Deep Learning

Contents

Deep Learning Tutorial

What is Machine Learning?

Artificial Intelligence (AI) systems learn by extracting patterns from input and output data. **Machine Learning (ML**) relies on learning patterns based on sample data. Programs learn from labeled data (supervised learning), unlabelled data (unsupervised learning), or a combination of both (semi-supervised learning).

Artificial Intelligence (AI) came around in the middle of 1900s when scientists tried to envision intelligent machines. Machine Learning evolved in the late 1900s. This allowed scientists to train machines for AI.

In the early 2000s, certain breakthroughs in multi-layered neural networks facilitated the advent of Deep Learning.

What is Deep Learning?

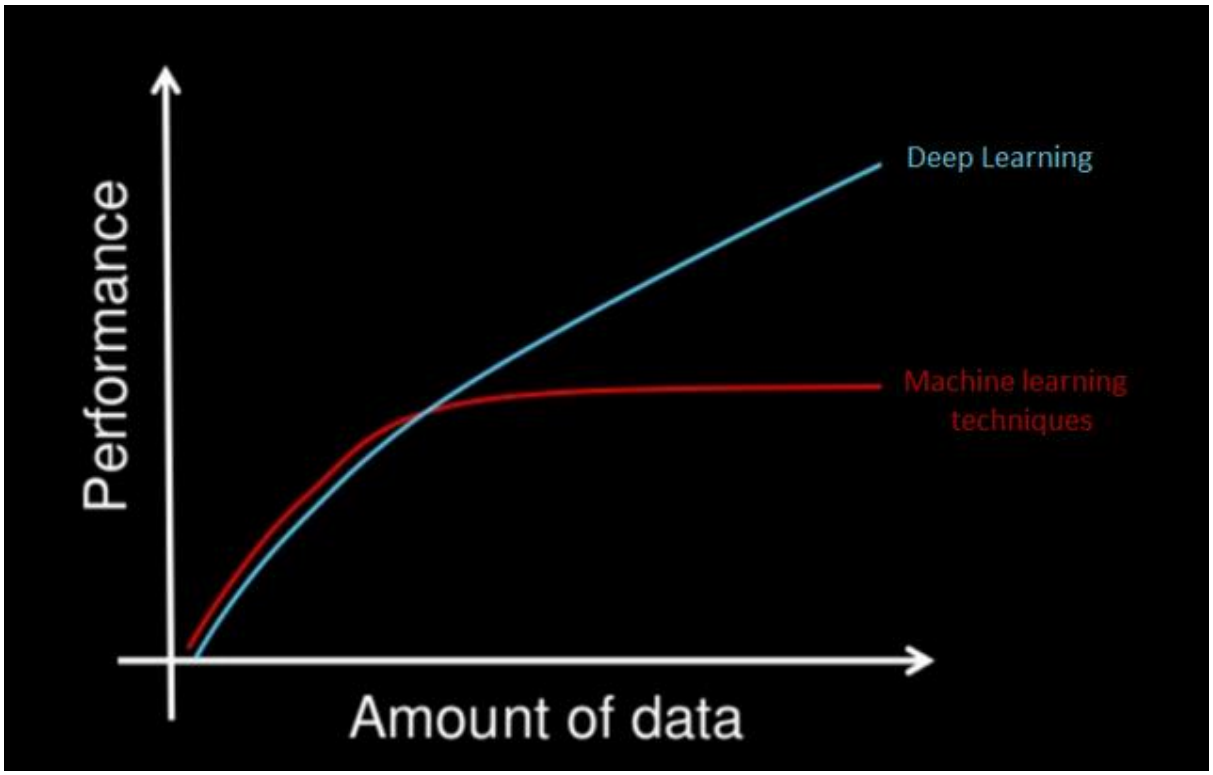
Deep Learning is a specialized form of Machine Learning that uses supervised, unsupervised, or semi-supervised learning to learn from data representations.

It is similar to the structure and function of the human nervous system, where a complex network of interconnected computation units work in a coordinated fashion to process complex information.

Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it. Deep Learning takes Machine Learning to the next level.

Why deep learning?

When the amounth of data is increased, machine learning techniques are insufficient in terms of performance and deep learning gives better performance like accuracy.



Usage fields of deep learning:

Speech recognition, image classification, natural language procession (nlp) or recommendation systems

Relationship Between Artificial Intelligence And Deep Learning

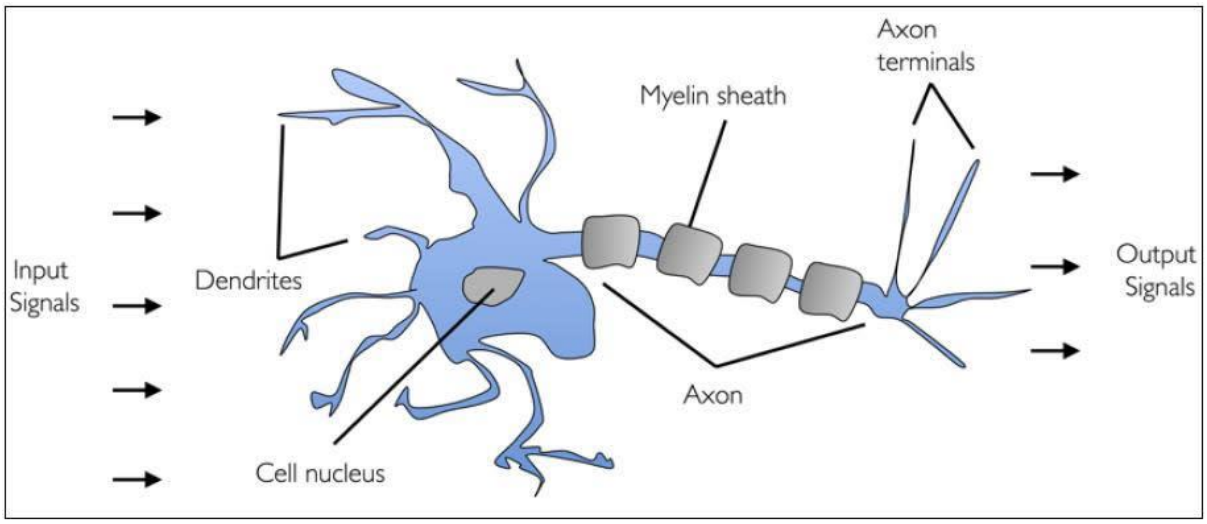
Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it. Deep Learning takes Machine Learning to the next level.

Artificial Neural Networks

In the coming sections, we would learn about Artificial Neural Networks starting with Biological Neuron.

Biological Neuron

A mammalian brain has billions of neurons. Neurons are interconnected nerve cells in the human brain that are involved in processing and transmitting chemical and electrical signals. They take input and pass along outputs.



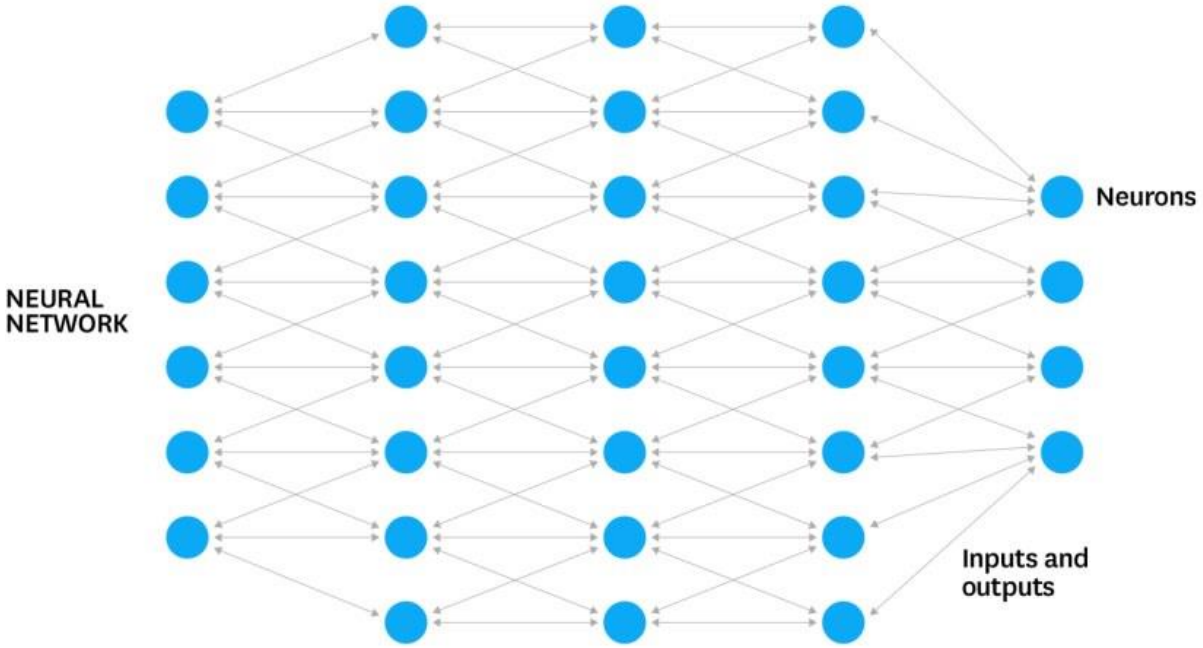
A human brain can learn how to identify objects from photos. For example, it can learn to identify the characteristics of chairs and thereby increase its probability of identifying them over time.

Following are the important parts of the biological neuron:

- **Dendrites** - branches that receive information from other neurons
- **Cell nucleus or Soma** - processes the information received from dendrites
- **Axon** - a cable that is used by neurons to send information
- **Synapse** - the connection between an axon and other neuron dendrites

Human Brain vs. Artificial Neural Networks

The computational models in Deep Learning are loosely inspired by the human brain. The multiple layers of training are called Artificial Neural Networks (ANN).



