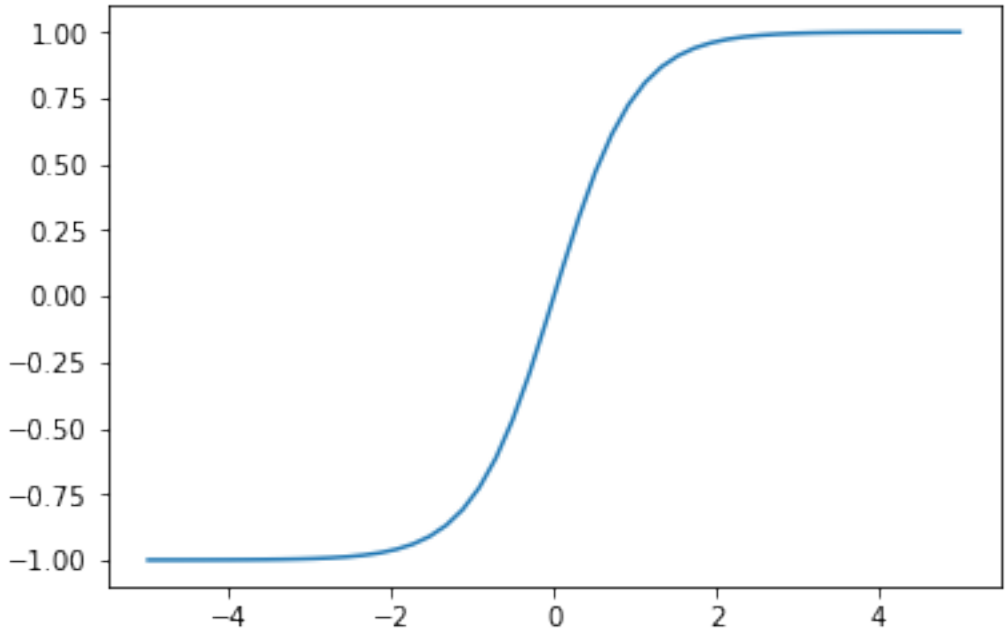


Unlike the ReLU, gradients can pass through the softmax when $x < 0$.

1.12.12 Tanh

Activation function that is used in the GRU and LSTM. It is between -1 and 1 and centered around 0, unlike the sigmoid.

$$f(x) = tanh(x)$$



Has problems with saturation like the sigmoid. This makes vanishing and exploding gradients a problem and initialization extremely important.

1.13 Convolutional networks

1.13.1 AlexNet

Performed considerably better than the state of the art at the time. Has 60 million parameters, 650,000 neurons and includes five convolutional layers.

The two 'streams' shown in the paper only exist to allow training on two GPUs.

Proposed in

ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

1.13.2 GoogLeNet

CNN that won the ILSVRC 2014 challenge. Composed of 9 inception layers.

Proposed in

Going Deeper with Convolutions, Szegedy et al. (2014)

1.13.3 LeNet5

A basic convolutional network, historically used for the MNIST dataset.

Proposed in
Gradient-based learning applied to document recognition, LeCun et al. (1998)

1.13.4 Residual network (ResNet)

An architecture that uses skip connections to create very deep networks. The original paper achieved 152 layers, 8 times deeper than VGG nets. Used for image recognition, winning first place in the ILSVRC 2015 classification task. Residual connections can also be used to create deeper RNNs such as Google's 16-layer RNN encoder-decoder (Wu et al., 2016).

Uses shortcut connections performing the identity mapping, which are added to the outputs of the stacked layers. Each residual block uses the equation:

$$x = f(x) + x$$

where f is a sequence of layers such as convolutions and nonlinearities.

Motivation

There are a number of hypothesized reasons for why residual networks are effective:

- Shorter paths: The skip connections provide short paths between the input and output, making residual networks able to avoid the vanishing gradient problem more easily.
- Increased depth: As a result of the reduced vanishing gradients problem ResNets can be trained with more layers, enabling more sophisticated functions to be learnt.
- Ensembling effect: Veit et al. (2016) demonstrate that a residual network can be seen as an ensemble of sub-networks of different lengths.

Comparison with Highway Networks

Highway Networks, Srivastava et al (2015) also use skip connections to attempt to make it easier to train very deep networks. In contrast to Residual Networks their connections are gated as follows:

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

Comparisons between the accuracies of the two approaches suggest the gating is not useful and so is detrimental overall as it increases the number of parameters and the computational complexity of the network.

Proposed in
Deep Residual Learning for Image Recognition, He et al. (2015)

1.13.5 VGG

A CNN that secured the first and second place in the 2014 ImageNet localization and classification tracks, respectively. VGG stands for the team which submitted the model, Oxford's Visual Geometry Group. The VGG model consists of 16–19 weight layers and uses small convolutional filters of size 3x3 and 1x1.

Proposed in
Very deep convolutional networks for large-scale image recognition, Simonyan and Zisserman (2015)

1.14 Embeddings

1.14.1 Distributed representation

A representation in which each number in a vector is used to store information about some attribute of an object. For example, brightness or size.

Distributed representations are much more powerful than one-hot representations. A one-hot vector of length n can store n states, whereas a distributed representation of the same length can store a number of states which is exponential in its length.

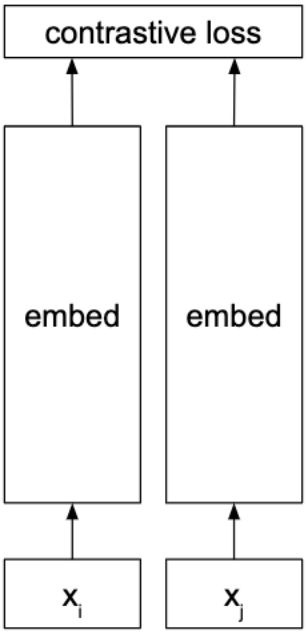
1.14.2 One-hot representation

A vector which has zeros everywhere except for in the indices representing the class or classes which are present.

1.14.3 Siamese network

An architecture that is often used for calculating similarities, as in face verification for example.

The network is trained with random pairs of inputs that are either positive (the examples are similar) or negative (they are not similar). Note that weights are shared between the two embedding sections.

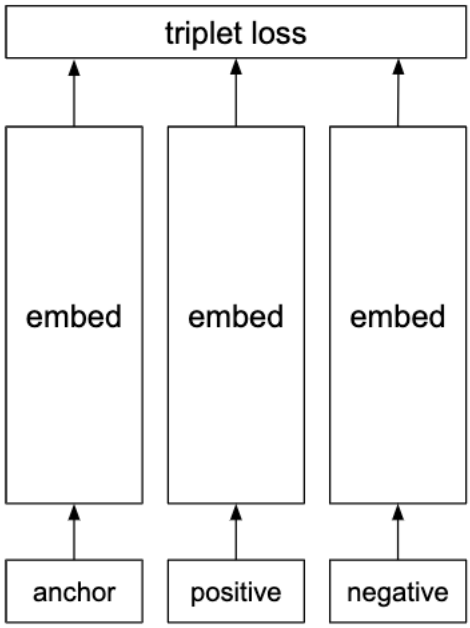


Often used with the contrastive loss.

1.14.4 Triplet network

Architecture for learning embeddings for calculating similarities. Useful for tasks like face verification.

During each iteration in training, an 'anchor' example is supplied along with a positive that is similar to it and a negative that is not. Each of the three inputs (the anchor, the positive and the negative) are processed separately to produce an embedding for each.



Note that the three embedding sections all share the same weights.

Uses the triplet loss.

1.14.5 Word vectors

The meaning of a word is represented by a vector of fixed size.

Polysemous words (words with multiple meanings) can be hard to model effectively with a single point if the dimensionality is too small.

CBOW (Continuous Bag of Words)

Used to create word embeddings. Predicts a word given its context. The context is the surrounding n words, as in the skip-gram model. Referred to as a bag of words model as the order of words within the window does not affect the embedding.

Several times faster to train than the skip-gram model and has slightly better accuracy for words which occur frequently.

Proposed in

Efficient Estimation of Word Representations in Vector Space, Mikolov et al. (2013)

GloVe

Method for learning word vectors. GloVe is short for 'Global Vectors'.

Unlike the CBOW and skip-gram methods which try to learn word vectors through a classification task (eg predict the word given its context), GloVe uses a regression task. The task is to predict the log frequency of word pairs given the similarity of their word vectors.

The loss function is:

$$L(X; w, b) = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log(X_{ij}))^2$$

where V is the size of the vocabulary and X_{ij} is the number of times word j occurs in the context of word i. $w_i^T w_j$ measures the similarity of the two word vectors.

$f(X_{ij})$ is a frequency weighting function. Below the threshold of x_{max} it gives more weight to frequently occuring pairs than rarer ones but beyond this all pairs are weighted equally. The function is defined as:

$$f(x) = \begin{cases} (x/x_{\text{max}})^{\alpha}, & x < x_{\text{max}} \\ 1, & \text{otherwise} \end{cases}$$

α and x_{max} are hyperparameters, set to 0.75 and 100 respectively.

https://nlp.stanford.edu/projects/glove/

Proposed in

GloVe: Global Vectors for Word Representation, Pennington et al. (2014)

Skip-gram

Word-embedding algorithm that works by predicting the context of a word given the word itself. The context is defined as other words appearing within a window of constant size, centered on the word.

For example, let the window size be 2. Then the relevant window is $\{w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}\}$. The model picks a random word $w_k \in \{w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}\}$ and attempts to predict w_k given w_i.

Increasing the window size improves the quality of the word vectors but also makes them more expensive to compute. Samples less from words that are far away from the known word, since the influence will be weaker. Works well with a small amount of data and can represent even rare words or phrases well.

Augmentations

The efficiency and quality of the skip-gram model is improved by two additions:

1. Subsampling frequent words. Words like 'the' and 'is' occur very frequently in most text corpora yet contain little useful semantic information about surrounding words. To reduce this inefficiency words are sampled according to $P(w_i) = 1 - t/f_i$ where f_i is the frequency of word i and t is a manually set threshold, usually around 10-5.
2. Negative sampling, a simplification of noise-contrastive estimation.

With some minor changes, skip-grams can also be used to calculate embeddings for phrases such as 'North Sea'. However, this can increase the size of the vocabulary dramatically.

Proposed in

Efficient Estimation of Word Representations in Vector Space, Mikolov et al. (2013)

Word2vec

The name of the implementation of the CBOW and skip-gram architectures in Mikolov et al. (2013)

https://code.google.com/archive/p/word2vec/

Efficient Estimation of Word Representations in Vector Space, Mikolov et al. (2013)

1.15 Generative networks

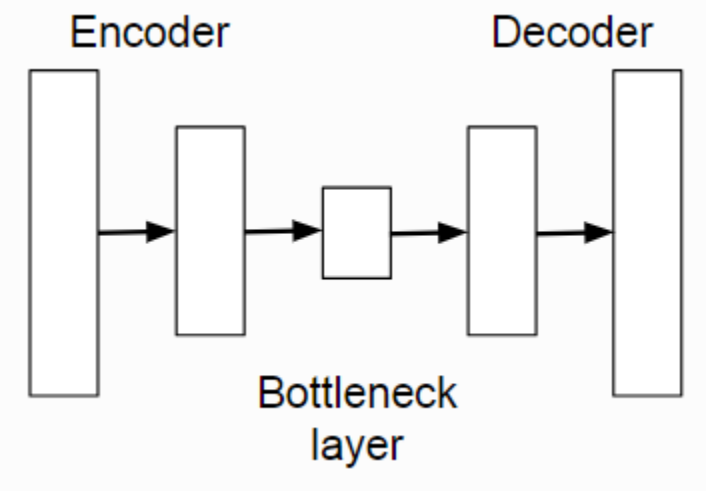
Models the joint distribution over the features $P(x; \theta)$, ignoring any labels.

The model can be estimated by trying to maximise the probability of the observations given the parameters. However, this can be close to intractable for cases like images where the number of possible outcomes is huge.

1.15.1 Autoencoders

A network for dimensionality reduction that can also be used for generative modelling.

In its simplest form, an autoencoder takes the original input (eg the pixel values of an image) and transforms them into a hidden layer with fewer features than the original. This 'bottleneck' means a compressed representation of the input. The part of the network which does this transformation is known as the encoder. The second part of an autoencoder is the decoder which takes the bottleneck layer and uses it to try and reconstruct the original input. This part is known as the decoder.



Autoencoders can be used as generative networks by sampling a new hidden state in the bottleneck layer and running it through the decoder.

Convolutional autoencoders

An autoencoder composed of standard convolutional layers and upsampling layers rather than fully connected layers.

Denoising Autoencoder (DAE)

Adds noise to prevent the hidden layer(s) from learning the identity function. This is particularly useful when the width of the narrowest hidden layer is at least as wide as the input layer.

Variational Autoencoder (VAE)

Unlike the standard autoencoder, the VAE can take noise as an input and use it to generate a sample from the distribution being modelled. 'Variational' refers to the Variational Bayes method which is used to approximate the true objective function with one that is more computable.

In order to modify the standard autoencoder to allow sampling, the distribution of the encoded image vectors is constrained to be roughly Normal(0,1). This means sampling can be done by sampling a random vector from N(0,1) and running the decoder on it.

There are two vectors outputted by the encoder, one for the mean and one for the variance. The closeness of these vectors to the unit Gaussian is measured by the KL-divergence.

The total loss is the sum of the reconstruction loss (mean squared error) and the KL-divergence:

$$L(y, \hat{y}) = \sum_i (y_i - \hat{y}_i)^2 + D_{KL}(N(\mu, \sigma)||N(0, 1))$$

where μ and σ are the mean and standard deviation of the encoding.

Proposed in

Auto-Encoding Variational Bayes, Kingma and Welling (2014)

Evidence-lower bound (ELBO)

A lower bound on the log probability of the data given the parameters. In a VAE this function is maximised instead of the true likelihood.

Reparameterization trick

A method for backpropagating through nodes in the graph that have random sampling. Suppose the first half of a network outputs μ and σ. We want to sample from the distribution they define and then compute the loss function on that sample.

We can rewrite $x \sim N(\mu, \sigma^2)$ as $x = \mu + \sigma \cdot \epsilon$ where $\epsilon \sim N(0, 1)$. This means the gradients no longer have to go through stochastic nodes in the graph.

Problems

- The use of the mean squared error means the network tends to produce blurry images. A GAN does not have this problem.
- The assumption of independence in the entries of the hidden vector may also contribute to poor results.

1.15.2 Autoregressive Networks

Unlike other generative models such as GANs or VAEs, these models generate their results sequentially. At each timestep they compute $x_i = \arg\max P(x|x_{i-1}, ..., x_1)$. The process is broadly the same as generating a sample of text using an RNN but can be used to generate images.

Autoregressive networks exploit the chain rule to express the joint probability as the product of conditional probabilities:

$$p(x) = \prod_{i=1}^{n} p(x_i|x_1, ..., x_{i-1})$$

PixelRNN

The model reads the image one pixel at a time and row by row, form the top left to the bottom right. Their best model used a 7 layer diagonal bidirectional LSTM with residual connections between the layers to ease training.

Pixels are modelled as being drawn from a discrete distribution with 256 values. The model has one 256-way output layer for each colour channel. When reading in the pixels, colour channels are handled sequentially so that the red channel is conditioned only on the previous pixels, the blue channel can use the red as well as the previous pixels and the green can use both the blue and red.

Proposed in

Pixel Recurrent Neural Networks, van den Oord et al. (2016)

PixelCNN

PixelCNN was also proposed in van den Oord et al. (2016) but the results were not as good as PixelRNN.

PixelCNN++ improves upon PixelCNN with a number of modifications, improving upon both it and PixelRNN. The modifications are:

- The 256-way softmax for each colour channel is replaced by a mixture of logistic distributions. This requires two parameters and one weight for each distribution, making it much more efficient given that only around 5 distributions are needed. The edge cases of 0 and 255 are handled specially.
- "Conditioning on whole pixels"
- Convolutions with a stride of 2 are used to downsample the image and effective increase the size of the convolutions' receptive fields.
- Residual connections are added between the convolutional layers. These help to prevent information being lost through the downsampling.
- Dropout is added on the model's residual connection to improve generalization.

Proposed in

Pixel Recurrent Neural Networks, van den Oord et al. (2016)

PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications, Salimans et al. (2016)

Further reading

Conditional Image Generation with PixelCNN Decoders, van den Oord et al. (2016b)

WaveNet

Proposed in

WaveNet: A Generative Model for Raw Audio, van den Oord et al. (2016)

Other papers

Neural Machine Translation in Linear Time, Kalchbrenner et al. (2017)

1.15.3 Energy-based Models

Also known as Undirected Graphical Models.

An energy function models the probability density. A model is learnt that minimises the energy for correct combinations of the variables and maximises it for incorrect ones. This function is minimised during inference.

The loss function is minimised during training. The energy function is a component of it.

A Tutorial on Energy-based Learning, LeCun (2006)

1.15.4 Generative Adversarial Network (GAN)

Unsupervised, generative image model. A GAN consists of two components; a generator, G which converts random noise into images and a discriminator, D which tries to distinguish between generated and real images. Here, 'real' means that the image came from the training set of images in contrast to the generated fakes.

Proposed in
Generative Adversarial Nets, Goodfellow et al. (2014)

Problems

- The training process can be unstable when trained solely with the adversarial loss as G can create images to confuse D that are not close to the actual image distribution. D will then learn to discriminate amongst these samples, causing G to create new confusing samples. This problem can be addressed by adding an L2 loss which penalizes a lack of similarity with the input distribution.
- Mode collapse. This is when the network stops generating certain classes (or more generally, modes). For example, it may only create 6's on MNIST.
- There is no way of telling how well it is doing except by manually inspecting the image outputs. This makes comparing different approaches difficult and early stopping impossible.

Notable variants

- A Style-Based Generator Architecture for Generative Adversarial Networks, Karras et al. (2018)
- Progressive Growing of GANs for Improved Quality, Stability, and Variation, Karras et al. (2017)
- Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, Zhu et al. (2017)
- BEGAN: Boundary Equilibrium Generative Adversarial Networks, Berthelot et al. (2017) - Gets similar quality results as the WGAN-GP.
- Improved Training of Wasserstein GANs, Gulrajani et al. (2017)
- Wasserstein GAN, Arjovsky et al. (2017) - Replaces the original loss function, improving stability. The WGAN-GP (2017) is a further improved version.

- InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets, Chen et al. (2016) - Is able to disentangle various aspects like pose vs lighting and digit shape vs writing style.
- Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, Radford et al. (2015) - Has a number of architectural improvements over the original GAN but is not fundamentally different.

Further reading

How to Train a GAN? Tips and tricks to make GANs work, Chintala (2016)
Fantastic GANs and where to find them part one and two by Guim Perarnau
The GAN Zoo
Are GANs Created Equal? A Large-Scale Study, Lucic et al. (2017)

1.16 Initialization

1.16.1 He initialization

The weights are drawn from the following normal distribution:

$$\theta^{(i)} \sim N(0, \sqrt{2/n_i})$$

where $\theta^{(i)}$ are the parameters for layer i of the network and n_i is the size of layer i of the network.

