Machine Learning Glossary

This glossary defines general machine learning terms, plus terms specific to TensorFlow.

Did You Know?

You can **filter the glossary** by choosing a topic from the Glossary dropdown in the top navigation bar. The hatching bird icon signifies definitions aimed at ML newcomers.

A

A/B testing

A statistical way of comparing two (or more) techniques—the *A* and the *B*. Typically, the *A* is an existing technique, and the *B* is a new technique. A/B testing not only determines which technique performs better but also whether the difference is statistically significant.

A/B testing usually compares a single **metric** (#metric) on two techniques; for example, how does model **accuracy** (#accuracy) compare for two techniques? However, A/B testing can also compare any finite number of metrics.

accelerator chip

A category of specialized hardware components designed to perform key computations needed for deep learning algorithms.

Accelerator chips (or just **accelerators**, for short) can significantly increase the speed and efficiency of training and inference tasks compared to a general-purpose CPU. They are ideal for training neural networks and similar computationally intensive tasks.

Examples of accelerator chips include:

- Google's Tensor Processing Units (**TPUs** (#TPU)) with dedicated hardware for deep learning.
- NVIDIA's GPUs which, though initially designed for graphics processing, are designed to enable parallel processing, which can significantly increase processing speed.

accuracy

The number of correct classification **predictions** (#prediction) divided by the total number of predictions. That is:

$$\text{Accuracy} = \frac{\text{correct predictions}}{\text{correct predictions} + \text{incorrect predictions}}$$

For example, a model that made 40 correct predictions and 10 incorrect predictions would have an accuracy of:

$$\text{Accuracy} = \frac{40}{40 + 10} = 80\%$$

Binary classification (#binary_classification) provides specific names for the different categories of *correct predictions* and *incorrect predictions*. So, the accuracy formula for binary classification is as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

where:

- TP is the number of **true positives** (#TP) (correct predictions).
- TN is the number of **true negatives** (#TN) (correct predictions).
- FP is the number of **false positives** (#FP) (incorrect predictions).
- FN is the number of **false negatives** (#FN) (incorrect predictions).

Compare and contrast accuracy with **precision** (#precision) and **recall** (#recall).

Click the icon for additional notes.

Although a valuable metric for some situations, accuracy is highly misleading for others. Notably, accuracy is usually a poor metric for evaluating classification models that process **class-imbalanced datasets** (#class_imbalanced_data_set).

For example, suppose snow falls only 25 days per century in a certain subtropical city. Since days without snow (the negative class) vastly outnumber days with snow (the positive class), the snow dataset for this city is class-imbalanced. Imagine a **binary classification** (#binary-classification) model that is supposed to predict either snow or no snow each day but simply predicts "no snow" every day. This model is highly accurate but has no predictive power. The following table summarizes the results for a century of predictions:

Category	Number
TP	0
TN	36500
FP	25
FN	0

The accuracy of this model is therefore:

```
accuracy = (TP + TN) / (TP + TN + FP + FN)
accuracy = (0 + 36500) / (0 + 36500 + 25 + 0) = 0.9993 = 99.93%
```

Although 99.93% accuracy seems like very a impressive percentage, the model actually has no predictive power.

Precision (#precision) and **recall** (#recall) are usually more useful metrics than **accuracy** for evaluating models trained on class-imbalanced datasets.

---

action RL

In **reinforcement learning** (#reinforcement_learning), the mechanism by which the **agent** (#agent) transitions between **states** (#state) of the **environment** (#environment). The agent chooses the action by using a **policy** (#policy).

---

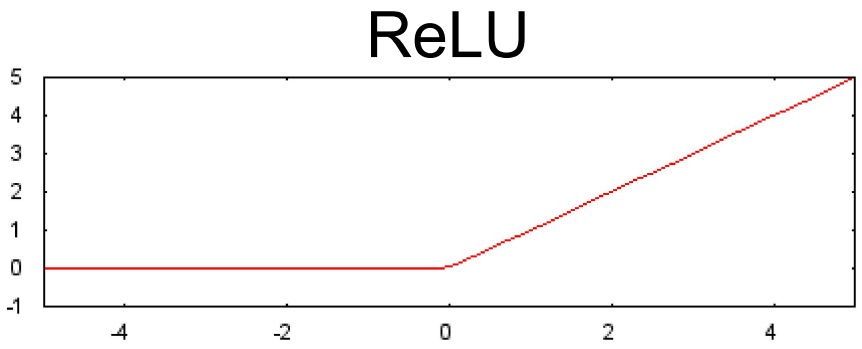
activation function

A function that enables **neural networks** (#neural_network) to learn **nonlinear** (#nonlinear) (complex) relationships between features and the label.

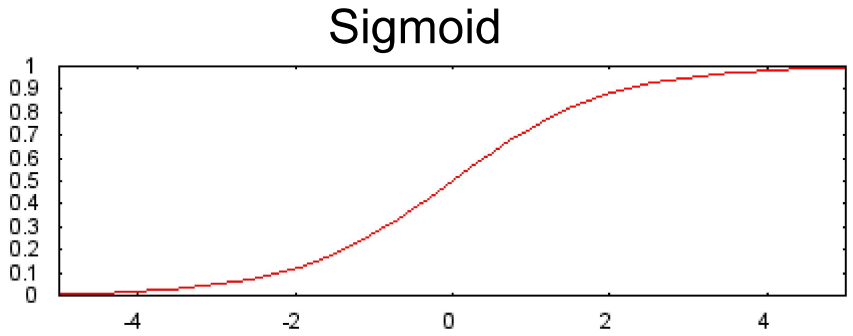
Popular activation functions include:

- **ReLU** (#ReLU)
- **Sigmoid** (#sigmoid-function)

The plots of activation functions are never single straight lines. For example, the plot of the ReLU activation function consists of two straight lines:



A plot of the sigmoid activation function looks as follows:



Click the icon to see an example.

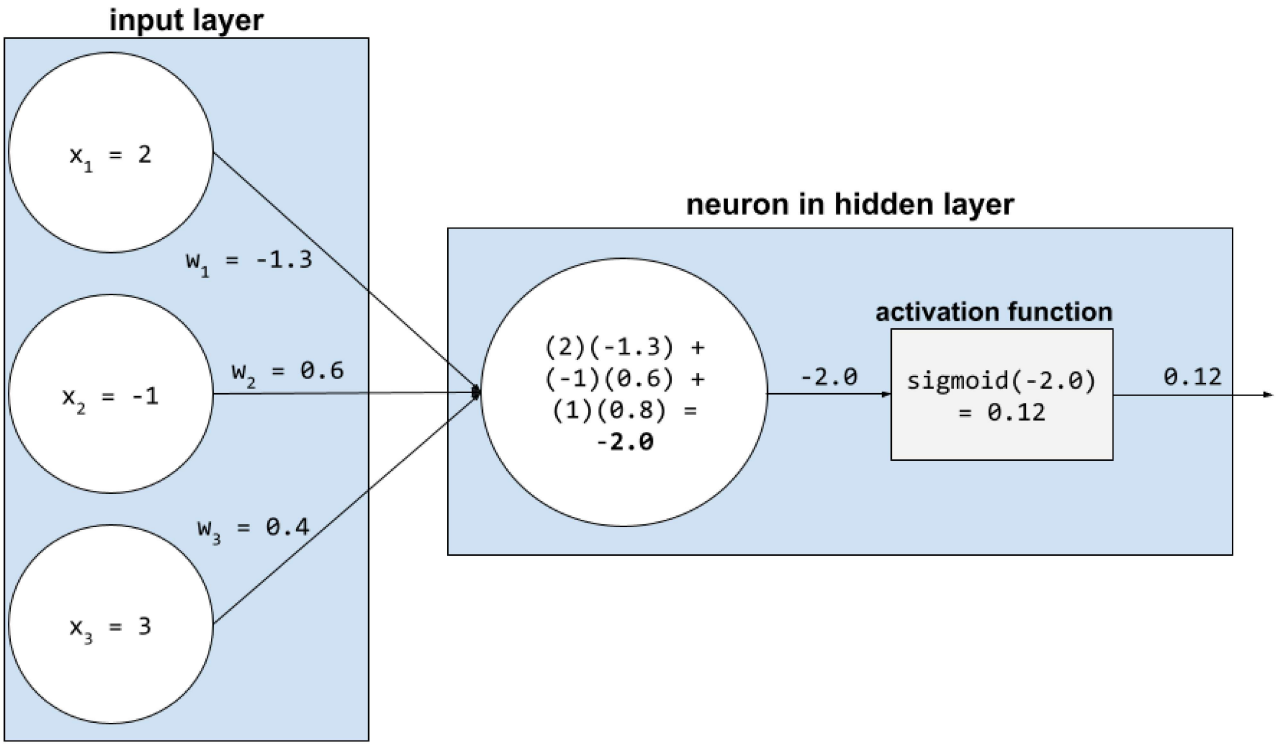
In a neural network, activation functions manipulate the weighted sum (#weighted_sum) of all the inputs to a neuron (#neuron). To calculate a weighted sum, the neuron adds up the products of the relevant values and weights. For example, suppose the relevant input to a neuron consists of the following:

input value	input weight
2	-1.3
-1	0.6
3	0.4

The weighted sum is therefore:

```
weighted sum = (2)(-1.3) + (-1)(0.6) + (3)(0.4) = -2.0
```

Suppose the designer of this neural network chooses the **sigmoid function** (#sigmoid-function) to be the activation function. In that case, the neuron calculates the sigmoid of -2.0, which is approximately 0.12. Therefore, the neuron passes 0.12 (rather than -2.0) to the next layer in the neural network. The following figure illustrates the relevant part of the process:



active learning

A **training** (#training) approach in which the algorithm *chooses* some of the data it learns from. Active learning is particularly valuable when **labeled examples** (#labeled_example) 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.

AdaGrad

A sophisticated gradient descent algorithm that rescales the gradients of each **parameter** (#parameter), effectively giving each parameter an independent **learning rate** (#learning_rate). For a full explanation, see this paper (http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf).

agent RL

In **reinforcement learning** (#reinforcement_learning), the entity that uses a **policy** (#policy) to maximize the expected **return** (#return) gained from transitioning between **states** (#state) of the **environment** (#environment).

agglomerative clustering

See **hierarchical clustering** (#hierarchical_clustering).

anomaly detection

The process of identifying **outliers** (#outliers). For example, if the mean for a certain **feature** (#feature) is 100 with a standard deviation of 10, then anomaly detection should flag a value of 200 as suspicious.

AR

Abbreviation for **augmented reality** (#augmented_reality).

area under the PR curve

See **PR AUC (Area under the PR Curve)** (#PR_AUC).

area under the ROC curve

See **AUC (Area under the ROC curve)** (#AUC).

artificial general intelligence

A non-human mechanism that demonstrates a *broad range* of problem solving, creativity, and adaptability. For example, a program demonstrating artificial general intelligence could translate text, compose symphonies, *and* excel at games that have not yet been invented.

artificial intelligence

A non-human program or **model** (#model) that can solve sophisticated tasks. For example, a program or model that translates text or a program or model that identifies diseases from radiologic images both exhibit artificial intelligence.

Formally, **machine learning** (#machine_learning) is a sub-field of artificial intelligence. However, in recent years, some organizations have begun using the terms *artificial intelligence* and *machine learning* interchangeably.

attention

A mechanism used in a **neural network** (#neural_network) that indicates the importance of a particular word or part of a word. Attention compresses the amount of information a model needs to predict the next token/word. A typical attention mechanism might consist of a **weighted sum** (#weighted_sum) over a set of inputs, where the **weight** (#weight) for each input is computed by another part of the neural network.

Refer also to **self-attention** (#self-attention) and **multi-head self-attention** (#multi-head-self-attention), which are the building blocks of **Transformers** (#Transformer).

attribute

Synonym for **feature** (#feature).

