An Introduction to Machine Learning

Srijith Rajamohan, Ph.D.
Virginia Tech

October 16, 2018

ML Basics

ML Taxonomy

Classification - Classify an input as belonging to a particular class from a group of classes

Regression - Produce a numerical result for an input

These can be further subdivided into *Supervised*, *Unsupervised*, *Semi-supervised* and *Reinforcement Learning* based on whether they are trained with human supervision or not

ML Taxonomy contd..

Instance-based - The system learns the training samples by heart and generalizes to new cases

Model-based - Build a model from the examples and use the model to predict new cases

ML Taxonomy contd..

Discriminative Learning

We do not model the data distribution explicitly

A functional form is adopted and the parameters are estimated by minimizing a cost function

Simpler models can be used but the input data distribution is ignored even though it can carry important information

Generative Learning

By contrast the data distribution is used here

We end up with one distribution per class, which has to be learned

Parametric Methods

Assume that the samples come from a distribution

We attempt to determine the parameters of this distribution, called the sufficient statistics

Once we have these parameters, we can now make predictions

Fixed number of parameters

Non-parametric Methods

Non-parametric method have parameters too

However the number of parameters depend on the size of the data and could theoretically be infinite

Increased need for memory and computation, as a function of data size, for non-parametric methods

A little bit of Probability

A *probability distribution* of a discrete random variable is given by $P(X)$ - Probability mass function

The probability distribution of continuous random variables is denoted as $p(x)$ - also called Probability density function

$p(x_i|C)$ is the *conditional probability* that an event belonging to class 'C' has value x_i, assuming continuous x. Or seen another way, given class 'C' the probability you would see the value x_i

A little bit of Probability contd..

A *prior* or knowledge that we have beforehand of the class is given by $P(C_i)$

$p(x_i)$ is the *evidence* or *marginal probability* that an observation 'x_i' is seen regardless of the class

Class likelihood is defined as $p(x_i|C_i)$, the conditional probability we saw earlier

For class prediction given an input, we define a *posterior* $P(C_i|x_i)$, which is the conditional probability that given 'x_i', the probability of it being in class 'C_i'

A little bit of Probability contd..

For example, think of two classes related to the weather- it rains/ does not rain and we have observations of temperatures for both of these classes

The prior is the probability of these classes, that it either rains or it doesn't. The evidence indicates the probability of seeing this temperature measurement.

The likelihood indicates, given a temperature, the probability that belonged to one of the classes, for .e.g 'Rained'

The posterior here is the probability, given a temperature measurement, the probability of it raining or not raining

Key Point - Bayes Rule

Bayes rule can be summarized as:

$$Posterior = \frac{Prior}{Evidence} \times Likelihood \tag{1}$$

Or in mathematical terms

$$P(C_i|x_i) = \frac{P(C_i) \times p(x_i|C_i)}{p(x_i)} \tag{2}$$

Entropy and Information

Given a discrete variable 'x', it's entropy measures randomness over all possible outcomes of that variable.

$$H(x) = -\sum_{x} P(x) log(P(x)) \tag{3}$$

Similarly the average mutual information between two random variables 'x' and 'y' is defined as

$$I(x,y) = -\sum_{x}\sum_{y} P(x,y) log(\frac{P(x,y)}{P(x)P(y)}) \tag{4}$$

If the average mutual information is 0, the two variables are statistically independent

Kullback-Leibler Divergence

Used to measure the dissimilarity between two PDFs.

$$KL(p||q) = \int_{-\infty}^{\infty} p(x) log \frac{p(x)}{q(x)} dx \tag{5}$$

KL divergence is 0 if both distributions are the same.

The KL divergence metric is not symmetric and does not satisfy the condition to be a distance metric, use Jensen-Shannon divergence for that

Mutual Information can be written as the KL divergence between $p(x, y)$ and $p(x)p(y)$

Maximum Likelihood Estimation

Given a i.i.d sample $X = \{x^t\}_1^N$, drawn from a distribution with a probability density function $p(x|\theta)$ defined up to parameters θ. The maximum likelihood estimate of the parameter θ given the sample is the θ that makes X the most likely to be drawn and can be determined from the product of the likelihoods of the individual points.

$$I(\theta|X) = \prod_{t=1}^{N} p(x^t|\theta) \tag{6}$$

Maximum Likelihood Estimation contd..

We want to maximize the likelihood term $I(\theta|X)$ above. Sometimes we take the logarithm of the above term to form the log likelihood.

$$\mathcal{L}(\theta|X) = \mathcal{P}(X|\theta) = \sum_{t=1}^{N} log(p(x^t|\theta)) \tag{7}$$

To find the θ that maximizes the chance of drawing the samples 'X', we differentiate $\mathcal{L}$ w.r.t. θ and we set $\frac{\partial \mathcal{L}}{\partial \theta} = 0$ to solve for θ

Maximum A Posteriori Estimate

If there is a priori information about the distribution of θ, we can use that information by considering θ as a random variable.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} \tag{8}$$

To make a prediction using this parametric distribution $p(\theta|X)$ and if we try to do prediction using an estimator given by $y = g(x|\theta)$ like we do in regression, we now have to integrate in the form to get the result

$$y = \int g(x|\theta)p(\theta|X)d\theta \tag{9}$$

since θ is a distribution instead of a point estimate

Maximum A Posteriori Estimate contd..

Usually not possible to do the integration over the density, so we replace the density with a point estimate.

If the posterior density $p(\theta|X)$ has a narrow peak around its mode, we can use *maximum a posteriori estimate* to compute an estimate for θ

$$\theta_{MAP} = argmax_{\theta}[p(\theta|X)] \quad (10)$$

$$y_{MAP} = g(x|\theta_{MAP}) \quad (11)$$

Note that in the first equation above on the previous page, the first term in the numerator is the same as the equation for the MLE estimate.

If we have no prior information about θ, the prior distribution is flat and the posterior will have the same form as the likelihood, i.e. MAP = MLE .

Bayes Estimator

Another estimate is to compute the expected value of θ

So instead of taking the maximum value from the mode of the distribution, we compute the weighted average of the θ over its posterior distribution (or in other words the expected value)

The assumption is that the mean value for a random variable is the best estimate for it

If the posterior is not unimodal both the MAP and Bayes' estimators lose information

$$\theta_{Bayes} = \int \theta p(\theta|X) d\theta \tag{12}$$

Mean-square Error of the Estimator

Since the parameter θ depends on the training samples, θ is a random variable and can be represented as $\hat{\theta}$. Let us assume that the true estimator is θ_0

$$MSE = E[(\hat{\theta} - \theta_0)^2] \tag{13}$$

Simplifying the above, we get

$$MSE = E[(\hat{\theta} - E[\hat{\theta}])^2] + (E[\hat{\theta}] - \theta_0)^2 \tag{14}$$

The first term above is variance and the second term is the square of bias

Bias-Variance Tradeoff

The MSE has error from the variance of the estimator around it's mean value

The second contribution to the error is from the difference of the estimator mean from the optimal estimate

One cannot make both terms small simultaneously

This is called the bias variance trade-off

Bias-Variance Tradeoff contd..

In order to reduce the bias one has to increase the model complexity, which increases the variance as we change the training data.

Increasing the model complexity can also result in over fitting

Decreasing the model complexity can reduce variance but this increases the bias

To reduce both terms simultaneously increase the number of training data points and increase the complexity of the model carefully. After a certain point, increasing either can result in an increase in the MSE

Hyperparameter

What is a *hyperparameter*?