The biases are initialized to zero as usual.

Effectiveness

Was used to improve the state of the art for image classification (He et al., 2015) but the improvement over ReLU activations with Xavier initialization was very small, reducing top-1 error on ImageNet from 33.9% to 33.8%.

Proposed in

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification , He et al. (2015)

1.16.2 Initialization with zeros

All of the weights are initialised to zero. Used for bias vectors since the weight matrix, which is initialized with random weights, provides the symmetry breaking.

1.16.3 Orthogonal initialization

Initializes the weights as an orthogonal matrix. Useful for training very deep networks. Can be used to help with vanishing and exploding gradients in RNNs.

The procedure is as follows:

```
1. Generate a matrix of random numbers, X (eg from the normal distribution)
2. Perform the QR decomposition X = QR, resulting in an orthogonal matrix Q and an␣
↪upper triangular matrix R.
3. Initialise with Q.
```

Further reading

Explaining and illustrating orthogonal initialization for recurrent neural networks, Merity (2016)

LSUV initialization

Layer-sequential unit-variance initialization. An iterative initialization procedure:

```
1. t_max = 10
2. tol_var = 0.05
3. pre-initialize the layers with orthonormal matrices as proposed in Saxe et al.␣
↪(2013)
4. for each layer:
5.     let w be the weights of the layer
6.     let b be the output of the layer
7.     for i in range(t_max):
8.         w = w / sqrt(var(b))
9.         if abs(var(b) - 1) < tol_var:
10.            break
```

Proposed in

All you need is a good init, Mishkin and Matas (2015)

Orthonormal initialization

1. Initialise the weights from a standard normal distribution: $\theta_i \sim N(0, 1)$.
2. Perform a QR decomposition and use Q as the initialization matrix. Alternatively, do SVD and pick U or V as the initialization matrix.

Proposed in

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, Saxe et al. (2013)

Used by

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, Cho et al. (2014)

1.16.4 Symmetry breaking

An essential property of good initialization for fully connected layers. In a fully connected layer every hidden node has exactly the same set of inputs so if all nodes are initialised to the same value their gradients will also be identical. Thus they will never take on different values.

1.16.5 Xavier initialization

Sometimes referred to as Glorot initialization.

$$\theta^{(i)} \sim U(-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}})$$

where $\theta^{(i)}$ are the parameters for layer i of the network and n_i is the size of layer i of the network.

Xavier initialization's derivation assumes linear activations. Despite this it has been observed to work well in practice for networks that whose activations are nonlinear.

Proposed in

Understanding the difficulty of training deep feedforward neural networks, Glorot and Bengio (2010)

1.17 Layers

1.17.1 Affine layer

Synomym for fully-connected layer.

1.17.2 Attention

An attention layer takes a query vector and uses it, combined with key vectors, to compute a weighted sum of value vectors. If a key is determined to be highly compatible with the query the weight for the associated value will be high.

Attention has been used to improve image classification, image captioning, speech recognition, generative models and learning algorithmic tasks, but has probably had the largest impact on neural machine translation.

Computational complexity

Let n be the length of a sequence and d be the embedding size.

A recurrent network's complexity will be $O(nd^2)$.

A soft attention mechanism must look over every item in the input sequence for every item in the output sequence, resulting in complexity that is quadratic in the sequence length: $O(n^2d)$.

Additive attention

Let $x = \{x_1, ..., x_T\}$ be the input sequence and $y = \{y_1, ..., y_U\}$ be the output sequence.

There is an encoder RNN whose hidden state at index t we refer to as h_t. The decoder RNN's state at time t is s_t.

Attention is calculated over all the words in the sequence form a weighted sum, known as the context vector. This is defined as:

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$$

where α_{tj} is the jth element of the softmax of e_t.

The attention given to a particular input word depends on the hidden states of the encoder and decoder RNNs.

$$e_{tj} = a(s_{t-1}, h_j)$$

The decoder's hidden state is computed according to the following expression, where f represents the decoder.

$$s_i = f(s_{t-1}, y_{t-1}, c_t)$$

To predict the output sequence we take the decoder hidden state and the context vector and feed them into a fully connected softmax layer g which gives a distribution over the output vocabulary.

$$y_t = g(s_t, c_t)$$

Proposed in

Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et al. (2015)

Dot-product attention

Returns a weighted average over the values, V.

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V$$

Where Q is the query matrix, K is the matrix of keys and V is the matrix of values. $\text{softmax}(QK^T)$ determines the weight of each value in the result, based on the similarity between the query and the value's corresponding key.

The queries and keys have the same dimension.

The query might be the hidden state of the decoder, the key the hidden state of the encoder and the value the word vector at the corresponding position.

Scaled dot-product attention

Adds a scaling factor $\sqrt{d_k}$, equal to the dimension of K:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

This addition to the formula is intended to ensure the gradients do not become small when d_k grows large.

Proposed in

Attention is All You Need (2017)

Hard attention

Form of attention that attends only to one input, unlike soft attention. Trained using the REINFORCE algorithm since, unlike other forms of attention, it is not differentiable.

Self-attention

TODO

Soft attention

Forms of attention that attend to every input to some extent, meaning they can be trained through backpropagation. Contrast with hard attention, which attends exclusively to one input.

1.17.3 Convolutional layer

Transforms an image according to the convolution operation shown below, where the image on the left is the input and the image being created on the right is the output:

TODO

Let x be a matrix representing the image and k be another representing the kernel, which is of size NxN. $c(x, k)$ is the matrix that results from convolving them together. Then, formally, convolution applies the following formula:

$$c(x,k)_{ij} = \sum_{r=-M}^{M} \sum_{s=-M}^{M} x_{i+r,j+s} k_{r+M,s+M}$$

Where $M = (N-1)/2$.

Padding

Applying the kernel to pixels near or at the edges of the image will result in needing pixel values that do not exist. There are two ways of resolving this:

- Only apply the kernel to pixels where the operation is valid. For a kernel of size k this will reduce the image by $(k-1)/2$ pixels on each side.
- Pad the image with zeros to allow the operation to be defined.

Efficiency

The same convolution operation is applied to every pixel in the image, resulting in a considerable amount of weight sharing. This means convolutional layers are quite efficient in terms of parameters. Additionally, if a fully connected layer was used to represent the functionality of a convolutional layer most of its parameters would be zero since the convolution is a local operation. This further increases efficiency.

The number of parameters can be further reduced by setting a stride so the convolution operation is only applied every m pixels.

1x1 convolution

These are actually matrix multiplications, not convolutions. They are a useful way of increasing the depth of the neural network since they are equivalent to $f(hW)$, where f is the activation function.

If the number of channels decreases from one layer to the next they can be also be used for dimensionality reduction.

http://iamaaditya.github.io/2016/03/one-by-one-convolution/

Dilated convolution

Increases the size of the receptive field of the convolution layer.

Used in WaveNet: A Generative Model for Raw Audio, van den Oord et al. (2016).

Separable convolution/filter

A filter or kernel is separable if it (a matrix) can be expressed as the product of a row vector and a column vector. This decomposition can reduce the computational cost of the convolution. Examples include the Sobel edge detection and Gaussian blur filters.

$$K = xx^T, x \in \mathbb{R}^{n \times 1}$$

Transposed convolutional layer

Sometimes referred to as a deconvolutional layer. Can be used for upsampling.

Pads the input with zeros and then applies a convolution. Has parameters which must be learned, unlike the upsampling layer.

1.17.4 Dense layer

Synomym for fully-connected layer.

1.17.5 Fully-connected layer

Applies the following function:

$$h' = f(hW + b)$$

f is the activation function. h is the output of the previous hidden layer. W is the weight matrix and b is known as the bias vector.

1.17.6 Hierarchical softmax

A layer designed to improve efficiency when the number of output classes is large. Its complexity is logarithmic in the number of classes rather than linear, as for a standard softmax layer.

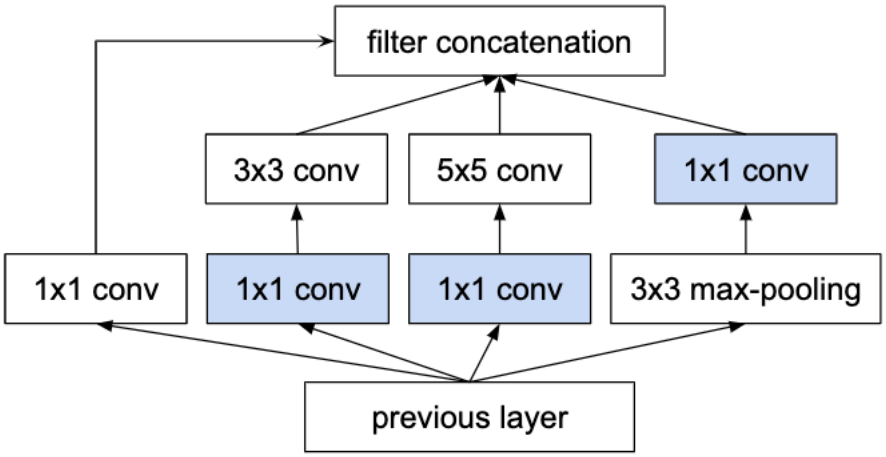
A tree is constructed where the leaves are the output classes.

Alternative methods include Noise Contrastive Estimation and Negative Sampling.

Classes for Fast Maximum Entropy Training, Goodman (2001)

1.17.7 Inception layer

Using convolutional layers means it is necessary to choose the kernel size (1x1, 3x3, 5x5 etc.). Inception layers negate this choice by using multiple convolutional layers with different kernel sizes and concatenating the results.



Inception layer with dimensionality reduction (blue backgrounds)

Padding can ensure the different convolution sizes still have the same size of output. The pooling component can be concatenated by using a stride of length 1 for the pooling.

5x5 convolutions are expensive so the 1x1 convolutions make the architecture computationally viable. The 1x1 convolutions perform dimensionality reduction by reducing the number of filters. This is not a characteristic necessarily found in all 1x1 convolutions. Rather, the authors have specified to have the number of output filters less than the number of input filters.

9 inception layers are used in GoogLeNet, a 22-layer deep network and state of the art solution for the ILSVRC in 2014.

Proposed in

Going deeper with convolutions, Szegedy et al. (2014)

1.17.8 Pooling layer

Max pooling

Transforms the input by taking the max along a particular dimension. In sequence processing this is usually the length of the sequence.

Mean pooling

Also known as average pooling. Identical to max-pooling except the mean is used instead of the max.

RoI pooling

Used to solve the problem that the regions of interest (RoI) identified by the bounding boxes can be different shapes in object recognition. The CNN requires all inputs to have the same dimensions.

The RoI is divided into a number of rectangles of fixed size (except at the edges). If doing 3x3 RoI pooling there will be 9 rectangles in each RoI. We do max-pooling over each RoI to get 3x3 numbers.

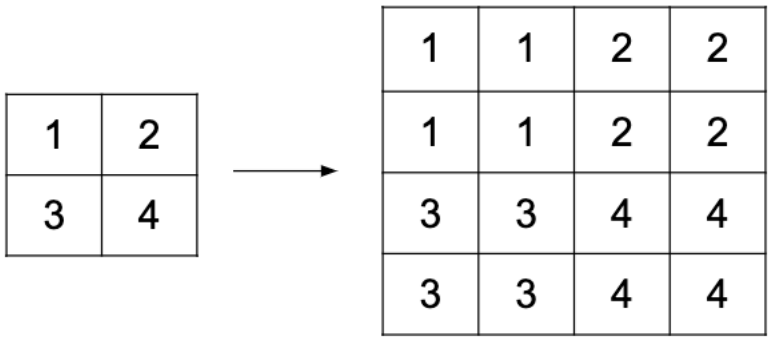
1.17.9 Softmax layer

A fully-connected layer with a softmax activation function.

1.17.10 Upsampling layer

Simple layer used to increase the size of its input by repeating its entries. Does not have any parameters.

Example of a 2D upsampling layer:



1.18 Loss functions

For classification problems, y is equal to 1 if the example is a positive and 0 if it is a negative. $\hat{y}$ can take on any value (although predicting outside of the (0,1) interval is unlikely to be useful).

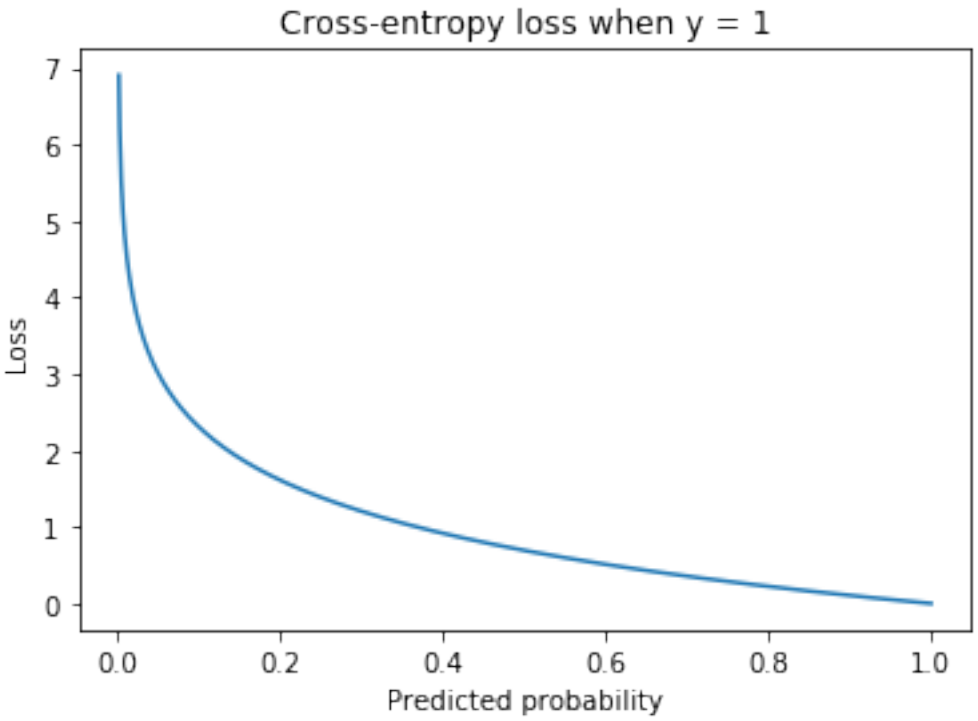
1.18.1 Classification

Cross-entropy loss

Loss function for classification.

$$L(y, \hat{y}) = -\sum_{i}\sum_{c} y_{i,c} \log(\hat{y}_{i,c})$$

where c are the classes. $y_{i,c}$ equals 1 if example i is in class c and 0 otherwise. $\hat{y}_{i,c}$ is the predicted probability that example i is in class c.



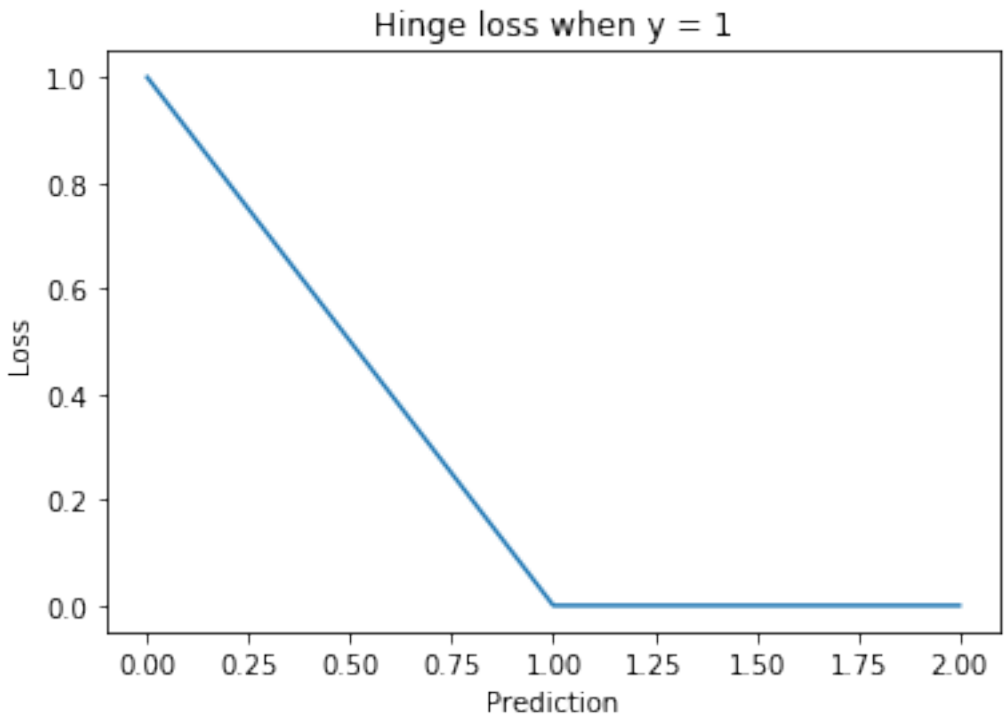
For discrete distributions (ie classification problems rather than regression) this is the same as the negative log-likelihood loss.

Hinge loss

Let positives be encoded as $y = 1$ and negatives as $y = -1$. Then the hinge loss is defined as:

$$L(y, \hat{y}) = \max\{0, m - y\hat{y}\}$$

The margin m is a hyperparameter that is commonly set to 1.



Used for training SVMs.

Focal loss

Variant of the cross-entropy loss, designed for use on datasets with severe class imbalance. It is defined as:

$$L(p) = -(1-p)^{\gamma} \log(p)$$

Where γ is a hyperparameter that determines the relative importance of the classes. If $\gamma = 0$ the focal loss is equivalent to the cross-entropy loss.

Proposed in
Focal Loss for Dense Object Detection, Lin et al. (2017)

Noise Contrastive Estimation

Like negative sampling, this is a technique for efficient learning when the number of output classes is large. Useful for language modelling.

A binary classification task is created to disambiguate pairs that are expected to be close to each other from 'noisy' examples put together at random.

In essence, rather than estimating $P(y|x)$, NCE estimates $P(C = 1|x, y)$ where $C = 1$ if y has been sampled from the real distribution and $C = 0$ if y has been sampled from the noise distribution.

NCE makes training time at the output layer independent of the number of classes. It remains linear in time at evaluation, however.

$$L(x, y) = -\sum_{i} \log(P(C_i = 1|x_i, y_i)) + \sum_{j=1}^{k} \log(1 - P(C_i = 1|x_i, y_j^n))$$

k is a hyperparameter, denoting the number of noise samples for each real sample. y_i is a label sampled from the data distribution and y_j^n is one sampled from the noise distribution. $C_i = 1$ if the pair (x, y) was drawn from the data distribution and 0 otherwise.

Used in

Noise Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models, Gutmann and Hyvarinen (2010)

Learning Word Embeddings Efficiently with Noise Contrastive Estimation, Mnih and Kavukcuoglu (2013)

RNNLM Training with NCE for Speech Recognition, Chen et al. (2015)

1.18.2 Embeddings

Contrastive loss

Loss function for learning embeddings, often used in face verification.