In machine learning fairness, attributes often refer to characteristics pertaining to individuals.

attribute sampling

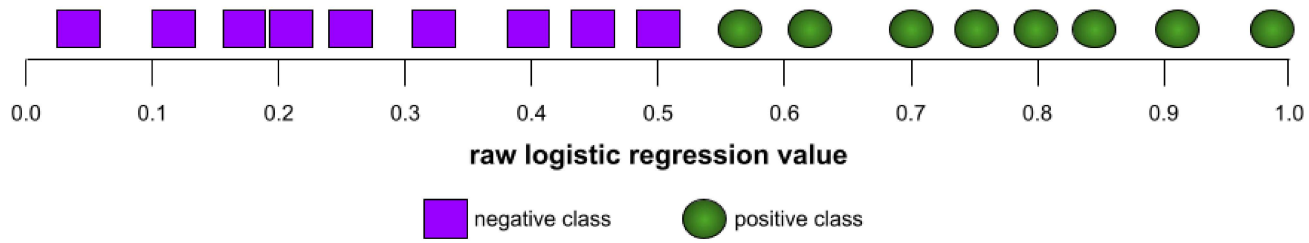
A tactic for training a **decision forest** (#decision-forest) in which each **decision tree** (#decision-tree) considers only a random subset of possible **features** (#feature) when learning the **condition** (#condition). Generally, a different subset of features is sampled for each **node**

(#node-decision-tree). In contrast, when training a decision tree without attribute sampling, all possible features are considered for each node.

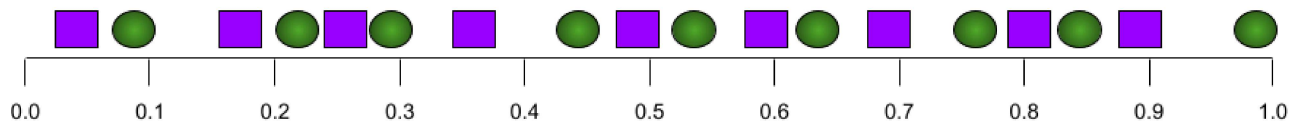
AUC (Area under the ROC curve)

A number between 0.0 and 1.0 representing a **binary classification** (#binary-classification) model's ability to separate **positive classes** (#positive_class) from **negative classes** (#negative_class). The closer the AUC is to 1.0, the better the model's ability to separate classes from each other.

For example, the following illustration shows a classifier model that separates positive classes (green ovals) from negative classes (purple rectangles) perfectly. This unrealistically perfect model has an AUC of 1.0:

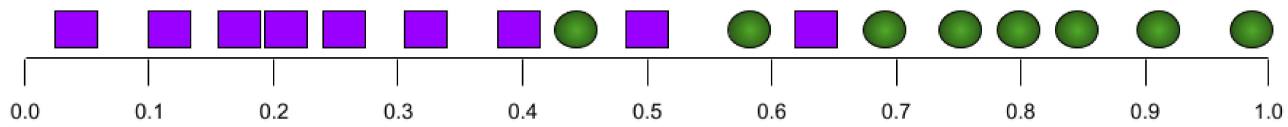


Conversely, the following illustration shows the results for a classifier model that generated random results. This model has an AUC of 0.5:



Yes, the preceding model has an AUC of 0.5, not 0.0.

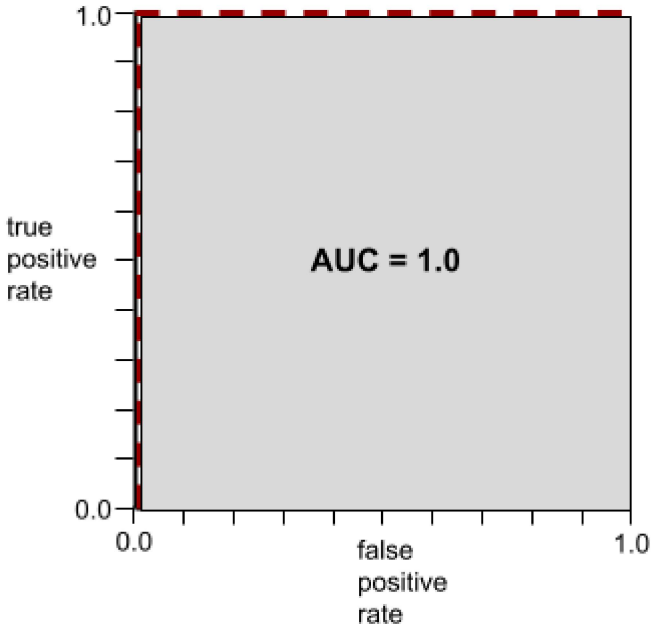
Most models are somewhere between the two extremes. For instance, the following model separates positives from negatives somewhat, and therefore has an AUC somewhere between 0.5 and 1.0:



AUC ignores any value you set for **classification threshold** (#classification_threshold). Instead, AUC considers *all* possible classification thresholds.

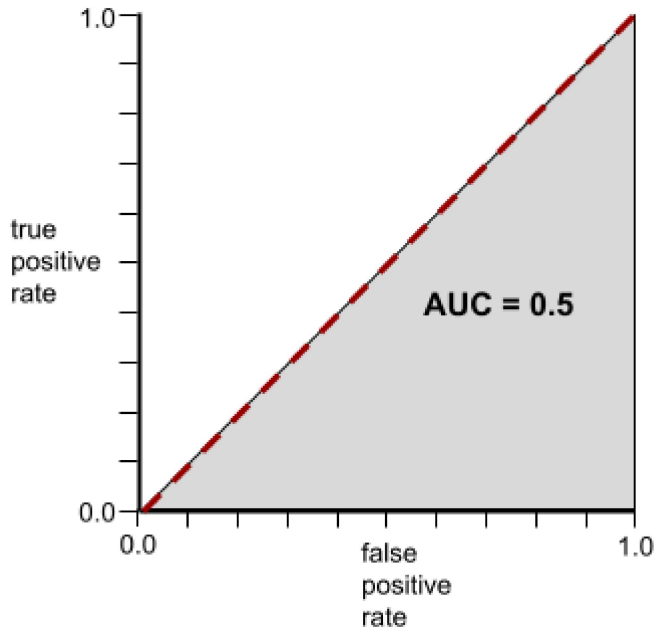
Click the icon to learn about the relationship between AUC and ROC curves.

AUC represents the *area* under an **ROC curve** (#ROC). For example, the ROC curve for a model that perfectly separates positives from negatives looks as follows:

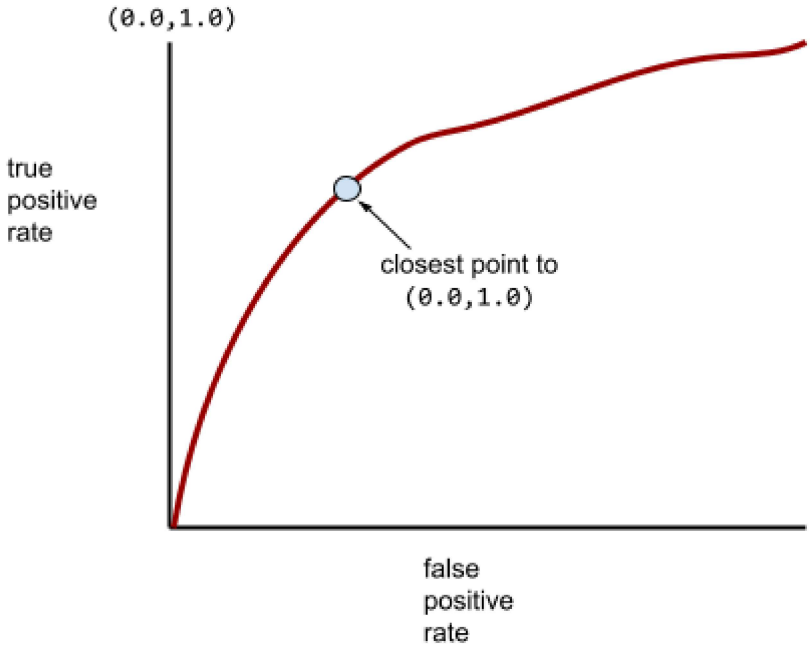


AUC is the area of the gray region in the preceding illustration. In this unusual case, the area is simply the length of the gray region (1.0) multiplied by the width of the gray region (1.0). So, the product of 1.0 and 1.0 yields an AUC of exactly 1.0, which is the highest possible AUC score.

Conversely, the ROC curve for a classifier that can't separate classes at all is as follows. The area of this gray region is 0.5.



A more typical ROC curve looks approximately like the following:



It would be painstaking to calculate the area under this curve manually, which is why a program typically calculates most AUC values.

Click the icon for a more formal definition of AUC.

AUC is the probability that a classifier will be more confident that a randomly chosen positive example is actually positive than that a randomly chosen negative example is positive.

augmented reality

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

automation bias

When a human decision maker favors recommendations made by an automated decision-making system over information made without automation, even when the automated decision-making system makes errors.

AutoML

Any automated process for building **machine learning** (#machine_learning) **models** (#model). AutoML can automatically do tasks such as the following:

- Search for the most appropriate model.
- Tune **hyperparameters** (#hyperparameter).
- Prepare data (including performing **feature engineering** (#feature_engineering)).
- Deploy the resulting model.

AutoML is useful for data scientists because it can save them time and effort in developing machine learning pipelines and improve prediction accuracy. It is also useful to non-experts, by making complicated machine learning tasks more accessible to them.

auxiliary loss

A **loss function** (#loss-function)—used in conjunction with a **neural network** (#neural-network) **model's** (#model) main loss function—that helps accelerate **training** (#training) during the early iterations when weights are randomly initialized.

Auxiliary loss functions push effective **gradients** (#gradient) to the earlier **layers** (#layer). This facilitates **convergence** (#convergence) during **training** (#training) by combating the **vanishing gradient problem** (#vanishing_gradient_problem).

average precision

A metric for summarizing the performance of a ranked sequence of results. Average precision is calculated by taking the average of the **precision** (#precision) values for each relevant result (each result in the ranked list where the recall increases relative to the previous result).

See also **Area under the PR Curve** (#area_under_the_pr_curve).

axis-aligned condition

In a **decision tree** (#decision-tree), a **condition** (#condition) that involves only a single **feature** (#feature). For example, if area is a feature, then the following is an axis-aligned condition:

```
area > 200
```

Contrast with **oblique condition** (#oblique-condition).

B

backpropagation

The algorithm that implements **gradient descent** (#gradient_descent) in **neural networks** (#neural_network).

Training a neural network involves many **iterations** (#iteration) of the following two-pass cycle:

1. During the **forward pass**, the system processes a **batch** (#batch) of **examples** (#example) to yield prediction(s). The system compares each prediction to each **label** (#label) value. The difference between the prediction and the label value is the **loss** (#loss) for that example. The system aggregates the losses for all the examples to compute the total loss for the current batch.
2. During the **backward pass** (backpropagation), the system reduces loss by adjusting the weights of all the **neurons** (#neuron) in all the **hidden layer(s)** (#hidden_layer).

Neural networks often contain many neurons across many hidden layers. Each of those neurons contribute to the overall loss in different ways. Backpropagation determines whether to increase or decrease the weights applied to particular neurons.

The **learning rate** (#learning_rate) is a multiplier that controls the degree to which each backward pass increases or decreases each weight. A large learning rate will increase or decrease each weight more than a small learning rate.

In calculus terms, backpropagation implements calculus' **chain rule** (https://www.khanacademy.org/math/ap-calculus-ab/ab-differentiation-2-new/ab-3-1a/v/chain-rule-introduction). That is, backpropagation calculates the **partial derivative** (#partial_derivative) of the error with respect to each parameter. For more details, see this **tutorial in Machine Learning Crash Course** (https://developers-dot-devsite-v2-prod.appspot.com/machine-learning/crash-course/backprop-scroll).