It is a learning parameter of the algorithm, not a parameter of a model, e.g. the learning rate of an algorithm

Determines how the algorithm learns/performs

Must be set prior to learning

Determining the right hyperparameters is problem-specific often

Batch vs Online Learning

Batch - The algorithm is run using the entire data. Better for smaller data size, e.g. using *Gradient Descent*

In *batch* (also called offline learning) learning, if you have new data you need to train the whole model from scratch

Online - Model learns incrementally on the fly. The only option for Big Data, e.g. using *Stochastic Gradient Descent*

In *online* learning the data can be fed individually or in small batches called *mini-batches*

Cross-validation

Once the model is trained you have to determine how well it performs

The data is divided into a training set and a test set

Tuning the *hyperparameters* can be done with a third set called a validation set

The goal of cross validation is two-fold: to get an estimate for the variance or consistency of the models performance across different datasets and to fine-tune the hyperparameters to get a better model

K-fold Cross-validation

In *k-fold cross-validation* the training set is divided into k non-overlapping folds, the training is performed on different k-1 folds and tested against the one left out

This gives us an understanding of how the model performance varies for those model parameters obtained using a certain combination of hyperparameters

Repeat this process by changing the hyperparamters

This process of fine-tuning will give you a better model whose performance is a predictable

Overfitting and Underfitting

Tension between optimization and generalization.

Optimization refers to the process of adjusting a model to get the best performance possible on the training data

Generalization refers to how well the trained model performs on data it has never seen before

The error rate on new cases is called the generalization error

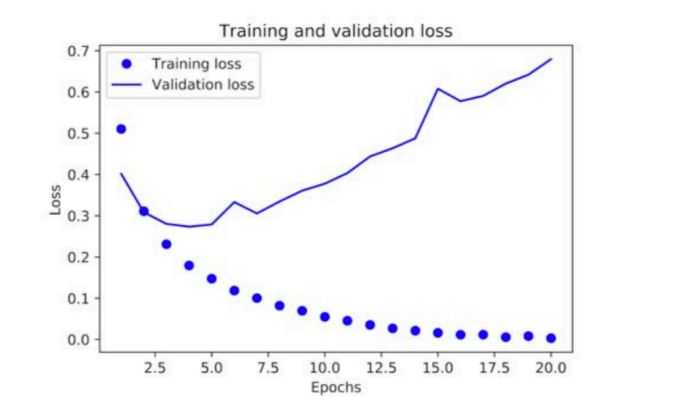
Overfitting and Underfitting contd...

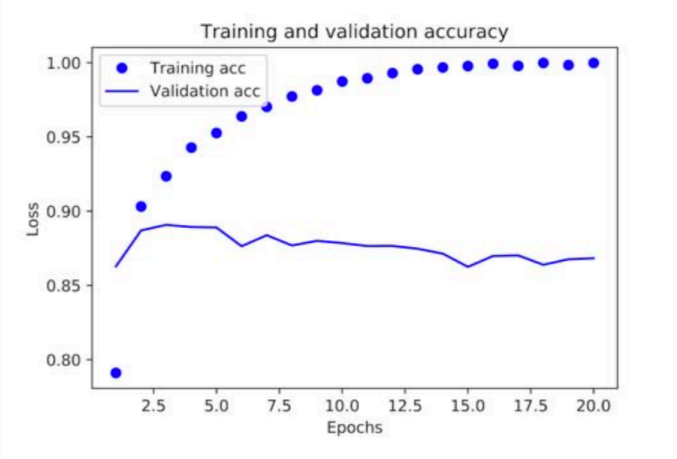
At the beginning of training, optimization and generalization are correlated: the lower the loss on training data, the lower the loss on test data. While this is happening, your model is said to be **underfit**: there is still progress to be made; the network hasn't yet modeled all relevant patterns in the training data

But after a certain number of iterations on the training data, generalization stops improving, and validation metrics stall and then begin to degrade: the model is starting to **overfit**

Overfitting

These graphs are typical of overfitting





Challenges in Machine Learning

Insufficient quantity of training data

Representative training data

Poor quality of data

Using irrelevant features

Over fitting the model

Under fitting the model

Algorithms

Gradient Descent

Can be used to find the solution of non-linear equations iteratively

Consider our solution space as a n-dimensional space

If it is a convex equation, we can think of it as an n-dimensional bowl that has a unique minimum

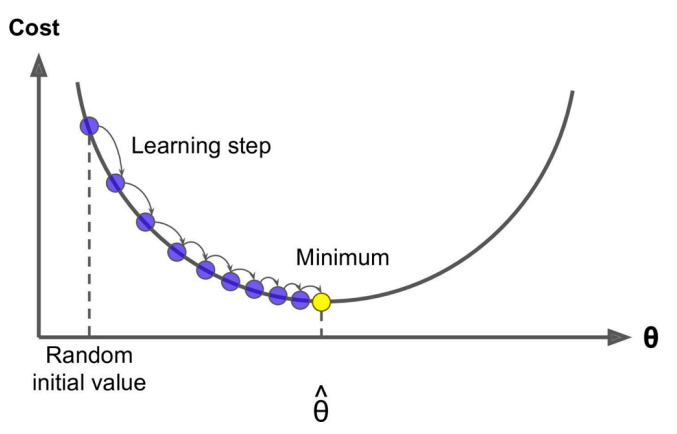
Starting from a random location in the bowl, we find the local gradient at that location for the cost function

We move down in the direction of decreasing gradient. Once the gradient is zero, you have attained the minimum

Gradient Descent contd..

The size of the steps are determined by the *learning rate* hyperparameter

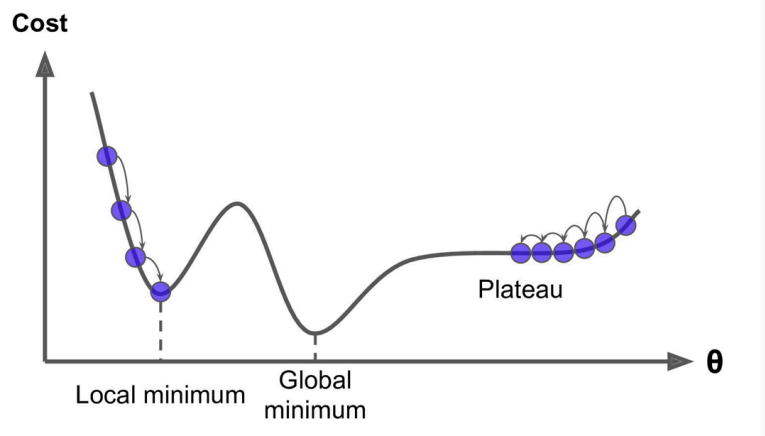
Too small and the algorithm will take an excessive amount of time to converge, too large and the algorithm may diverge



Gradient Descent and Non-convex Functions

Several local minima and GD may fail to converge to the global minimum

Depends on the starting point

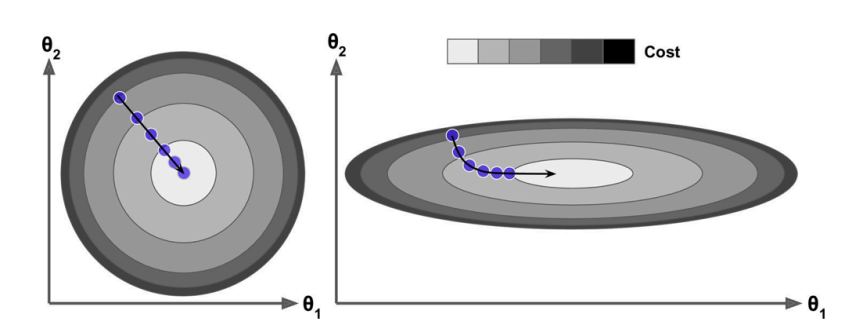


Gradient Descent and Feature Scaling

All features should have the same scale, if not convergence can be slow

Two ways to perform feature scaling: normalization and standardization

Standardization is less affected by outliers



Batch Gradient Descent

Compute the gradient with respect to each model parameter θ_j

Computed using the full dataset

Scales well with the number of features

Can be slow for really large data

Update using batch gradient for Linear Regression

The derivative of the MSE cost function w.r.t each parameter indicated by index 'j' is shown below. The summation over 'i' is for each instance of the observation 'x'

$$\frac{\partial}{\partial \theta_j} MSE(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^{m} (\boldsymbol{\theta}^T \cdot \boldsymbol{x}^i - \boldsymbol{y}^i) x_j^i \tag{15}$$