The inputs are pairs of examples x_1 and x_2 where $y = 1$ if the two examples are of the similar and 0 if not.

$$L(x_1, x_2, y) = yd(x_1, x_2)^2 + (1 - y)\max\{0, m - d(x_1, x_2)\}^2$$

Where x_1 and x_2 are the embeddings for the two examples and m is a hyperparameter called the margin. $d(x, y)$ is a distance function, usually the Euclidean distance.

Intuition

If $y = 1$ the two examples x_1 and x_2 are similar and we want to minimize the distance $d(x_1, x_2)$. Otherwise ($y = 0$) we wish to maximize it.

The margin

If $y = 0$ we want to make $d(x_1, x_2)$ as large as possible to minimize the loss. However, beyond the threshold for classifying the example as a negative increasing this distance will not have any effect on the accuracy. The margin ensures this intuition is reflected in the loss function. Using the margin means increasing $d(x_1, x_2)$ beyond m has no effect.

There is no margin for when $y = 1$. This case is naturally bounded by 0 as the Euclidean distance cannot be negative.

Example paper

Deep Learning Face Representation by Joint Identification-Verification, Sun et al. (2014)

Negative sampling

The problem is reframed as a binary classification problem.

$$L(x_0, x_1, y) = y \log \sigma(f(x_0) \cdot f(x_1)) + (1 - y_i) \log(\sigma(-f(x_0) \cdot f(x_1)))$$

where x_0 and x_1 are two examples, f is the learned embedding function and $y = 1$ if the pair (x_0, x_1) are expected to be similar and $y = 0$ otherwise. The dot product measures the distance between the two embeddings.

Noise Contrastive Estimation

A binary classification task is created to disambiguate pairs that are expected to be close to each other from 'noisy' examples put together at random.

$$L(x_0, x_1, y) = y \log \sigma(f(x_0) \cdot f(x_1)) + (1 - y_i) \log(1 - \sigma(f(x_0) \cdot f(x_1)))$$

where x_0 and x_1 are two examples, f is the learned embedding function and $y = 1$ if the pair (x_0, x_1) are expected to be similar and $y = 0$ if not (because they have been sampled from the noise distribution). The dot product measures the distance between the two embeddings and the sigmoid function transforms it to be between 0 and 1 so it can be interpreted as a prediction for a binary classifier.

This means maximising the probability that actual samples are in the dataset and that noise samples aren't in the dataset. Parameter update complexity is linear in the size of the vocabulary. The model is improved by having more noise than training samples, with around 15 times more being optimal.

1.18.3 Triplet loss

Used for training embeddings with triplet networks. A triplet is composed of an anchor (a), a positive example (p) and a negative example (n). The positive examples are similar to the anchor and the negative examples are dissimilar.

$$L(a, p, n) = \sum_i \max\{0, m + d(a_i, p_i) - d(a_i, n_i)\}$$

Where m is a hyperparameter called the margin. $d(x, y)$ is a distance function, usually the the Euclidean distance.

The margin

We want to minimize $d(a_i, p_i)$ and maximize $d(a_i, n_i)$. The former is lower-bounded by 0 but the latter has no upper bound (distances can be arbitrarily large). However, beyond the threshold to classify a pair as a negative, increasing this distance will not help improve the accuracy, a fact which needs to be reflected in the loss function. The margin does this by ensuring that there is no gain from increasing $d(a_i, n_i)$ beyond $m + d(a_i, p_i)$ since the loss will be set to 0 by the maximum.

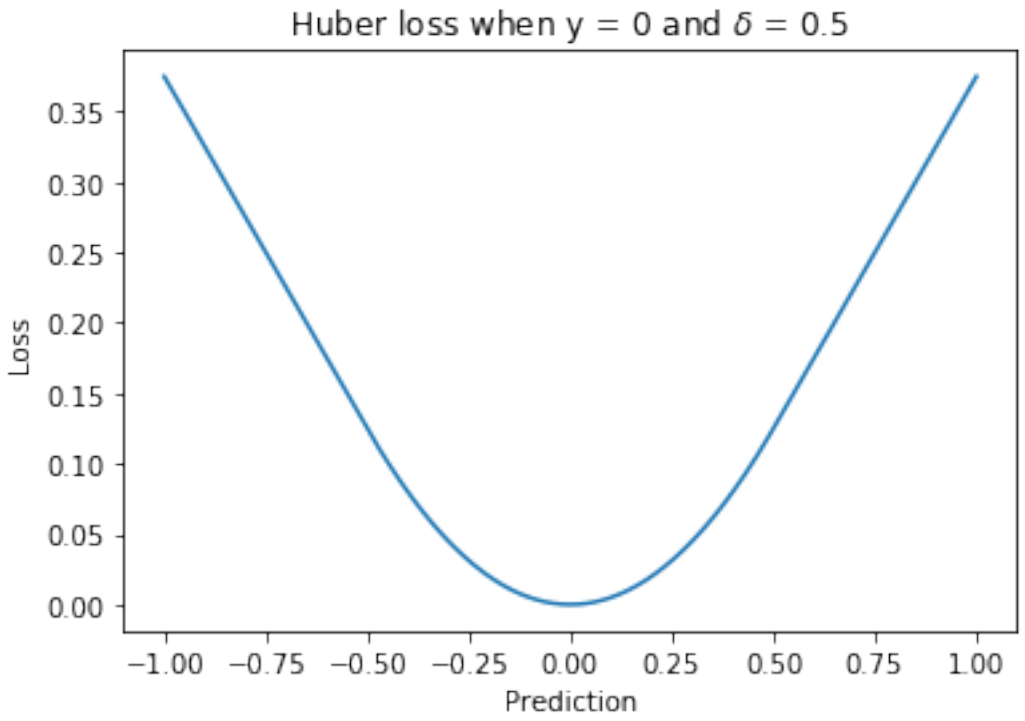
1.18.4 Regression

Huber loss

A loss function used for regression. It is less sensitive to outliers than the squared loss since there is only a linear relationship between the size of the error and the loss beyond δ.

$$L(y, \hat{y}; \delta) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta(|y - \hat{y}| - \frac{1}{2}\delta), & \text{otherwise} \end{cases}$$

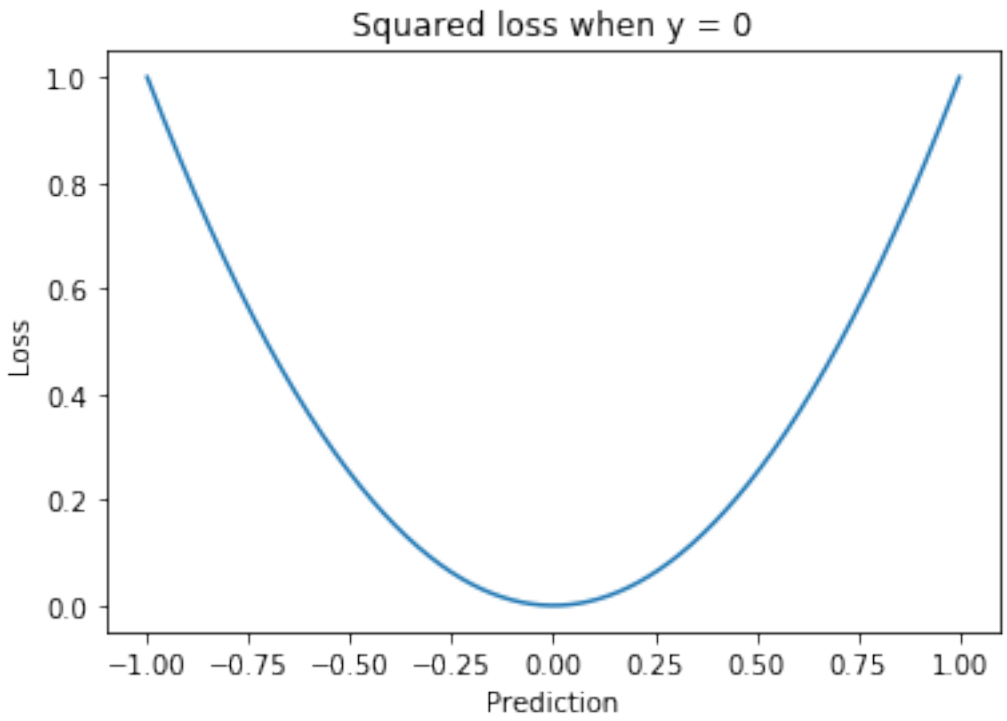
Where δ is a hyperparameter.



Squared loss

A loss function used for regression.

$$L(y, \hat{y}) = \sum_i (y_i - \hat{y}_i)^2$$



Disadvantages

The squaring means this loss function weights large errors more than smaller ones, relative to the magnitude of the error. This can be particularly harmful in the case of outliers. One solution is to use the Huber loss.

1.19 Normalization

1.19.1 Batch normalization

Normalizes the input vector to a layer to have zero mean and unit variance, making training more efficient. Training deep neural networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization. This phenomenon is referred to as internal covariate shift.

Adding β to the normalized input and scaling it by γ ensures the model does not lose representational power as a result of the normalization.

Batch Normalization is often found to improve generalization performance (Zhang et al. (2016)).

Proposed in

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015)

Training

The batch-normalized version of the inputs, $x \in \mathbb{R}^{n \times d}$, to a layer is:

$$BN(x) = \gamma \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} + \beta$$

Where γ and β are learned and ϵ is a small hyperparameter that prevents division by zero. If there are multiple batch normalization layers a separate γ and β will be learned for each of them.

$\mu_x \in \mathbb{R}^d$ and $\sigma_x^2 \in \mathbb{R}^d$ are moving averages of the mean and variance of x. They do not need to be learned. The moving averages are calculated independently for each feature in x.

Batch normalization does not work well with small batch sizes (Wu and He, 2018). Small batches cause the statistics to become inaccurate. This can cause problems when training models with large images where large batches will not fit in memory.

Inference

Batch normalization's stabilizing effect is helpful during training but unnecessary at inference time. Therefore, once the network is trained the population mean and variance are used for normalization, rather than the batch mean and variance. This means the networks output can depend only on the input, not also on other examples in the batch.

Application to RNNs

Batch normalization is difficult to apply to RNNs since it requires storing the batch statistics for every time step in the sequence. This can be problematic if a sequence input during inference is longer than those seen during training.

Coojimans et al. (2016) propose a variant of the LSTM that applies batch normalization to the hidden-to-hidden transitions.

Recurrent Batch Normalization, Coojimans et al. (2016)

Conditional batch normalization

The formula is exactly the same as normal batch normalization except γ and β are not learned parameters, but rather the outputs of functions.

Was used to achieve state of the art results on the CLEVR visual reasoning benchmark.

Learning Visual Reasoning Without Strong Priors, Perez et al. (2017)

1.19.2 Feature normalization

This class of normalizations refers to methods that transform the inputs to the model, as opposed to the activations within it.

Feature scaling

Scaling the inputs to the network with a linear transformation. Examples include min-max and z-score normalization.

Feature scaling is motivated by practical considerations in optimization.

Min-max normalization

Rescales the features so they have a specified minimum and maximum.

To rescale to between a and b:

$$x_{ij} := \frac{(x_{ij} - min_j x_{ij})(b-a)}{max_j x_{ij} - min_j x_{ij}}$$

When computing the min and max be sure to use only the training data, as opposed to calculating these statistics on the entire dataset.

Principal Component Analysis (PCA)

Decomposes a matrix $X \in \mathbb{R}^{n \times m}$ into a set of k orthogonal vectors. The matrix X represents a dataset with n examples and m features.

Method for PCA via eigendecomposition:

1. Center the data by subtracting the mean for each dimension.
2. Compute the covariance matrix on the centered data $C = (X^T X)/(n-1)$.
3. Do eigendecomposition of the covariance matrix to get $C = Q \Lambda Q^*$.
4. Take the k largest eigenvalues and their associated eigenvectors. These eigenvectors are the 'principal components'.
5. Construct the new matrix from the principal components by multiplying the centered X by the truncated Q.

PCA can also be done via SVD.

Whitening

The process of transforming the inputs so that they have zero mean and a covariance matrix which is the identity. This means the features will be linearly uncorrelated with each other and have variances equal to 1.

Examples:

- PCA
- ZCA

ZCA

Like PCA, ZCA converts the data to have zero mean and an identity covariance matrix. Unlike PCA, it does not reduce the dimensionality of the data and tries to create a whitened version that is minimally different from the original.

Z-score normalization

The features are transformed by subtracting their mean and dividing by their standard deviation:

$$x_{ij} := \frac{x_{ij} - \mu_i}{\sigma_i}$$

where x_{ij} is the jth instance of feature i and μ_i and σ_i are the mean and standard deviation of feature x_i respectively.

Ensure that the mean and standard deviation are calculated on the training set, not on the entire dataset.

1.19.3 Group normalization

Group normalization implements the same formula as batch normalization but takes the average over the feature dimension(s) rather than the batch dimension. This means it can be used with small batch sizes, unlike batch normalization, which is useful for many computer vision applications where memory-consuming high resolution images naturally restrict the batch size.

$$GN(x) = \gamma \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}} + \beta$$

Where γ and β are learned and ϵ is a small hyperparameter that prevents division by zero. Separate gamma and beta are learned for each group normalization layer. β and γ make sure the model does not lose any representational power from the normalization.

Proposed in
Group Normalization, Wu and He. (2018)

1.19.4 Layer normalization

Can be easily applied to RNNs, unlike batch normalization.

If the hidden state at time t of an RNN is given by:

$$h_t = f(Wx_t + b) = f(a_t + b)$$

Then the layer normalized version is:

$$h_t = f(\frac{g}{\sigma_t} * (a - \mu_t) + b)$$

where μ_t and σ_t are the mean and variance of a_t.

Proposed in
Layer Normalization, Ba et al. (2016)

Used in
Attention is All You Need, Vaswani et al. (2017)

1.19.5 Weight normalization

The weights of the network are reparameterized as:

$$w = \frac{g}{||v||} v$$

where g is a learnt scalar and v is a learnt vector.

This guarantees that $||w|| = g$ without the need for explicit normalization.

Simple to use in RNNs, unlike batch normalization.

Unlike batch normalization, weight normalization only affects the weights - it does not normalize the activations of the network.

Proposed in
Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, Salimans and Kingma (2016)

1.20 Optimization

1.20.1 Automatic differentiation

Has two distinct modes - forward and reverse.

Forward mode takes an input to the graph and evaluates the derivative of all subsequent nodes with respect to it.

Reverse mode takes an output (eg the loss) and differentiates it with respect to all inputs. This is usually more useful in neural networks since it can be used to get the derivatives for all the parameters in one pass.

1.20.2 Backpropagation

Naively summing the product of derivatives over all paths to a node is computationally intractable because the number of paths increases exponentially with depth.

Instead, the sum over paths is calculated by merging paths back together at the nodes. Derivatives can be computed either forward or backward with this method.

Further reading
Calculus on Computational Graphs: Backpropagation, Olah (2015)

Backpropagation through time (BPTT)

Used to train RNNs. The RNN is unfolded through time.

When dealing with long sequences (hundreds of inputs), a truncated version of BPTT is often used to reduce the computational cost. This stops backpropagating the errors after a fixed number of steps, limiting the length of the dependencies that can be learned.

1.20.3 Batch size

Pros of large batch sizes:

- Decreases the variance of the updates, making convergence more stable. From this perspective, increasing the batch size has very similar effects to decreasing the learning rate.
- Matrix computation is more efficient.

Cons of large batch sizes:

- Very large batch sizes may not fit in memory.
- Smaller number of updates for processing the same amount of data, slowing training.
- Hypothesized by Keskar et al. (2016) to have worse generalization performance since they result in sharper local minima being reached.

Further reading

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, Keskar et al. (2016)
Coupling Adaptive Batch Sizes with Learning Rates, Balles et al. (2016)
Big Batch SGD: Automated Inference using Adaptive Batch Sizes, De et al. (2016)

1.20.4 Curriculum learning

Training the classifier with easy examples initially and gradually transitioning to the harder ones. Useful for architectures which are very hard to train.

1.20.5 Depth

Depth increases the representational power of a network exponentially, for a given number of parameters. However, deeper networks can also be considerably harder to train, due to vanishing and exploding gradients or dying ReLUs. Problems stemming from depth are seen both in deep feedforward networks and in recurrent networks, where the depth comes from being unfolded over a large number of timesteps.

Potential solutions include:

- Using a smaller learning rate
- Skip connections
- Batch normalization
- Memory cells. Used in the Neural Turing Machine for learning long dependencies.
- Auxiliary loss functions (eg Szegedy et al. (2016))
- Orthogonal initialization

1.20.6 Distributed training

Training a neural network using multiple machines in parallel. Unfortunately gradient descent is a naturally sequential process since each iteration uses the parameters from the previous one. This means distributed training requires algorithms written specifically for that purpose.

Distributed training methods are broadly classified as being either synchronous or asynchronous, more details of which are given in their sections below.

Asynchronous SGD

Each worker processes a batch of data and computes the gradients. A central server holds the model parameters. The workers fetch the parameters from the parameter server and use them to compute gradients, which are then sent to the parameter server so the weights can be updated.

It is likely that while a worker is computing gradients other worker(s) have already finished their gradients and used them to update the parameters. Therefore the update can be several steps out-of-date when the gradient is finally computed. This problem is more severe the more workers there are.

Example papers

Project Adam: Building an Efficient and Scalable Deep Learning Training System , Chilimbi et al. (2014)
Massively Parallel Methods for Deep Reinforcement Learning, Nair et al. (2015)

Large Scale Distributed Deep Networks, Dean et al. (2012)

Synchronous SGD

Gradients are accumulated from the workers and summed before updating the network parameters.

Parameter updates can only occur once all the workers have computed their gradients which can slow down learning, unlike in asynchronous SGD. The whole system is limited to the speed of the lowest worker.

Means using larger batch sizes. This could be counteracted by reducing the batch sizes on each of the workers but this would reduce efficiency.

Example papers
Revisiting Distributed Synchronous SGD, Chen et al. (2016)

1.20.7 Early stopping

Halting training when the validation loss has stopped decreasing but the training loss is still going down.

1.20.8 End-to-end

The entire model is trained in one process, not as separate modules. For example, a pipeline consisting of object recognition and description algorithms that are trained individually would not be trained end-to-end.

1.20.9 Epoch

A single pass through the training data.

1.20.10 Error surface

The surface obtained by plotting the weights of the network against the loss. For a linear network with a squared loss function, the surface is a quadratic bowl.

1.20.11 Exploding gradient problem

When the gradient grows exponentially as we move backward through the layers.

Gradient clipping can be an effective antidote.

Further reading
On the difficulty of training recurrent neural networks, Pascanu et al. (2012)

1.20.12 Gradient clipping

Used to avoid exploding gradients in very deep networks by normalizing the gradients of the parameter vector. Clipping can be done either by value or by norm.

Clipping by value

$$g_i = \min\{a, \max\{b, g_i\}\}$$

Where g_i is the gradient of the parameter θ_i and a and b are hyperparameters.