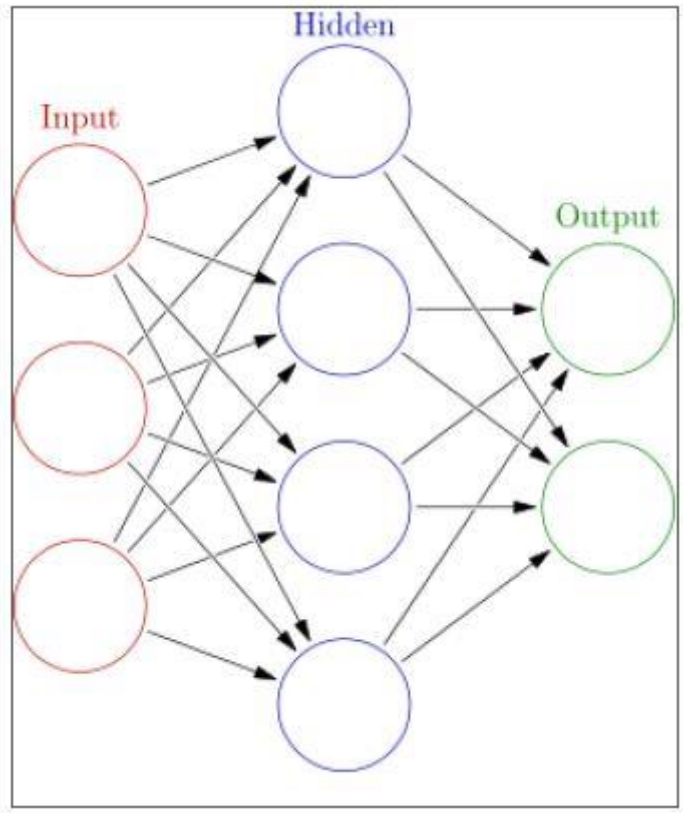
ANNs are processing devices (algorithms or actual hardware) that are modeled on the neuronal structure of the mammalian cerebral cortex but on a much smaller scale. It is a computing system made up of a number of simple, highly interconnected processing elements which process information through their dynamic state response to external inputs.

Artificial Neural Networks Process

Artificial Neural Networks consist of the following four main parts:

- Neurons
- Nodes
- Input
- Output

Let us discuss each of them in detail.



Neuron

Artificial Neural Networks contain layers of neurons. A neuron is a computational unit that calculates a piece of information based on weighted input parameters. Inputs accepted by the neuron are separately weighted.

Inputs are summed and passed through a non-linear function to produce output. Each layer of neurons detects some additional information, such as edges of things in a picture or tumors in a human body. Multiple layers of neurons can be used to detect additional information about input parameters.

Nodes

Artificial Neural Network is an interconnected group of nodes akin to the vast network of layers of neurons in a brain. Each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

Inputs

Inputs are passed into the first layer. Individual neurons receive the inputs, with each of them receiving a specific value. After this, an output is produced based on these values.

Outputs

The outputs from the first layer are then passed into the second layer to be processed. This continues until the final output is produced. The assumption is that the correct output is predefined.

Each time data is passed through the network, the end result is compared with the correct one, and tweaks are made to their values until the network creates the correct final output each time.

In the next section, let us study the types of Neural Networks.

Types of Neural Networks

Some of the commonly used neural networks are as follows:

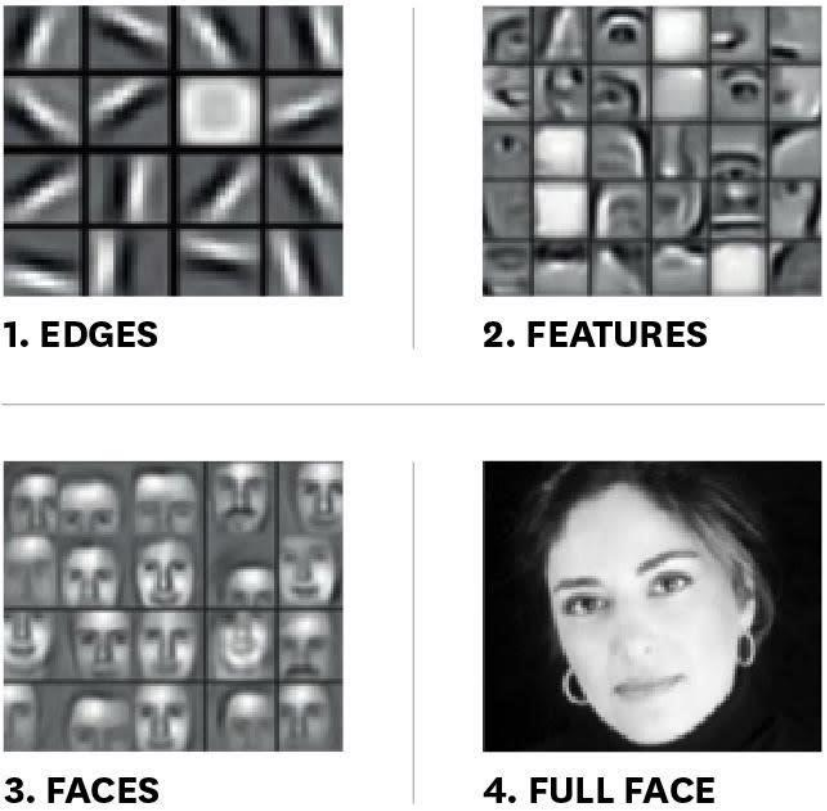
- Artificial Neural Network (ANN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Deep Neural Network (DNN)
- Deep Belief Network (DBN)

The use cases of different neural networks are listed below:

Neural Network	Use Case
ANN	Computational Neuroscience
CNN	Image Processing
RNN	Speech Recognition
DNN	Acoustic Modeling
DBN	Drug Discovery

Case Study: Deep Face

DeepFace is Facebook's facial recognition system created using Deep Learning. It uses a nine-layer neural network with more than 120 million connection weights. The researchers used four million images uploaded by Facebook users. Facebook has been using this technology since 2015.



Layers of virtual neurons are trained to identify edges, features, a face, and so on. In the example given below, the first layer recognizes edges.
The second layer recognizes facial features like a nose or an ear. The third layer recognizes faces. The full face is eventually recognized in the fourth layer.
The layers of a neural network work in coordination with each other to progressively add more insight as the data passes from input layers toward the output layers.

Deep Learning platforms include:

- Tensorflow(Python Based)
- Keras(Python)
- Torch(C/C++)
- Deeplearning4j(JAVA)

Train and Test Sets

Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally.Most classification data sets do not have exactly equal number of instances in each class, but a small difference often does not matter. You thus need to make sure that all two classes of wine are present in the training model. What's more, the

amount of instances of all two wine types needs to be more or less equal so that you do not favour one or the other class in your predictions.

In this case, there seems to be an imbalance, but you will go with this for the moment. Afterwards, you can evaluate the model and if it underperforms, you can resort to undersampling or oversampling to cover up the difference in observations.

The Problem of Model Generalization and Overfitting

The objective of a neural network is to have a final model that performs well both on the data that we used to train it (e.g. the training dataset) and the new data on which the model will be used to make predictions.

We require that the model learn from known examples and generalize from those known examples to new examples in the future. We use methods like a train/test split or k-fold cross-validation only to estimate the ability of the model to generalize to new data.

Learning and also generalizing to new cases is hard.

Too little learning and the model will perform poorly on the training dataset and on new data. The model will underfit the problem. Too much learning and the model will perform well on the training dataset and poorly on new data, the model will overfit the problem. In both cases, the model has not generalized.

- **Underfit Model**. A model that fails to sufficiently learn the problem and performs poorly on a training dataset and does not perform well on a holdout sample.
- **Overfit Model**. A model that learns the training dataset too well, performing well on the training dataset but does not perform well on a hold out sample.
- **Good Fit Model**. A model that suitably learns the training dataset and generalizes well to the old out dataset.

A model fit can be considered in the context of the bias-variance trade-off.

An underfit model has high bias and low variance. Regardless of the specific samples in the training data, it cannot learn the problem. An overfit model has low bias and high variance. The model learns the training data too well and performance varies widely with new unseen examples or even statistical noise added to examples in the training dataset.

We can address underfitting by increasing the capacity of the model. Capacity refers to the ability of a model to fit a variety of functions; more capacity, means that a model can fit more types of functions for mapping inputs to outputs. Increasing the capacity of a model is easily

achieved by changing the structure of the model, such as adding more layers and/or more nodes to layers.

Because an underfit model is so easily addressed, it is more common to have an overfit model.

An overfit model is easily diagnosed by monitoring the performance of the model during training by evaluating it on both a training dataset and on a holdout validation dataset. Graphing line plots of the performance of the model during training, called learning curves, will show a familiar pattern.

For example, line plots of the loss (that we seek to minimize) of the model on train and validation datasets will show a line for the training dataset that drops and may plateau and a line for the validation dataset that drops at first, then at some point begins to rise again.

A learning curve plot tells the story of the model learning the problem until a point at which it begins overfitting and its ability to generalize to the unseen validation dataset begins to get worse.

Reduce Overfitting by Constraining Model Complexity

There are two ways to approach an overfit model:

1. Reduce overfitting by training the network on more examples.
2. Reduce overfitting by changing the complexity of the network.

A benefit of very deep neural networks is that their performance continues to improve as they are fed larger and larger datasets. A model with a near-infinite number of examples will eventually plateau in terms of what the capacity of the network is capable of learning.

A model can overfit a training dataset because it has sufficient capacity to do so. Reducing the capacity of the model reduces the likelihood of the model overfitting the training dataset, to a point where it no longer overfits.