Update the weights or parameters as shown below

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_\theta MSE(\boldsymbol{\theta}) \tag{16}$$

Stochastic Gradient Descent

Instead of using the entire data set, only one instance is used to update the gradient

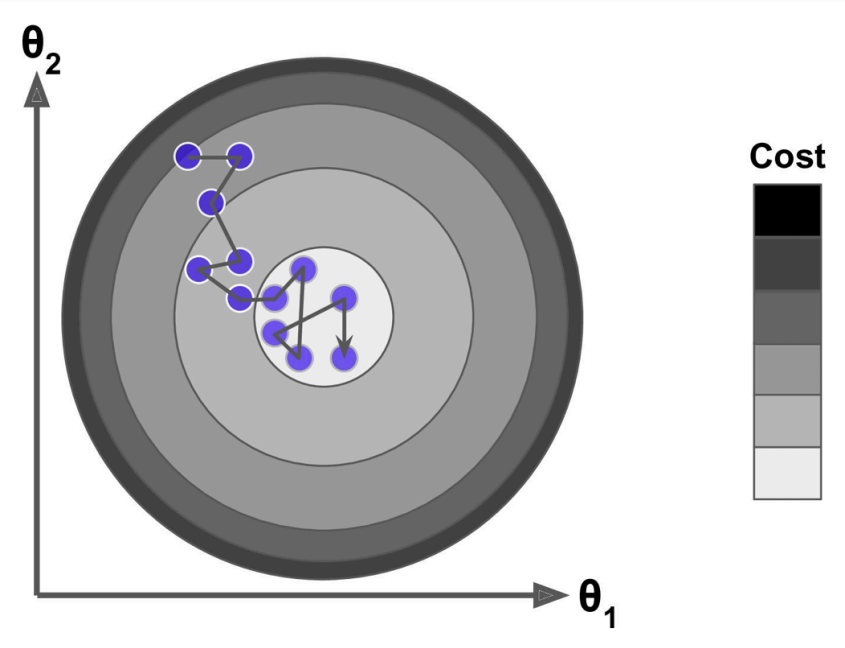
Stochastic in nature and therefore much less regular than batch gradient descent

The convergence irregularity can also be advantage for cost functions with local minimum because it can kick out of local minimum

Gradual reduction of the learning parameter called *simulated annealing*

Stochastic Gradient Descent contd..

Convergence issues of Stochastic GD



Mini-batch Gradient Descent

Instead of using the entire data set or only one instance, small mini-batch samples are used to update the gradient

Randomly sampled

Convergence less erratic than Stochastic Gradient Descent

Supervised Learning

Examples of Supervised Learning

k-Nearest Neighbors

Linear Regression

Logistic Regression

Support Vector Machines (SVMs)

Decision Trees

Random Forests

Neural networks

Linear Regression

Assume a linear relationship between the data denoted by **x**, which is a vector of features for a single training instance and the output denoted by $\hat{y}$. $\boldsymbol{\theta}$ is a vector of model parameters

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \tag{17}$$

In vector notation we get

$$\hat{y} = \boldsymbol{\theta}^T \cdot \boldsymbol{x} = h_\theta(x) \tag{18}$$

The term $h_\theta(x)$ is called the hypothesis function

MSE for Linear Regression

For a set of data denoted by the matrix **X**, we can find the list of parameters θ_i that satisfies the condition for all data instances

$$MSE(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{\theta}^T \cdot \boldsymbol{x}^i - y^i)^2 \tag{19}$$

The Normal Equation gives the closed-form solution of θ that minimizes the error above

$$\hat{\theta} = (\boldsymbol{X}^T \cdot \boldsymbol{X})^{-1} \cdot \boldsymbol{X}^T \cdot \boldsymbol{y} \tag{20}$$

Computational Complexity

Computational complexity of the normal equation is the order of $O(n^{2.4})$ to $O(n^3)$

Works well for large data sets if they fit into memory, if so we get the solution from all the parameters in closed-form

If the data is too large we have to resort to iterative training algorithms such as Gradient Descent

Logistic Regression

Inspite of the name it is actually used for classification

Estimates the probability that an instance belongs to the particular class

Uses the logistic/logit function (also called a sigmoid function) to compute a probability $\hat{p}$

$$\hat{p} = \sigma(\boldsymbol{\theta}^T \cdot \boldsymbol{x}) \tag{21}$$

where

$$\sigma(t) = \frac{1}{1 + e^{-t}} \tag{22}$$

MSE for Logistic Regression

Cost function is similar to linear regression

We compute the gradient with respect to all θ_i

The MSE is proportional to the term shown below

$$MSE(\theta) \propto \frac{1}{m} \sum_{i=1}^{m} (\sigma(\boldsymbol{\theta}^T \cdot \boldsymbol{x}^i) - y^i)^2 \quad (23)$$

Cost Function for Logistic Regression

So how do we train this for binary classification?

Let y be the real output and $\hat{y}$ be the computed output and $\hat{p}$ be the computed probability

We compute the gradient with respect to all θ_i

$$\hat{y} = \left\{ \begin{array}{l} 0, \hat{p} < 0.5 \\ 1, \hat{p} > 0.5 \end{array} \right\} \tag{24}$$

The cost function is given by

$$c(\theta) = \left\{ \begin{array}{rl} -log(\hat{p}), & y = 1 \\ -log(1-\hat{p}), & y = 0 \end{array} \right\} \tag{25}$$

Log Loss Cost Function.

The cost function is designed such that it gives a high value for an incorrect prediction

A log loss function using the above cost function, but averaged out over the entire training data, can be written as shown below

$$C(\theta) = -1.0 \cdot \frac{1}{m} \sum_{i=1}^{m} [y^i \cdot log(\hat{p}^i) + (1 - y^i) \cdot log(1 - \hat{p}^i)]$$

(26)

Compute the partial derivatives with respect to the parameters θ and solve for them using a gradient based approach

Artificial Neural Network - Perceptrons

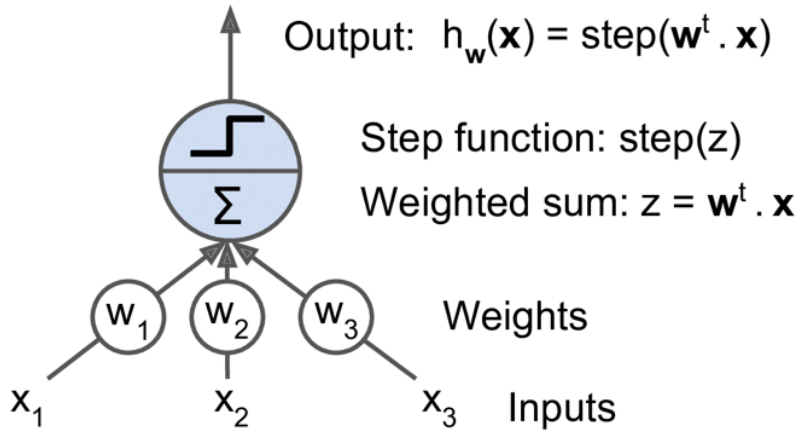
Computes a weighted sum of inputs then uses a linear threshold