Clipping by norm

$$g_i = g_i * a / ||g||_2$$

Where g_i is the gradient of the parameter θ_i and a is a hyperparameter.

On the difficulty of training recurrent neural networks, Pascanu et al. (2012)

1.20.13 Learning rate

Pros of large learning rates:

- Training is faster if the large learning rate does not cause problems.
- Lowers the risk of overfitting.

Cons of large learning rates:

- Increases the risk of oscillations during training, especially when not using an optimizer with a momentum term.
- Can make it harder to train deeper networks.

Learning rate decay

Also known as learning rate annealing. Changing the learning rate throughout the training process according to some schedule.

Cosine learning rate decay

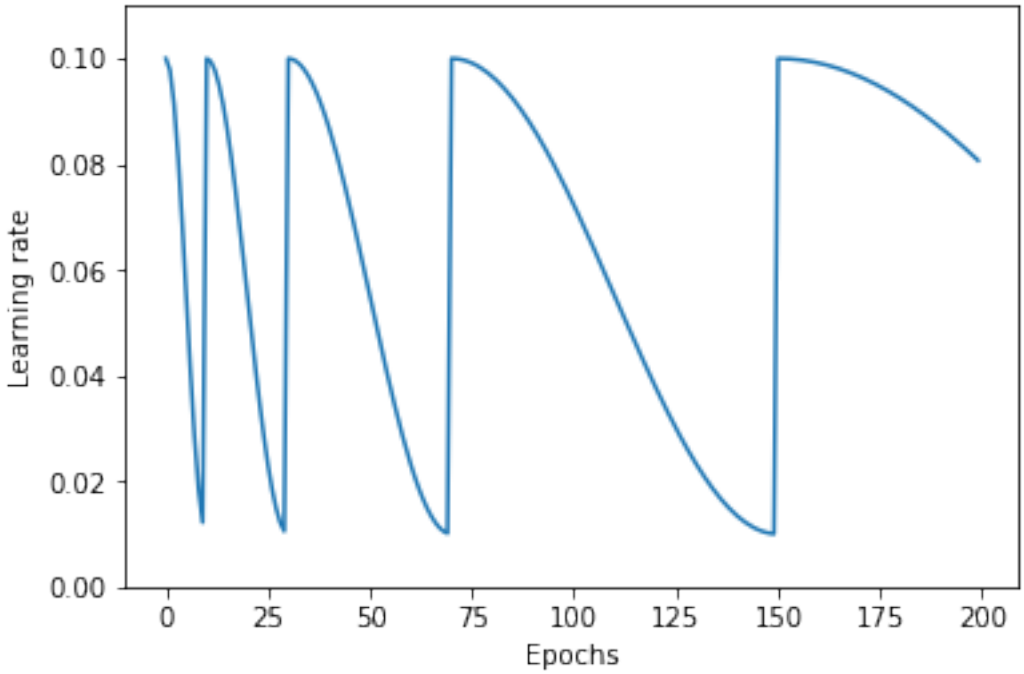
The learning rate decays according to a cosine function but is reset to its maximum value once its minimum is reached. The equation for the learning rate at epoch t is:

$$\eta_t = \eta_{\text{min}} + \frac{1}{2}(\eta_{\text{max}} - \eta_{\text{min}})(1 + \cos(\frac{T_{\text{cur}}}{T_i}\pi))$$

where T_i is the number of epochs between warm restarts and T_{cur} is the number of epochs that have been performed since the last warm restart. The learning rate fluctuates between η_{max} and η_{min}.

Multiplying T_i by $T_{\text{mult}} > 1$ after every restart was found to increase performance.

The graph below shows cosine learning rate decay with $T_i = 10$, $T_{\text{mult}} = 2$, $\eta_{\text{max}} = 0.1$ and $\eta_{\text{min}} = 0.01$:



Was shown (Loschilov and Hutter (2016)) to increase accuracy on CIFAR-10 and CIFAR-100 compared to the conventional approach of decaying the learning rate monotonically with a step function.

Note that warm restarts can temporarily make the model's performance worse. The best model can usually be found when the learning rate is at its minimum.

The following Python code shows how to implement cosine learning rate decay:

```
t_i = 10 # number of epochs between warm restarts.
t_mult = 2 # double t_i at every restart. set to 1 to ignore.
t_cur = 0 # how many epochs have been performed since the last restart.

min_lr = 0.01
max_lr = 0.1

for epoch in range(num_epochs):
    # warm restart
    if epoch > 0 and t_cur == t_i:
        t_cur = 0
        t_i *= t_mult

    lr = min_lr + 0.5 * (max_lr - min_lr) * (1 + np.cos(np.pi * t_cur / t_i))
    t_cur += 1
```

Proposed in

SGDR: Stochastic Gradient Descent with Warm Restarts, Loschilov and Hutter (2016)

1.20.14 Momentum

Adds a fraction of the update from the previous time step to the current time step. The parameter update at time t is given by:

$$u_t = -\alpha v_t$$

$$v_t = \rho v_{t-1} + g_t$$

Deep architectures often have deep ravines in their landscape near local optimas. They can lead to slow convergence with vanilla SGD since the negative gradient will point down one of the steep sides rather than towards the optimum. Momentum pushes optimization to the minimum faster. Commonly set to 0.9.

Further reading
Why Momentum Really Works, Goh (2017)

1.20.15 Optimizers

AdaDelta

AdaDelta is a gradient descent based learning algorithm that adapts the learning rate per parameter over time. It was proposed as an improvement over AdaGrad, which is more sensitive to hyperparameters and may decrease the learning rate too aggressively.

Proposed in
AdaDelta: An Adaptive Learning Rate Method, Zeiler (2012)

AdaGrad

Proposed in
Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, Duchi et al. (2011)

Adam

Adam is an adaptive learning rate algorithm similar to RMSProp, but updates are directly estimated using EMAs of the first and uncentered second moment of the gradient. Designed to combine the advantages of RMSProp and AdaGrad. Does not require a stationary objective and works with sparse gradients. Is invariant to the scale of the gradients.

Has hyperparameters α, β_1, β_2 and ϵ.

The biased first moment (mean) estimate at iteration t:

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

The biased second moment (variance) estimate at iteration t:

$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias correction for the first and second moment estimates:

$$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$$

$$\hat{v}_t \leftarrow v_t/(1-\beta_2^t)$$

The bias correction terms counteracts bias caused by initializing the moment estimates with zeros which makes them biased towards zero at the start of training.

Update the parameters of the network:

$$\theta_t \leftarrow \theta_{t-1} - \alpha\hat{m}_t/(\sqrt{\hat{v}_t}+\epsilon))$$

This can be interpreted as a signal-to-noise ratio, with the step-size increasing when the signal is higher, relative to the noise. This leads to the step-size naturally becoming smaller over time. Using the square root for the variance term means it can be seen as computing the EMA of $g/|g|$. This reduces the learning rate when the gradient is a mixture of positive and negative values as they cancel out in the EMA to produce a number closer to 0.

Proposed in

Adam: A Method for Stochastic Optimization, Kingma et al. (2015)

Averaged SGD (ASGD)

Runs like normal SGD but replaces the parameters with their average over time at the end.

BFGS

Iterative method for solving nonlinear optimization problems that approximates Newton's method. BFGS stands for Broyden–Fletcher–Goldfarb–Shanno. L-BFGS is a popular memory-limited version of the algorithm.

Conjugate gradient

Iterative algorithm for solving SLEs where the matrix is symmetric and positive-definite.

Coordinate descent

Minimizes a function by adjusting the input along only one dimension at a time.

Krylov subspace descent

Second-order optimization method. Inferior to SGD.

Proposed in

Krylov Subspace Descent for Deep Learning, Vinyals and Povey (2011)

Natural gradient

At each iteration attempts to perform the update which minimizes the loss function subject to the constraint that the KL-divergence between the probability distribution output by the network before and after the update is equal to a constant.

Revisiting natural gradient for deep networks, Pascanu and Bengio (2014)

Newton's method

An iterative method for finding the roots of an equation, $f(x)$. An initial guess (x_0) is chosen and iteratively refined by computing x_{n+1}.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Applied to gradient descent

In the context of gradient descent, Newton's method is applied to the derivative of the function to find the points where the derivative is equal to zero (the local optima). Therefore in this context it is a second order method.

$x_t = H_{t-1} g_t$ where H_{t-1} is the inverse of the Hessian matrix at iteration $t - 1$.

Picks the optimal step size for quadratic problems but is also prohibitively expensive to compute for large models due to the size of the Hessian matrix, which is quadratic in the number of parameters of the network.

Nesterov's method

Attempts to solve instabilities that can arise from using momentum by keeping the history of previous update steps and combining this with the next gradient step.

RMSProp

Similar to Adagrad, but introduces an additional decay term to counteract AdaGrad's rapid decrease in the learning rate. Divides the gradient by a running average of its recent magnitude. 0.001 is a good default value for the learning rate (η) and 0.9 is a good default value for α. The name comes from Root Mean Square Propagation.

$$\mu_t = \alpha \mu_{t-1} + (1 - \alpha) g_t^2$$
$$u_t = -\eta \frac{g_t}{\sqrt{\mu_t + \epsilon}}$$

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

http://ruder.io/optimizing-gradient-descent/index.html#rmsprop

Subgradient method

A class of iterative methods for solving convex optimization problems. Very similar to gradient descent except the subgradient is used instead of the gradient. The subgradient can be taken even at non-differentiable kinks in a function, enabling convergence on these functions.

1.20.16 Polyak averaging

The final parameters are set to the average of the parameters from the last n iterations.

Proposed in

Acceleration of Stochastic Approximation by Averaging, Polyak and Juditsky (1992)

1.20.17 Saddle points

A point on a function which is not a local or global optimum but where the derivatives are zero.

Gradients around saddle points are close to zero which makes learning slow. The problem can be partially solved by using a noisy estimate of the gradient, which SGD does implicitly.

Further reading

Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, Dauphin et al. (2014)

1.20.18 Saturation

When the input to a neuron is such that the gradient is close to zero. This makes learning very slow. This is a common problem for sigmoid and tanh activations, which saturate for inputs that are too high or too low.

1.20.19 Vanishing gradient problem

The gradients of activation functions like the sigmoid are all between 0 and 1. When gradients are computed via the chain rule they become smaller, increasingly so towards the beginning of the network. This means the affected layers train slowly.

If the gradients are larger than 1 this can cause the *exploding gradient problem*.

See also the dying ReLU problem.

1.21 Regularization

Used to reduce overfitting and improve generalization to data that was not seen during the training process.

Identifying Generalization Properties in Neural Networks, Wang et al. (2018)
Understanding Deep Learning Requires Rethinking Generalization, Zhang et al. (2016)

1.21.1 General principles

- Small changes in the inputs should not produce large changes in the outputs.
- Sparsity. Most features should be inactive most of the time.
- It should be possible to model the data well using a relatively low dimensional distribution of independent latent factors.

1.21.2 Methods

- Dropout
- Weight decay
- Early stopping

- Unsupervised pre-training
- Data augmentation
- Semi-supervised learning
- Noise injection
- Bagging and ensembling
- Optimisation algorithms like SGD that prefer wide minima
- Batch normalization
- Label smoothing

1.21.3 Dropout

For each training case, omit each hidden unit with some constant probability. This results in a network for each training case, the outputs of which are combined through averaging. If a unit is not omitted, its value is shared across all the models. Prevents units from co-adapting too much.

Dropout's effectiveness could be due to:

- An ensembling effect. 'Training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing' - Srivastava et al. (2014)
- Restricting the network's ability to co-adapt weights. The idea is that if a node is not reliably included, it would be ineffective for nodes in the next layer to rely on it's output. Weights that depend strongly on each other correspond to a sharp local minimum as a small change in the weights is likely to damage accuracy significantly. Conversely, nodes that take input from a variety of sources will be more resilient and reside in a shallower local minimum.

Can be interpreted as injecting noise inside the network.

Proposed in
Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Srivastava et al. (2014)

Variational dropout

Applied to RNNs. Unlike normal dropout, the same dropout mask is retained over all timesteps, rather than sampling a new one each time the cell is called. Compared to normal dropout, this is less likely to disrupt the RNN's ability to learn long-term dependencies.

Proposed in
Variational Dropout and the Local Reparameterization Trick, Kingma et al. (2015)

1.21.4 Generalization error

The difference between the training error and the test error.

1.21.5 Label smoothing

Replaces the labels with a weighted average of the true labels and the uniform distribution.

Further reading
When Does Label Smoothing Help?, Müller, R. et al. (2019)

Used in
Attention is All You Need, Vaswani et al. (2017)

1.21.6 Overfitting

When the network fails to generalize well, leading to worse performance on the test set but better performance on the training set. Caused by the model fitting on noise resulting from the dataset being only a finite representation of the true distribution.

1.21.7 Weight decay

L1 weight decay

Adds the following term to the loss function:

$$C \sum_{i=1}^{k} |\theta_i|$$

$C > 0$ is a hyperparameter.

L1 weight decay is mathematically equivalent to MAP estimation with a Laplacian prior on the parameters.

L2 weight decay

Adds the following term to the loss function:

$$C \sum_{i=1}^{k} {\theta_i}^2$$

$C > 0$ is a hyperparameter.

L2 weight decay is mathematically equivalent to doing MAP estimation where the prior on the parameters is Gaussian:

$$q(\theta) = N(0, C^{-1})$$

Intuition

Weight decay works by making large parameters costly. Therefore during optimisation the most important parameters will tend to have the largest magnitude. The unimportant ones will be close to zero.

Sometimes referred to as ridge regression or Tikhonov regularisation in statistics.

1.21.8 Zoneout

Method for regularizing RNNs. A subset of the hidden units are randomly set to their previous value ($h_t = h_{t-1}$).

Proposed in

Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations, Kreuger et al. (2016)

1.22 Sequence models

1.22.1 Beam search

Search algorithm to find the most likely output sequence.

Motivation

In a sequence to sequence problem, the next element in the decoded sequence is highly dependent on the previous one. If the output vocabulary is of size n and the sequence is of length m the complexity of finding the best sequence is $O(n^m)$ by brute force. Therefore a good heuristic algorithm is needed.

Greedy search runs in $O(mn)$ time but has poor accuracy. Beam search is a compromise between these two extremes.

The algorithm

In the context sequence-to-sequence beam search is a tree search algorithm. The inner nodes are partial solutions and the leaves are full solutions. At each iteration beam search keeps track of the k best partial solutions. The parameter k is known as the beam width.

Pseudocode:

```
1. # frontier maps partial solutions to scores
2. initialise frontier to contain the root node with a score of 0

3. while the end of the sequence has not been reached:
4.    select the candidate from frontier with the best score

5.    # expand the chosen candidate
6.    add all the children of the candidate to frontier
7.    compute the scores of all the new nodes in frontier

8.    # prune candidates
9.    remove all entries not in the top k from frontier
```

https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/

1.22.2 Bidirectional RNN

Combines the outputs of two RNNs, one processing the input sequence from left to right (forwards in time) and the other from right to left (backwards in time). The two RNNs are stacked on top of each other and their states are typically combined by appending the two vectors. Bidirectional RNNs are often used in Natural Language problems, where we want to take the context from both before and after a word into account before making a prediction.

The basic bidirectional RNN can be defined as follows:

$$
\begin{aligned}
h_t^f &= \tanh(W_h^f x_t + U_h^f h_{t-1}^f) \\
h_t^b &= \tanh(W_h^b x_{T-t} + U_h^b h_{t-1}^b) \\
h_t &= \text{concat}(h_t^f, h_t^b) \\
o_t &= V h_t
\end{aligned}
$$

Where h_t^f and h_t^b are the hidden states in the forwards and backwards directions respectively. T is the length of the sequence. Biases have been omitted for simplicity. x_t and o_t are the input and output states at time t, respectively.

Proposed in

Bidirectional Recurrent Neural Networks, Schuster and Paliwal (1997)

1.22.3 Differentiable Neural Computer (DNC)

The memory is an NxW matrix. There are N locations, which can be selectively read and written to. Read vectors are weighted sums over the memory locations.

The heads use three forms of differentiable attention which:

- Look up content.
- Record transitions between consecutively written locations in an NxN temporal link matrix L.
- Allocate memory for writing.

Proposed in

Hybrid computing using a neural network with dynamic external memory, Graves et al. (2016)

Further reading

https://deepmind.com/blog/article/differentiable-neural-computers

1.22.4 GRU (Gated Recurrent Unit)

Variation of the LSTM that is simpler to compute and implement, mergeing the cell and the hidden state.

Comparable performance to LSTMs on a translation task. Has two gates, a reset gate r and an update gate z. Not reducible from LSTM as there is only one tanh nonlinearity. Cannot 'count' as LSTMs can. Partially negates the vanishing gradient problem, as LSTMs do.

The formulation is:

$$
\begin{aligned}
r_t &= \sigma(W_r x_t + U_r h_{t-1}) \\
z_t &= \sigma(W_z x_t + U_z h_{t-1}) \\
\tilde{h}_t &= \tanh(W x + U(h_{t-1} * r)) \\
h_t &= z_t * h_{t-1} + (1 - z) * \tilde{h}_t
\end{aligned}
$$

Where $*$ represents element-wise multiplication and W_r, U_r, W_z, U_z, W and U are parameters to be learnt. Note the lack of bias terms, in contrast with the LSTM.

z is used for constructing the new hidden vector and dictates which information is updated from the new output and which is remembered from the old hidden vector. r is used for constructing the output and decides which parts of the hidden vector will be used and which won't be. The input for the current time-step is always used.

Proposed by

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, Cho et al. (2014)

Further reading

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Chung et al. (2014)

1.22.5 LSTM (Long Short-Term Memory)

A type of RNN with a memory cell as the hidden state. Uses a gating mechanism to ensure proper propagation of information through many timesteps. Traditional RNNs struggle to train for behaviour requiring long lags due to the exponential loss in error as back propagation proceeds through time (vanishing gradient problem). LSTMs store the error in the memory cell, making long memories possible. However, repeated access to the cell means the issue remains for many problems.

Can have multiple layers. The input gate determines when the input is significant enough to remember. The output gate decides when to output the value. The forget gate determines when the value should be forgotten.

The activations of the input, forget and output gates are i_t, f_t and o_t respectively. The state of the memory cell is C_t.

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
\tilde{C}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
C_t &= i_t * \tilde{C}_t + f_t * C_{t-1} \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + V_o C_t + b_o) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

Where $*$ represents element-wise multiplication.

Each of the input, output and forget gates is surrounded by a sigmoid nonlinearity. This squashes the input so it is between 0 (let nothing through the gate) and 1 (let everything through).

The new cell state is the candidate cell state scaled by the input gate activation, representing how much we want to remember each value and added to the old cell state, scaled by the forget gate activation, how much we want to forget each of those values.

The $\tanh$ functions serve to add nonlinearities.

Using an LSTM does not protect from exploding gradients.

Proposed by

Long Short-Term Memory, Hochreiter and Schmidhuber (1997)

Forget bias initialization

Helpful to initialize the bias of the forget gate to 1 in order to reduce the scale of forgetting at the start of training. This is done by default in TensorFlow.