The capacity of a neural network model, it's complexity, is defined by both it's structure in terms of nodes and layers and the parameters in terms of its weights. Therefore, we can reduce the complexity of a neural network to reduce overfitting in one of two ways:

1. Change network complexity by changing the network structure (number of weights).
2. Change network complexity by changing the network parameters (values of weights).

For example, the structure could be tuned such as via grid search until a suitable number of nodes and/or layers is found to reduce or remove overfitting for the problem. Alternately, the model could be overfit and pruned by removing nodes until it achieves suitable performance on a validation dataset.

It is more common to instead constrain the complexity of the model by ensuring the parameters (weights) of the model remain small. Small parameters suggest a less complex and, in turn, more stable model that is less sensitive to statistical fluctuations in the input data.

Regularization Methods for Neural Networks

The simplest and perhaps most common regularization method is to add a penalty to the loss function in proportion to the size of the weights in the model.

1. **Weight Regularization (weight decay)**: Penalize the model during training based on the magnitude of the weights.
   This will encourage the model to map the inputs to the outputs of the training dataset in such a way that the weights of the model are kept small. This approach is called weight regularization or weight decay and has proven very effective for decades for both simpler linear models and neural networks.

Below is a list of five of the most common additional regularization methods.

1. **Activity Regularization**: Penalize the model during training base on the magnitude of the activations.
2. **Weight Constraint**: Constrain the magnitude of weights to be within a range or below a limit.
3. **Dropout**: Probabilistically remove inputs during training.
4. **Noise**: Add statistical noise to inputs during training.
5. **Early Stopping**: Monitor model performance on a validation set and stop training when performance degrades.
   Most of these methods have been demonstrated (or proven) to approximate the effect of adding a penalty to the loss function.

Each method approaches the problem differently, offering benefits in terms of a mixture of generalization performance, configurability, and/or computational complexity.

tutorial video

- Machine Learning intro
- Deep Learning
- Artificial Neural Networks
- Layers in a Neural Network
- Activation Functions in a Neural Network
- Training a Neural Network
- How a Neural Network Learns
- Loss in a Neural Network
- Learning Rate in a Neural Network
- Train, Test, & Validation Sets
- Predicting with a Neural Network
- Overfitting in a Neural Network
- Underfitting in a Neural Network
- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Data Augmentation
- Convolutional Neural Networks (CNNs)
- Max Pooling in Convolutional Neural Networks
- Backpropagation | Part 1 - The intuition
- Backpropagation | Part 2 - The mathematical notation
- Backpropagation | Part 3 - Mathematical observations
- Backpropagation | Part 4 - Calculating the gradient
- Backpropagation | Part 5 - What puts the "back" in backprop?
- Regularization in a Neural Network
- Batch Size in a Neural Network
- How Gradient Descent Works. Simple Explanation

Terms

- accuracy
- activation function
- active learning
- augmented reality
- backpropagation
- baseline
- batch
- batch normalization
- batch size
- Bayesian neural network
- bias (ethics/fairness)
- bias (math)
- binary classification
- Boosting
- bounding box
- broadcasting
- bucketing
- centroid
- centroid-based clustering
- checkpoint
- class
- classification model
- classification threshold
- class-imbalanced dataset
- Clustering
- co-adaptation
- convergence
- convex function
- convex optimization
- convolution
- convolutional filter
- convolutional layer
- convolutional neural network
- convolutional operation
- cost
- cross-validation
- data analysis
- data augmentation
- decision boundary
- dense feature
- dense layer
- depth
- Downsampling
- dropout regularization
- early stopping

- Epoch
- false negative (FN)
- false positive (FP)
- false positive rate (FPR)
- feature
- fine tuning
- fully connected layer
- generalization
- generalization curve
- generator
- gradient
- gradient descent
- ground truth
- Hidden layer
- image recognition
- input layer
- iteration
- Keras
- k-means
- label
- layer
- Layers API (tf.layers)
- learning rate
- Logistic regression
- loss
- loss curve
- machine learning
- Mean Absolute Error (MAE)
- Mean Squared Error(MSE)
- mini-batch stochastic gradient descent (SGD)
- model
- Multi-class classification
- neural network
- neuron
- node (neural network)
- normalization
- optimizer
- output layer
- overfitting
- perceptron
- performance
- pooling
- positive class
- post-processing
- precision
- preprocessing

- recurrent neural network(RNN)
- regularization
- semi-supervised learning
- sigmoid function
- softmax
- supervised machine learning
- Tensor
- TensorFlow
- test set
- tf.keras
- training
- training set
- transfer learning
- true negative (TN)
- true positive (TP)
- true positive rate (TPR)
- underfitting
- Unsupervised machine learning
- validation
- validation set
- weight
- width

accuracy

The fraction of predictions that a classification model got right. In multi-class classification, accuracy is defined as follows:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Number Of Examples}}$$

In binary classification, accuracy has the following definition:

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Number Of Examples}}$$

Accuracy=True Positives+True NegativesTotal Number Of Examples

activation function

A function (for example, **ReLU** or **sigmoid**) that takes in the weighted sum of all of the inputs from the previous layer and then generates and passes an output value (typically nonlinear) to the next laye

active learning

A **training** approach in which the algorithm *chooses* some of the data it learns from. Active learning is particularly valuable when **labeled examples** are scarce or expensive to obtain. Instead of blindly seeking a diverse range of labeled examples, an active learning algorithm selectively seeks the particular range of examples it needs for learning.

augmented reality

A technology that superimposes a computer-generated image on a user's view of the real world, thus providing a composite view.

backpropagation

The primary algorithm for performing **gradient descent** on **neural networks**. First, the output values of each node are calculated (and cached) in a forward pass. Then, the **partial derivative** of the error with respect to each parameter is calculated in a backward pass through the graph.

baseline

A **model** used as a reference point for comparing how well another model (typically, a more complex one) is performing. For example, a **logistic regression model** might serve as a good baseline for a **deep model**.

For a particular problem, the baseline helps model developers quantify the minimal expected performance that a new model must achieve for the new model to be useful.

batch

The set of examples used in one **iteration** (that is, one **gradient** update) of **model training**.

batch normalization

Normalizing the input or output of the **activation functions** in a **hidden layer**. Batch normalization can provide the following benefits:

- Make **neural networks** more stable by protecting against **outlier** weights.
- Enable higher **learning rates**.
- Reduce **overfitting**.

batch size

The number of examples in a **batch**. For example, the batch size of **SGD** is 1, while the batch size of a **mini-batch** is usually between 10 and 1000. Batch size is usually

fixed during **training** and **inference**; however, **TensorFlow** does permit dynamic batch sizes.

Bayesian neural network

A probabilistic **neural network** that accounts for uncertainty in **weights** and outputs. A standard neural network regression model typically **predicts** a scalar value; for example, a model predicts a house price of 853,000. By contrast, a Bayesian neural network predicts a distribution of values; for example, a model predicts a house price of 853,000 with a standard deviation of 67,200. A Bayesian neural network relies on Bayes' Theorem to calculate uncertainties in weights and predictions. A Bayesian neural network can be useful when it is important to quantify uncertainty, such as in models related to pharmaceuticals. Bayesian neural networks can also help prevent **overfitting**.

bias (ethics/fairness)

1. Stereotyping, prejudice or favoritism towards some things, people, or groups over others. These biases can affect collection and interpretation of data, the design of a system, and how users interact with a system. Forms of this type of bias include:

- **automation bias**
- **confirmation bias**
- **experimenter's bias**
- **group attribution bias**
- **implicit bias**
- **in-group bias**
- **out-group homogeneity bias**

2. Systematic error introduced by a sampling or reporting procedure. Forms of this type of bias include:

- coverage bias
- non-response bias
- participation bias
- reporting bias
- **sampling bias**

- **selection bias**

  Not to be confused with the bias term in machine learning models or **prediction bias**.

bias (math)

An intercept or offset from an origin. Bias (also known as the **bias term**) is referred to as *b* or w_0 in machine learning models. For example, bias is the *b* in the following formula:

$$y' = b + w_1 x_1 + w_2 x_2 + \ldots w_n x_n$$

Not to be confused with **bias in ethics and fairness** or **prediction bias**.

binary classification

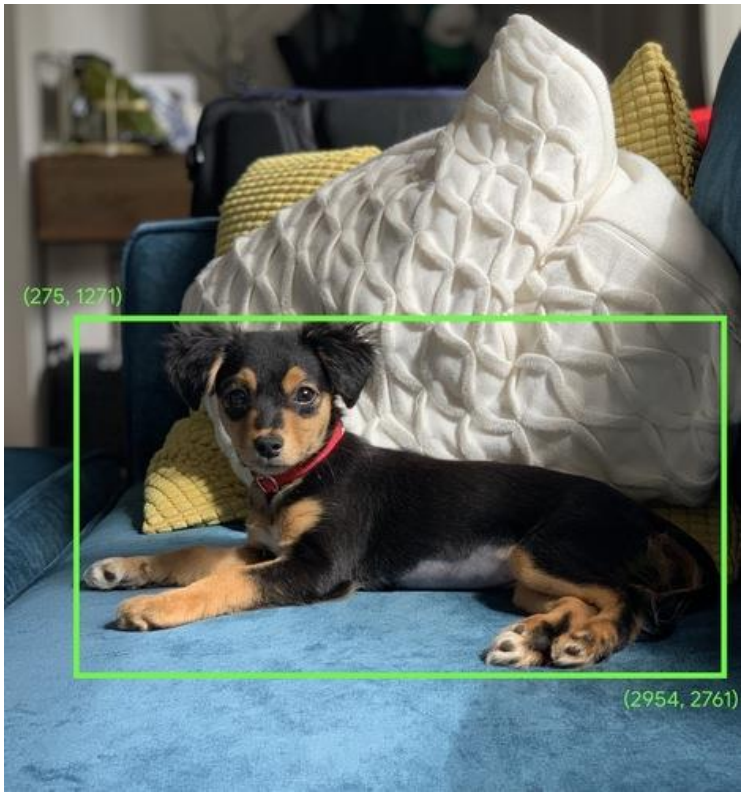
A type of **classification** task that outputs one of two mutually exclusive **classes**. For example, a machine learning model that evaluates email messages and outputs either "spam" or "not spam" is a **binary classifier**.

boosting

A machine learning technique that iteratively combines a set of simple and not very accurate classifiers (referred to as "weak" classifiers) into a classifier with high accuracy (a "strong" classifier) by **upweighting** the examples that the model is currently misclassfying.

bounding box

In an image, the (*x*, *y*) coordinates of a rectangle around an area of interest, such as the dog in the image below.



broadcasting

Expanding the shape of an operand in a matrix math operation to **dimensions** compatible for that operation. For instance, linear algebra requires that the two operands in a matrix addition operation must have the same dimensions. Consequently, you can't add a matrix of shape (m, n) to a vector of length n. Broadcasting enables this operation by virtually expanding the vector of length n to a matrix of shape (m,n) by replicating the same values down each column.