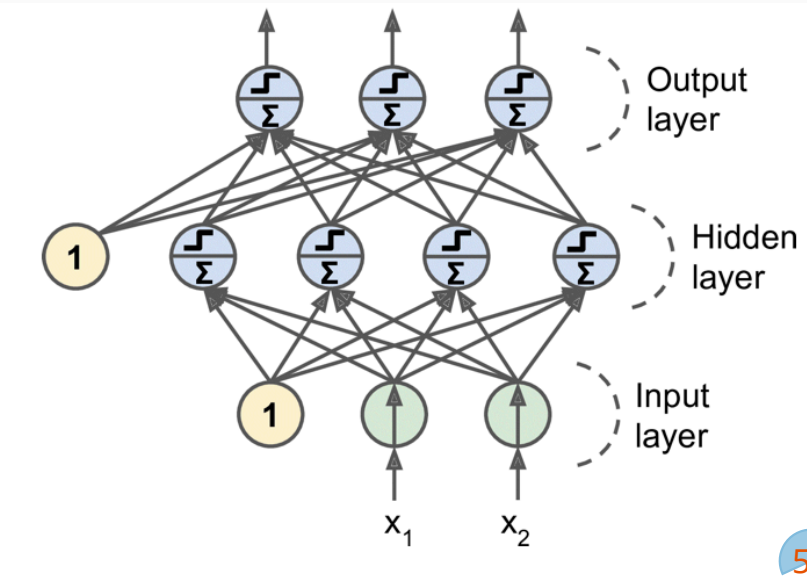
Instances have to be linearly separable

Unlike logistic regression does not output class probabilities, instead uses a hard threshhold

Artificial Neural Network - Perceptrons contd..

- Linear threshold unit
- Multi-layer perceptron added a *hidden layer*
- In 1986 introduced Back propagation also now known as the Reverse-mode autodiff
- Non-linear threshold units such as the hyperbolic tangent function





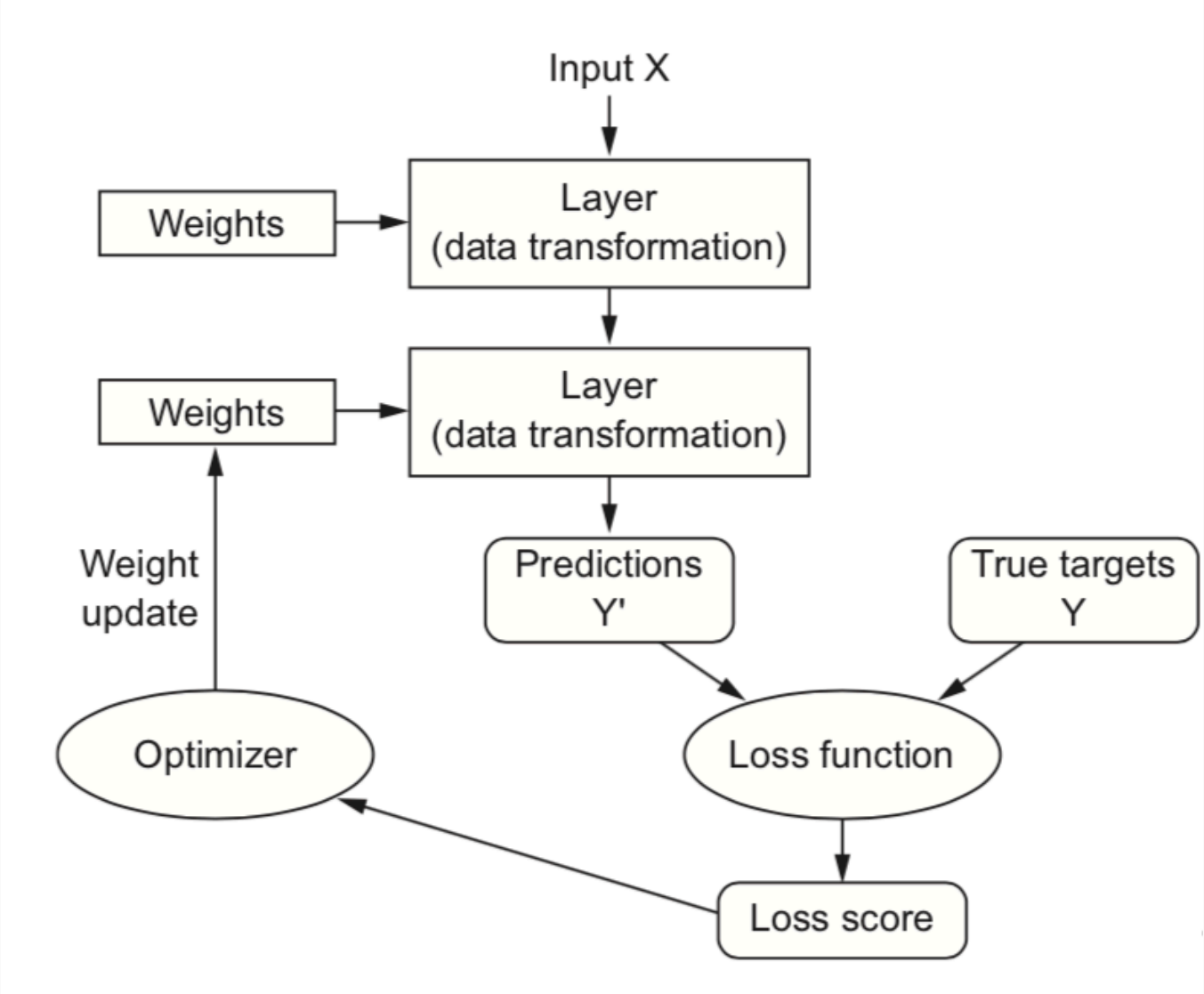
Multilayer Perceptrons

Multilayer Perceptrons are universal approximators

You can approximate any function (including non-linear) with a hidden layer that has non-linear activation functions

Although theoretically you can do this, not practical since ’d’ inputs might need 2^d hidden units

Structure of a multi-layer neural network



Neural Network

Compute weighted sum of outputs across all layers as $\mathbf{w}^T \cdot \mathbf{x}$

Apply an activation function to the outputs as $\sigma(\mathbf{w}^T \cdot \mathbf{x})$, where σ could be a step function, ReLU, sigmoid, tanh etc.

Compute the loss function.

Use back-propagation to calculate weight updates for all nodes

Repeat till weight updates are below a certain defined threshold

Neural Network

For a 'k-parallel' perceptron use the online update rule to increment the weights (i is output, j is input, t is index)

$$\Delta w_{ij}^t = -\eta \frac{\partial E_i}{\partial w_j} = \eta(y_i^t - \hat{y}_i^t)x_j^t \tag{27}$$

The above can be derived from gradient descent, or taking the derivative of the squared error function $\nabla_w \mathbf{E}$. For a single output neural network with multiple layers, use the backpropgation algorithm to compute the gradient w.r.t any node w_j in layer 'h' as

$$\Delta w_{ij}^t = -\eta \frac{\partial E}{\partial w_j} = -\eta \frac{\partial E_i}{\partial y^t} \frac{\partial y^t}{\partial zn^t} \cdots \frac{\partial zh^t}{\partial wh_j^t} \tag{28}$$

More info on autodiff:

`https://srijithr.gitlab.io/post/autodiff/`

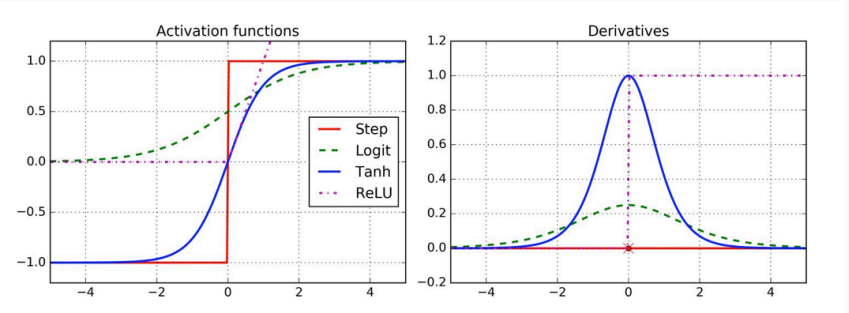
Neural Network - Activation functions

Logistic function has a mean of 0.5; saturates towards the ends with a derivative close to zero

Hyperbolic tangent has a mean of zero and is slightly better in deep networks

ReLU does not saturate, faster to compute and can possibly result in sparse networks

Activation functions and their derivatives



Tuning Network Size

Simpler networks are better generalizers (!)