Weight tying

Tie the input and output embeddings. May only be applicable to generative models. Discriminative ones do not have an output embedding.

Using the Output Embedding to Improve LMs, Press and Wolf (2016)

Cell clipping

Clip the activations of the memory cells to a range such as [-3,3] or [-50,50]. Helps with convergence problems by preventing exploding gradients and saturation in the sigmoid/tanh nonlinearities.

Used in
Deep Recurrent Neural Networks for Acoustic Modelling, Chan and Lane (2015)
LSTM RNN Architectures for Large Scale Acoustic Modeling, Sak et al. (2014)

Peep-hole connections

Allows precise timing to be learned, such as the frequency of a signal and other periodic patterns.

Used in
Learning Precise Timing with LSTM Recurrent Networks, Ger et al. (2002)
LSTM RNN Architectures for Large Scale Acoustic Modeling, Sak et al. (2014)

1.22.6 Neural Turing Machine (NTM)

Can infer simple algorithms like copying, sorting and associative recall.

Has two principal components:

1. A controller, an LSTM. Takes the inputs and emits the outputs for the NTM as a whole.
2. A memory matrix.

The controller interacts with the memory via a number of read and write heads. Read and write operations are 'blurry'. A read is a convex combination of 'locations' or rows in the memory matrix, according to a weight vector over locations assigned to the read head. Writing uses an erase vector and an add vector. Both content-based and location-based addressing systems are used.

Similarity between vectors is measured by the cosine similarity.

Location-based addressing is designed for both iteration across locations and random-access jumps.

Content addressing

Compares a key vector to each location in memory, $M_t(i)$ to produce a normalised weighting, $w_t^c(i)$. $t > 0$ is the key strength, used to amplify or attenuate the focus.

Interpolation

Blends the weighting produced at the previous time step and the content weighting. An 'interpolation gate' is emitted by each head. If $g_t = 1$ the addressing is entirely content-based. If $g_t = 0$, the addressing is entirely location-based.

Convolutional shift

Provides a rotation to the weight vector w_t^g. All index arithmetic is computed modulo N. The shift weighting s_t is a vector emitted by each head and defines a distribution over the allowed integer shifts.

Sharpening

Combats possible dispersion of weightings over time.

$$w_t(i) := \frac{w_t(i)^{\gamma_t}}{\sum_j w_t(j)^{\gamma_t}}$$

Proposed in
Neural Turing Machines, Graves et al. (2014)

1.22.7 RNN (Recurrent Neural Network)

A type of network which processes a sequence and outputs another of the same length. It maintains a hidden state which is updated as new inputs are read in.

The most basic type of RNN has the functional form:

$$h_t = \tanh(W_h x_t + U_h h_{t-1} + b_h)$$
$$o_t = V h_t + b_o$$

Where x_t, o_t and h_t are the input, output and hidden states at time t, respectively.

1.22.8 RNN Encoder-Decoder

A simple architecture for sequence-to-sequence tasks.

It consists of two RNNs. One encodes the input sequence into a fixed-length vector representation, the other decodes it into an output sequence. The original, proposed in Cho et al. (2014), uses the GRU to model sequential information using fewer parameters than the LSTM. Can be augmented with sampled softmax, bucketing and padding.

Proposed in
Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, Cho et al. (2014)

1.22.9 Sequence to sequence

Any machine learning task that takes one sequence and turns it into another.

Examples include:

- Translation

- Text-to-speech
- Speech-to-text
- Part of speech tagging (POS tagging)

1.22.10 Transformer

Sequence model notable for not using recurrence or convolutions - only attention.

Attained state of the art accuracy on translation tasks (Vaswani et al., 2017) and has subsequently been used to get new records on a variety of other tasks (see 'Used in').

Attention layers use scaled-dot product attention.

Both the encoder and decoder are comprised of multiple blocks each with a multi-ahead attention layer, two fully-connected layers and two layer-normalisation components.

Multi-head attention

Concatenates the output of multiple parallel attention layers. Each layer has the same inputs (Q, K and V) but different weights. Vaswani et al. (2017) use 8 layers in each multi-head attention component but reduce the dimensionality of each from 512 to 64, which keeps the computational cost the same overall.

Positional encoding

Positional encodings are added (summed, not concatenated) to the input embeddings to allow the model to be aware of the sequence order.

Self-attention

Usage in pre-trained language models

Devlin et al. (2018) pre-train a bidirectional transformer and use this model to attain state of the art accuracy on a variety of natural language tasks. The transformer is first pre-trained to predict masked out tokens and predict next sentences and then fine-tuned on the tasks to be evaluated.

Proposed in
Attention is All You Need, Vaswani et al. (2017)

Used in
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Devlin et al. (2018)
Improving Language Understanding by Generative Pre-Training, Radford et al. (2018)
Language Models are Unsupervised Multitask Learners, Radford et al. (2019)

Further reading
The Annotated Transformer, Rush (2018)

1.23 Decision Trees

1.23.1 Decision Tree

Advantages

- Easy to interpret.
- Efficient inference. Inference time is proportional to the depth of the tree, not its size.
- No risk of local optima during training.

Disadvantages

- Can easily overfit.
- Requires a quantity of data that is exponential in the depth of the tree. This means learning a complex function can require a prohibitive amount of data.

Training

The simplest approach for training decision trees is:

- At each node find the optimal variable to split on. This is the variable whose split yields the largest information gain (decrease in entropy).

Regularization

Some options for avoiding overfitting when using decision trees include:

- Specifying a maximum depth for the tree
- Setting a minimum number of samples to create a split.

1.23.2 Random forest

Each tree is built from a sample of the dataset, drawn with replacement. This randomises how splits in the tree are chosen (rather than simply choosing the best), decreasing variance at the expense of bias, but with a positive overall effect on accuracy.

1.24 Ensemble models

1.24.1 AdaBoost

A boosting algorithm. Each classifier in the ensemble attempts to correctly predict the instances misclassified by the previous iteration.

Decision trees are often used as the weak learners.

1.24.2 Bagging

A way to reduce overfitting by building several models independently and averaging their predictions. As bagging reduces variance it is well suited to models like decision trees as they are prone to overfitting.

1.24.3 Boosting

Build models sequentially, each one trying to reduce the bias of the combined estimator. AdaBoost is an example.

Gradient boosting

Learns a weighted sum of weak learners:

$$\hat{y}_i = \sum_{i=1}^{M} \gamma_i h_i(x)$$

where γ_i is the weight associated with the weak learner h_i. M is the total number of weak learners.

The first learner predicts a constant for all examples. All subsequent learners try to predict the residuals.

Can learn with any differentiable loss function.

1.24.4 Weak learner

The individual algorithms that make up the ensemble.

1.25 Gaussian processes

Gaussian processes model a probability distribution over functions.

Let $f(x)$ be some function mapping vectors to vectors. Then we can write:

$$f(x) \sim GP(m(x), k(x, x'))$$

where $m(x)$ represents the mean vector:

$$m(x) = \mathbb{E}[f(x)]$$

and $k(x, x')$ is the kernel function.

1.25.1 Kernel function

The kernel is a function that represents the covariance function for the Gaussian process.

$$k(x, x') = \text{Cov}(f(x), f(x'))$$

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))^T]$$

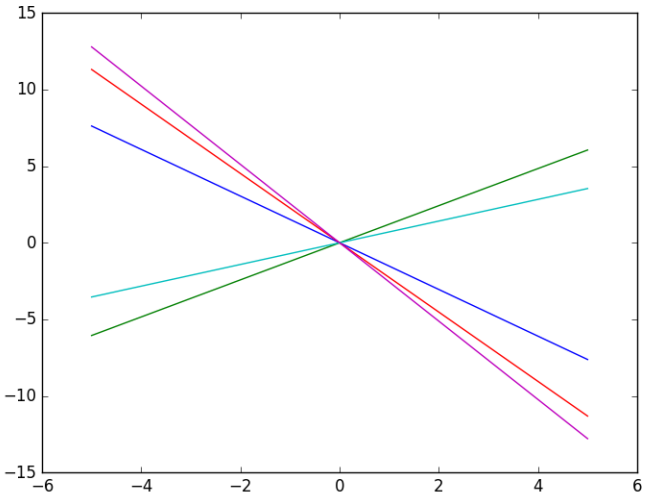
The kernel can be thought of as a prior for the shape of the function, encoding our expectations for the amount of smoothness or non-linearity.

Not all conceivable kernels are valid. The kernel must produce covariance matrices that are positive-definite.

Linear kernel

$$k(x, x') = x \cdot x'$$

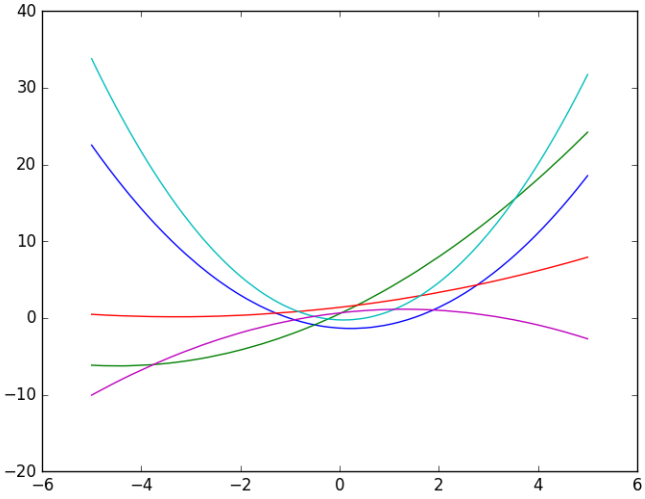
Some functions sampled from a Gaussian process with a linear kernel:



Polynomial kernel

$$k(x, x'; a, b) = (x \cdot x' + a)^b$$

Functions sampled from a Gaussian process with a polynomial kernel where $a = 1$ and $b = 2$:

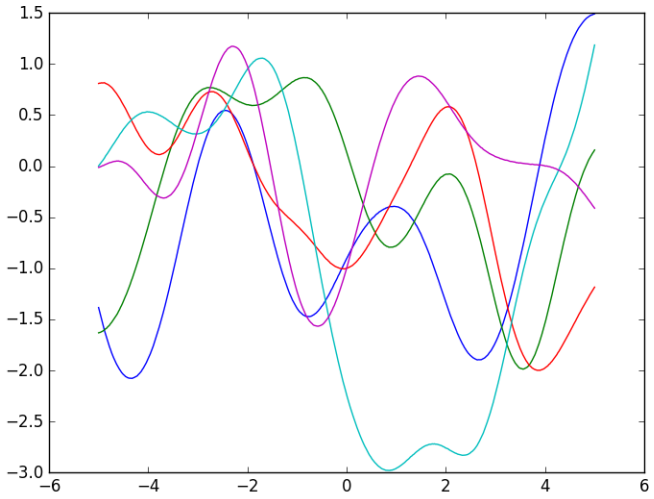


Gaussian kernel

Also known as the **radial basis function or RBF** kernel.

$$k(x, x'; \sigma, l) = \sigma_2 \exp(-\frac{(x - x')_2}{2l^2})$$

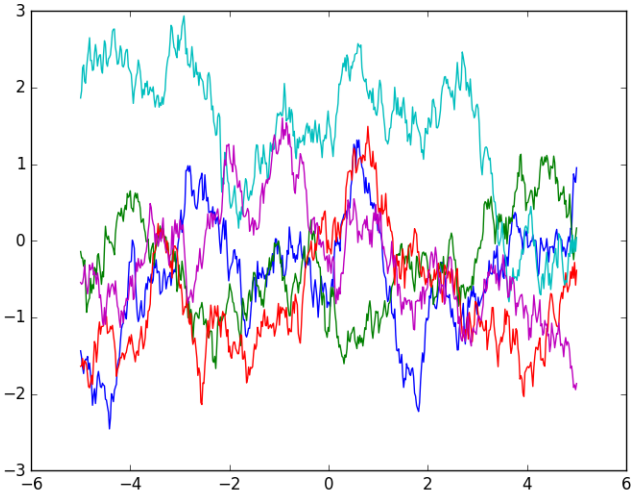
Some functions sampled from a GP with a Gaussian kernel:



Laplacian kernel

$$k(x, x') = \exp(-|x - x'|)$$

Functions sampled from a GP with a Laplacian kernel:



1.25.2 Sampling from a Gaussian process

The method is as follows:

1. Decide on a vector of inputs x for which we want to compute $f(x)$, where f is some function which we will sample from the Gaussian process.
2. Compute the matrix K where $K_{ij} = k(x_i, x_j)$.
3. Perform Cholesky decomposition on K, yielding a lower triangular matrix L.
4. Sample a vector of numbers from a standard Gaussian distribution, $s_i \sim N(0, 1)$.
5. Take the dot product of L and the vector s to get the samples $f(x) = L \cdot s$.

1.26 Graphical models

1.26.1 Bayesian network

A directed acyclic graph where the nodes represent random variables.

Not to be confused with Bayesian neural networks.

The chain rule for Bayesian networks

The joint distribution for all the variables in a network is equal to the product of the distributions for all the individual variables, conditional on their parents.

$$P(X_1, ..., X_n) = \prod_i P(X_i | Par(X_i))$$

where $Par(X_i)$ denotes the parents of the node X_i in the graph.

1.26.2 Boltzmann Machines

Restricted Boltzmann Machines (RBMs)

Trained with contrastive divergence.

Deep Belief Networks (DBNs)

Deep Belief Machines (DBMs)

1.26.3 Clique

A subset of a graph where the nodes are fully-connected, ie each node has an edge with every other node in the set.

1.26.4 Conditional Random Field (CRF)

Discriminative model that can be seen as a generalization of logistic regression.

Common applications of CRFs include image segmentation and named entity recognition.

Used in

Seed, Expand and Constrain: Three Principles for Weakly-Supervised Image Segmentation, Kolesnikov and Lampert (2016)

Linear Chain CRFs

A simple sequential CRF.

1.26.5 Hidden Markov Model (HMM)

A simple generative sequence model in which there is an observable state and a latent state, which must be inferred.

At each time step the model is in a latent state x_t and outputs an observation y_t. The observation is solely a function of the latent state, as is the probability distribution over the next state, x_{t+1}. Hence the model obeys the Markov property.

The model is defined by:

- A matrix T of transition probabilities where T_{ij} is the probability of going from state i to state j.
- A matrix E of emission probabilities where E_{ij} is the probability of emitting observation j in state i.

The parameters can be learnt with the Baum-Welch algorithm.

1.26.6 Markov chain

A simple state transition model where the next state depends only on the current state. At any given time, if the current state is node i, there is a probability T_{ij} of transitioning to node j, where T is the transition matrix.

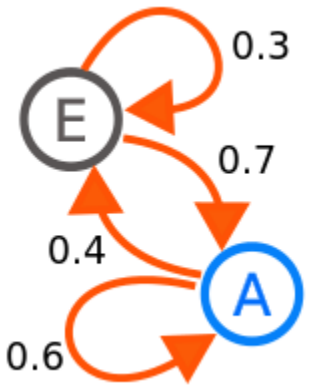


Fig. 1: Source: Wikipedia

1.26.7 Markov property

A process is said to have the Markov property if the next state depends only on the current state, not any of the previous ones.

1.26.8 Markov Random Field (MRF)

A type of undirected graphical model which defines the joint probability distribution over a set of variables. Each variable is represented by one node in the graph.

One use for an MRF could be to model the distribution over the pixel values for a set of images. In order to keep the model tractable edges are only drawn between neighbouring pixels.

1.26.9 Naive Bayes Model

A simple classifier that models all of the features as independent, given the label.

$$P(Y|X_1, ..., X_n) = P(Y) \prod_{i=1}^{n} P(Y|X_i)$$

1.27 Regression

1.27.1 Confidence intervals

The confidence interval for a point estimate measures is the interval within which we have a particular degree of confidence the true value resides. For example, the 95% confidence interval for the mean height in a population may be [1.78m, 1.85m].

Confidence intervals can be calculated in this way:

1. Let α be the specified confidence level. eg $\alpha = 0.95$ for the 95% confidence level.
2. Let $f(x; n-1)$ be the pdf for Student's t distribution, parameterised by the number of degrees of freedom which is the sample size (n) minus 1.
3. Calculate $t = f(1 - \alpha/2; n - 1)$
4. Then the confidence interval for the point estimate is:

$$\bar{x} - t\frac{s}{\sqrt{n}} \leq x \leq \bar{x} + t\frac{s}{\sqrt{n}}$$

Where $\bar{x}$ is the estimated value of the statistic, x is the true value and s is the sample standard deviation.

1.27.2 Isotonic regression

Fits a step-wise monotonic function to the data. A useful way to avoid overfitting if there is a strong theoretical reason to believe that the function $y = f(x)$ is monotonic. For example, the relationship between the floor area of houses and their prices.

1.27.3 Linear regression

The simplest form of regression. Estimates a model with the equation:

$$\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

where the β_i are parameters to be estimated by the model and the x_i are the features.

The loss function is usually the squared error.

Normal equation

The equation that gives the optimal parameters for a linear regression.

Rewrite the regression equation as $\hat{y} = X\beta$.

Then the formula for β which minimizes the squared error is:

$$\beta = (X^T X)^{-1} X^T y$$

1.27.4 Logistic regression

Used for modelling probabilities. It uses the sigmoid function (σ) to ensure the predicted values are between 0 and 1. Values outside of this range would not make sense when predicting a probability. The functional form is:

$$\hat{y} = \sigma(\beta_0 + \beta_1 x_1 + ... + \beta_n x_n)$$

1.27.5 Multicollinearity

When one of the features is a linear function of one or more of the others.

1.27.6 P-values

Measure the statistical significance of the coefficients of a regression. The closer the p-value is to 0, the more statistically significant that result is.

The p-value is the probability of seeing an effect greater than or equal to the one observed if there is in fact no relationship.

In a regression the formula for calculating the p-value of a coefficient is:

TODO

1.28 SVMs

Support Vector Machines.

Binary classifier. Their objective is to find a hyperplane that optimally separates the two classes (maximises the margin).

1.28.1 Hard margin

Can be used when the data is linearly separable.

The decision function for a linear hard-margin SVM is:

$$\hat{y}_i = \text{sgn}(wx_i - b)$$

All positive examples should have $wx_i - b \geq 1$ and all negative examples should have $wx_i - b \leq -1$.

1.28.2 Soft-margin

Necessary when the data is not linearly separable.

The loss function for a linear soft-margin SVM is:

$$L(x; w, b) = \sum_i \max\{0, m - y_i(wx_i - b)\} + C||w||^2$$

Where w and b are parameters to be learnt and C is a hyperparameter.

1.28.3 Dual form

1.28.4 Primal form

1.28.5 Training

Quadratic programming

1.28.6 Kernels

The kernel is used to map the data into a high-dimensional space in which it is easier to separate it linearly. This is known as the **kernel trick**.