Years ago, ML practitioners had to write code to implement backpropagation. Modern ML APIs like TensorFlow now implement backpropagation for you. Phew!

bagging

A method to **train** (#training) an **ensemble** (#ensemble) where each constituent **model** (#model) trains on a random subset of training examples **sampled with replacement** (#sampling-with-replacement). For example, a **random forest** (#random-forest) is a collection of **decision trees** (#decision-tree) trained with bagging.

The term **bagging** is short for **b**ootstrap **agg**regat**ing**.

bag of words

A representation of the words in a phrase or passage, irrespective of order. For example, bag of words represents the following three phrases identically:

- the dog jumps
- jumps the dog
- dog jumps the

Each word is mapped to an index in a **sparse vector** (#sparse_vector), where the vector has an index for every word in the vocabulary. For example, the phrase *the dog jumps* is mapped into a

feature vector with non-zero values at the three indices corresponding to the words *the*, *dog*, and *jumps*. The non-zero value can be any of the following:

- A 1 to indicate the presence of a word.
- A count of the number of times a word appears in the bag. For example, if the phrase were *the maroon dog is a dog with maroon fur*, then both *maroon* and *dog* would be represented as 2, while the other words would be represented as 1.
- Some other value, such as the logarithm of the count of the number of times a word appears in the bag.

baseline

A **model** (#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** (#logistic_regression) might serve as a good baseline for a **deep model** (#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** (#example) used in one training **iteration** (#iteration). The **batch size** (#batch_size) determines the number of examples in a batch.

See **epoch** (#epoch) for an explanation of how a batch relates to an epoch.

batch normalization

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

- Make **neural networks** (#neural_network) more stable by protecting against **outlier** (#outliers) weights.
- Enable higher **learning rates** (#learning_rate), which can speed training.
- Reduce **overfitting** (#overfitting).

batch size

The number of **examples** (#example) in a **batch** (#batch). For instance, if the batch size is 100, then the model processes 100 examples per **iteration** (#iteration).

The following are popular batch size strategies:

- **Stochastic Gradient Descent (SGD)** (#SGD), in which the batch size is 1.
- full batch, in which the batch size is the number of examples in the entire **training set** (#training_set). For instance, if the training set contains a million examples, then the batch size would be a million examples. Full batch is usually an inefficient strategy.
- **mini-batch** (#mini-batch) in which the batch size is usually between 10 and 1000. Mini-batch is usually the most efficient strategy.

Bayesian neural network

A probabilistic **neural network** (#neural_network) that accounts for uncertainty in **weights** (#weight) and outputs. A standard neural network regression model typically **predicts** (#prediction) a scalar value; for example, a standard model predicts a house price of 853,000. In

contrast, a Bayesian neural network predicts a distribution of values; for example, a Bayesian model predicts a house price of 853,000 with a standard deviation of 67,200.

A Bayesian neural network relies on Bayes' Theorem (https://betterexplained.com/articles/an-intuitive-and-short-explanation-of-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** (#overfitting).

Bayesian optimization

A **probabilistic regression model** (#probabilistic-regression-model) technique for optimizing computationally expensive **objective functions** (#objective_function) by instead optimizing a surrogate that quantifies the uncertainty via a Bayesian learning technique. Since Bayesian optimization is itself very expensive, it is usually used to optimize expensive-to-evaluate tasks that have a small number of parameters, such as selecting **hyperparameters** (#hyperparameter).

Bellman equation RL

In reinforcement learning, the following identity satisfied by the optimal **Q-function** (#q-function):

$$Q(s, a) = r(s, a) + \gamma \mathbb{E}_{s'|s,a} \max_{a'} Q(s', a')$$

Reinforcement learning (#reinforcement_learning) algorithms apply this identity to create **Q-learning** (#q-learning) via the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r(s, a) + \gamma \max_{a_1} Q(s', a') - Q(s, a) \right]$$

Beyond reinforcement learning, the Bellman equation has applications to dynamic programming. See the Wikipedia entry for Bellman Equation

(https://wikipedia.org/wiki/Bellman_equation).

BERT (Bidirectional Encoder Representations from Transformers)

A model architecture for text **representation** (#representation). A trained BERT model can act as part of a larger model for text classification or other ML tasks.

BERT has the following characteristics:

- Uses the **Transformer** (#Transformer) architecture, and therefore relies on **self-attention** (#self-attention).
- Uses the **encoder** (#encoder) part of the Transformer. The encoder's job is to produce good text representations, rather than to perform a specific task like classification.
- Is **bidirectional** (#bidirectional).
- Uses **masking** (#masked-language-model) for **unsupervised training** (#unsupervised_machine_learning).

BERT's variants include:

- ALBERT (https://ai.googleblog.com/2019/12/albert-lite-bert-for-self-supervised.html), which is an acronym for **A** Light **BERT**.
- LaBSE (https://ai.googleblog.com/2020/08/language-agnostic-bert-sentence.html).

See Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing (https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html) for an overview of BERT.

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** (#automation_bias)
- **confirmation bias** (#confirmation_bias)
- **experimenter's bias** (#confirmation_bias)
- **group attribution bias** (#group_attribution_bias)
- **implicit bias** (#implicit_bias)
- **in-group bias** (#in-group_bias)
- **out-group homogeneity bias** (#out-group_homogeneity_bias)

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

- **coverage bias** (#selection_bias)
- **non-response bias** (#selection_bias)
- **participation bias** (#participation_bias)
- **reporting bias** (#reporting_bias)
- **sampling bias** (#selection_bias)
- **selection bias** (#selection_bias)

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

bias (math) or bias term

An intercept or offset from an origin. Bias is a parameter in machine learning models, which is symbolized by either of the following:

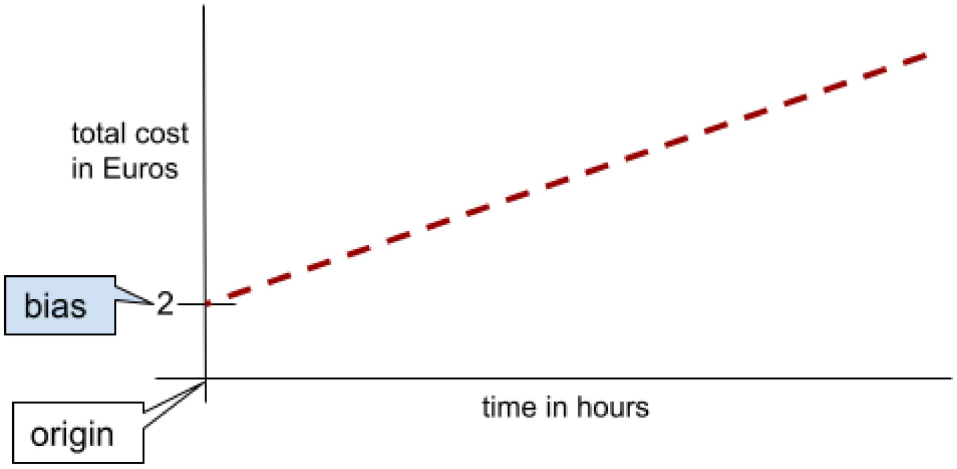
- *b*

- w_0

For example, bias is the *b* in the following formula:

$$y' = b + w_1x_1 + w_2x_2 + \ldots w_nx_n$$

In a simple two-dimensional line, bias just means "y-intercept." For example, the bias of the line in the following illustration is 2.



Bias exists because not all models start from the origin (0,0). For example, suppose an amusement park costs 2 Euros to enter and an additional 0.5 Euro for every hour a customer stays. Therefore, a model mapping the total cost has a bias of 2 because the lowest cost is 2 Euros.

Bias is not to be confused with **bias in ethics and fairness** (#bias_ethics) or **prediction bias** (#prediction_bias).

bigram

An **N-gram** (#N-gram) in which N=2.

bidirectional

A term used to describe a system that evaluates the text that both *precedes* and *follows* a target section of text. In contrast, a **unidirectional** (#unidirectional) system only evaluates the text that *precedes* a target section of text.

For example, consider a **masked language model** (#masked-language-model) that must determine probabilities for the word or words representing the underline in the following question:

What is the _____ with you?

A unidirectional language model would have to base its probabilities only on the context provided by the words "What", "is", and "the". In contrast, a bidirectional language model could also gain context from "with" and "you", which might help the model generate better predictions.

bidirectional language model

A **language model** (#language-model) that determines the probability that a given token is present at a given location in an excerpt of text based on the *preceding* and *following* text.

binary classification

A type of **classification** (#classification_model) task that predicts one of two mutually exclusive classes:

- the **positive class** (#positive_class)
- the **negative class** (#negative_class)

For example, the following two machine learning models each perform binary classification:

- A model that determines whether email messages are *spam* (the positive class) or *not spam* (the negative class).
- A model that evaluates medical symptoms to determine whether a person has a particular disease (the positive class) or doesn't have that disease (the negative class).

Contrast with **multi-class classification** (#multi-class).

See also **logistic regression** (#logistic_regression) and **classification threshold** (#classification_threshold).

binary condition

In a **decision tree** (#decision-tree), a **condition** (#condition) that has only two possible outcomes, typically *yes* or *no*. For example, the following is a binary condition:

```
temperature >= 100
```

Contrast with **non-binary condition** (#non-binary-condition).

binning

Synonym for **bucketing** (#bucketing).

BLEU (Bilingual Evaluation Understudy)

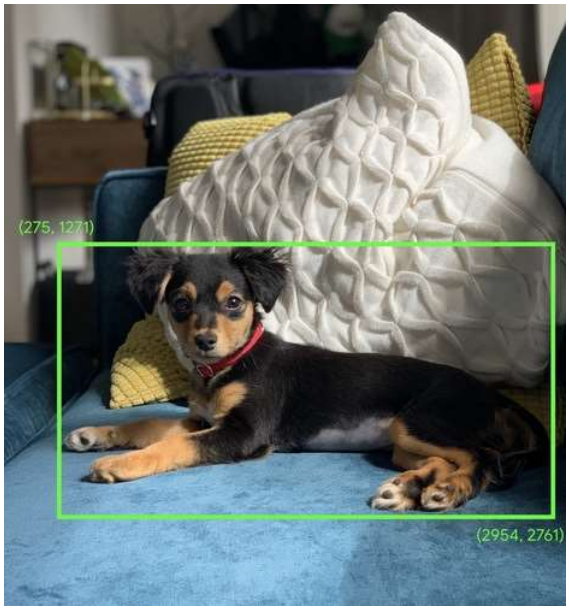
A score between 0.0 and 1.0, inclusive, indicating the quality of a translation between two human languages (for example, between English and Russian). A BLEU score of 1.0 indicates a perfect translation; a BLEU score of 0.0 indicates a terrible translation.

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** (#upweighting) the examples that the model is currently misclassifying.