For example, given the following definitions, linear algebra prohibits A+B because A and B have different dimensions:

```
A = [[7, 10, 4],
     [13, 5, 9]]
B = [2]
```

However, broadcasting enables the operation A+B by virtually expanding B to:

```
 [[2, 2, 2],
  [2, 2, 2]]
```

Thus, A+B is now a valid operation:

```
[[7, 10, 4],  +  [[2, 2, 2],  =  [[ 9, 12, 6],
 [13, 5, 9]]      [2, 2, 2]]      [15, 7, 11]]
```

bucketing

Converting a (usually **continuous**) feature into multiple binary features called buckets or bins, typically based on value range. For example, instead of representing temperature as a single continuous floating-point feature, you could chop ranges of temperatures into discrete bins. Given temperature data sensitive to a tenth of a degree, all temperatures between 0.0 and 15.0 degrees could be put into one bin, 15.1 to 30.0 degrees could be a second bin, and 30.1 to 50.0 degrees could be a third bin.

centroid

The center of a cluster as determined by a **k-means** or **k-median** algorithm. For instance, if k is 3, then the k-means or k-median algorithm finds 3 centroids.

centroid-based clustering

A category of **clustering** algorithms that organizes data into nonhierarchical clusters. **k-means** is the most widely used centroid-based clustering algorithm.

Contrast with **hierarchical clustering** algorithms.

checkpoint

Data that captures the state of the variables of a model at a particular time. Checkpoints enable exporting model **weights**, as well as performing training across multiple sessions. Checkpoints also enable training to continue past errors (for example, job preemption). Note that the **graph** itself is not included in a checkpoint.

class

One of a set of enumerated target values for a label. For example, in a **binary classification** model that detects spam, the two classes are *spam* and *not spam*. In

a **multi-class classification** model that identifies dog breeds, the classes would be *poodle*, *beagle*, *pug*, and so on.

classification model

A type of machine learning model for distinguishing among two or more discrete classes. For example, a natural language processing classification model could determine whether an input sentence was in French, Spanish, or Italian. Compare with **regression model**.

classification threshold

A scalar-value criterion that is applied to a model's predicted score in order to separate the **positive class** from the **negative class**. Used when mapping **logistic regression** results to **binary classification**. For example, consider a logistic regression model that determines the probability of a given email message being spam. If the classification threshold is 0.9, then logistic regression values above 0.9 are classified as *spam* and those below 0.9 are classified as *not spam*.

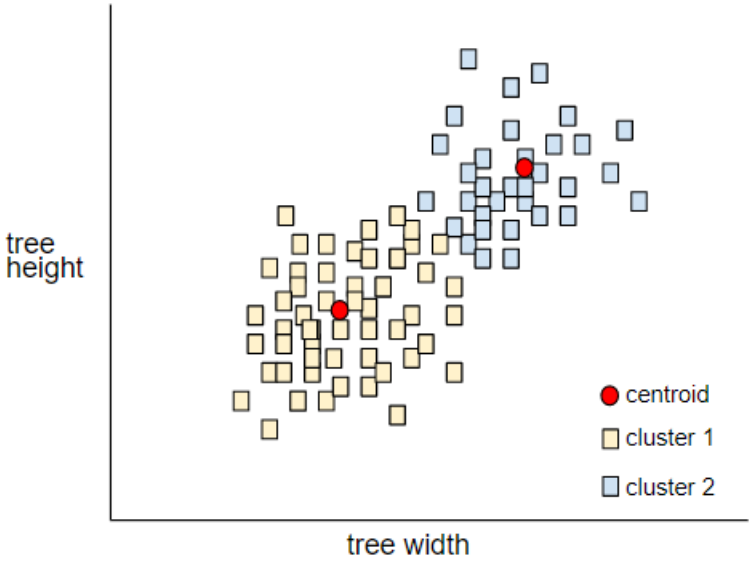
class-imbalanced dataset

A **binary classification** problem in which the **labels** for the two classes have significantly different frequencies. For example, a disease dataset in which 0.0001 of examples have positive labels and 0.9999 have negative labels is a class-imbalanced problem, but a football game predictor in which 0.51 of examples label one team winning and 0.49 label the other team winning is *not* a class-imbalanced problem.

clustering

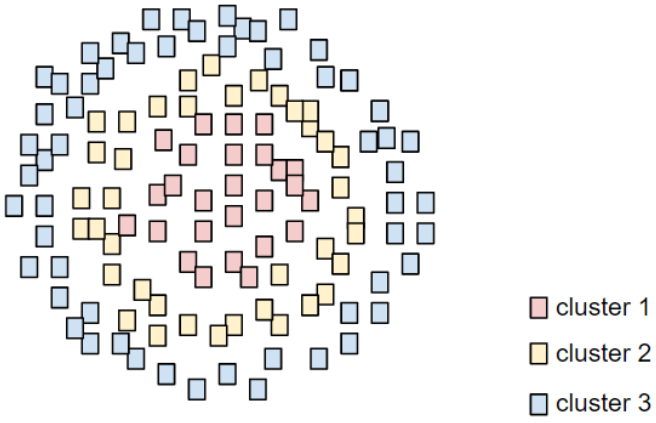
Grouping related **examples**, particularly during **unsupervised learning**. Once all the examples are grouped, a human can optionally supply meaning to each cluster.

Many clustering algorithms exist. For example, the **k-means** algorithm clusters examples based on their proximity to a **centroid**, as in the following diagram:



A human researcher could then review the clusters and, for example, label cluster 1 as "dwarf trees" and cluster 2 as "full-size trees."

As another example, consider a clustering algorithm based on an example's distance from a center point, illustrated as follows:



co-adaptation

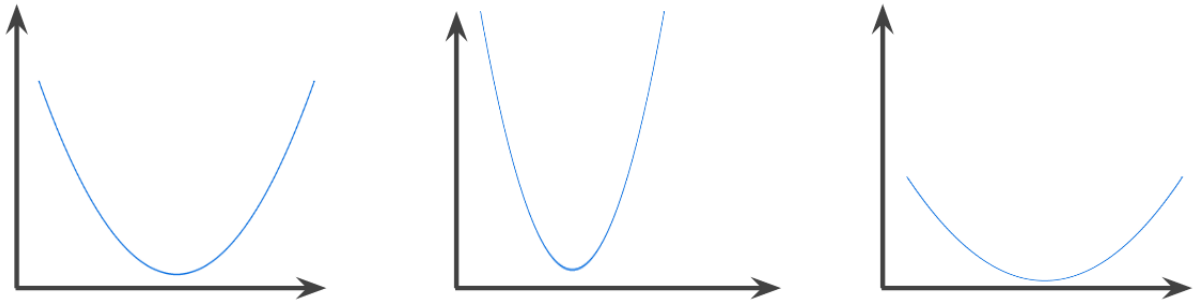
When **neurons** predict patterns in training data by relying almost exclusively on outputs of specific other neurons instead of relying on the network's behavior as a whole. When the patterns that cause co-adaption are not present in validation data, then co-adaptation causes overfitting. **Dropout regularization** reduces co-adaptation because dropout ensures neurons cannot rely solely on specific other neurons.

convergence

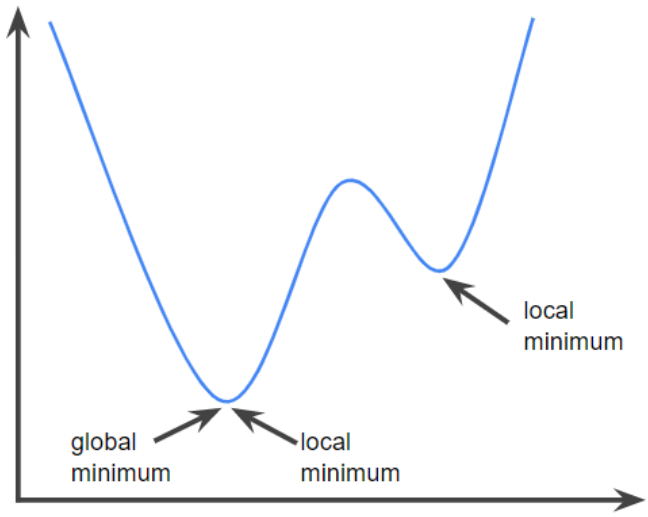
Informally, often refers to a state reached during **training** in which training **loss** and **validation** loss change very little or not at all with each iteration after a certain number of iterations. In other words, a model reaches convergence when additional training on the current data will not improve the model. In **deep learning**, loss values sometimes stay constant or nearly so for many iterations before finally descending, temporarily producing a false sense of convergence.

convex function

A function in which the region above the graph of the function is a convex set. The prototypical convex function is shaped something like the letter U. For example, the following are all convex functions:



By contrast, the following function is not convex. Notice how the region above the graph is not a convex set:



A **strictly convex function** has exactly one local minimum point, which is also the global minimum point. The classic U-shaped functions are strictly convex functions. However, some convex functions (for example, straight lines) are not U-shaped.