Linear

$$k(x, y) = x \cdot y$$

Polynomial

$$k(x, y) = (ax \cdot y + b)^d$$

Sigmoid

$$k(x, y) = \tanh(ax \cdot y + b)$$

RBF

$$k(x, y) = \exp(-||x - y||^2/2\sigma^2)$$

1.28.7 Advantages

- The optimisation problem is convex so local optima are not a problem.

1.28.8 Disadvantages

- Cannot naturally learn multiclass classification problems. Applying an SVM to these requires reformulating the problem as a series of binary classification tasks, either n one-vs-all or n^2 one-vs-one tasks. Learning these separately is inefficient and poor for generalisation.

1.28.9 One-Class SVMs

See here

1.29 Adversarial examples

Examples that are specially created so that image classification algorithms predict the wrong class with high confidence even though the image remains easy for humans to classify correctly. Only small perturbations in the pixel-values are necessary to create adversarial examples.

They can be created without knowledge of the weights of the classifier. The same adversarial example can be misclassified by many classifiers, trained on different subsets of the dataset and with different architectures.

The direction of perturbation, not the point itself matters most when generating adversarial examples. Adversarial perturbations generalize across different clean examples.

Kurakin et al. (2016) showed that adversarial examples are still effective, even when perceived through a cellphone camera.

1.29.1 Generating adversarial examples

Perform gradient descent on the image by taking the derivative of the score for the desired class with respect to the pixels.

Note that this is almost the same technique as was used by Google for understanding convnet predictions but without an additional constraint. They specify that the output image should look like a natural image (eg by having neighbouring pixels be correlated).

1.29.2 Explanations

Adversarial examples are made possible when the input has a large number of dimensions. This means many individually small effects can have a very large effect on the overall prediction.

Goodfellow et al. (2015) suggest that the effectiveness of adversarial examples is down to the linearity of neural networks. While the function created by the network is indeed nonlinear, it is not as nonlinear as often thought. Goodfellow says "... neural nets are piecewise linear, and the linear pieces with non-negligible slope are much bigger than we expected."

1.29.3 Mitigation techniques

- Regularization - Karpathy (2015) showed that regularization is effective for linear classifiers. It reduces the size of the weights so the image has to be changed more drastically in order to get the same misclassification. However, this comes at a cost in accuracy.
- Adding noise - Somewhat effective but hurts accuracy, Gu et al. (2014)
- Blurring - Somewhat effective but hurts accuracy, Gu et al. (2014)
- Binarization - Highly effective where it is applicable without hurting accuracy, such as reading text, Graese et al. (2016)
- Averaging over multiple crops - Can be sufficient to correctly classify the majority of adversarial examples.
- RBF networks (Goodfellow et al. (2015)) are resistant to adversarial examples due to their non-linearity. In general using more non-linear models (trained with a better optimization algorithm to make them feasible) may be the best approach.

1.29.4 Papers

Breaking Linear Classifiers on ImageNet, Karpathy (2015)
Explaining and Harnessing Adversarial Examples, Goodfellow et al. (2015)
Intriguing Properties of Neural Networks, Szegedy et al. (2013)
Towards Deep Neural Network Architectures Robust to Adversarial Examples, Gu et al. (2014)
Adversarial examples in the physical world, Kurakin et al. (2016)
Assessing Threat of Adversarial Examples on Deep Neural Networks, Graese et al. (2016)

1.30 Anomaly detection

This problem can be solved well through methods for density estimation. If the density predicted for an example falls below a threshold it can be declared an anomaly. In addition, the following methods also exist:

1.30.1 Isolation Forest

An ensemble of decision trees. The key idea is that points in less dense areas will require fewer splits to be uniquely identified since they are surrounded by fewer points.

Features and split values are randomly chosen, with the split value being somewhere between the min and the max observed values of the feature.

1.30.2 Local Outlier Factor

A nearest-neighbour model.

1.30.3 One-Class SVM

Learns the equation for the smallest possible hypersphere that totally encapsulates the data points.

Proposed by Estimating the Support of a High-Dimensional Distribution, Schölkopf et al. (2001)

1.31 Computer vision

Tasks which have an image or video as their input. This includes:

- Image captioning
- Image classification
- Image segmentation
- Image-to-image translation
- Object detection

1.31.1 Challenges

- Parts of the object may be obscured.
- Photos can be taken at different angles.
- Different lighting conditions. Both the direction and amount of light may differ, as well as the number of light sources.
- Objects belonging to one class can come in a variety of forms.

1.31.2 Data augmentation

The images in the training set are randomly altered in order to improve the generalization of the network.

Cubuk et al. (2018), who evaluate a number of different data augmentation techniques, use the following transforms:

- Blur - The entire image is blurred by a random amount.
- Brightness
- Color balance
- Contrast
- Cropping - The image is randomly cropped and the result is fed into the network instead.
- Cutout - Mask a random square region of the image, replacing it with grey. Was used to get state of the art results on the CIFAR-10, CIFAR-100 and SVHN datasets. Proposed in Improved Regularization of Convolutional Neural Networks with Cutout, DeVries and Taylor (2017)
- Equalize - Perform histogram equalization on the image. This adjusts the contrast.
- Flipping - The image is flipped with probability 0.5 and left as it is otherwise. Normally only horizontal flipping is used but vertical flipping can be used where it makes sense - satellite imagery for example.
- Posterize - Decrease the bits per pixel
- Rotation
- Sample pairing - Combine two random images into a new synthetic image. See Data Augmentation by Pairing Samples for Images Classification, Inoue (2018).
- Shearing
- Solarize - Pixels above a random value are inverted.
- Translation

1.31.3 Datasets

CIFAR-10/100

CIFAR-10 is a dataset of 60000 32x32 colour images in 10 classes with 6000 images each. CIFAR-100 has 100 classes, with only 600 images for each. The dataset comprises 50000 images in the training set and 10000 in the test.

Notable results - CIFAR-10

- 98.9% - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Tan and Le (2019)
- 98.5% - AutoAugment: Learning Augmentation Strategies from Data, Cubuk et al. (2018)
- 97.6% - Learning Transferable Architectures for Scalable Image Recognition, Zoph et al. (2017)
- 97.4% - Improved Regularization of Convolutional Neural Networks with Cutout, DeVries and Taylor (2017)
- 96.1% - Wide Residual Networks, Zagoruyko and Komodakis (2016)
- 94.2% - All you need is a good init, Mishkin and Matas (2015)
- 93.6% - Deep Residual Learning for Image Recognition, He et al. (2015)
- 93.5% - Fast and Accurate Deep Network Learning by Exponential Linear Units, Clevert et al. (2015)

Notable results - CIFAR-100

- 91.7% - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Tan and Le (2019)
- 89.3% - AutoAugment: Learning Augmentation Strategies from Data, Cubuk et al. (2018)
- 84.8% - Improved Regularization of Convolutional Neural Networks with Cutout, de Vries and Taylor (2017)
- 81.1% - Wide Residual Networks, Zagoruyko and Komodakis (2016)
- 75.7% - Fast and Accurate Deep Network Learning by Exponential Linear Units, Clevert et al. (2015)
- 72.3% - All you need is a good init, Mishkin and Matas (2015)

https://keras.io/datasets/#cifar10-small-image-classification

COCO

Common Objects in COntext. A dataset for image recognition, segmentation and captioning.

Detection task - Notable results (mAP):

- 51.0% - EfficientDet: Scalable and Efficient Object Detection, Tan et al. (2019)
- 48.3% - NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection, Ghaisi et al. (2019)
- 42.1% - AutoAugment: Learning Augmentation Strategies from Data, Cubuk et al. (2018)
- 35.9% - Fast R-CNN, Girshick et al. (2015)

ImageNet (ILSVRC)

ILSVRC stands for Imagenet Large Scale Recognition Challenge. Popular image classification task in which the algorithm must use a dataset of ~1.4m images to classify 1000 classes.

Notable results (top-1 accuracy):

- 87.4% - Self-training with Noisy Student improves ImageNet classification, Xie et al. (2019)
- 85.0% - RandAugment: Practical data augmentation with no separate search, Cubuk et al. (2019)

- 84.4% - EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Tan and Le (2019)
- 83.9% - Regularized Evolution for Image Classifier Architecture Search, Real et al. (2018)
- 83.5% - AutoAugment: Learning Augmentation Strategies from Data, Cubuk et al. (2018)
- 82.7% - Learning Transferable Architectures for Scalable Image Recognition, Zoph et al. (2017)
- 78.6% - Deep Residual Learning for Image Recognition, He et al. (2015)
- 76.3% - Very deep convolutional networks for large-scale image recognition, Simonyan and Zisserman (2014)
- 62.5% - ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky et al. (2012)

NB: Xie et al. (2019) also use unlabeled data.

MNIST

70000 28x28 pixel grayscale images of handwritten digits (10 classes), 60000 in the training set and 10000 in the test set.

http://yann.lecun.com/exdb/mnist/

https://keras.io/datasets/#mnist-database-of-handwritten-digits

Pascal VOC

PASCAL Visual Object Classes Homepage

SVHN

Street View House Numbers. Contains images of the numbers 0-9 (10 classes) in over 600,000 images. Harder than MNIST since the images come from natural scenes.

http://ufldl.stanford.edu/housenumbers/

1.31.4 Face recognition

The name of the general topic. Includes face identification and verification.

The normal face recognition pipeline is:

- Face detection - Identifying the area of the photo that corresponds to the face.
- Face alignment - Often done by detecting facial landmarks like the nose, eyes and mouth.
- Feature extraction and similarity calculation

Challenges

In addition to the standard challenges in computer vision facial recognition also encounters the following problems:

- Changes in facial hair.
- Glasses, which may not always be worn.
- People aging over time.

Datasets

- LFW
- YouTube-Faces
- CASIA-Webface
- CelebA

Face identification

Multiclass classification problem. Given an image of a face, determine the identity of the person.

Face verification

Binary classification problem. Given two images of faces, assess whether they are from the same person or not.

Commonly used architectures for solving this problem include Siamese and Triplet networks.

1.31.5 Image segmentation

Partitions an object into meaningful parts with associated labels. May also be referred to as per-pixel classification.

Further reading
U-Net: Convolutional Networks for Biomedical Image Segmentation, Ronneberger et al. (2015)

Instance segmentation

Unlike semantic segmentation, different instances of the same object type have to be labelled as separate objects (eg person 1, person 2). Harder than semantic segmentation.

Semantic segmentation

Unlike instance segmentation, in semantic segmentation it is only necessary to predict what class each pixel belongs to, not separate out different instances of the same class.

Weakly-supervised segmentation

Learning to segment from only image-level labels. The labels will describe the classes that exist within the image but not what the class is for every pixel.

The results from weak-supervision are generally poorer than otherwise but datasets tend to be much cheaper to acquire.

When the dataset is only weakly-supervised it can be very hard to correctly label highly-correlated objects that are usually only seen together, such as a train and rails.

1.31.6 Image-to-image translation

Examples:

- Daytime to nighttime
- Greyscale to colour
- Streetmap to satellite view

Example papers
Image-to-Image Translation with Conditional Adversarial Networks, Isola et al. (2016)

1.31.7 Object detection

The task of finding objects of interest in a scene and determining what they are.

Object detection algorithms can generally be divided into two categories:

- One-stage detectors
- Two-stage detectors

One-stage detector

A class of object detection algorithm. Contrast with two-stage detectors.

Example papers
Focal Loss for Dense Object Detection, Lin et al. (2017)
YOLO9000: Better, Faster, Stronger, Redmon and Farhadi (2016)
You Only Look Once: Unified, Real-Time Object Detection, Redmon et al. (2015)
SSD: Single Shot MultiBox Detector, Liu et al. (2015)

Region of interest

See 'region proposal'.

Region proposal

A region in an image (usually defined by a rectangle) identified as containing an object of interest with high probability, relative to the background.

Two-stage detector

A type of object detection algorithm.

The first stage proposes regions that may contain objects of interest. The second stage classifies these regions as either background or one of the classes.

There is often a significant class-imbalance problem since background regions greatly outnumber the other classes.

Contrast with one-stage detectors.

Example papers for the first stage
Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, Ren et al. (2015)
Edge Boxes: Locating Object Proposals from Edges, Zitnick and Dollar (2014)
Selective Search for Object Recognition, Uijlings et al. (2012)

Example papers for the second stage
Mask R-CNN, He et al. (2017)
Fast R-CNN, Girshick et al. (2015)

1.31.8 Saliency map

A heatmap over an image which shows each pixel's importance for the predicted classification. This makes them very useful for making image classifiers explainable.

1.32 Density estimation

The problem of estimating the probability density function for a given set of observations.

1.32.1 Empirical distribution function

Compute the empirical CDF and numerically differentiate it.

1.32.2 Histogram

Take the range of the sample and split it up into n bins, where n is a hyperparameter. Then assign a probability to each bin according to the proportion of the sample that fell within its bounds.

1.32.3 Kernel Density Estimation

The predicted density function given an a sample x is:

$$\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i)$$

Where K is the kernel and $h > 0$ is a smoothing parameter.

$$K_h(x) = \frac{1}{h}K(\frac{x}{h})$$

A variety of kernels can be used. A common one is the Gaussian, defined as:

$$K(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$$

Disadvantages

The complexity at inference time is linear in the size of the sample.

1.32.4 Mixture Model

Estimates the density as a weighted sum of parametric distributions. The predicted density function for a sample x is:

$$\hat{f}(x) = \sum_{i=1}^{k} w_i \phi(x; \theta_i)$$

Where k is the number of distributions and each distribution, ϕ, is parameterised by θ. It is also weighted by a single scalar w_i where $\sum_{i=1}^{k} w_i = 1$.

The Gaussian is a common choice for the distribution. In this case the estimator is known as a **Gaussian Mixture Model**.

All of the parameters can be learnt using Expectation-Maximization, except for k which is a hyperparameter.

1.33 Evaluation metrics

1.33.1 Classification

AUC (Area Under the Curve)

Summarises the ROC curve with a single number, equal to the integral of the curve.

Sometimes referred to as AUROC (Area Under the Receiver Operating Characteristics).

F1-score

The F1-score is the harmonic mean of the precision and the recall.

Using the harmonic mean has the effect that a good F1-score requires both a good precision and a good recall.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Precision

The probability that an example is in fact a positive, given that it was classified as one.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Where TP is the number of true positives and FP is the number of false positives.

Recall

The probability of classifying an example as a positive given that it is infact a positive.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Where TP is the number of true positives and FN is the number of false negatives.

ROC curve

Plots the true positive rate against the false positive rate for different values of the threshold in a binary classifier.

ROC stands for Receiver Operating Characteristic.

The ROC curve can be described with one number using the ROC.

1.33.2 Language modelling

Bits per character (BPC)

Used for assessing character-level language models.

Identical to the cross-entropy loss, but uses base 2 for the logarithm:

$$BPC(y, \hat{y}) = -\sum_{i}\sum_{c} y_{i,c} \log_2(\hat{y}_{i,c})$$

where c are the character classes. $y_{i,c}$ equals 1 if example i is character c and 0 otherwise. $\hat{y}_{i,c}$ is the predicted probability that example i is character c.

Perplexity

Used to measure how well a probabilistic model predicts a sample. It is equivalent to the exponential of the cross-entropy loss.

1.33.3 Object detection

Intersection over Union (IoU)

An accuracy score for two bounding boxes, where one is the prediction and the other is the target. It is equal to the area of their intersection divided by the area of their union.

Mean Average Precision

The main evaluation metric for object detection.

To calculate it first define the overlap criterion. This could be that the IoU for two bounding boxes be greater than 0.5. Since the ground truth is always that the class is present, this means each predicted box is either a true-positive or a false-positive. This means the precision can be calculated using TP/(TP+FN).

1.33.4 Ranking

Cumulative Gain

A simple metric for ranking that does not take position into account.

$$CG_p = \sum_{i=1}^{p} r_i$$

Where r_i is the relevance of document i.

Discounted Cumulative Gain (DCG)

Used for ranking. Takes the position of the documents in the ranking into account.

$$DCG_p = \sum_{i=1}^{p} \frac{r_i}{\log_2 (i+1)}$$

Where r_i is the relevance of the document in position i.

Mean Reciprocal Rank (MRR)

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{rank(q)}$$

Where $q \in Q$ is a query taken from a set of queries and $rank(q)$ is the rank of the first document that is relevant for query q.

Normalized Discounted Cumulative Gain (NDCG)

Used for ranking. Normalizes the DCG by dividing by the score that would be achieved by a perfect ranking. NDCG is always between 0 and 1.

$$NDCG_p = \frac{DCG_p}{IDCG_p}$$

Where

$$DCG_p = \sum_{i=1}^{p} \frac{r_i}{\log_2 (i+1)}$$

and IDCG is the Ideal Discounted Cumulative Gain, the DCG that would be produced by a perfect ranking:

$$IDCG_p = \sum_{i=1}^{p} \frac{2^{r_i} - 1}{\log_2 (i+1)}$$

Precision @ k

The proportion of documents returned in the top k results which are relevant. ie the number of relevant documents divided by k.

1.33.5 Regression

RMSE

Root Mean Squared Error.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

R-squared

A common metric for evaluating regression algorithms that is easier to interpret than the RMSE but only valid for linear models.

Intuitively, it is the proportion of the variance in the y variable that has been explained by the model. As long as the model contains an intercept term the R-squared should be between 0 and 1.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

where $\bar{y} = \sum_{i=1}^{n} y_i$, the mean of y.

1.33.6 Translation

BLEU

Score for assessing translation tasks. Also used for image captioning. Stands for BiLingual Evaluation Understudy.

Ranges from 0 to 1, where 1 corresponds to being identical to the reference translation. Often uses multiple reference translations.

BLEU: a Method for Automatic Evaluation of Machine Translation, Papineni et al. (2002)

1.34 Hyperparameter optimization

A hyperparameter is a parameter of the model which is set according to the design of the model rather than learnt through the training process. Examples of hyperparameters include the learning rate, the dropout rate and the number of layers. Since they cannot be learnt by gradient descent hyperparameter optimization is a difficult problem.

1.34.1 Gaussian processes

Gaussian processes are used to model the function we are trying to optimise.

1.34.2 Bayesian optimization

Note that much of the below explanation references states. These are irrelevant for hyperparameter optimisation since each training run is initialized in the same way.

Acquisition function

A function that decides the next point to sample while trying to maximize the cumulative reward, balancing exploration and exploitation. They are useful not just in hyperparameter optimization but also in reinforcement learning.

https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf

Probability of Improvement

Pick the action which maximises the chance of getting to a state with a value greater than the current best state. The reward is 1 if the new state is better and 0 otherwise. This means that it will eschew possible large improvements in favour of more certain small ones.

If all nearby states are known to be worse this strategy can lead to getting stuck in local optima.

Expected Improvement

Pick the action which maximises the expected improvement of that new state over the current best state. The reward is the difference between the values if the new state is better than the old one and zero otherwise.

A higher expected improvement can be obtained either by increasing either the variance or the mean of the value distribution of the next state.

Upper Confidence Bound

Calculate the upper bound of the confidence interval for the rewards from each action in a given state. Pick the action for which the upper bound of the reward is greatest. This will lead to actions with greater uncertainty being chosen since their confidence interval will be larger.