Structural adaptation into the learning algorithm, either a constructive or destructive approach

Weight decay is an example of a destructive method

Dynamic node creation is an example of a constructive method

Decision Trees

Consists of tree-like structure where decisions are made at each node

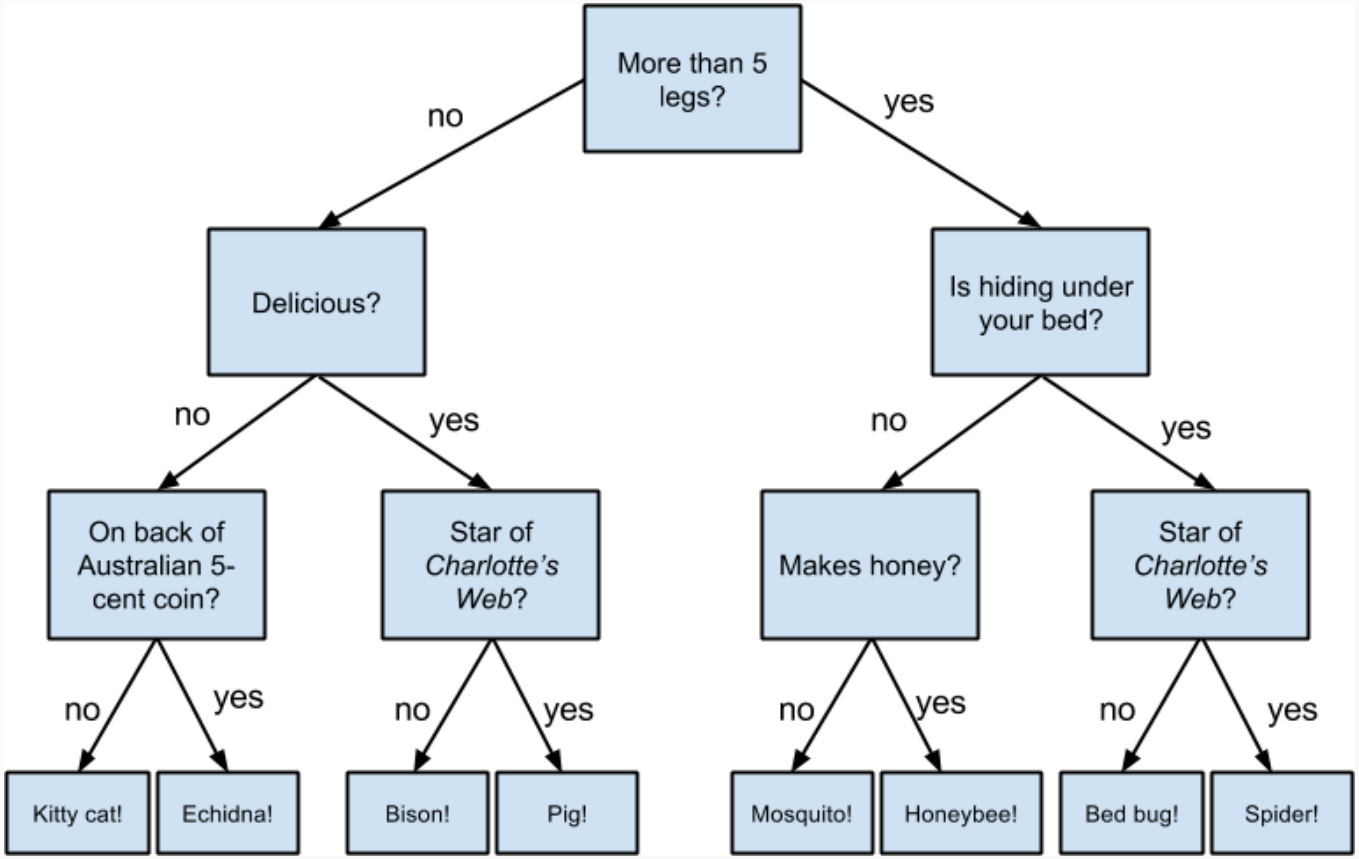
The goal at each node is to determine which feature is to be tested and the threshold for this feature

Once traversed all the way down, it ends up with a class(for a classifier)

Feature space is partitioned using hyperplanes, using a series of questions

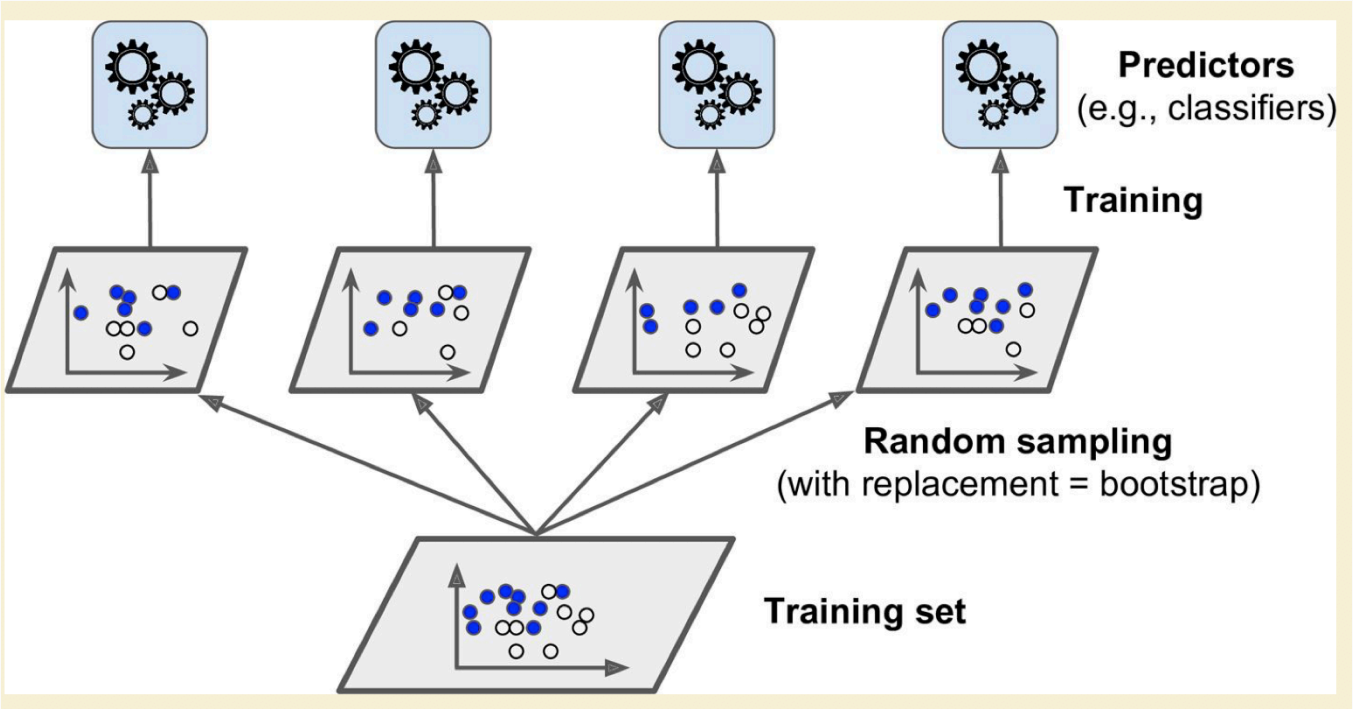
Splitting criterion, stop splitting rule and class assignment rule

Example of a Decision Tree



Bootstrap Aggregation - Bagging

One way to get a diverse set of classifiers is to use very different training algorithms. Another approach is to use the same training algorithm (classifier) but to train them on different random subsets of the training set.