A lot of the common loss functions, including the following, are convex functions:

1. L2 loss
2. Log Loss
3. L1 regularization
4. L2 regularization

Many variations of gradient descent are guaranteed to find a point close to the minimum of a strictly convex function. Similarly, many variations of stochastic gradient descent have a high probability (though, not a guarantee) of finding a point close to the minimum of a strictly convex function.

The sum of two convex functions (for example, L2 loss + L1 regularization) is a convex function.

Deep models are never convex functions. Remarkably, algorithms designed for convex optimization tend to find reasonably good solutions on deep networks anyway, even though those solutions are not guaranteed to be a global minimum.

convex optimization

The process of using mathematical techniques such as **gradient descent** to find the minimum of a **convex function**. A great deal of research in machine learning has focused on formulating various problems as convex optimization problems and in solving those problems more efficiently.

For complete details, see Boyd and Vandenberghe, Convex Optimization.

convolution

In mathematics, casually speaking, a mixture of two functions. In machine learning, a convolution mixes the convolutional filter and the input matrix in order to train **weights**.

The term "convolution" in machine learning is often a shorthand way of referring to either **convolutional operation** or **convolutional layer**.

Without convolutions, a machine learning algorithm would have to learn a separate weight for every cell in a large **tensor**. For example, a machine learning algorithm training on 2K x 2K images would be forced to find 4M separate weights. Thanks to convolutions, a machine learning algorithm only has to find weights for every cell in the **convolutional filter**, dramatically reducing the memory needed to train the model.

When the convolutional filter is applied, it is simply replicated across cells such that each is multiplied by the filter.

convolutional filter

One of the two actors in a **convolutional operation**. (The other actor is a slice of an input matrix.) A convolutional filter is a matrix having the same **rank** as the input matrix, but a smaller shape. For example, given a 28x28 input matrix, the filter could be any 2D matrix smaller than 28x28.

In photographic manipulation, all the cells in a convolutional filter are typically set to a constant pattern of ones and zeroes. In machine learning, convolutional filters are typically seeded with random numbers and then the network **trains** the ideal values.

convolutional layer

A layer of a deep neural network in which a convolutional filter passes along an input matrix. For example, consider the following 3x3 convolutional filter:

0	1	0
1	0	1
0	1	0

The following animation shows a convolutional layer consisting of 9 convolutional operations involving the 5x5 input matrix. Notice that each convolutional operation works on a different 3x3 slice of the input matrix. The resulting 3x3 matrix (on the right) consists of the results of the 9 convolutional operations:

128	97	53	201	198
35	22	25	200	195
37	24	28	197	182
33	28	92	195	179
31	40	100	192	177

181		

convolutional neural network

A **neural network** in which at least one layer is a **convolutional layer**. A typical convolutional neural network consists of some combination of the following layers:

- **convolutional layers**
- **pooling layers**
- **dense layers**

Convolutional neural networks have had great success in certain kinds of problems, such as image recognition.

convolutional operation

The following two-step mathematical operation:

1. Element-wise multiplication of the **convolutional filter** and a slice of an input matrix. (The slice of the input matrix has the same rank and size as the convolutional filter.)
2. Summation of all the values in the resulting product matrix.

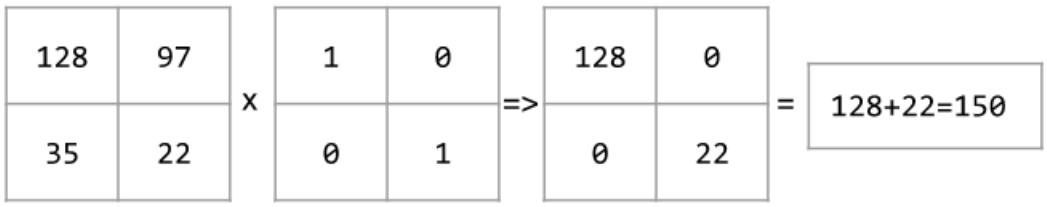
For example, consider the following 5x5 input matrix:

128	97	53	201	198
35	22	25	200	195
37	24	28	197	182
33	28	92	195	179
31	40	100	192	177

Now imagine the following 2x2 convolutional filter:

1	0
0	1

Each convolutional operation involves a single 2x2 slice of the input matrix. For instance, suppose we use the 2x2 slice at the top-left of the input matrix. So, the convolution operation on this slice looks as follows:



A **convolutional layer** consists of a series of convolutional operations, each acting on a different slice of the input matrix.

cost

Synonym for **loss**.

cross-validation

A mechanism for estimating how well a model will generalize to new data by testing the model against one or more non-overlapping data subsets withheld from the **training set**.

data analysis

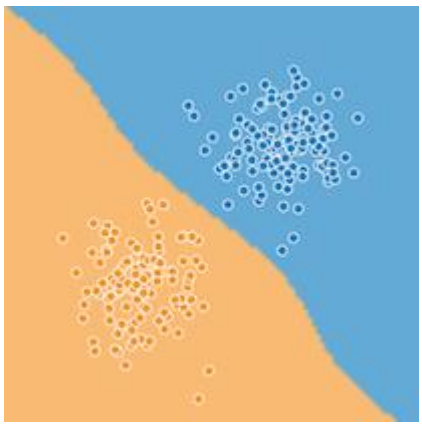
Obtaining an understanding of data by considering samples, measurement, and visualization. Data analysis can be particularly useful when a dataset is first received, before one builds the first model. It is also crucial in understanding experiments and debugging problems with the system.

data augmentation

Artificially boosting the range and number of training examples by transforming existing examples to create additional examples. For example, suppose images are one of your features, but your dataset doesn't contain enough image examples for the model to learn useful associations. Ideally, you'd add enough labeled images to your dataset to enable your model to train properly. If that's not possible, data augmentation can rotate, stretch, and reflect each image to produce many variants of the original picture, possibly yielding enough labeled data to enable excellent training.

decision boundary

The separator between classes learned by a model in a **binary class** or **multi-class classification problems**. For example, in the following image representing a binary classification problem, the decision boundary is the frontier between the orange class and the blue class:



dense feature

A **feature** in which most values are non-zero, typically a **Tensor** of floating-point values. Contrast with **sparse feature**.

dense layer

Synonym for **fully connected layer**.

depth

The number of **layers** (including any **embedding** layers) in a **neural network** that learn weights. For example, a neural network with 5 **hidden layers** and 1 output layer has a depth of 6.

downsampling

Overloaded term that can mean either of the following:

- Reducing the amount of information in a feature in order to train a model more efficiently. For example, before training an image recognition model, downsampling high-resolution images to a lower-resolution format.
- Training on a disproportionately low percentage of over-represented class examples in order to improve model training on under-represented classes. For example, in a **class-imbalanced dataset**, models tend to learn a lot about the **majority class** and not enough about the **minority class**. Downsampling helps balance the amount of training on the majority and minority classes.

dropout regularization

A form of regularization **useful in training** neural networks**. Dropout regularization works by removing a random selection of a fixed number of the units in a network layer for a single gradient step. The more units dropped out, the stronger the regularization. This is analogous to training the network to emulate an exponentially** large ensemble of smaller networks. For full details

early stopping

A method for **regularization** that involves ending model training *before* training loss finishes decreasing. In early stopping, you end model training when the loss on a **validation dataset** starts to increase, that is, when **generalization** performance worsens.

epoch

A full training pass over the entire dataset such that each example has been seen once. Thus, an epoch represents N/**batch size** training **iterations**, where N is the total number of examples.

false negative (FN)

An example in which the model mistakenly predicted the **negative class**. For example, the model inferred that a particular email message was not spam (the negative class), but that email message actually was spam.

false positive (FP)

An example in which the model mistakenly predicted the **positive class**. For example, the model inferred that a particular email message was spam (the positive class), but that email message was actually not spam.

false positive rate (FPR)

The x-axis in an ROC curve. The false positive rate is defined as follows:

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

feature

An input variable used in making predictions.

fine tuning

Perform a secondary optimization to adjust the parameters of an already trained **model** to fit a new problem. Fine tuning often refers to refitting the weights of a trained **unsupervised** model to a **supervised** model.

fully connected layer

A **hidden layer** in which each **node** is connected to *every* node in the subsequent hidden layer.