Using a Gaussian distribution gives a simple expression for the bound, that it is β standard deviations away from the mean of the distribution of rewards given an action a in some state s:

$$UCB_{s,a} = \mu_{s,a} + \beta\sigma_{s,a}$$

1.34.3 Cross-validation

k-fold cross validation

```
1. Randomly split the dataset into K equally sized parts
2. For i = 1,...,K
3.     Train the model on every part apart from part i
4.     Evaluate the model on part i
5. Report the average of the K accuracy statistics
```

1.34.4 Grid search

A simple algorithm for exhaustively testing different combinations of hyperparameters.

It starts by manually specifying the hyperparameters to be evaluated. For example:

```
learning_rates = [0.001, 0.01, 0.1]
dropout_rates = [0.0, 0.2, 0.4, 0.6, 0.8]
num_layers = [12, 16, 20, 24]
```

Then every combination is tested one by one by training a model with those settings and calculating the accuracy on the validation set.

Effectiveness

Grid search is believed to be less efficient than random search, particularly when tuning a large number of parameters. (Bergstra and Bengio (2012).

The reasoning is that typically in neural networks a few hyperparameters matter a great deal and most do not change the results much. Grid search looks at exponentially fewer values of each hyperparameter than random search does. In essence, grid search wastes far more time evaluating combinations of variables that don't matter.

1.34.5 Neural Architecture Search

The automatic design of the architecture of neural networks. Typically involves deciding aspects like the size, connections and type of layers as well as their activations.

Notable papers

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Tan and Le (2019)
Efficient Neural Architecture Search via Parameter Sharing, Pham et al. (2018)
Regularized Evolution for Image Classifier Architecture Search, Real et al. (2018)
Learning Transferable Architectures for Scalable Image Recognition, Zoph et al. (2017)
Neural Architecture Search with Reinforcement Learning, Zoph and Le (2016)

1.34.6 Random search

A simple algorithm that tests random combinations of hyperparameters.

As with grid search, it begins by deciding the hyperparameters to be evaluated. For example:

```
learning_rates = [0.001, 0.01, 0.1]
dropout_rates = [0.0, 0.2, 0.4, 0.6, 0.8]
num_layers = [12, 16, 20, 24]
```

Then random combinations of hyperparameters are chosen. For each one we train a model and calculate the accuracy on the validation set.

Extremely simple to implement and easy to parallelize.

Further reading

Random Search for Hyper-Parameter Optimization, Bergstra and Bengio (2012)

1.34.7 Reinforcement learning

Hyperparameter optimisation can be framed as a problem for reinforcement learning by letting the accuracy on the validation set be the reward and training with a standard algorithm like REINFORCE.

Used by

Neural Architecture Search with Reinforcement Learning, Zoph and Le (2016)
Efficient Neural Architecture Search via Parameter Sharing, Pham et al. (2018)

1.35 Modelling uncertainty

1.35.1 Calibration

The problem of getting accurate estimates of the uncertainty of the prediction(s) of a classifier or regressor.

For example, if a binary classifier gives scores of 0.9 and 0.1 for classes A and B that does not necessarily mean it has a 90% chance of being correct. If the actual probability of being correct (class A) is far from 90% we say that the classifier is **poorly calibrated**. On the other hand, if the model if it really does have a close to 90% chance of being correct we can say the classifier is **well calibrated**.

Binary classification

1. Train the classifier $\hat{y} = f(x)$ in the normal way
2. Construct a dataset with, for each row in the original dataset, the predicted score and the actual label.
3. Fit an isotonic regression $\bar{y} = g(\hat{y})$ to this data, trying to predict the label given the score. $\bar{y}$ can be used as a well-calibrated estimate of the true probability.

Multi-class classification

Reduce the problem to n one-vs-all binary classification tasks and use the method in the preceding section for each of them. Normalise the resulting distribution to ensure it sums to 1.

1.35.2 Measuring uncertainty

This section describes methods for estimating the uncertainty of a classifier. Note that additional methods may be necessary to ensure that this estimate is well-calibrated.

Classification

The uncertainty for a predicted probability distribution over a set of classes can be measured by calculating its entropy.

Regression

Unlike in classification we do not normally output a probability distribution when making predictions for a regression problem. The solution is to make the model output additional scalars, describing a probability distribution.

This could be:

- The Gaussian distribution. This only requires two parameters but may be over-simplifying if there aren't strong theoretical reasons to believe the distribution ought to be Gaussian or at least unimodal.
- A categorical distribution. This option allows a great degree of flexibility but requires a relatively large number of parameters. It also makes learning harder since the model has to learn for itself that the 14th category is numerically close to the 13th and 15th (Salimans et al., 2017).
- A mixture model. If the number of mixtures is chosen well this can represent a good middle ground between descriptiveness and efficiency.

Here is an example in full, using the normal distribution:

The network outputs two numbers describing the Normal distribution $N(\mu, \sigma^2)$. μ is the predicted value and σ^2 describes the level of uncertainty.

- The mean μ, outputted by a fully-connected layer with a linear activation.
- The variance σ^2, outputted by a fully-connected layer with a softplus activation. Using the softplus ensures the variance is always positive without having zero gradients when the input is below zero, as with the ReLU.

The loss function is the negative log likelihood of the observation under the predicted distribution:

$$L(y, \mu, \sigma) = -\frac{1}{2}\log(\sigma^2) - \frac{1}{2n\sigma^2}\sum_{i=1}^{n}(y-\mu)^2$$

Example papers

PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications, Salimans et al. (2017)

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al. (2016) - Section 9

1.36 Multimodal learning

Multimodal learning concerns tasks which require being able to make sense of multiple types of input, such as images and text, simultaneously.

1.36.1 Datasets

CLEVR

A synthetic dataset for visual question answering. Given a scene of objects of different shapes, colours and sizes the algorithm must answer questions such as *"What color is the cube to the right of the yellow sphere?"* and *"How many cylinders are in front of the small thing and on the left side of the green object?"*.

Notable results:

- 99.1% - Transparency by Design: Closing the Gap Between Performance and Interpretability in Visual Reasoning, Mascharka et al. (2018)
- 97.6% - Learning Visual Reasoning Without Strong Priors, Perez et al. (2017)

Introduced in

CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning, Johnson et al. (2016)

1.36.2 Image Captioning

TODO

1.36.3 Visual Question Answering (VQA)

Tasks where the algorithm must answer questions given some piece of visual information.

Datasets:

- CLEVR
- VQA

1.37 Natural language processing (NLP)

1.37.1 Datasets

Labelled

WMT

Parallel corpora for translation. Aligned on the sentence level.

Notable results in BLEU (higher is better):

English-to-German (2014)

- 28.4 - Attention is All You Need, Vaswani et al. (2017)
- 24.7 - Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Wu et al. (2016)
- 23.8 - Neural Machine Translation in Linear Time, Kalchbrenner et al. (2016)

English-to-French (2014)

- 41.8 - Attention is All You Need, Vaswani et al. (2017)
- 39.0 - Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Wu et al. (2016)

Other datasets

- bAbI - Dataset for question answering
- GLUE - Stands for General Language Understanding Evaluation. Assesses performance across 11 different tasks including sentiment analysis, question answering and entailment, more details of which can be found on their website. Leaderboard here.
- IMDB - Dataset of movie reviews, used for sentiment classification. Each review is labelled as either positive or negative.
- RACE - Reading comprehension dataset. Leaderboard here.
- RACE: Large-scale ReAding Comprehension Dataset From Examinations, Lai et al. (2017)
- SQuAD - Stanford Question Answering Dataset
- SuperGLUE - Harder successor to the GLUE dataset. Assesses performance across 10 different tasks (more details here). Leaderboard here.
- TIMIT - Speech corpus

Unlabelled

A list of some of the most frequently used unlabelled datasets and text corpora, suitable for tasks like language modelling and learning word embeddings.

PTB

Stands for 'Penn Treebank'. Notable results, given in word-level perplexity (lower is better):

- 35.8 - Language Models are Unsupervised Multitask Learners, Radford et al. (2019)
- 47.7 - Breaking the Softmax Bottleneck: A High-Rank RNN Language Model, Yang et al. (2017)
- 55.8 - Efficient Neural Architecture Search via Parameter Sharing, Pham et al. (2018)
- 62.4 - Neural Architecture Search with Reinforcement Learning, Zoph and Le (2016)
- 68.7 - Recurrent Neural Network Regularization, Zaremba et al. (2014)

Other datasets

- 1 Billion Word Language Model Benchmark
- Common Crawl
- Gigaword 5
- Project Gutenberg
- WikiText-2
- Shakespeare

1.37.2 Entailment

The task of deciding whether one piece of text follows logically from another.

1.37.3 Entity linking

The task of finding the specific entity which words or phrases refer to. Not to be confused with Named Entity Recognition.

1.37.4 FastText

A simple baseline method for text classification.

The architecture is as follows:

1. The inputs are n-grams features from the original input sequence. Using n-grams means some of the word-order information is preserved without the large increase in computational complexity characteristic of recurrent networks.
2. An embedding layer.
3. A mean-pooling layer averages the embeddings over the length of the inputs.
4. A softmax layer gives the class probabilities.

The model is trained with the cross-entropy loss as normal.

Proposed in
Bag of Tricks for Efficient Text Classification

1.37.5 Latent Dirichlet Allocation (LDA)

Topic modelling algorithm.

Each item/document is a finite mixture over the set of topics. Each topic is a distribution over words. The parameters can be estimated with expectation maximisation. Unlike a simple clustering approach, LDA allows a document to be associated with multiple topics.

Latent Dirichlet Allocation, Blei et al. (2003)

1.37.6 Morpheme

A word or a part of a word that conveys meaning on its own. For example, 'ing', 'un', 'dog' or 'cat'.

1.37.7 Named Entity Recognition (NER)

Labelling words and word sequences with the type of entity they represent, such as person, place or time.

Not to be confused with entity linking which finds the specific entity (eg the city of London) rather than only the type (place).

1.37.8 Part of speech tagging (POS tagging)

Labelling words with ADV, ADJ, PREP etc. Correct labelling is dependent on context - 'bananas' can be a noun or an adjective.

1.37.9 Phoneme

A unit of sound in a language, shorter than a syllabel. English has 44 phonemes. For example, the long 'a' sound in 'train' and 'sleigh' and the 't' sound in 'bottle' and 'sit'.

1.37.10 Polysemy

The existence of multiple meanings for a word.

1.37.11 Stemming

Reducing a word to its basic form. This often involves removing suffixes like 'ed', 'ing' or 's'.

1.38 Ranking

The task of retrieving the most relevant documents from a set given a query. If the ranking is personalized, a context including user history or location may also be taken into account. Often referred to as 'learning to rank'.

Ranking problems tend to be hard to optimise for since the performance of the algorithm depends on the order of the documents when they are sorted according to their predicted scores. This is non-differentiable.

1.38.1 Inversion

An instance where two documents have been ranked in the wrong order given the ground truth. That is to say the less relevant document is ranked above the more relevant one.

1.38.2 LambdaLoss

Builds upon LambdaRank.

NDCG-Loss2

$$L(y, s) = \sum_{y_i > y_j} \Delta NDCG(i, j) \log(1 + \exp(-\sigma(s_i - s_j)))$$

$$\Delta NDCG(i, j) = |G_i - G_j||\frac{1}{D_{|i-j|}} - \frac{1}{D_{|i-j+1|}}|$$

Proposed in

The LambdaLoss Framework for Ranking Metric Optimization, Wang et al. (2018)

1.38.3 LambdaMART

Combines the boosted tree model MART (Friedman, 1999) with LambdaRank.

Further reading

From RankNet to LambdaRank to LambdaMART: An Overview, Burges (2010)

1.38.4 LambdaRank

Builds upon RankNet.

The loss is a function of the labeled relevance y and the predicted score s, summing over pairs of relevance labels where $y_i > y_j$:

$$L(y, s) = \sum_{y_i > y_j} \Delta NDCG(i, j) \log(1 + \exp(-\sigma(s_i - s_j)))$$

where $\Delta NDCG(i, j)$ is the change in NDCG that would result from the ranking of documents i and j being swapped:

$$\Delta NDCG(i, j) = |G_i - G_j||\frac{1}{D_i} - \frac{1}{D_j}|$$

G and D are the gain and discount functions:

$$G_i = \frac{2^{y_i} - 1}{maxDCG}$$

$$D_i = \log(1 + i)$$

Proposed in
Learning to Rank with Nonsmooth Cost Functions, Burges et al. (2006)

Further reading
From RankNet to LambdaRank to LambdaMART: An Overview, Burges (2010)
The LambdaLoss Framework for Ranking Metric Optimization, Wang et al. (2018)

1.38.5 Listwise ranking

The loss function is defined over the list of documents, as opposed to just a pair of documents for example.

Example papers
SoftRank: Optimising Non-Smooth Rank Metrics, Taylor et al. (2008)

1.38.6 Metrics

See the main section on metrics or jump to one of its subsections:

- Cumulative Gain
- Discounted Cumulative Gain (DCG)
- Mean Reciprocal Rank (MRR)
- Normalized Discounted Cumulative Gain (NDCG)
- Precision@k

1.38.7 Pairwise ranking

Learning to rank is seen as a classification problem where the task is to predict whether a document A is more relevant than some other document B given a query.

Simple to train using the cross-entropy loss but requires more computational effort at inference time since there are $O(n^2)$ possible comparisons in a list of n items.

Example papers
Learning to Rank using Gradient Descent, Burges et al. (2005)
Learning to Rank with Nonsmooth Cost Functions, Burges et al. (2006)

1.38.8 Pointwise ranking

Poses learning to rank as a regression problem where a relevance score must be predicted given a document and query. A squared loss is typically used to minimise the difference between the predicted and target relevance.

Example papers

1.38.9 RankNet

A pairwise ranking algorithm. Can be built using any differentiable model such as neural networks or boosted trees. For a given pair of documents i and j the model computes the probability that i should be ranked higher than j:

$$P_{ij} = P(y_i > y_j) = \frac{1}{1 + \exp(-\sigma(s_i - s_j))}$$

Given the prediction, the model is then trained using the cross-entropy loss.

Proposed in
Learning to Rank using Gradient Descent, Burges et al. (2005)

Further reading
From RankNet to LambdaRank to LambdaMART: An Overview, Burges (2010)

1.38.10 SoftRank

A listwise ranking algorithm. Optimises a smoothed approximation of NDCG which is obtained by treating the scores as random variables.

Each score is viewed as being sampled from a Gaussian distribution centered on the true score.

Proposed in
SoftRank: Optimising Non-Smooth Rank Metrics, Taylor et al. (2008)

1.39 Training with limited data

1.39.1 Active learning

The learning algorithm requests examples to be labelled as part of the training process. Useful when there is a small set of labelled examples and a larger set of unlabelled examples and labelling is expensive.

1.39.2 Class imbalance problem

When one or more classes occur much more frequently in the dataset than others. This can lead to classifiers maximising their objective by predicting the majority class(es) all of the time, ignoring the features.

Methods for addressing the problem include:

- Focal loss
- Weight the loss function (increase the weight for the minority class)
- Oversampling the minority class
- Undersampling the majority class

1.39.3 Datasets

Omniglot

1623 handwritten characters from 50 alphabets with 20 examples of each character. Useful for one-shot learning. Introduced in One shot learning of simple visual concepts, Lake et al. (2011).

Notable results
20-way one shot accuracies are reported. This means one labelled example is provided from each of the 20 classes that were not in the training set. The task is then to classify unlabelled examples into these 20 classes.

- 98.2% - Object-Level Representation Learning for Few-Shot Image Classification, Long et al. (2018)
- 93.8% - Matching Networks for One Shot Learning, Vinyals et al. (2016)
- 88.1% - Siamese Neural Networks for One-shot Image Recognition, Koch et al. (2015)

miniImageNet

60,000 84x84 images from 100 classes, each with 600 examples. There are 80 classes in the training set and 20 in the test set. Much harder than Omniglot.

Introduced in Vinyals et al. (2016).

1.39.4 Few-shot learning

Classification where only a few (normally < 20) members of that class have been seen before.

1.39.5 Meta-learning

Learning from tasks with the goal of using that knowledge to solve other unseen tasks.

Further reading
Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML, Raghu et al. (2019)
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, Finn et al. (2017)

1.39.6 One-shot learning

Classification where only one member of that class has been seen before. Matching Networks achieve 93.2% top-5 accuracy on ImageNet compared to 96.5% for Inception v3.

Further reading

Object-Level Representation Learning for Few-Shot Image Classification, Long et al. (2018)

Matching Networks for One Shot Learning, Vinyals et al. (2016)

Siamese Neural Networks for One-shot Image Recognition, Koch et al. (2015)

1.39.7 Semi-supervised learning

Training using a limited set of labelled data and a (usually much larger) set of unlabelled data.

Ladder Network

A network designed for semi-supervised learning that also works very well for permutation invariant MNIST.

Simultaneously minimize the sum of supervised and unsupervised cost functions by backpropagation, avoiding the need for layer-wise pre-training. The learning task is similar to that of a denoising autoencoder, but minimizing the reconstruction error at every layer, not just the inputs. Each layer contributes a term to the loss function.

The architecture is an autoencoder with skip-connections from the encoder to the decoder. Can work with both fully-connected and convolutional layers.

There are two encoders - one for clean and one for noisy data. The clean one is used to predict labels and get the supervised loss. The noisy one links with the decoder and helps create the unsupervised losses. Both encoders have the same parameters.

The loss is the sum of the supervised and the unsupervised losses. The supervised cost is the cross-entropy loss as normal. The unsupervised cost (reconstruction error) is the squared difference.

The hyperparameters are the weight for the denoising cost of each layer as well as the amount of noise to be added within the corrupted encoder.

Achieved state of the art performance for semi-supervised MNIST and CIFAR-10 and permutation invariant MNIST.

Semi-Supervised Learning with Ladder Networks, Rasmus et al. (2015)

Self-training

Method for semi-supervised learning. A model is trained on the labelled data and then used to classify the unlabelled data, creating more labelled examples. This process then continues iteratively. Usually only the most confident predictions are used at each stage.

Unsupervised pre-training

Layers are first trained using an auto-encoder and then fine tuned over labelled data. Improves the initialization of the weights, making optimization faster and reducing overfitting. Most useful in semi-supervised learning.

Further reading

Why Does Unsupervised Pre-training Help Deep Learning?, Erhan et al. (2010)

1.39.8 Transfer learning

The process of taking results (usually weights) that have been obtained on one dataset and applying them to another to improve accuracy on that one.

Useful for reducing the amount of training time and data required.

1.39.9 Zero-shot learning

Learning without any training examples. This is made possible by generalising from a wider dataset.

An example is learning to recognise a cat having only read information about them - no images of cats are seen. This could be done by using Wikipedia with a dataset like ImageNet to learn a joint embedding between words and images.

Further reading
Zero-Shot Learning Through Cross-Modal Transfer, Socher et al. (2013)

1.40 Applications

1.40.1 Atari

Notable results

Below is the median human-normalized performance on the 57 Atari games dataset with human starts. The numbers are from Hessel et al. (2017).