Boostrap Aggregating - Bagging

Used to reduce the variance and improve the generalization performance

Uses bootstrapping which is sampling with replacement, pasting is sampling without replacement

This is done a number of times and the results are aggregated

For classification the class is predicted by a majority vote

Random Forests

Ensemble of trees trained with bagging (sometimes pasting)

Combines bagging with random feature selection - uses a subset of the features at each node

This extra randomness improves performance

More successful with classification tasks as opposed to regression tasks

Boosting is combining several *weak learners* sequentially to correct its predecessor to form a *strong learner*

Unsupervised Learning

Unsupervised Learning

We wish to learn the inherent structure of our data without using explicitly-provided labels

Since no labels are provided, there is no specific way to compare model performance in most unsupervised learning methods.

Two common use-cases for unsupervised learning are exploratory analysis and dimensionality reduction.

Examples of Unsupervised Learning

Clustering - e.g. k-Means, Hierarchical Cluster Analysis, Expectation Maximization

Visualization and Dimensionality Reduction - e.g. Principal Component Analysis, Kernel PCA, Multidimensional Scaling, t-distributed Stochastic Neighbor Embedding

Association Rule learning - e.g. Apriori, Eclat

Unsupervised Learning - Clustering

Class labels are not known

We want to find hidden structures in the data

We want to find a grouping of data such that similar items are together

Exploratory in nature

k-Means Clustering

Randomly pick k centroids for the clusters

Assign each sample to the nearest centroid

Compute the centroid again after all points have been assigned to clusters

Repeat steps two and three above till the points do not change cluster assignment

Distance can be based on any metric, commonly euclidean

The Curse of Dimensionality

Distance between two points is very large in high-dimensional space

New instances will be far away from training instances making predictions less reliable

Working in higher dimensions can lead to overfitting the model

The size of the training set required for higher dimensions grows exponentially with the number of dimensions

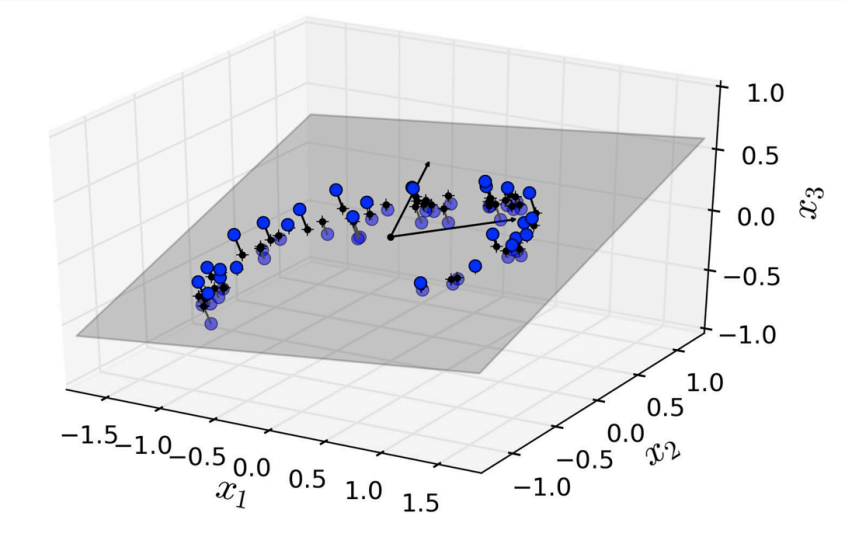
Fortunately, most real data lives in a lower-dimensional subspace, sometimes referred to as the *manifold assumption*

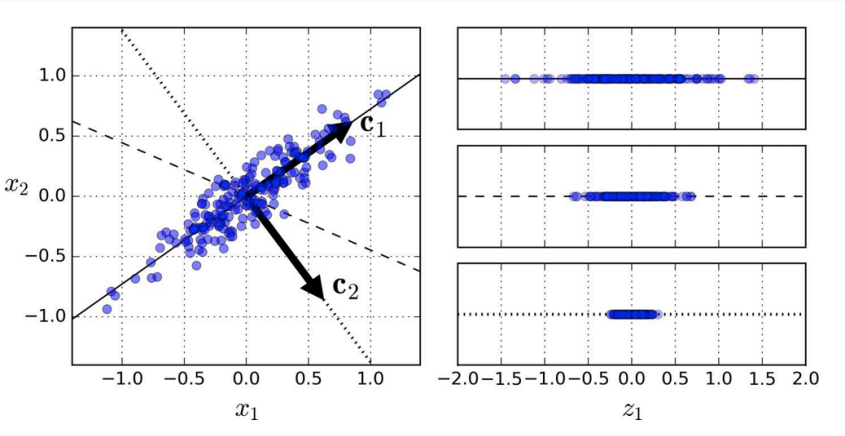
Principal Component Analysis

One of the most popular dimensionality reduction techniques

Projects data to the hyperplane closest to the data

Select the axis that preserves the maximum amount of variance





Singular Value Decomposition for PCA

PCA finds a series of axes on to which the data can be projected

Each axis is orthogonal to every other axis

Compute the Singular Value Decomposition

Σ is a diagonal matrix of singular values, **U** and **V** are orthogonal column matrices

$$\mathbf{X} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T \qquad (29)$$

$$\mathbf{V}^T = \begin{pmatrix} | & | & & | \\ \mathbf{c_1} & \mathbf{c_2} & \cdots & \mathbf{c_n} \\ | & | & & | \end{pmatrix}$$

V is the matrix of principal axes or directions and its columns are the principal axes (The image above needs correction)

Projecting using the Principal Components

Reduce dimensionality down to 'd' dimensions

Singular values are usually ordered from increasing to decreasing magnitude

Pick the 'd' dimensions corresponding to the 'd' largest singular values

$$\mathbf{X}_{dproj} = \mathbf{X} \cdot \mathbf{W}_d \tag{30}$$

Here $\mathbf{X}_{dproj}$ is the projected data (rows are coordinates of new data) and $\mathbf{W}_d$ is the matrix containing the first 'd' dimensions, i.e. use the first 'd' columns of $\mathbf{V}$

Some confusing terminology

Two sets of conventions exist regarding PCA terminology

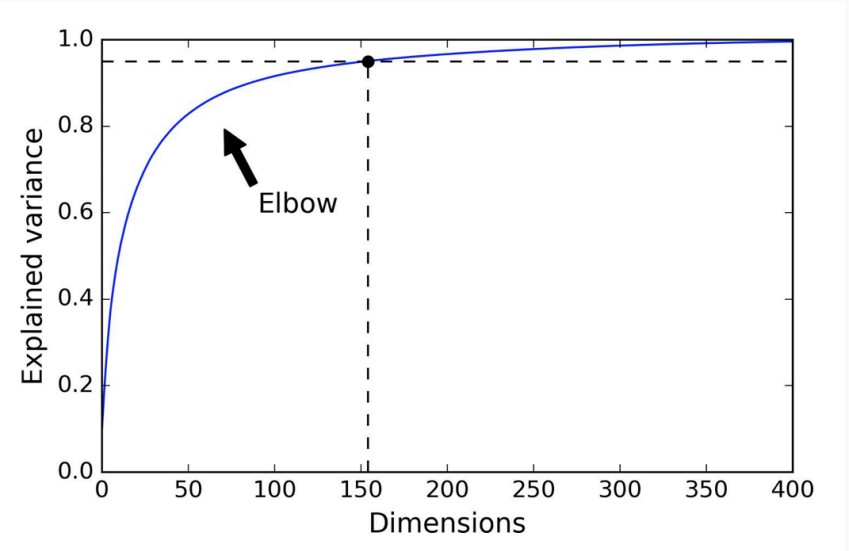
	Convention 1	Convention 2
V	Principal axis/direction	Principal component
XV	Principal component	Principal component score

Explained Variance Ratio

It indicates the proportion of the dataset's variance that lies along the axis of each principal component.