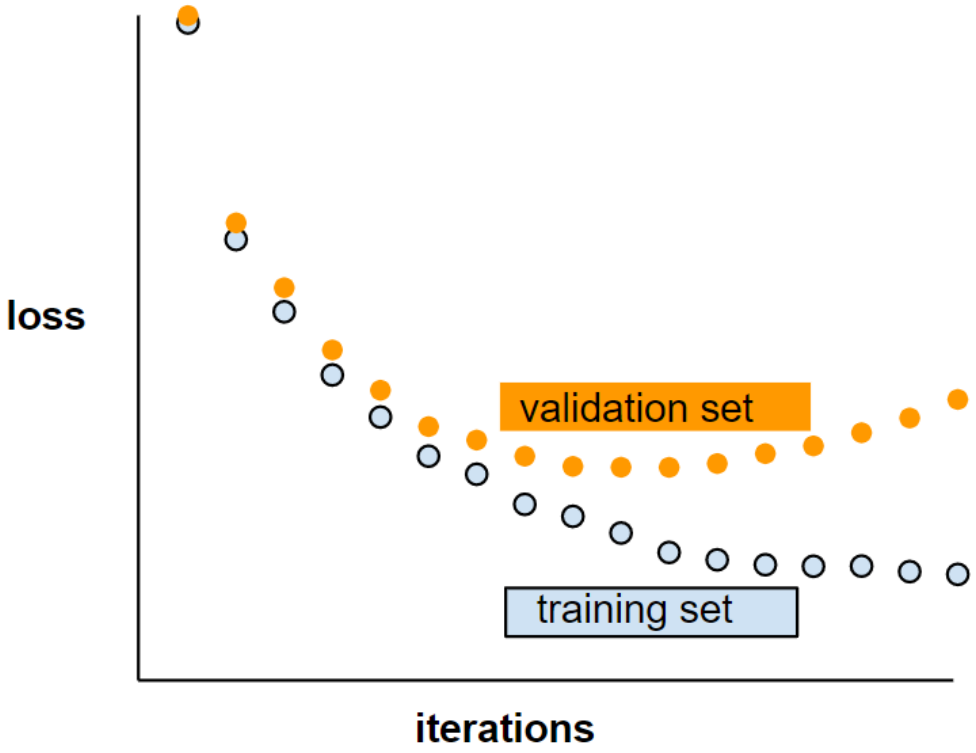
A fully connected layer is also known as a **dense layer**.

generalization

Refers to your model's ability to make correct predictions on new, previously unseen data as opposed to the data used to train the model.

generalization curve

A **loss curve** showing both the **training set** and the **validation set**. A generalization curve can help you detect possible **overfitting**. For example, the following generalization curve suggests overfitting because loss for the validation set ultimately becomes significantly higher than for the training set.



generator

The subsystem within a **generative adversarial network** that creates new **examples**.

Contrast with **discriminative model**.

gradient

The vector of **partial derivatives** with respect to all of the independent variables. In machine learning, the gradient is the vector of partial derivatives of the model function. The gradient points in the direction of steepest ascent.

gradient descent

A technique to minimize **loss** by computing the gradients of loss with respect to the model's parameters, conditioned on training data. Informally, gradient descent iteratively adjusts parameters, gradually finding the best combination of **weights** and bias to minimize loss.

ground truth

The correct answer. Reality. Since reality is often subjective, expert **raters** typically are the proxy for ground truth.

hidden layer

A synthetic layer in a **neural network** between the **input layer** (that is, the features) and the **output layer** (the prediction). Hidden layers typically contain an **activation function** (such as **ReLU**) for training. A **deep neural network** contains more than one hidden layer.

image recognition

#

A process that classifies object(s), pattern(s), or concept(s) in an image. Image recognition is also known as **image classification**.

input layer

The first layer (the one that receives the input data) in a **neural network**.

iteration

A single update of a model's weights during training. An iteration consists of computing the gradients of the parameters with respect to the loss on a single **batch** of data.

Keras

A popular Python machine learning API. Keras runs on several deep learning frameworks, including TensorFlow, where it is made available as tf.keras.

k-means

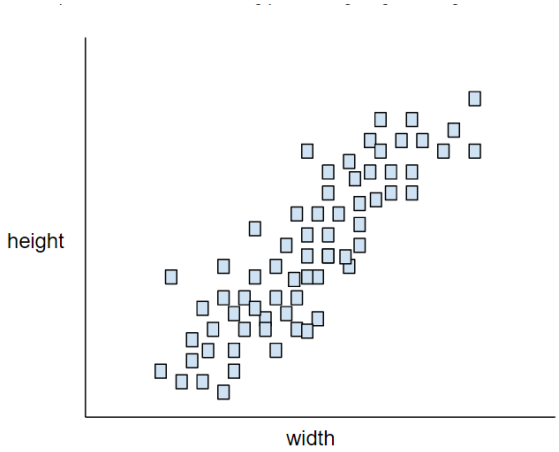
#clustering

A popular **clustering** algorithm that groups examples in unsupervised learning. The k-means algorithm basically does the following:

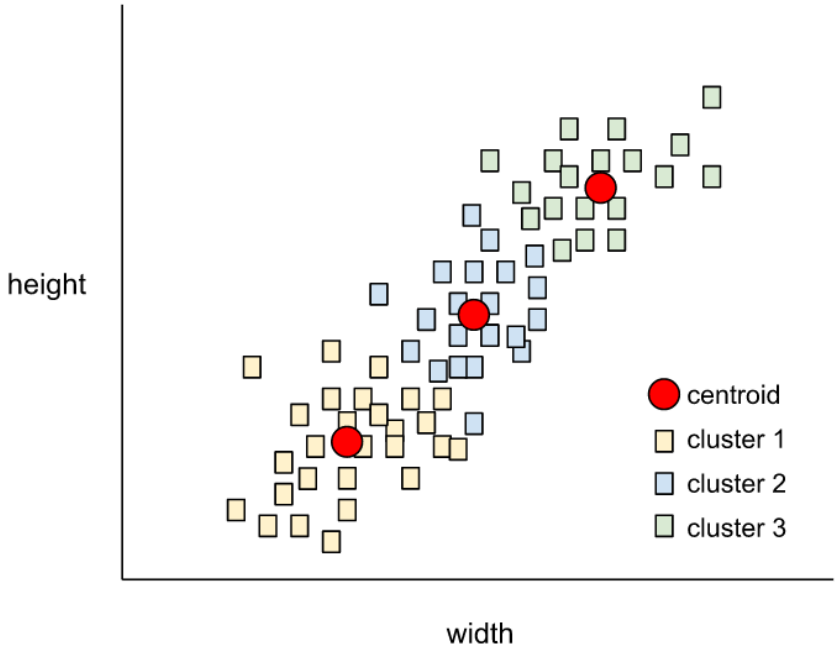
- Iteratively determines the best k center points (known as **centroids**).
- Assigns each example to the closest centroid. Those examples nearest the same centroid belong to the same group.

The k-means algorithm picks centroid locations to minimize the cumulative *square* of the distances from each example to its closest centroid.

For example, consider the following plot of dog height to dog width:



If k=3, the k-means algorithm will determine three centroids. Each example is assigned to its closest centroid, yielding three groups:



Imagine that a manufacturer wants to determine the ideal sizes for small, medium, and large sweaters for dogs. The three centroids identify the mean height and mean width of each dog in that cluster. So, the manufacturer should probably base sweater sizes on those three centroids. Note that the centroid of a cluster is typically *not* an example in the cluster.

The preceding illustrations shows k-means for examples with only two features (height and width). Note that k-means can group examples across many features.

label

In supervised learning, the "answer" or "result" portion of an **example**. Each example in a labeled dataset consists of one or more features and a label. For instance, in a housing dataset, the features might include the number of bedrooms, the number of bathrooms, and the age of the house, while the label might be the house's price. In a spam detection dataset, the features might include the subject line, the sender, and the email message itself, while the label would probably be either "spam" or "not spam."

layer

A set of **neurons** in a **neural network** that process a set of input features, or the output of those neurons.

Also, an abstraction in TensorFlow. Layers are Python functions that take **Tensors** and configuration options as input and produce other tensors as output. Once the necessary Tensors have been composed, the user can convert the result into an **Estimator** via a **model function**.

Layers API (tf.layers)

A TensorFlow API for constructing a **deep** neural network as a composition of layers. The Layers API enables you to build different types of **layers**, such as:

- `tf.layers.Dense` for a **fully-connected layer**.
- `tf.layers.Conv2D` for a convolutional layer.

When writing a custom Estimator**, you compose Layers objects to define the characteristics of all the** hidden layers**.**

The Layers API **follows the** Keras **layers API** conventions. That is, aside from a different prefix, all functions in the Layers API have the same names and signatures as their counterparts in the Keras layers API.

learning rate

A scalar used to train a model via gradient descent. During each iteration, the **gradient descent** algorithm multiplies the learning rate by the gradient. The resulting product is called the **gradient step**.

logistic regression

A **classification model** that uses a **sigmoid function** to convert a **linear model's** raw prediction (y′) into a value between 0 and 1. You can interpret the value between 0 and 1 in either of the following two ways:

- As a probability that the example belongs to the **positive class** in a binary classification problem.
- As a value to be compared against a **classification threshold**. If the value is equal to or above the classification threshold, the system classifies the example as the positive class. Conversely, if the value is below the given threshold, the system classifies the example as the **negative class**. For example, suppose the classification threshold is 0.82:
- Imagine an example that produces a raw prediction (y′) of 2.6. The sigmoid of 2.6 is 0.93. Since 0.93 is greater than 0.82, the system classifies this example as the positive class.
- Imagine a different example that produces a raw prediction of 1.3. The sigmoid of 1.3 is 0.79. Since 0.79 is less than 0.82, the system classifies that example as the negative class.