- 153% - Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al. (2017)
- 128% - Prioritized Experience Replay, Schaul et al. (2015)
- 125% - A Distributional Perspective on Reinforcement Learning, Bellemare et al. (2017)
- 117% - Dueling Network Architectures for Deep Reinforcement Learning, Wang et al. (2015)
- 116% - Asynchronous Methods for Deep Reinforcement Learning, Minh et al. (2016)
- 110% - Deep Reinforcement Learning with Double Q-learning, Hassely et al. (2015)
- 102% - Noisy Networks for Exploration, Fortunato et al. (2017)
- 68% - Human-level control through deep reinforcement learning, Mnih et al. (2015)

Human starts refers to starting episodes at points randomly chosen from the set of human expert trajectories. They are used in the evaluation to avoid rewarding agents that have overfitted to their own trajectories.

1.40.2 Go

AlphaGo

Go-playing algorithm by Google DeepMind.

First learns a supervised policy network that predicts moves by expert human players. A reinforcement learning policy network is initialized to this network and then improved by policy gradient learning against previous versions of the policy network.

Finally, a supervised value-network is trained to predict the outcome (which player wins) from positions in the self-play dataset.

The value and policy networks are combined in an Monte Carlo Tree Search (MCTS) algorithm that selects actions by lookahead search. Both the value and policy networks are composed of many convolutional layers.

AlphaGo Zero

An advanced version of AlphaGo that beat its predecessor 100-0 without having been trained on any data from human games.

Note: AlphaGo Zero is not Alpha Zero applied to Go. They are different algorithms. AlphaGo Zero has some features specific to Go that Alpha Zero does not.

Training

AlphaGo Zero is trained entirely from self-play. The key idea is to learn a policy which can no longer be improved by MCTS. The algorithm maintains a 'best network' which is updated when a new network beats it in at least 55% of games.

During training moves are picked stochastically, with the amount of noise being decreased over time. This aids exploration.

Architecture and loss functions

The core network is a 20-layer ResNet with batch norm and ReLUs. It has two outputs:

The first predicts the value of the current game position. This is trained with a mean-squared error from the actual outcomes of played games. 1 if the player won and -1 if they lost. The second predicts the policy, given the current game position. This is trained with a cross-entropy loss and the policy resulting from MCTS.

Paper

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, Silver et al. (2017)

Blog posts

http://www.inference.vc/alphago-zero-policy-improvement-and-vector-fields

1.40.3 Poker

Unlike games like Chess and Go, Poker is an imperfect information game. This means that as well as having to maintain a probability distribution over the hidden state of the game, strategies like bluffing must also be considered.

Due to the amount of luck involved who is the better of two Poker players can take a very large number of games to evaluate.

Heads up no limit Texas Hold 'em

- Two players
- The cards start off dealt face down to each player.
- Cards in later rounds are dealt face up.
- The bets can be of any size, subject to an overall limit on the amount wagered in the game.

DeepStack

DeepStack is an AI for playing sequential imperfect-information games, most notably applied to heads up no-limit Texas Hold 'em Poker. It was the first algorithm to beat human professional players with statistical significance.

https://www.deepstack.ai/

DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker, Moravcik et al. (2017)

1.40.4 Starcraft

Compared to games like Go, Starcraft is hard for the following reasons:

1. Continuous action space. Means the conventional tree-search method cannot be applied. Even if it could be, the number of states to search through would be far too large.
2. Imperfect information. Not all of the environment is visible at once. This means the agent must not only seek to improve its position but also explore the environment.
3. More complex rules. There are multiple types of units, all of which have different available actions and interact in different ways.
4. Requires learning both low-level tasks (like positioning troops) and high-level strategy.
5. May require feints and bluffs.

2 and 5 may have been solved by the DeepStack poker playing system.

1.41 Basics

1.41.1 Absorbing state

See terminal state.

1.41.2 Action space

The space of all possible actions. May be discrete as in Chess or continuous as in many robotics tasks.

1.41.3 Behaviour distribution

The probability distribution over sequences of state-action pairs that describes the behaviour of an agent.

1.41.4 Bellman equation

Computes the value of a state given a policy. Represents the intuition that if the value at the next timestep is known for all possible actions, the optimal strategy is to select the action that maximizes that value plus the immediate reward.

$$V(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \max_{a'} V(s', a')]$$

Where r is the immediate reward, γ is the discount rate, s and s' are states and a and a' are actions. $V(s, a)$ is the value function for executing action a in state s.

1.41.5 Breadth

In the context of games with a discrete action space like Chess and Go, breadth is the average number of possible moves.

1.41.6 Control policy

See policy.

1.41.7 Credit assignment problem

The problem of not knowing which actions helped and which hindered in getting to a particular reward.

1.41.8 Depth

Length of the game on average.

1.41.9 Discount factor

Between 0 and 1. Values closer to 0 make the agent concentrate on short-term rewards.

1.41.10 Episode

Analogous to a game. Ends when a terminal state is reached or after a predetermined number of steps.

1.41.11 Markov Decision Process (MDP)

Models the environment using Markov chains, extended with actions and rewards.

1.41.12 Partially Observable Markov Decision Process (POMDP)

Generalization of the MDP. The agent cannot directly observe the underlying state.

1.41.13 Policy

A function, π that maps states to actions.

1.41.14 Regret

The difference in the cumulative reward between performing optimally and executing the given policy.

1.41.15 Reward function

Maps state-action pairs to rewards.

1.41.16 REINFORCE

Simple policy learning algorithm.

If a policy π_θ executes action a in state s with some corresponding value v the update rule is:

$$\Delta\theta = \alpha\nabla_\theta\pi_\theta(s,a)v$$

Where ∇_θ means the derivative with respect to θ.

Proposed in
Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning, Williams (1992)

1.41.17 Terminal state

A state which ends the episode when reached. No further actions by the agent are possible.

1.41.18 Trajectory

The sequence of states and actions experienced by the agent.

1.41.19 Transition function

Maps a state and an action to a new state.

1.41.20 Value function

The value of a state is equal to the expectation of the reward function given the state and the policy.

$$V(s) = E[R|s,\pi]$$

1.42 Explore-exploit dilemma

Refers to the trade-off between exploitation, which maximises reward in the short-term, and exploration which sacrifices short-term reward for knowledge which can increase rewards in the long term.

See the multi-armed bandit problem for an example.

1.42.1 Acquisition function

A function that decides the next point to sample while trying to maximize the cumulative reward, balancing exploration and exploitation.

Read the full explanation under Bayesian Optimization.

Examples of acquisition functions:

- Probability of Improvement
- Expected Improvement
- Upper Confidence Bound

Further reading
Lecture notes on Bayesian Optimization and Acquisition Functions, Roman Garnett

1.42.2 Boltzmann exploration

Method for addressing the explore-exploit dilemma and choosing actions off-policy. Decrease the temperature over time. High temperatures incentivize exploration and low temperatures prioritize exploitation.

$$\pi(a|s) = e^{\frac{Q(s,a)}{\tau}} / \sum_{a' \in A} e^{\frac{Q(s,a')}{\tau}}$$

As τ approaches 0, it approximates a maximum function. As τ becomes large (above 3), the uniform distribution is approximated. The place where the distribution of probabilities is proportional to the differences in the Q-values does not appear to be fixed but usually occurs around 0.5.

1.42.3 Count-based exploration

Use the count of the number of times an agent has visited a state to guide exploration.

Count-based exploration is only useful if the number of possible states is very small. If there are a large number of states the count for the vast majority of them will be zero, even if a very similar state has been visited before.

1.42.4 Entropy of the policy

The entropy of the policy's distribution over actions $H(\pi(s;\theta))$ can be added to the loss function to incentivize exploration.

Proposed in
Function optimization using connectionist reinforcement learning algorithms, Williams and Peng (1991)

Used by
Asynchronous Methods for Deep Reinforcement Learning, Mnih et al. (2016)

1.42.5 Epsilon-greedy policy

A method for inducing exploration during training. Choose the greedy policy with probability $1 - \epsilon$ and a random action with probability ϵ.

Unlike more sophisticated methods, epsilon-greedy cannot automatically decrease the exploration rate over time as the model becomes more certain.

1.42.6 Greedy policy

A policy that always selects the best action according to its value or Q function without any regard for exploration.

1.42.7 Intrinsic motivation

Technique in which exploration is motivated by the change in prediction error from learning a new piece of information.

Further reading
Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation, Kulkarni et al. (2016)

1.42.8 Multiple Armed Bandit Problem

Example of the explore-exploit tradeoff. Each machine (or bandit) provides a random reward from a probability distribution specific to that machine. A balance must be struck between picking the machine which has the best expected return (exploiting) and picking one's about which not much is known in the hope that they may be better (exploring).

The objective is to maximise the sum of the rewards over a predetermined number of moves.

A simple approach that works reasonably well is to use an epsilon-greedy policy.

1.42.9 Off-policy and on-policy learning

In **off-policy learning** the behaviour distribution does not follow the policy. This allows a more exploratory behaviour distribution to be chosen, using an epsilon-greedy policy or Boltzmann exploration, for example.

In **on-policy learning** the policy determines the samples the network is trained on. An example is SARSA.

1.42.10 Thompson sampling

Method for addressing the explore-exploit dilemma. An action is chosen with probability equal to the probability that that action maximises the expected reward.

This can be done by maintaining a probability distribution for the parameters of the environment (eg which actions are associated with which rewards in a given state). Then parameters can be sampled from this distribution and the expected optimal action chosen given the sampled parameters.

1.43 Search

In the context of reinforcement learning and most commonly in games, search refers to trying to find the value of an action in a particular state by looking ahead into the future, imagining possible moves and countermoves. Search is also sometimes referred to as lookahead search.

1.43.1 Alpha-beta pruning

A technique to reduce the number of nodes that need to be evaluated when doing search with the Minimax algorithm.

1.43.2 Minimax algorithm

Recursive search algorithm for computing the value of a state in a two-player zero-sum game.

It models the players in this way:

1. It (the first player) always picks the move that will lead to the state that maximizes its value function.
2. Its opponent always picks the move that will lead to the state which minimizes the value function.

It alternately picks and evaluates moves for itself and its opponent up to some maximum depth using depth-first search.

1.43.3 Rollout

A simulation of a possible future game trajectory.

Monte Carlo rollout

Searches to maximum depth without branching by sampling long sequences of actions with a policy. Can average over these to achieve super-human performance in backgammon and Scrabble.

1.43.4 Monte Carlo Tree Search

Uses Monte Carlo rollouts to estimate the value of each state in a search tree in order to improve a policy. The policy and value networks will be evaluated multiple times in each branch of the tree search. Converges to optimal play.

1.44 Temporal-difference learning

Temporal-difference learning optimizes the model to make predictions of the total return more similar to other, more accurate, predictions. These latter predictions are more accurate because they were made at a later point in time, closer to the end.

The TD error is defined as:

$$r_t + V(s_{t+1}) - V(s_t)$$

Q-learning is an example of TD learning.

1.44.1 Action-value function

Another term for the Q-function.

1.44.2 Actor-critic method

A type of on-policy temporal-difference method, as well as a policy-gradient algorithm.

The policy is the actor and the value function is the critic, with the 'criticism' being the TD error. If the TD error is positive the value of the action was greater than expected, suggesting the chosen action should be taken more often. If the TD error was negative the action had a lower value than expected, and so will be done less often in future states which are similar.

Unlike pure policy or value-based methods, actor-critic learns both a policy and a value function.

Apart from being off-policy, Q-learning is different as it estimates the value as a function of the state and the action, not just the state.

Asynchronous Advantage Actor-Critic (A3C)

An on-policy asynchronous RL algorithm. Can train both feedforward and recurrent agents.

Maintains a policy (the actor) $\pi(a_t|s_t;\theta)$ and a value function (the critic) $V(s_t;\theta_v)$ The policy and value functions are updated after t_{max} steps or when a terminal state is reached. The policy and value functions share all parameters apart from those in the final output layers. The policy network has a softmax over all actions (in the discrete case) and the value network has a single linear output.

The advantage function for doing action a_t in state s_t is the sum of discounted rewards plus the difference in the value functions between the states:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v), k \leq t_{max}$$

The loss function is:

$$L = \log \pi(a_t|s_t;\theta)(R_t - V(s_t;\theta_v)) + \beta H(\pi(s_t;\theta))$$

Where

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

and $H(\pi(s_t;\theta)$ is the entropy of the policy's distribution over actions. This term is used to incentivize exploration. β is a hyperparameter.

$R_t - V(s_t;\theta_v)$ is the temporal difference term.

It's multiplied by the probability assigned by the policy for the action at time t. This means policies which are more certain will be penalized more heavily for incorrectly estimating the value function.

Proposed in

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al. (2016)

1.44.3 Q-learning

Model-free iterative algorithm to find the optimal policy and a form of temporal difference learning.

Uses the update rule:

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

where $Q(a,s)$ is the value of performing action a in state s and performing optimally thereafter. s' is the state that results from performing action a in state s.

The Q-function

Expresses the expected total reward from taking the action in the given state and following the policy thereafter. Also known as the action-value function.

$$Q^{\pi}(s,a) = E[R|s,a,\pi]$$

Eventually converges to the optimal policy in any finite MDP. In its simplest form it uses tables to store values for the Q function, although this only works for very small state and action spaces.

Deep Q-learning

A neural network is used to approximate the optimal action-value function, $Q(s,a)$ and the actions which maximise Q are chosen.

The action-value function is defined according to the Bellman equation.

The CNN takes an image of the game state as input and outputs a Q-value for each action in that state. This is more computationally efficient than having the action as an input to the network. The action with the largest corresponding Q-value is chosen.

Uses the loss function:

$$L = \mathbb{E}_{s,a}[(y - Q(s,a;\theta))^2]$$

where the target, y is defined as:

$$y = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q(s',a';\theta)|s,a]$$

This means the target depends on the network weights, unlike in supervised learning. The loss function tries to change the parameters such that the estimate and the true Q-values are as close as possible, making forecasts of action-values more accurate.

Periodically freezing the target Q network helps prevent oscillations or divergence in the learning process.

Further reading

Playing Atari with Deep Reinforcement Learning, Mnih et al. (2013)

Human-level control through deep reinforcement learning, Mnih et al. (2015)

Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al. (2017)

Experience Replay

Sample experiences (s_t, a_t, r_t, s_{t+1}) to update the Q-function from a **replay memory** which retains the last N experiences. Mnih et al. (2013) set N to 1 million when training over a total of 10 million frames.

Contrast this with on-policy learning algorithms learn from events as they experience them. This can cause two problems:

1. Most gradient descent algorithms rely on the assumption that updates are identically and independently distributed. Learning on-policy can break that assumption since the update at time t influences the state at the next timestep.
2. Events are forgotten quickly. This can be particularly harmful in the case of rare but important events.

Both of these problems are solved by using experience replay.

The use of a replay memory means it is necessary to learn off-policy.

Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching, Lin (1992)
Playing Atari with Deep Reinforcement Learning, Mnih et al. (2013)

Prioritized Experience Replay

Samples from the replay memory according to a function of the loss. In contrast, in the standard approach (eg Mnih et al. (2013)) past experiences are selected uniformly at random from the replay memory.

TODO

Proposed in
Prioritized Experience Replay, Schaul et al. (2015)

Distributional Q-learning

Models the distribution of the value function, rather than simply its expectation.

Proposed in
A Distributional Perspective on Reinforcement Learning, Bellemare et al. (2017)

Multi-step bootstrap targets

Replace the expression for the target y in the original deep Q-learning loss function with a sum of discounted rewards and action-values:

$$y = R_t^{(n)} + \gamma^n \max_{a'} Q(s_{t+n}, a')$$

where

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}$$

The motivation for multi-step bootstrap targets is to speed up learning.

Hessel et al. (2017) set the hyperparameter n equal to 3.

Asynchronous Methods for Deep Reinforcement Learning, Mnih et al. (2016)
Learning to Predict by the Methods of Temporal Differences, Sutton (1988)

Noisy parameters

A method for helping exploration when training that can be more effective than traditional epsilon-greedy appraoch. The linear component $y = wx + b$ of the layers in the network are replaced with:

$$y = (\mu_w + \sigma_w * \epsilon_w)x + (\mu_b + \sigma_b * \epsilon_b)$$

where μ_w and σ_w are learned parameter matrices of the same shape as w in the original equation. Similarly, μ_b and σ_b are learned parameter vectors and have the same shape as b. ϵ_w and ϵ_b also have the same shape as w and b respectively, but are not learnt - they are random variables.

Since the amount of noise is learnt no hyperparameter-tuning is required, unlike epsilon-greedy, for example.

The noise parameters can be specified in two ways:

- Independent Gaussian noise - Learn one noise parameter for each parameter in the main network.
- Factorised Gaussian noise - The matrix of noise parameters is factorized into two vectors. This means the number of noise parameters needed for each layer is linear in its size rather than quadratic, as it is with independent Gaussian noise.

Proposed in
Noisy Networks for Exploration, Fortunato et al. (2017)

Rainbow

Approach that combines a number of existing improvemenets on the DQN algorithm to reach state of the art performance on the Atari 2600 benchmark.

It combines:

- Double DQN
- Prioritized Experience Replay
- Dueling Networks
- Multi-step bootstrap targets
- Distributional DQN
- Noisy DQN

Rainbow: Combining Improvements in Deep Reinforcement Learning, Hessel et al. (2017)

1.44.4 SARSA

An algorithm for learning a policy. Stands for state-action-reward-state-action.

The update rule for learning the Q-function is:

$$Q(s_t, a_t) := Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Where $0 < \alpha < 1$ is the learning rate.

Pseudocode:

```
1. Randomly initialize Q(s,a)
2. While not converged:
3.   Choose the action that maximizes Q(s,a)
4.   Compute the next state, given s and a.
5.   Apply the update rule for the Q-function.
```

Unlike Q-learning, SARSA is an on-policy algorithm and thus learns the Q-values associated with the policy it follows itself. Q-learning on the other hand is an off-policy algorithm and learns the value function while following an exploitation/exploration policy.

1.45 Types of policy-learning algorithms

1.45.1 Model-based reinforcement learning

Models the environment in order to predict the distribution over states that will result from a given state-action pair.

1.45.2 Model-free reinforcement learning

Algorithms that learn the policy without requiring a model of the environment. Q-learning is an example.

1.45.3 Off-policy learning

The behaviour distribution does not follow the policy. Typically a more exploratory behaviour distribution is chosen. An example is Q-learning.

1.45.4 On-policy learning

The policy determines the samples the network is trained on. Can introduce bias to the estimator. An example is SARSA.

1.45.5 Policy-based method

Does not use a value function. Learns the policy explicitly, unlike value-based methods which instead choose the action which maximises the value function.

1.45.6 Policy gradient method

Policy learning algorithm. Iteratively alternates between improving the policy given the value function and the value function under the current policy.

1.45.7 Value-based methods

Have an implicit policy based on choosing the action which maximises the value function.