Choose the number of dimensions that add up to a percentage of the variance

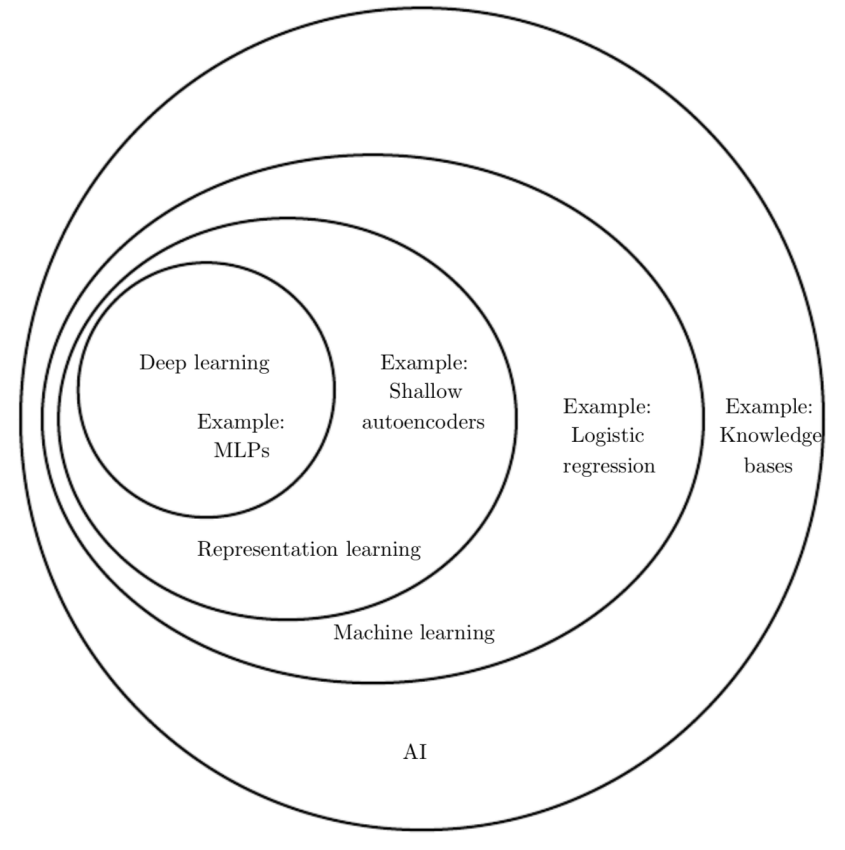
If you're using it for visualization two or three make sense



Deep Learning

Deep Learning or AI?

Deep Learning ⊂ Representation Learning ⊂ Machine Learning ⊂ AI



Deep learning

So far we have seen shallow neural networks

Deep neural networks usually have more than 10 layers

Vanishing/Exploding gradients problem, gradients get smaller and smaller or diverge as they propagate down to the lower layers

Suffer from severe overfitting

Deep learning

Deep Learning eliminates the need for feature engineering

No need to manually extract features

You learn all the features

Deep learning - Weight Initialization

Initializing weights by drawing from a normal distribution did not work

Xavier initialization for weights

The variance of inputs and outputs needed to be maintained

When using the logistic activation function, draw from normal distribution with mean 0 and standard deviation given by the following

$$\sigma = \sqrt{\frac{2}{n_{inputs} + n_{inputs}}} \tag{31}$$

Batch Normalization

The distribution of each layer's inputs changes during training, as the parameters of the previous layers change

Before the activation function of each layer, zero-center and normalize the inputs

Scale and shift the result using two parameters

This operation lets the model learn the optimal scale and mean of the inputs for each layer

Mean and standard deviation computed over the mini-batch

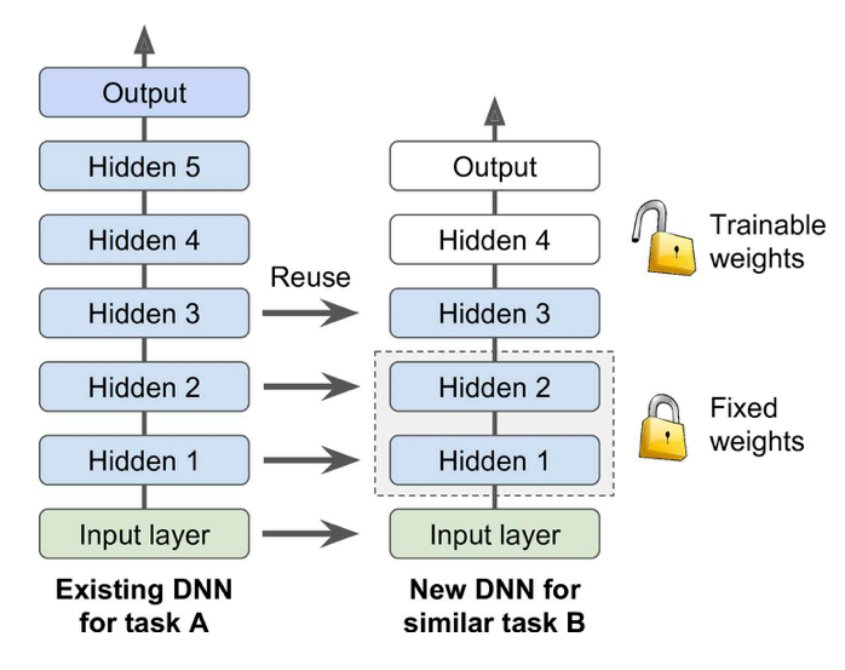
Batch Normalization can also act like a regularizer

Transfer Learning

Try not to train a large DNN from scratch

Find an existing DNN that does something similar to what you want to do and then retrain the higher layers