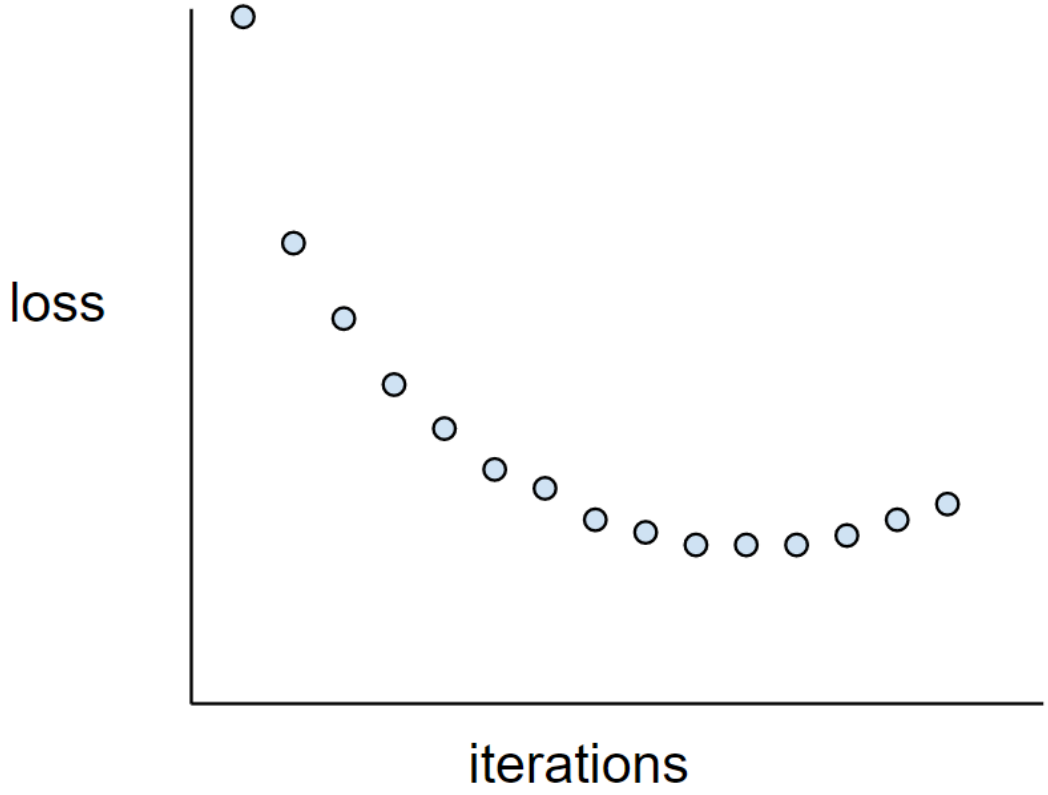
Although logistic regression is often used in binary classification **problems, logistic regression can also be used in** multi-class classification **problems (where it becomes called** multi-class logistic regression **or** multinomial regression**).**

loss

A measure of how far a model's **predictions** are from its **label**. Or, to phrase it more pessimistically, a measure of how bad the model is. To determine this value, a model must define a loss function. For example, linear regression models typically use **mean squared error** for a loss function, while logistic regression models use **Log Loss**.

loss curve

A graph of **loss** as a function of training **iterations**. For example:



The loss curve can help you determine when your model is **converging**, **overfitting**, or **underfitting**.

machine learning

A program or system that builds (trains) a predictive model from input data. The system uses the learned model to make useful predictions from new (never-before-seen) data drawn from the same distribution as the one used to train the model. Machine learning also refers to the field of study concerned with these programs or systems.

Mean Absolute Error (MAE)

An error metric calculated by taking an average of absolute errors. In the context of evaluating a model's accuracy, MAE is the average absolute difference between the expected and predicted values across all training examples. Specifically, for n examples, for each value y and its prediction y^, MAE is defined as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=0}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE)

The average squared loss per example. MSE is calculated by dividing the squared loss by the number of examples. The values that TensorFlow Playground displays for "Training loss" and "Test loss" are MSE.

mini-batch stochastic gradient descent (SGD)

A **gradient descent** algorithm that uses **mini-batches**. In other words, mini-batch SGD estimates the gradient based on a small subset of the training data. **Vanilla SGD** uses a mini-batch of size 1.

model

The representation of what a machine learning system has learned from the training data. Within TensorFlow, model is an overloaded term, which can have either of the following two related meanings:

- The **TensorFlow** graph that expresses the structure of how a prediction will be computed.
- The particular weights and biases of that TensorFlow graph, which are determined by **training**.

multi-class classification

Classification problems that distinguish among more than two classes. For example, there are approximately 128 species of maple trees, so a model that categorized maple tree species would be multi-class. Conversely, a model that divided emails into only two categories (*spam* and *not spam*) would be a **binary classification model**.

neural network

A model that, taking inspiration from the brain, is composed of layers (at least one of which is **hidden**) consisting of simple connected units or **neurons** followed by nonlinearities.

neuron

A node in a **neural network**, typically taking in multiple input values and generating one output value. The neuron calculates the output value by applying an **activation function** (nonlinear transformation) to a weighted sum of input values.

node (neural network)

A **neuron** in a **hidden layer**.

normalization

The process of converting an actual range of values into a standard range of values, typically -1 to +1 or 0 to 1. For example, suppose the natural range of a certain feature is 800 to 6,000. Through subtraction and division, you can normalize those values into the range -1 to +1.

optimizer

A specific implementation of the **gradient descent** algorithm. TensorFlow's base class for optimizers is tf.train.Optimizer. Popular optimizers include:

- AdaGrad, which stands for ADAptive GRADient descent.
- Adam, which stands for ADAptive with Momentum.

Different optimizers may leverage one or more of the following concepts to enhance the effectiveness of gradient descent on a given **training set**:

- momentum (Momentum)
- update frequency
- sparsity/regularization (Ftrl)
- more complex math (Proximal, and others)

output layer

The "final" layer of a neural network. The layer containing the answer(s).

overfitting

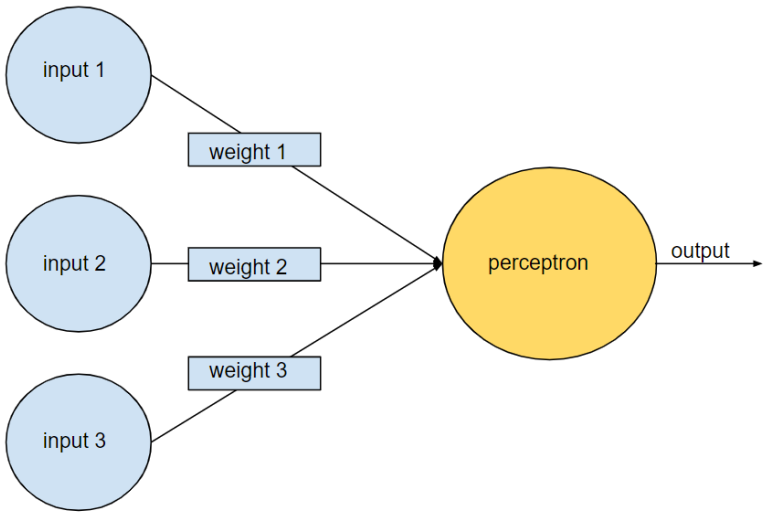
Creating a model that matches the **training data** so closely that the model fails to make correct predictions on new data.

perceptron

A system (either hardware or software) that takes in one or more input values, runs a function on the weighted sum of the inputs, and computes a single output value. In machine learning, the function is typically nonlinear, such as ReLU, sigmoid, or tanh. For example, the following perceptron relies on the sigmoid function to process three input values:

$$f(x_1, x_2, x_3) = \text{sigmoid}(w_1 x_1 + w_2 x_2 + w_3 x_3)$$

In the following illustration, the perceptron takes three inputs, each of which is itself modified by a weight before entering the perceptron:



Perceptrons are the (**nodes**) in **deep neural networks**. That is, a deep neural network consists of multiple connected perceptrons, plus a **backpropagation** algorithm to introduce feedback.

performance

Overloaded term with the following meanings:

- The traditional meaning within software engineering. Namely: How fast (or efficiently) does this piece of software run?

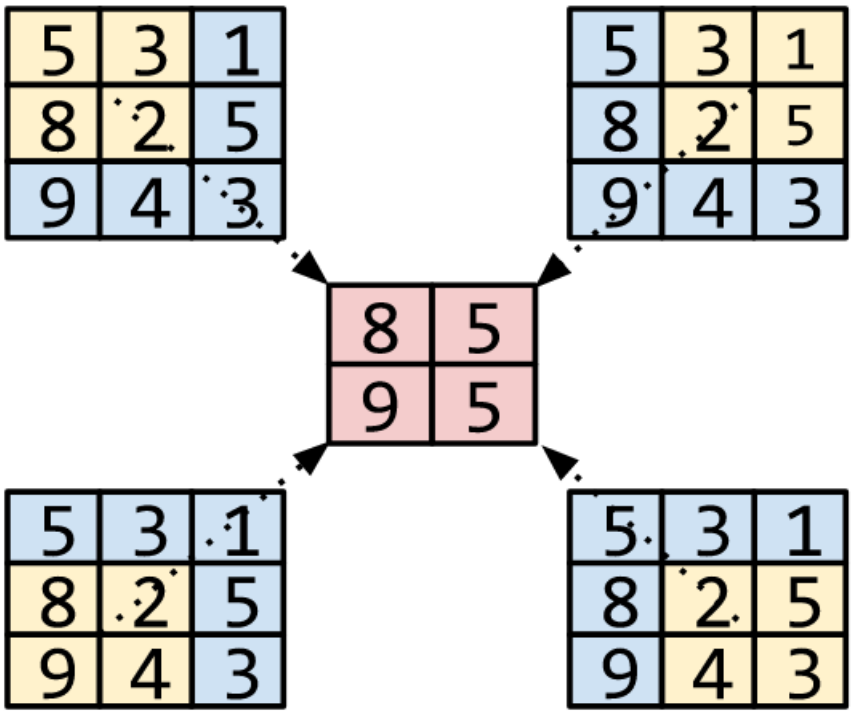
- The meaning within machine learning. Here, performance answers the following question: How correct is this **model**? That is, how good are the model's predictions?

pooling

Reducing a matrix (or matrices) created by an earlier convolutional layer to a smaller matrix. Pooling usually involves taking either the maximum or average value across the pooled area. For example, suppose we have the following 3x3 matrix:

5	3	1
8	2	5
9	4	3

A pooling operation, just like a convolutional operation, divides that matrix into slices and then slides that convolutional operation by strides. For example, suppose the pooling operation divides the convolutional matrix into 2x2 slices with a 1x1 stride. As the following diagram illustrates, four pooling operations take place. Imagine that each pooling operation picks the maximum value of the four in that slice:



Pooling helps enforce translational invariance in the input matrix.

Pooling for vision applications is known more formally as **spatial pooling**. Time-series applications usually refer to pooling as **temporal pooling**. Less formally, pooling is often called **subsampling** or **downsampling**.

positive class

In **binary classification**, the two possible classes are labeled as positive and negative. The positive outcome is the thing we're testing for. (Admittedly, we're simultaneously testing for both outcomes, but play along.) For example, the positive class in a medical test might be "tumor." The positive class in an email classifier might be "spam."

Contrast with **negative class**.

post-processing

Processing the output of a model after the model has been run. Post-processing can be used to enforce fairness constraints without modifying models themselves.

For example, one might apply post-processing to a binary classifier by setting a classification threshold such that equality of opportunity is maintained for some attribute by checking that the true positive rate is the same for all values of that attribute.

precision

A metric for classification models. Precision identifies the frequency with which a model was correct when predicting the positive class. That is:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

preprocessing

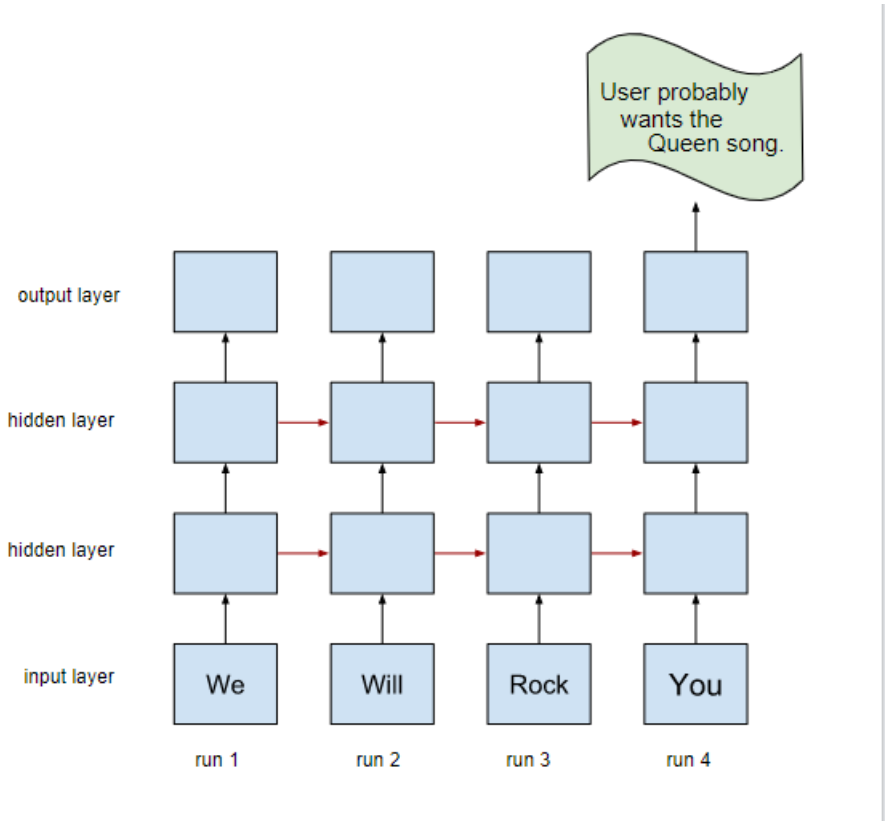
Processing data before it's used to train a model. Preprocessing could be as simple as removing words from an English text corpus that don't occur in the English dictionary, or could be as complex as re-expressing data points in a way that

eliminates as many attributes that are correlated with sensitive attributes as possible. Preprocessing can help satisfy fairness constraints.

recurrent neural network(RNN)

A neural network that is intentionally run multiple times, where parts of each run feed into the next run. Specifically, hidden layers from the previous run provide part of the input to the same hidden layer in the next run. Recurrent neural networks are particularly useful for evaluating sequences, so that the hidden layers can learn from previous runs of the neural network on earlier parts of the sequence.

For example, the following figure shows a recurrent neural network that runs four times. Notice that the values learned in the hidden layers from the first run become part of the input to the same hidden layers in the second run. Similarly, the values learned in the hidden layer on the second run become part of the input to the same hidden layer in the third run. In this way, the recurrent neural network gradually trains and predicts the meaning of the entire sequence rather than just the meaning of individual words.



regularization

The penalty on a model's complexity. Regularization helps prevent **overfitting**. Different kinds of regularization include:

- **L1 regularization**
- **L2 regularization**
- **dropout regularization**
- **early stopping** (this is not a formal regularization method, but can effectively limit overfitting)

semi-supervised learning

Training a model on data where some of the training examples have labels but others don't. One technique for semi-supervised learning is to infer labels for the unlabeled examples, and then to train on the inferred labels to create a new model. Semi-supervised learning can be useful if labels are expensive to obtain but unlabeled examples are plentiful.

sigmoid function

A function that maps logistic or multinomial regression output (log odds) to probabilities, returning a value between 0 and 1. The sigmoid function has the following formula:

$$y = \frac{1}{1 + e^{-\sigma}}$$

where σ in logistic regression problems is simply:

$$\sigma = b + w_1 x_1 + w_2 x_2 + \ldots w_n x_n$$

In other words, the sigmoid function converts σ into a probability between 0 and 1. In some neural networks, the sigmoid function acts as the activation function.

softmax

A function that provides probabilities for each possible class in a **multi-class classification model**. The probabilities add up to exactly 1.0. For example, softmax might determine that the probability of a particular image being a dog at 0.9, a cat at 0.08, and a horse at 0.02. (Also called **full softmax**.)
Contrast with **candidate sampling**.

supervised machine learning

Training a **model** from input data and its corresponding **labels**. Supervised machine learning is analogous to a student learning a subject by studying a set of questions and their corresponding answers. After mastering the mapping between questions and answers, the student can then provide answers to new (never-before-seen) questions on the same topic. Compare with **unsupervised machine learning**.

Tensor

The primary data structure in TensorFlow programs. Tensors are N-dimensional (where N could be very large) data structures, most commonly scalars, vectors, or matrices. The elements of a Tensor can hold integer, floating-point, or string values.

TensorFlow

A large-scale, distributed, machine learning platform. The term also refers to the base API layer in the TensorFlow stack, which supports general computation on dataflow graphs.

Although TensorFlow is primarily used for machine learning, you may also use TensorFlow for non-ML tasks that require numerical computation using dataflow graphs.

test set

The subset of the dataset that you use to test your **model** after the model has gone through initial vetting by the validation set.
Contrast with **training set** and **validation set**.

tf.keras

An implementation of Keras integrated into TensorFlow.

training

The process of determining the ideal **parameters** comprising a model.

training set

The subset of the dataset used to train a model.

transfer learning

Transferring information from one machine learning task to another. For example, in multi-task learning, a single model solves multiple tasks, such as a **deep model** that has different output nodes for different tasks. Transfer learning might involve transferring knowledge from the solution of a simpler task to a more complex one, or involve transferring knowledge from a task where there is more data to one where there is less data.

Most machine learning systems solve a *single* task. Transfer learning is a baby step towards artificial intelligence in which a single program can solve *multiple* tasks.

true negative (TN)

An example in which the model *correctly* predicted the negative class. For example, the model inferred that a particular email message was not spam, and that email message really was not spam.

true positive (TP)

An example in which the model *correctly* predicted the positive class. For example, the model inferred that a particular email message was spam, and that email message really was spam.

true positive rate (TPR)

Synonym for recall. That is:

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

True positive rate is the y-axis in an ROC curve.

underfitting

Producing a model with poor predictive ability because the model hasn't captured the complexity of the training data. Many problems can cause underfitting, including:

- Training on the wrong set of features.
- Training for too few epochs or at too low a learning rate.
- Training with too high a regularization rate.
- Providing too few hidden layers in a deep neural network.

unsupervised machine learning

Training a **model** to find patterns in a dataset, typically an unlabeled dataset.

The most common use of unsupervised machine learning is to cluster data into groups of similar examples. For example, an unsupervised machine learning algorithm can cluster songs together based on various properties of the music. The resulting clusters can become an input to other machine learning algorithms (for example, to a music recommendation service). Clustering can be helpful in domains where true labels are hard to obtain. For example, in domains such as anti-abuse and fraud, clusters can help humans better understand the data.

Another example of unsupervised machine learning is principal component analysis (PCA). For example, applying PCA on a dataset containing the contents of millions of shopping carts might reveal that shopping carts containing lemons frequently also contain antacids.

Compare with **supervised machine learning**

validation

A process used, as part of **training**, to evaluate the quality of a **machine learning** model using the **validation set**. Because the validation set is disjoint from the training set, validation helps ensure that the model's performance generalizes beyond the training set.

validation set

A subset of the dataset—disjoint from the training set—used in **validation**.

weight

A coefficient for a **feature** in a linear model, or an edge in a deep network. The goal of training a linear model is to determine the ideal weight for each feature. If a weight is 0, then its corresponding feature does not contribute to the model.

width

The number of **neurons** in a particular **layer** of a **neural network**.

Course recommendation

- coursera :Neural Networks and Deep Learning by Andrew Ng
- Stanford CS230: Deep Learning
- Stanford University CS231n
- CS231n: Convolutional Neural Networks for Visual Recognition
- https://developers.google.com/machine-learning/crash-course

credit

https://www.simplilearn.com/deep-learning-tutorial

https://developers.google.com/machine-learning

https://www.youtube.com/playlist?list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

https://en.wikipedia.org/wiki/Deep_learning

https://www.kaggle.com/kanncaa1/deep-learning-tutorial-for-beginners

https://www.datacamp.com/community/tutorials/deep-learning-python

https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/