Requires much less training time and training data



Fast Optimizers

Momentum

Nesterov Accelerated Gradient

AdaGrad

RMSProp

Adam

Avoiding Overfitting

L1 and L2 Regularization

Early stopping

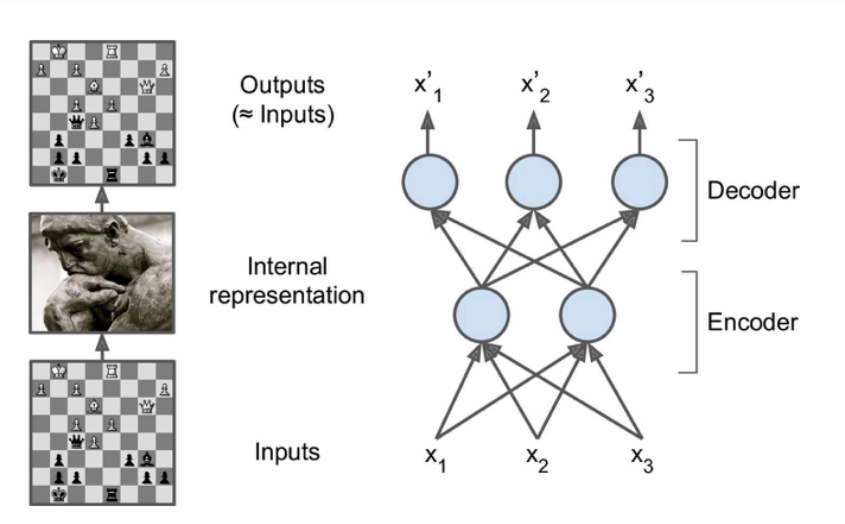
Dropout

Max-Norm Regularization

Data Augmentation

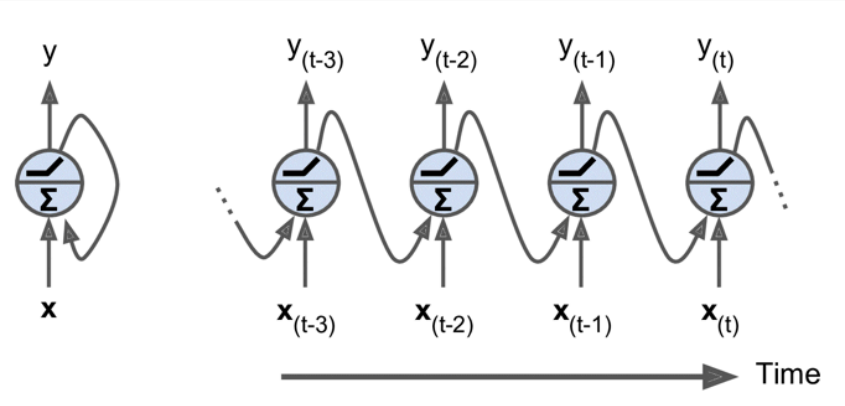
Autoencoder

- Composed of an encoder and a decoder
- Number of inputs be the same as the number of outputs
- Outputs are often called reconstructions since the autoencoder tries to reconstruct the inputs
- Penalizes output when different from inputs



Recurrent Neural Networks

- Unlike regular networks has connections pointing backwards
- Works with time series data and can do predictions
- Can work with arbitrary length sequences
- Useful in Natural Language Processing problems



Recurrent Neural Networks contd..

Output of a RNN for a single input

$$y_t = \sigma(\mathbf{x}_t^T \cdot \mathbf{w}_x + \mathbf{y}_{t-1}^T \cdot \mathbf{w}_y + b) \tag{32}$$

where σ is any of the activation functions we have seen. You can define a state for a RNN cell as $h_t = f(y_{t-1}, x_t)$. You can think of this as a type of memory.

Issues with Long Sequences in RNN

For long sequences, the RNN gets quite deep

Suffers from all the gradient issues we saw earlier as well as long training time

Truncated backpropagation through time but loses fine-grained data

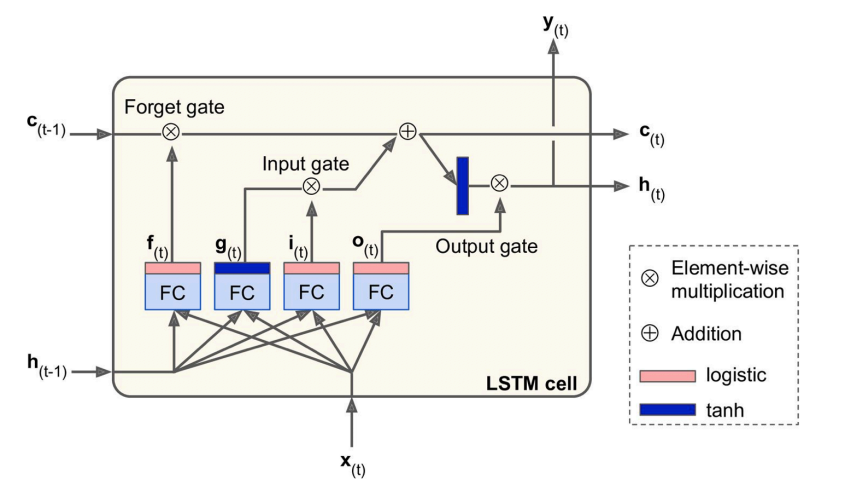
Memory of first inputs gradually fades away

Long Short-Term memory Cell

Has two state vectors, long and short-term state vectors

$\mathbf{h}_t$ is short term state and $\mathbf{c}_t$ is long term state

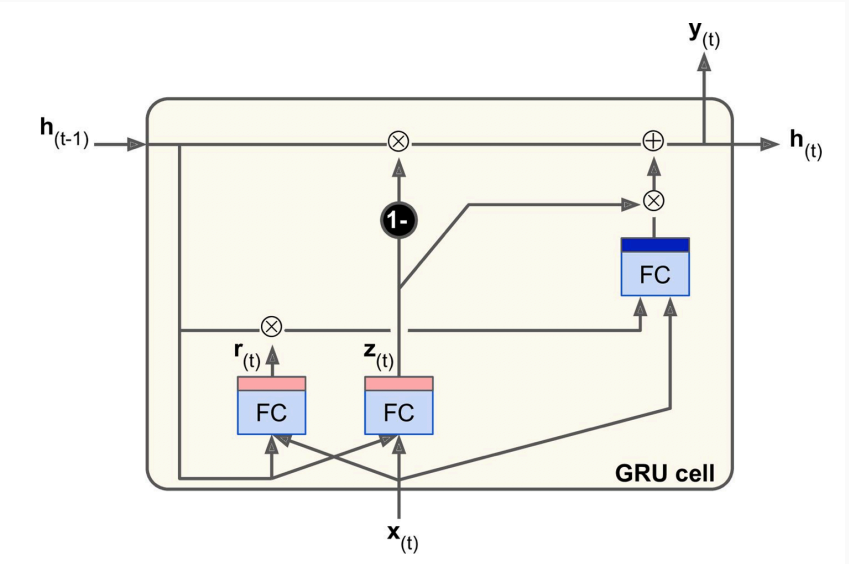
Detects long-term dependencies better



Gated Recurrent Unit

Simplified version of the LSTM, performs just as well in a lot of cases

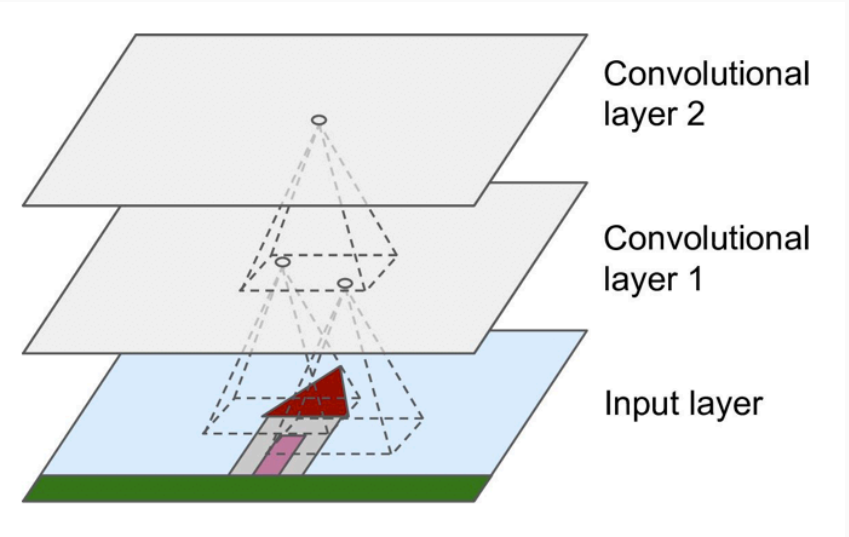
Only one state vector



Convolutional Neural Networks

Two new blocks: convolutional layers and pooling layers

CNNs preserve the spatial patterns in data, great for image classification



Convolutional Neural Networks - Pooling

- Convolutional layers apply a 2D convolution filter
- Pooling layers subsample the data
- A complete CNN architecture is shown here