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** (#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]]
```

See the following description of broadcasting in NumPy (https://docs.scipy.org/doc/numpy-1.15.0/user/basics.broadcasting.html) for more details.

bucketing

Converting a single **feature** (#feature) into multiple binary features called **buckets** or **bins**, typically based on a value range. The chopped feature is typically a **continuous feature** (#continuous_feature).

For example, instead of representing temperature as a single continuous floating-point feature, you could chop ranges of temperatures into discrete buckets, such as:

- <= 10 degrees Celsius would be the "cold" bucket.
- 11 - 24 degrees Celsius would be the "temperate" bucket.
- >= 25 degrees Celsius would be the "warm" bucket.

The model will treat every value in the same bucket identically. For example, the values 13 and 22 are both in the temperate bucket, so the model treats the two values identically.

Click the icon for additional notes.

If you represent temperature as a continuous feature, then the model treats temperature as a single feature. If you represent temperature as three buckets, then the model treats each bucket as a separate feature. That is, a model can learn separate relationships of each bucket to the **label** (#label). For example, a **linear regression** (#linear_regression) model can learn separate **weights** (#weight) for each bucket.

Increasing the number of buckets makes your model more complicated by increasing the number of relationships that your model must learn. For example, the cold, temperate, and warm buckets are essentially three separate features for your model to train on. If you decide to add two more buckets--for example, freezing and hot--your model would now have to train on five separate features.

How do you know how many buckets to create, or what the ranges for each bucket should be? The answers typically require a fair amount of experimentation.

C

calibration layer

A post-prediction adjustment, typically to account for **prediction bias** (#prediction_bias). The adjusted predictions and probabilities should match the distribution of an observed set of labels.

candidate generation

The initial set of recommendations chosen by a **recommendation system** (#recommendation_system). For example, consider a bookstore that offers 100,000 titles. The candidate generation phase creates a much smaller list of suitable books for a particular user, say 500. But even 500 books is way too many to recommend to a user. Subsequent, more expensive, phases of a recommendation system (such as **scoring** (#scoring) and **re-ranking** (#re-ranking)) reduce those 500 to a much smaller, more useful set of recommendations.

candidate sampling

A training-time optimization that calculates a probability for all the **positive** (#positive_class) labels, using, for example, **softmax** (#softmax), but only for a random sample of negative labels. For instance, given an example labeled *beagle* and *dog*, candidate sampling computes the predicted probabilities and corresponding loss terms for:

- *beagle*

- *dog*
- a random subset of the remaining negative classes (for example, *cat*, *lollipop*, *fence*).

The idea is that the **negative classes** (#negative_class) can learn from less frequent negative reinforcement as long as **positive classes** (#positive_class) always get proper positive reinforcement, and this is indeed observed empirically.

Candidate sampling is more computationally efficient than training algorithms that compute predictions for *all* negative classes, particularly when the number of negative classes is very large.

categorical data

Features (#feature) having a specific set of possible values. For example, consider a categorical feature named `traffic-light-state`, which can only have one of the following three possible values:

- `red`
- `yellow`
- `green`

By representing `traffic-light-state` as a categorical feature, a model can learn the differing impacts of `red`, `green`, and `yellow` on driver behavior.

Categorical features are sometimes called **discrete features** (#discrete_feature).

Contrast with **numerical data** (#numerical_data).

causal language model

Synonym for **unidirectional language model** (#unidirectional-language-model).

See **bidirectional language model** (#bidirectional-language-model) to contrast different directional approaches in language modeling.

centroid

The center of a cluster as determined by a **k-means** (#k-means) or **k-median** (#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** (#clustering) algorithms that organizes data into nonhierarchical clusters. **k-means** (#k-means) is the most widely used centroid-based clustering algorithm.

Contrast with **hierarchical clustering** (#hierarchical_clustering) algorithms.

checkpoint

Data that captures the state of a model's **parameters** (#parameter) at a particular training iteration. Checkpoints enable exporting model **weights** (#weight), or performing **training** (#training) across multiple sessions. Checkpoints also enable training to continue past errors (for example, job preemption).

When **fine tuning** (#fine_tuning), the starting point for **training** (#training) the new **model** (#model) will be a specific checkpoint of the **pre-trained model** (#pre-trained_model).

class

A category that a **label** (#label) can belong to. For example:

- In a **binary classification** (#binary_classification) model that detects spam, the two classes might be *spam* and *not spam*.
- In a **multi-class classification** (#multi-class) model that identifies dog breeds, the classes might be *poodle*, *beagle*, *pug*, and so on.

A **classification model** (#classification_model) predicts a class. In contrast, a **regression model** (#regression_model) predicts a number rather than a class.

classification model

A **model** (#model) whose prediction is a **class** (#class). For example, the following are all classification models:

- A model that predicts an input sentence's language (French? Spanish? Italian?).
- A model that predicts tree species (Maple? Oak? Baobab?).
- A model that predicts the positive or negative class for a particular medical condition.

In contrast, **regression models** (#regression_model) predict numbers rather than classes.

Two common types of classification models are:

- **binary classification** (#binary-classification)
- **multi-class classification** (#multi-class)

classification threshold

In a **binary classification** (#binary-classification), a number between 0 and 1 that converts the raw output of a **logistic regression** (#logistic_regression) model into a prediction of either the **positive class** (#positive_class) or the **negative class** (#negative_class). Note that the classification threshold is a value that a human chooses, not a value chosen by model training.

A logistic regression model outputs a raw value between 0 and 1. Then:

- If this raw value is *greater than* the classification threshold, then the positive class is predicted.
- If this raw value is *less than* the classification threshold, then the negative class is predicted.

For example, suppose the classification threshold is 0.8. If the raw value is 0.9, then the model predicts the positive class. If the raw value is 0.7, then the model predicts the negative class.

The choice of classification threshold strongly influences the number of **false positives** (#FP) and **false negatives** (#FN).

Click the icon for additional notes.

As models or datasets evolve, engineers sometimes also change the classification threshold. When the classification threshold changes, positive class predictions can suddenly become negative classes and vice-versa.

For example, consider a binary classification disease prediction model. Suppose that when the system runs in the first year:

- The raw value for a particular patient is 0.95.
- The classification threshold is 0.94.

Therefore, the system diagnoses the positive class. (The patient gasps, "Oh no! I'm sick!")

A year later, perhaps the values now look as follows:

- The raw value for the same patient remains at 0.95.
- The classification threshold changes to 0.97.

Therefore, the system now reclassifies that patient as the negative class. ("Happy day! I'm not sick.") Same patient. Different diagnosis.

class-imbalanced dataset

A dataset for a classification problem in which the total number of **labels** (#label) of each class differs significantly. For example, consider a binary classification dataset whose two labels are divided as follows:

- 1,000,000 negative labels
- 10 positive labels

The ratio of negative to positive labels is 100,000 to 1, so this is a class-imbalanced dataset.

In contrast, the following dataset is *not* class-imbalanced because the ratio of negative labels to positive labels is relatively close to 1:

- 517 negative labels
- 483 positive labels

Multi-class datasets can also be class-imbalanced. For example, the following multi-class classification dataset is also class-imbalanced because one label has far more examples than the other two:

- 1,000,000 labels with class "green"
- 200 labels with class "purple"
- 350 labels with class "orange"

See also **entropy** (#entropy), **majority class** (#majority_class), and **minority class** (#minority_class).

clipping

A technique for handling **outliers** (#outliers) by doing either or both of the following:

- Reducing **feature** (#feature) values that are greater than a maximum threshold down to that maximum threshold.
- Increasing feature values that are less than a minimum threshold up to that minimum threshold.

For example, suppose that <0.5% of values for a particular feature fall outside the range 40–60. In this case, you could do the following:

- Clip all values over 60 (the maximum threshold) to be exactly 60.
- Clip all values under 40 (the minimum threshold) to be exactly 40.

Outliers can damage models, sometimes causing **weights** (#weight) to overflow during training. Some outliers can also dramatically spoil metrics like **accuracy** (#accuracy). Clipping is a common technique to limit the damage.

Gradient clipping (#gradient_clipping) forces **gradient** (#gradient) values within a designated range during training.

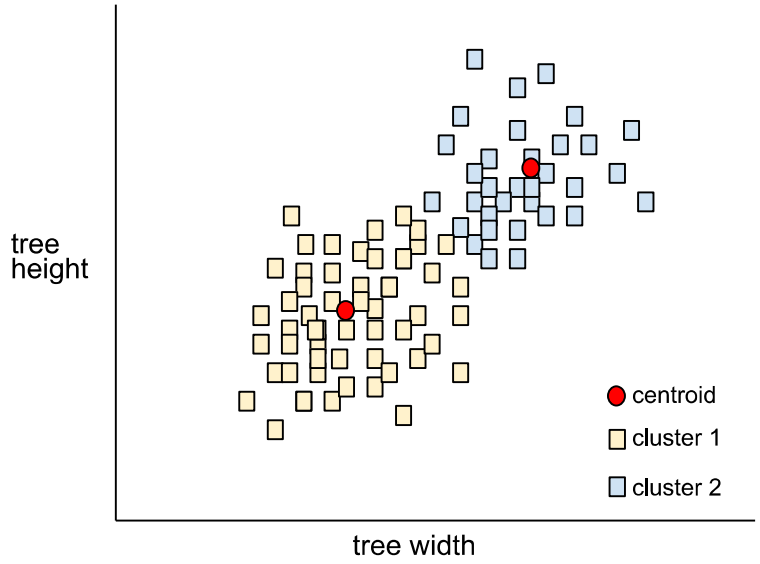
Cloud TPU

A specialized hardware accelerator designed to speed up machine learning workloads on Google Cloud Platform.

clustering

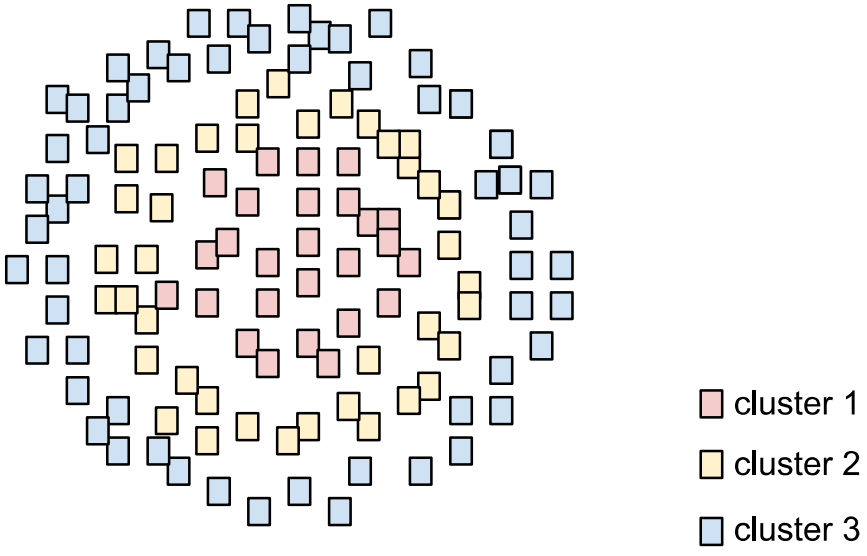
Grouping related **examples** (#example), particularly during **unsupervised learning** (#unsupervised_machine_learning). Once all the examples are grouped, a human can optionally supply meaning to each cluster.

Many clustering algorithms exist. For example, the **k-means** (#k-means) algorithm clusters examples based on their proximity to a **centroid** (#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

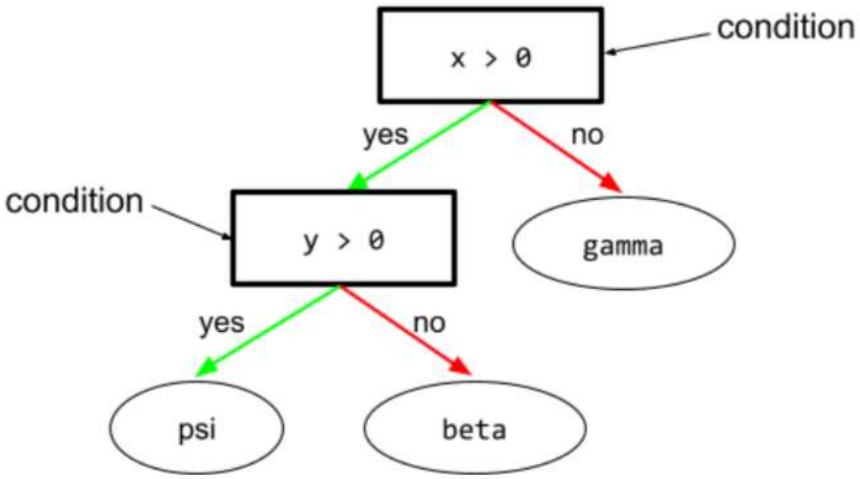
When **neurons** (#neuron) predict patterns in training data by relying almost exclusively on outputs of specific other neurons instead of relying on the network's behavior as a whole. When the patterns that cause co-adaption are not present in validation data, then co-adaptation causes overfitting.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

collaborative filtering

Making **predictions** (#prediction) about the interests of one user based on the interests of many other users. Collaborative filtering is often used in **recommendation systems** (#recommendation_system).

condition

In a **decision tree** (#decision-tree), any **node** (#node) that evaluates an expression. For example, the following portion of a decision tree contains two conditions:



A condition is also called a split or a test.

Contrast condition with **leaf** (#leaf).

See also:

- **binary condition** (#binary-condition)
- **non-binary condition** (#non-binary-condition).
- **axis-aligned-condition** (#axis-aligned-condition)
- **oblique-condition** (#oblique-condition)

configuration

The process of assigning the initial property values used to train a model, including:

- the model's composing **layers** (#layer)
- the location of the data
- **hyperparameters** (#hyperparameter) such as:
  - **learning rate** (#learning_rate)
  - **iterations** (#iteration)
  - **optimizer** (#optimizer)
  - **loss function** (#loss-function)

In machine learning projects, configuration can be done through a special configuration file or via configuration libraries such as the following:

- HParam (https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams)
- Gin (https://github.com/google/gin-config)
- **Fiddle** (#fiddle)

confirmation bias

The tendency to search for, interpret, favor, and recall information in a way that confirms one's preexisting beliefs or hypotheses. Machine learning developers may inadvertently collect or label data in ways that influence an outcome supporting their existing beliefs. Confirmation bias is a form of **implicit bias** (#implicit_bias).

Experimenter's bias is a form of confirmation bias in which an experimenter continues training models until a preexisting hypothesis is confirmed.

confusion matrix

An NxN table that summarizes the number of correct and incorrect predictions that a **classification model** (#classification_model) made. For example, consider the following confusion matrix for a **binary classification** (#binary_classification) model:

	Tumor (predicted)	Non-Tumor (predicted)
Tumor (ground truth)	18 (TP)	1 (FN)
Non-Tumor (ground truth)	6 (FP)	452 (TN)

The preceding confusion matrix shows the following:

- Of the 19 predictions in which **ground truth** (#ground_truth) was Tumor, the model correctly classified 18 and incorrectly classified 1.
- Of the 458 predictions in which ground truth was Non-Tumor, the model correctly classified 452 and incorrectly classified 6.

The confusion matrix for a **multi-class classification** (#multi-class) problem can help you identify patterns of mistakes. For example, consider the following confusion matrix for a 3-

class multi-class classification model that categorizes three different iris types (Virginica, Versicolor, and Setosa). When the ground truth was Virginica, the confusion matrix shows that the model was far more likely to mistakenly predict Versicolor than Setosa:

	Setosa (predicted)	Versicolor (predicted)	Virginica (predicted)
Setosa (ground truth)	88	12	0
Versicolor (ground truth)	6	141	7
Virginica (ground truth)	2	27	109

As yet another example, a confusion matrix could reveal that a model trained to recognize handwritten digits tends to mistakenly predict 9 instead of 4, or mistakenly predict 1 instead of 7.

Confusion matrices contain sufficient information to calculate a variety of performance metrics, including **precision** (#precision) and **recall** (#recall).

continuous feature

A floating-point **feature** (#feature) with an infinite range of possible values, such as temperature or weight.

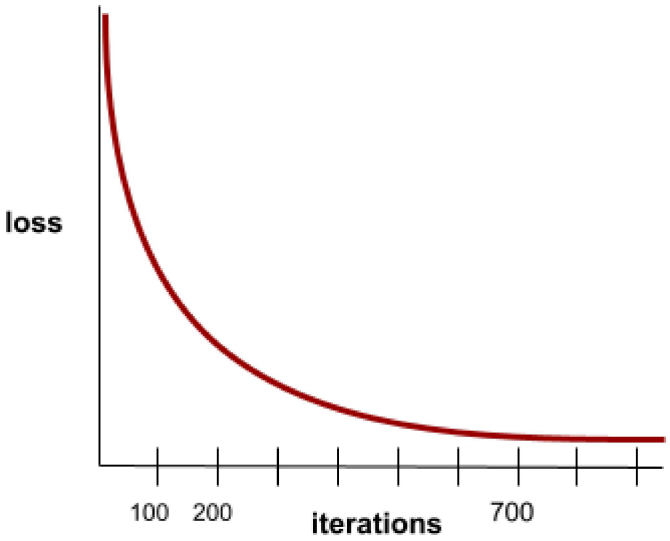
Contrast with **discrete feature** (#discrete_feature).

convenience sampling

Using a dataset not gathered scientifically in order to run quick experiments. Later on, it's essential to switch to a scientifically gathered dataset.

convergence

A state reached when **loss** (#loss) values change very little or not at all with each **iteration** (#iteration). For example, the following **loss curve** (#loss_curve) suggests convergence at around 700 iterations:



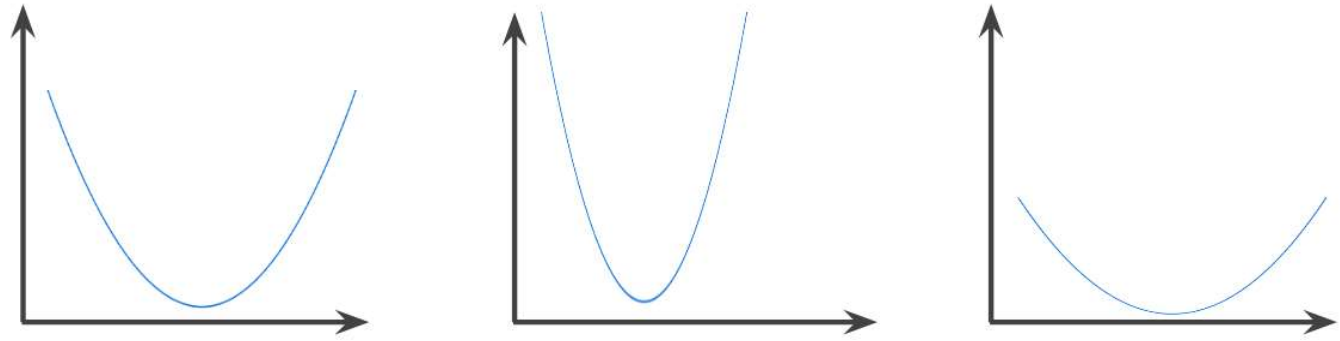
A model **converges** when additional training will not improve the model.

In **deep learning** (#deep_model), loss values sometimes stay constant or nearly so for many iterations before finally descending. During a long period of constant loss values, you may temporarily get a false sense of convergence.

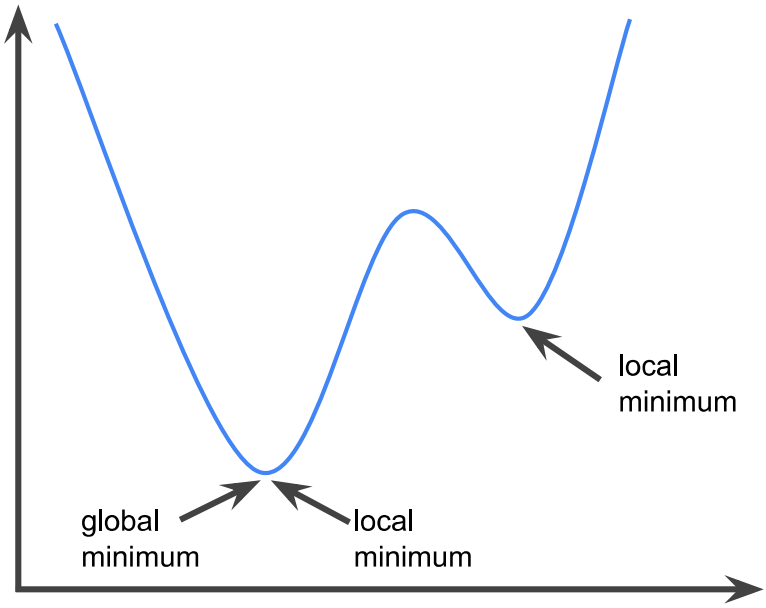
See also **early stopping** (#early_stopping).

convex function

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



In 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.

Click the icon for a deeper look at the math.

A lot of the common loss functions (#loss-function), including the following, are convex functions:

- **L_2 loss** (#L2_loss)
- **Log Loss** (#Log_Loss)
- **L_1 regularization** (#L1_regularization)

- **L_2 regularization** (#L2_regularization)

Many variations of **gradient descent** (#gradient_descent) are guaranteed to find a point close to the minimum of a strictly convex function. Similarly, many variations of **stochastic gradient descent** (#SGD) 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, L_2 loss + L_1 regularization) is a convex function.

Deep models (#deep_model) are never convex functions. Remarkably, algorithms designed for convex optimization (#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

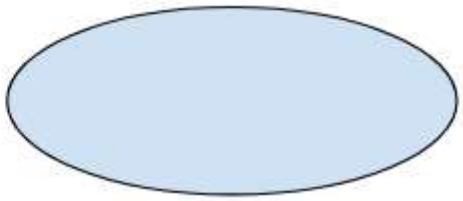
The process of using mathematical techniques such as **gradient descent** (#gradient_descent) to find the minimum of a **convex function** (#convex_function). A great deal of research in machine learning has focused on formulating various problems as convex optimization problems and in solving those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

For complete details, see Boyd and Vandenberghe, Convex Optimization (https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf).

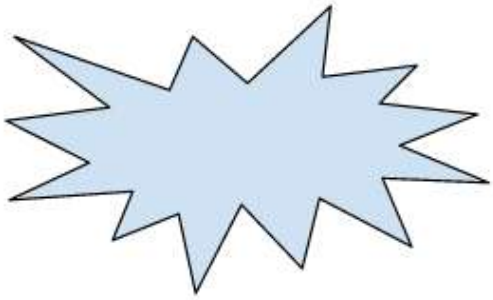
---

convex set

A subset of Euclidean space such that a line drawn between any two points in the subset remains completely within the subset. For instance, the following two shapes are convex sets:



In contrast, the following two shapes are not convex sets:



convolution

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

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

Without convolutions, a machine learning algorithm would have to learn a separate weight for every cell in a large **tensor** (#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** (#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** (#convolutional_operation). (The other actor is a slice of an input matrix.) A convolutional filter is a matrix having the same **rank** (#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** (#training) the ideal values.

convolutional layer

A layer of a **deep neural network** (#deep_model) in which a **convolutional filter** (#convolutional_filter) passes along an input matrix. For example, consider the following 3x3 **convolutional filter** (#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** (#neural_network) in which at least one layer is a **convolutional layer** (#convolutional_layer). A typical convolutional neural network consists of some combination of the following layers:

- **convolutional layers** (#convolutional_layer)
- **pooling layers** (#pooling)
- **dense layers** (#dense_layer)

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** (#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:

128	97	x	1	0	=>	128	0	=	128+22=150
35	22		0	1		0	22		

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

cost

Synonym for **loss** (#loss).

co-training

A **semi-supervised learning** (#semi-supervised_learning) approach particularly useful when all of the following conditions are true:

- The ratio of **unlabeled examples** (#unlabeled_example) to **labeled examples** (#labeled_example) in the dataset is high.
- This is a classification problem (**binary** (#binary_classification) or **multi-class** (#multi-class)).
- The **dataset** (#dataset) contains two different sets of predictive features that are independent of each other and complementary.

Co-training essentially amplifies independent signals into a stronger signal. For instance, consider a **classification model** (#classification_model) that categorizes individual used cars as either *Good* or *Bad*. One set of predictive features might focus on aggregate characteristics such as the year, make, and model of the car; another set of predictive features might focus on the previous owner's driving record and the car's maintenance history.

The seminal paper on co-training is Combining Labeled and Unlabeled Data with Co-Training (https://www.cs.cmu.edu/%7Eavrim/Papers/cotrain.pdf) by Blum and Mitchell.

counterfactual fairness

A **fairness metric** (#fairness_metric) that checks whether a classifier produces the same result for one individual as it does for another individual who is identical to the first, except with respect to one or more **sensitive attributes** (#sensitive_attribute). Evaluating a classifier for counterfactual fairness is one method for surfacing potential sources of bias in a model.

See "When Worlds Collide: Integrating Different Counterfactual Assumptions in Fairness" (https://papers.nips.cc/paper/2017/file/1271a7029c9df08643b631b02cf9e116-Paper.pdf) for a more detailed discussion of counterfactual fairness.

coverage bias

See **selection bias** (#selection_bias).

crash blossom

A sentence or phrase with an ambiguous meaning. Crash blossoms present a significant problem in **natural language understanding** (#natural_language_understanding). For example, the headline *Red Tape Holds Up Skyscraper* is a crash blossom because an NLU model could interpret the headline literally or figuratively.

Click the icon for additional notes.

Just to clarify that mysterious headline:

- **Red Tape** could refer to either of the following:
  - An adhesive
  - Excessive bureaucracy
- **Holds Up** could refer to either of the following:
  - Structural support
  - Delays

critic

RL

Synonym for **Deep Q-Network** (#deep_q-network).

cross-entropy

A generalization of **Log Loss** (#Log_Loss) to **multi-class classification problems** (#multi-class). Cross-entropy quantifies the difference between two probability distributions. See also **perplexity** (#perplexity).

cross-validation

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

D

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** (#model). It is also crucial in understanding experiments and debugging problems with the system.

data augmentation

Artificially boosting the range and number of **training** (#training) examples by transforming existing **examples** (#example) to create additional examples. For example, suppose images are one of your **features** (#feature), but your dataset doesn't contain enough image examples for the model to learn useful associations. Ideally, you'd add enough **labeled** (#label) 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.

DataFrame

A popular **pandas** (#pandas) datatype for representing **datasets** (#dataset) in memory.

A DataFrame is analogous to a table or a spreadsheet. Each column of a DataFrame has a name (a header), and each row is identified by a unique number.

Each column in a DataFrame is structured like a 2D array, except that each column can be assigned its own data type.

See also the official **pandas.DataFrame reference page** (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html).

data parallelism

A way of scaling **training** (#training) or **inference** (#inference) that replicates an entire model onto multiple devices and then passes a subset of the input data to each device. Data parallelism

can enable training and inference on very large **batch sizes** (#batch_size); however, data parallelism requires that the model be small enough to fit on all devices.

See also **model parallelism** (#model-parallelism).

data set or dataset

A collection of raw data, commonly (but not exclusively) organized in one of the following formats:

- a spreadsheet
- a file in CSV (comma-separated values) format

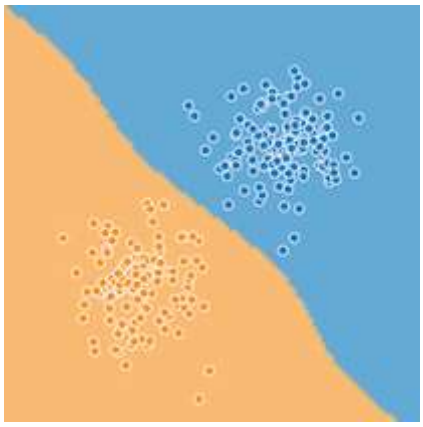
Dataset API (tf.data)

A high-level **TensorFlow** (#TensorFlow) API for reading data and transforming it into a form that a machine learning algorithm requires. A `tf.data.Dataset` object represents a sequence of elements, in which each element contains one or more **Tensors** (#tensor). A `tf.data.Iterator` object provides access to the elements of a `Dataset`.

For details about the Dataset API, see tf.data: Build TensorFlow input pipelines (https://www.tensorflow.org/guide/data) in the *TensorFlow Programmer's Guide*.

decision boundary

The separator between **classes** (#class) learned by a **model** (#model) in a **binary class** (#binary_classification) or **multi-class classification problems** (#multi-class). 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:



decision forest

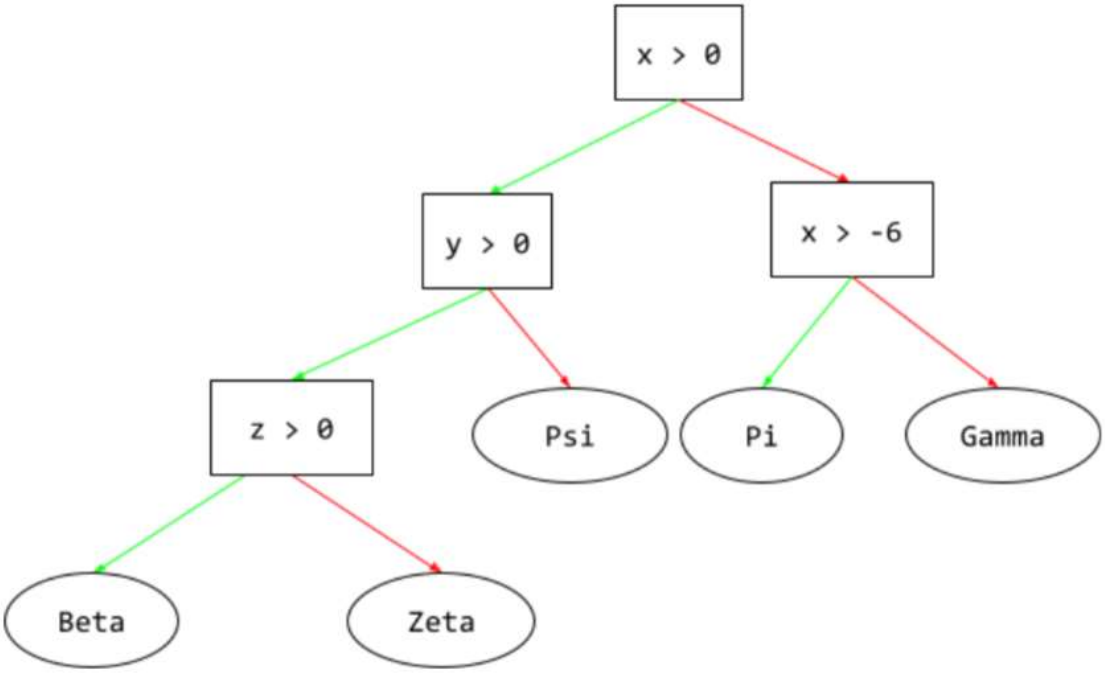
A model created from multiple **decision trees** (#decision-tree). A decision forest makes a prediction by aggregating the predictions of its decision trees. Popular types of decision forests include **random forests** (#random-forest) and **gradient boosted trees** (#gbt).

decision threshold

Synonym for **classification threshold** (#classification_threshold).

decision tree

A supervised learning model composed of a set of **conditions** (#condition) and **leaves** (#leaf) organized hierarchically. For example, the following is a decision tree:



deep model

A **neural network** (#neural_network) containing more than one **hidden layer** (#hidden_layer).

A deep model is also called a **deep neural network**.

Contrast with **wide model** (#wide_model).

decoder

In general, any ML system that converts from a processed, dense, or internal representation to a more raw, sparse, or external representation.

Decoders are often a component of a larger model, where they are frequently paired with an **encoder** (#encoder).

In **sequence-to-sequence tasks** (#sequence-to-sequence-task), a decoder starts with the internal state generated by the encoder to predict the next sequence.

Refer to **Transformer** (#Transformer) for the definition of a decoder within the Transformer architecture.

deep neural network

Synonym for **deep model** (#deep_model).

Deep Q-Network (DQN) RL

In **Q-learning** (#q-learning), a deep **neural network** (#neural_network) that predicts **Q-functions** (#q-function).

Critic is a synonym for Deep Q-Network.

demographic parity ⚖

A **fairness metric** (#fairness_metric) that is satisfied if the results of a model's classification are not dependent on a given **sensitive attribute** (#sensitive_attribute).

For example, if both Lilliputians and Brobdingnagians apply to Glubbdubdrib University, demographic parity is achieved if the percentage of Lilliputians admitted is the same as the

percentage of Brobdingnagians admitted, irrespective of whether one group is on average more qualified than the other.

Contrast with **equalized odds** (#equalized_odds) and **equality of opportunity** (#equality_of_opportunity), which permit classification results in aggregate to depend on sensitive attributes, but do not permit classification results for certain specified ground-truth labels to depend on sensitive attributes. See "Attacking discrimination with smarter machine learning" (http://research.google.com/bigpicture/attacking-discrimination-in-ml/) for a visualization exploring the tradeoffs when optimizing for demographic parity.

denoising

A common approach to **self-supervised learning** (#self-supervised-learning) in which:

1. **Noise** (#noise) is artificially added to the dataset.
2. The **model** (#model) tries to remove the noise.

Denoising enables learning from **unlabeled examples** (#unlabeled_example). The original **dataset** (#dataset) serves as the target or **label** (#label) and the noisy data as the input.

Some **masked language models** (#masked-language-model) use denoising as follows:

1. Noise is artificially added to an unlabeled sentence by masking some of the tokens.
2. The model tries to predict the original tokens.

dense feature

A **feature** (#feature) in which most or all values are nonzero, typically a **Tensor** (#tensor) of floating-point values. For example, the following 10-element Tensor is dense because 9 of its values are nonzero:

8	3	7	5	2	4	0	4	9	6

Contrast with **sparse feature** (#sparse_features).

dense layer

Synonym for **fully connected layer** (#fully_connected_layer).

depth

The sum of the following in a **neural network** (#neural_network):

- the number of **hidden layers** (#hidden_layer)
- the number of **output layers** (#output_layer), which is typically 1
- the number of any **embedding layers** (#embedding_layer)

For example, a neural network with five hidden layers and one output layer has a depth of 6.

Notice that the **input layer** (#input_layer) does not influence depth.

depthwise separable convolutional neural network (sepC🖼️N)

A **convolutional neural network** (#convolutional_neural_network) architecture based on Inception (https://github.com/tensorflow/tpu/tree/master/models/experimental/inception), but where Inception modules are replaced with depthwise separable convolutions. Also known as Xception.

A depthwise separable convolution (also abbreviated as separable convolution) factors a standard 3-D convolution into two separate convolution operations that are more computationally efficient: first, a depthwise convolution, with a depth of 1 ($n \times n \times 1$), and then second, a pointwise convolution, with length and width of 1 ($1 \times 1 \times n$).

To learn more, see Xception: Deep Learning with Depthwise Separable Convolutions (https://arxiv.org/pdf/1610.02357.pdf).

derived label

Synonym for **proxy label** (#proxy_labels).

device

A category of hardware that can run a TensorFlow session, including CPUs, GPUs, and **TPUs** (#TPU).

dimension reduction

Decreasing the number of dimensions used to represent a particular feature in a feature vector, typically by converting to an **embedding vector** (#embedding_vector).

dimensions

Overloaded term having any of the following definitions:

- The number of levels of coordinates in a **Tensor** (#tensor). For example:
  - A scalar has zero dimensions; for example, `["Hello"]`.
  - A vector has one dimension; for example, `[3, 5, 7, 11]`.
  - A matrix has two dimensions; for example, `[[2, 4, 18], [5, 7, 14]]`.

  You can uniquely specify a particular cell in a one-dimensional vector with one coordinate; you need two coordinates to uniquely specify a particular cell in a two-dimensional matrix.
- The number of entries in a **feature vector** (#feature_vector).
- The number of elements in an **embedding layer** (#embedding_layer).

discrete feature

A **feature** (#feature) with a finite set of possible values. For example, a feature whose values may only be *animal*, *vegetable*, or *mineral* is a discrete (or categorical) feature.

Contrast with **continuous feature** (#continuous_feature).

discriminative model

A **model** (#model) that predicts **labels** (#label) from a set of one or more **features** (#feature). More formally, discriminative models define the conditional probability of an output given the features and **weights** (#weight); that is:

```
p(output | features, weights)
```

For example, a model that predicts whether an email is spam from features and weights is a discriminative model.

The vast majority of supervised learning models, including classification and regression models, are discriminative models.

Contrast with **generative model** (#generative_model).

discriminator

A system that determines whether **examples** (#example) are real or fake.

Alternatively, the subsystem within a **generative adversarial network** (#generative_adversarial_network) that determines whether the examples created by the **generator** (#generator) are real or fake.

disparate impact

Making decisions about people that impact different population subgroups disproportionately. This usually refers to situations where an algorithmic decision-making process harms or benefits some subgroups more than others.

For example, suppose an algorithm that determines a Lilliputian's eligibility for a miniature-home loan is more likely to classify them as “ineligible” if their mailing address contains a certain postal code. If Big-Endian Lilliputians are more likely to have mailing addresses with this postal code than Little-Endian Lilliputians, then this algorithm may result in disparate impact.

Contrast with **disparate treatment** (#disparate_treatment), which focuses on disparities that result when subgroup characteristics are explicit inputs to an algorithmic decision-making process.

disparate treatment

Factoring subjects' **sensitive attributes** (#sensitive_attribute) into an algorithmic decision-making process such that different subgroups of people are treated differently.

For example, consider an algorithm that determines Lilliputians' eligibility for a miniature-home loan based on the data they provide in their loan application. If the algorithm uses a Lilliputian's affiliation as Big-Endian or Little-Endian as an input, it is enacting disparate treatment along that dimension.

Contrast with **disparate impact** (#disparate_impact), which focuses on disparities in the societal impacts of algorithmic decisions on subgroups, irrespective of whether those subgroups are inputs to the model.

Warning: Because sensitive attributes are almost always correlated with other features the data may have, explicitly removing sensitive attribute information does not guarantee that subgroups will be treated equally. For example, removing sensitive demographic attributes from a training data set that still includes postal code as a feature may address disparate treatment of subgroups, but there still might be disparate impact upon these groups because postal code might serve as a **proxy** (#proxy_sensitive_attributes) for other demographic information.

divisive clustering

See **hierarchical clustering** (#hierarchical_clustering).

downsampling

Overloaded term that can mean either of the following:

- Reducing the amount of information in a **feature** (#feature) in order to **train** (#training) 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** (#class) examples in order to improve model training on under-represented classes. For example, in a **class-imbalanced dataset** (#class_imbalanced_data_set), models tend to learn a lot about the **majority class** (#majority_class) and not enough about the **minority class** (#minority_class). Downsampling helps balance the amount of training on the majority and minority classes.

DQN

RL

Abbreviation for **Deep Q-Network** (#deep_q-network).

dropout regularization

A form of **regularization** (#regularization) useful in training **neural networks** (#neural_network). Dropout regularization removes 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** (#ensemble) of smaller networks. For full details, see Dropout: A Simple Way to Prevent Neural Networks from Overfitting (http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf).

dynamic

Something done frequently or continuously. The terms **dynamic** and **online** are synonyms in machine learning. The following are common uses of **dynamic** and **online** in machine learning:

- A **dynamic model** (#dynamic_model) (or **online model**) is a model that is retrained frequently or continuously.
- **Dynamic training** (or **online training**) is the process of training frequently or continuously.
- **Dynamic inference** (or **online inference**) is the process of generating predictions on demand.

dynamic model

A **model** (#model) that is frequently (maybe even continuously) retrained. A dynamic model is a "lifelong learner" that constantly adapts to evolving data. A dynamic model is also known as an **online model**.

Contrast with **static model** (#static-model).

E

eager execution

A TensorFlow programming environment in which **operations** (#Operation) run immediately. In contrast, operations called in **graph execution** (#graph_execution) don't run until they are

explicitly evaluated. Eager execution is an imperative interface (https://wikipedia.org/wiki/Imperative_programming), much like the code in most programming languages. Eager execution programs are generally far easier to debug than graph execution programs.

early stopping

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

Click the icon for additional notes.

Early stopping may seem counterintuitive. After all, telling a model to halt training while the loss is still decreasing may seem like telling a chef to stop cooking before the dessert has fully baked. However, training a model for too long can lead to overfitting (#overfitting). That is, if you train a model too long, the model may fit the training data so closely that the model doesn't make good predictions on new examples.

earth mover's distance (EMD)

A measure of the relative similarity between two documents. The lower the earth mover's distance, the more similar the documents.

edit distance

A measurement of how similar two text strings are to each other. In machine learning, edit distance is useful because it is simple and easy to compute, and an effective way to compare two strings that are known to be similar or to find strings that are similar to a given string.

There are several definitions of edit distance, each using different string operations. For example, the Levenshtein distance (https://wikipedia.org/wiki/Levenshtein_distance) considers the fewest delete, insert, and substitute operations.

For example, the Levenshtein distance between the words "heart" and "darts" is 3 because the following 3 edits are the fewest changes to turn one word into the other:

1. heart → deart (substitute “h” with "d")
2. deart → dart (delete "e")
3. dart → darts (insert "s")

Einsum notation

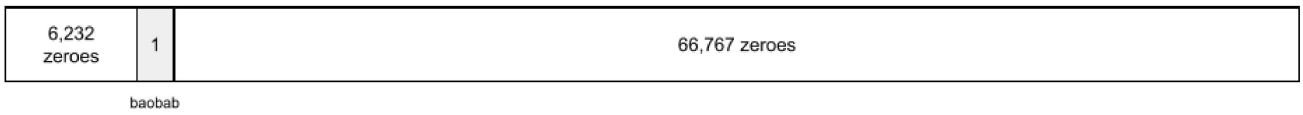
An efficient notation for describing how two **tensors** (#tensor) are to be combined. The tensors are combined by multiplying the elements of one tensor by the elements of the other tensor and then summing the products. Einsum notation uses symbols to identify the axes of each tensor, and those same symbols are rearranged to specify the shape of the new resulting tensor.

NumPy (#numpy) provides a common Einsum implementation.

embedding layer

A special **hidden layer** (#hidden_layer) that trains on a high-dimensional **categorical** (#categorical_data) feature to gradually learn a lower dimension embedding vector. An embedding layer enables a neural network to train far more efficiently than training just on the high-dimensional categorical feature.

For example, Earth currently supports about 73,000 tree species. Suppose tree species is a **feature** (#feature) in your model, so your model's input layer includes a **one-hot vector** (#one-hot_encoding) 73,000 elements long. For example, perhaps **baobab** would be represented something like this:



A 73,000-element array is very long. If you don't add an embedding layer to the model, training is going to be very time consuming due to multiplying 72,999 zeros. Perhaps you pick the embedding layer to consist of 12 dimensions. Consequently, the embedding layer will gradually learn a new embedding vector for each tree species.

In certain situations, **hashing** (#hashing) is a reasonable alternative to an embedding layer.

embedding space

The d-dimensional vector space that features from a higher-dimensional vector space are mapped to. Ideally, the embedding space contains a structure that yields meaningful mathematical results; for example, in an ideal embedding space, addition and subtraction of embeddings can solve word analogy tasks.

The dot product (https://wikipedia.org/wiki/Dot_product) of two embeddings is a measure of their similarity.

embedding vector

Broadly speaking, an array of floating-point numbers taken from **any hidden layer** (#hidden_layer) that describe the inputs to that hidden layer. Often, an embedding vector is the array of floating-point numbers trained in an embedding layer. For example, suppose an embedding layer must learn an embedding vector for each of the 73,000 tree species on Earth. Perhaps the following array is the embedding vector for a baobab tree:

0.819245	0.539102	0.391284	0.181923	0.519247	0.912043	0.291529	0.519284	0.710153	0.109152	0.851024	0.381528

An embedding vector is not a bunch of random numbers. An embedding layer determines these values through training, similar to the way a neural network learns other weights during training. Each element of the array is a rating along some characteristic of a tree species. Which element represents which tree species' characteristic? That's very hard for humans to determine.

The mathematically remarkable part of an embedding vector is that similar items have similar sets of floating-point numbers. For example, similar tree species have a more similar set of floating-point numbers than dissimilar tree species. Redwoods and sequoias are related tree species, so they'll have a more similar set of floating-pointing numbers than redwoods and coconut palms. The numbers in the embedding vector will change each time you retrain the model, even if you retrain the model with identical input.

empirical risk minimization (ERM)

Choosing the function that minimizes loss on the training set. Contrast with **structural risk minimization** (#SRM).

encoder

In general, any ML system that converts from a raw, sparse, or external representation into a more processed, denser, or more internal representation.

Encoders are often a component of a larger model, where they are frequently paired with a **decoder** (#decoder). Some **Transformers** (#Transformer) pair encoders with decoders, though other Transformers use only the encoder or only the decoder.

Some systems use the encoder's output as the input to a classification or regression network.

In **sequence-to-sequence tasks** (#sequence-to-sequence-task), an encoder takes an input sequence and returns an internal state (a vector). Then, the **decoder** (#decoder) uses that internal state to predict the next sequence.

Refer to **Transformer** (#Transformer) for the definition of an encoder in the Transformer architecture.

ensemble

A collection of **models** (#model) trained independently whose predictions are averaged or aggregated. In many cases, an ensemble produces better predictions than a single model. For example, a **random forest** (#random-forest) is an ensemble built from multiple **decision trees** (#decision-tree). Note that not all **decision forests** (#decision-forest) are ensembles.

entropy

In information theory (https://wikipedia.org/wiki/Information_theory), a description of how unpredictable a probability distribution is. Alternatively, entropy is also defined as how much information each **example** (#example) contains. A distribution has the highest possible entropy when all values of a random variable are equally likely.

The entropy of a set with two possible values "0" and "1" (for example, the labels in a **binary classification** (#binary_classification) problem) has the following formula:

```
H = -p log p - q log q = -p log p - (1-p) * log (1-p)
```

where:

- `H` is the entropy.
- `p` is the fraction of "1" examples.
- `q` is the fraction of "0" examples. Note that q = (1 - p)
- `log` is generally $\log_2$. In this case, the entropy unit is a bit.

For example, suppose the following:

- 100 examples contain the value "1"
- 300 examples contain the value "0"

Therefore, the entropy value is:

- `p = 0.25`
- `q = 0.75`
- `H = (-0.25)log2(0.25) - (0.75)log2(0.75) = 0.81` bits per example

A set that is perfectly balanced (for example, 200 "0"s and 200 "1"s) would have an entropy of 1.0 bit per example. As a set becomes more **imbalanced** (#class_imbalanced_data_set), its entropy moves towards 0.0.

In **decision trees** (#decision-tree), entropy helps formulate **information gain** (#information-gain) to help the **splitter** (#splitter) select the **conditions** (#condition) during the growth of a classification decision tree.

Compare entropy with:

- **gini impurity** (#gini-impurity)
- **cross-entropy** (#cross-entropy) loss function

Entropy is often called Shannon's entropy.

environment RL

In reinforcement learning, the world that contains the **agent** (#agent) and allows the agent to observe that world's **state** (#state). For example, the represented world can be a game like chess, or a physical world like a maze. When the agent applies an **action** (#action) to the environment, then the environment transitions between states.

episode RL

In reinforcement learning, each of the repeated attempts by the **agent** (#agent) to learn an **environment** (#environment).

epoch

A full training pass over the entire **training set** (#training_set) such that each **example** (#example) has been processed once.

An epoch represents `N`/**batch size** (#batch_size) training **iterations** (#iteration), where `N` is the total number of examples.

For instance, suppose the following:

- The dataset consists of 1,000 examples.
- The batch size is 50 examples.

Therefore, a single epoch requires 20 iterations:

```
1 epoch = (N/batch size) = (1,000 / 50) = 20 iterations
```

epsilon greedy policy RL

In reinforcement learning, a **policy** (#policy) that either follows a **random policy** (#random_policy) with epsilon probability or a **greedy policy** (#greedy_policy) otherwise. For example, if epsilon is 0.9, then the policy follows a random policy 90% of the time and a greedy policy 10% of the time.

Over successive episodes, the algorithm reduces epsilon's value in order to shift from following a random policy to following a greedy policy. By shifting the policy, the agent first randomly explores the environment and then greedily exploits the results of random exploration.

equality of opportunity ⚖️

A **fairness metric** (#fairness_metric) that checks whether, for a preferred **label** (#label) (one that confers an advantage or benefit to a person) and a given **attribute** (#attribute), a classifier predicts that preferred label equally well for all values of that attribute. In other words, equality of opportunity measures whether the people who should qualify for an opportunity are equally likely to do so regardless of their group membership.

For example, suppose Glubbdubdrib University admits both Lilliputians and Brobdingnagians to a rigorous mathematics program. Lilliputians' secondary schools offer a robust curriculum of math classes, and the vast majority of students are qualified for the university program. Brobdingnagians' secondary schools don't offer math classes at all, and as a result, far fewer of their students are qualified. Equality of opportunity is satisfied for the preferred label of "admitted" with respect to nationality (Lilliputian or Brobdingnagian) if qualified students are equally likely to be admitted irrespective of whether they're a Lilliputian or a Brobdingnagian.

For example, let's say 100 Lilliputians and 100 Brobdingnagians apply to Glubbdubdrib University, and admissions decisions are made as follows:

Table 1. Lilliputian applicants (90% are qualified)

	Qualified	Unqualified
Admitted	45	3
Rejected	45	7
Total	90	10

Percentage of qualified students admitted: 45/90 = 50%
Percentage of unqualified students rejected: 7/10 = 70%
Total percentage of Lilliputian students admitted: (45+3)/100 = 48%

Table 2. Brobdingnagian applicants (10% are qualified):

	Qualified	Unqualified
Admitted	5	9
Rejected	5	81
Total	10	90

Percentage of qualified students admitted: 5/10 = 50%
Percentage of unqualified students rejected: 81/90 = 90%
Total percentage of Brobdingnagian students admitted: (5+9)/100 = 14%

The preceding examples satisfy equality of opportunity for acceptance of qualified students because qualified Lilliputians and Brobdingnagians both have a 50% chance of being admitted.

Note: While equality of opportunity is satisfied, the following two fairness metrics are not satisfied:

- **demographic parity** (#demographic_parity): Lilliputians and Brobdingnagians are admitted to the university at different rates; 48% of Lilliputians students are admitted, but only 14% of Brobdingnagian students are admitted.
- **equalized odds** (#equalized_odds): While qualified Lilliputian and Brobdingnagian students both have the same chance of being admitted, the additional constraint that unqualified Lilliputians and Brobdingnagians both have the same chance of being rejected is not satisfied. Unqualified Lilliputians have a 70% rejection rate, whereas unqualified Brobdingnagians have a 90% rejection rate.

See "Equality of Opportunity in Supervised Learning" (https://arxiv.org/pdf/1610.02413.pdf) for a more detailed discussion of equality of opportunity. Also see "Attacking discrimination with smarter machine learning" (http://research.google.com/bigpicture/attacking-discrimination-in-ml/) for a visualization exploring the tradeoffs when optimizing for equality of opportunity.

equalized odds

A **fairness metric** (#fairness_metric) that checks if, for any particular label and attribute, a classifier predicts that label equally well for all values of that attribute.

For example, suppose Glubbdubdrib University admits both Lilliputians and Brobdingnagians to a rigorous mathematics program. Lilliputians' secondary schools offer a robust curriculum of math classes, and the vast majority of students are qualified for the university program. Brobdingnagians' secondary schools don't offer math classes at all, and as a result, far fewer of their students are qualified. Equalized odds is satisfied provided that no matter whether an applicant is a Lilliputian or a Brobdingnagian, if they are qualified, they are equally as likely to get admitted to the program, and if they are not qualified, they are equally as likely to get rejected.

Let's say 100 Lilliputians and 100 Brobdingnagians apply to Glubbdubdrib University, and admissions decisions are made as follows:

Table 3. Lilliputian applicants (90% are qualified)

	Qualified	Unqualified
Admitted	45	2
Rejected	45	8
Total	90	10

Percentage of qualified students admitted: 45/90 = 50%
Percentage of unqualified students rejected: 8/10 = 80%
Total percentage of Lilliputian students admitted: (45+2)/100 = 47%

Table 4. Brobdingnagian applicants (10% are qualified):

	Qualified	Unqualified
Admitted	5	18
Rejected	5	72
Total	10	90

Percentage of qualified students admitted: 5/10 = 50%
Percentage of unqualified students rejected: 72/90 = 80%
Total percentage of Brobdingnagian students admitted: (5+18)/100 = 23%

Equalized odds is satisfied because qualified Lilliputian and Brobdingnagian students both have a 50% chance of being admitted, and unqualified Lilliputian and Brobdingnagian have an 80% chance of being rejected.

Note: While equalized odds is satisfied here, **demographic parity** (#demographic_parity) is *not satisfied*. Lilliputian and Brobdingnagian students are admitted to Glubbdubdrib University at different rates; 47% of Lilliputian students are admitted, and 23% of Brobdingnagian students are admitted.

Equalized odds is formally defined in "Equality of Opportunity in Supervised Learning" (https://arxiv.org/pdf/1610.02413.pdf) as follows: "predictor $\hat{Y}$ satisfies equalized odds with

respect to protected attribute A and outcome Y if Ŷ and A are independent, conditional on Y."

Note: Contrast equalized odds with the more relaxed **equality of opportunity** (#equality_of_opportunity) metric.

Estimator

A deprecated TensorFlow API. Use **tf.keras** (#tf.keras) instead of Estimators.

example

The values of one row of **features** (#feature) and possibly a **label** (#label). Examples in **supervised learning** (#supervised_machine_learning) fall into two general categories:

- A **labeled example** (#labeled_example) consists of one or more features and a label. Labeled examples are used during training.
- An **unlabeled example** (#unlabeled_example) consists of one or more features but no label. Unlabeled examples are used during inference.

For instance, suppose you are training a model to determine the influence of weather conditions on student test scores. Here are three labeled examples:

Features			Label
Temperature	**Humidity**	**Pressure**	**Test score**
15	47	998	Good
19	34	1020	Excellent

18	92	1012	Poor

Here are three unlabeled examples:

Temperature	Humidity	Pressure
12	62	1014
21	47	1017
19	41	1021

The row of a **dataset** (#dataset) is typically the raw source for an example. That is, an example typically consists of a subset of the columns in the dataset. Furthermore, the features in an example can also include **synthetic features** (#synthetic_feature), such as **feature crosses** (#feature_cross).

experience replay RL

In reinforcement learning, a **DQN** (#deep_q-network) technique used to reduce temporal correlations in training data. The **agent** (#agent) stores state transitions in a **replay buffer** (#replay_buffer), and then samples transitions from the replay buffer to create training data.

experimenter's bias 

See **confirmation bias** (#confirmation_bias).

exploding gradient problem

The tendency for **gradients** (#gradient) in **deep neural networks** (#deep_neural_network) (especially **recurrent neural networks** (#recurrent_neural_network)) to become surprisingly steep (high). Steep gradients often cause very large updates to the **weights** (#weight) of each **node** (#node) in a deep neural network.

Models suffering from the exploding gradient problem become difficult or impossible to train. **Gradient clipping** (#gradient_clipping) can mitigate this problem.

Compare to **vanishing gradient problem** (#vanishing_gradient_problem).

F

F_1

A "roll-up" **binary classification** (#binary-classification) metric that relies on both **precision** (#precision) and **recall** (#recall). Here is the formula:

$$F_1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

For example, given the following:

- precision = 0.6
- recall = 0.4

$$F_1 = \frac{2 * 0.6 * 0.4}{0.6 + 0.4} = 0.48$$

When precision and recall are fairly similar (as in the preceding example), F_1 is close to their mean. When precision and recall differ significantly, F_1 is closer to the lower value. For example:

- precision = 0.9
- recall = 0.1

$$F_1 = \frac{2 * 0.9 * 0.1}{0.9 + 0.1} = 0.18$$

fairness constraint

Applying a constraint to an algorithm to ensure one or more definitions of fairness are satisfied. Examples of fairness constraints include:

- **Post-processing** (#post-processing) your model's output.
- Altering the **loss function** (#loss) to incorporate a penalty for violating a **fairness metric** (#fairness_metric).
- Directly adding a mathematical constraint to an optimization problem.

fairness metric

A mathematical definition of “fairness” that is measurable. Some commonly used fairness metrics include:

- **equalized odds** (#equalized_odds)
- **predictive parity** (#predictive_parity)
- **counterfactual fairness** (#counterfactual_fairness)
- **demographic parity** (#demographic_parity)

Many fairness metrics are mutually exclusive; see **incompatibility of fairness metrics** (#incompatibility_of_fairness_metrics).

false negative (FN)

An example in which the model mistakenly predicts the **negative class** (#negative_class). For example, the model predicts that a particular email message is *not spam* (the negative class), but that email message *actually is spam*.

false negative rate

The proportion of actual positive examples for which the model mistakenly predicted the negative class. The following formula calculates the false negative rate:

$$\text{false negative rate} = \frac{\text{false negatives}}{\text{false negatives} + \text{true positives}}$$

false positive (FP)

An example in which the model mistakenly predicts the **positive class** (#positive_class). For example, the model predicts that a particular email message is *spam* (the positive class), but that email message is *actually not spam*.

false positive rate (FPR)

The proportion of actual negative examples for which the model mistakenly predicted the positive class. The following formula calculates the false positive rate:

$$\text{false positive rate} = \frac{\text{false positives}}{\text{false positives} + \text{true negatives}}$$

The false positive rate is the x-axis in an **ROC curve** (#ROC).

feature

An input variable to a machine learning model. An **example** (#example) consists of one or more features. For instance, suppose you are training a model to determine the influence of weather conditions on student test scores. The following table shows three examples, each of which contains three features and one label:

Features			Label
Temperature	Humidity	Pressure	Test score
15	47	998	92
19	34	1020	84
18	92	1012	87

Contrast with **label** (#label).

feature cross

A **synthetic feature** (#synthetic_feature) formed by "crossing" **categorical** (#categorical_data) or **bucketed** (#bucketing) features.

For example, consider a "mood forecasting" model that represents temperature in one of the following four buckets:

- `freezing`
- `chilly`
- `temperate`
- `warm`

And represents wind speed in one of the following three buckets:

- `still`
- `light`
- `windy`

Without feature crosses, the linear model trains independently on each of the preceding seven various buckets. So, the model trains on, for instance, `freezing` independently of the training on, for instance, `windy`.

Alternatively, you could create a feature cross of temperature and wind speed. This synthetic feature would have the following 12 possible values:

- `freezing-still`
- `freezing-light`
- `freezing-windy`
- `chilly-still`
- `chilly-light`
- `chilly-windy`
- `temperate-still`
- `temperate-light`
- `temperate-windy`

- `warm-still`
- `warm-light`
- `warm-windy`

Thanks to feature crosses, the model can learn mood differences between a `freezing-windy` day and a `freezing-still` day.

If you create a synthetic feature from two features that each have a lot of different buckets, the resulting feature cross will have a huge number of possible combinations. For example, if one feature has 1,000 buckets and the other feature has 2,000 buckets, the resulting feature cross has 2,000,000 buckets.

Formally, a cross is a Cartesian product (https://wikipedia.org/wiki/Cartesian_product).

Feature crosses are mostly used with linear models and are rarely used with neural networks.

feature engineering

A process that involves the following steps:

1. Determining which **features** (#feature) might be useful in training a model.
2. Converting raw data from the dataset into efficient versions of those features.

For example, you might determine that `temperature` might be a useful feature. Then, you might experiment with **bucketing** (#bucketing) to optimize what the model can learn from different `temperature` ranges.

Feature engineering is sometimes called **feature extraction**.

Click the icon for additional notes about TensorFlow.

In TensorFlow, feature engineering often means converting raw log file entries to **tf.Example** (#tf.Example) protocol buffers. See also tf.Transform (https://github.com/tensorflow/transform).

feature extraction

Overloaded term having either of the following definitions:

- Retrieving intermediate feature representations calculated by an **unsupervised** (#unsupervised_machine_learning) or pretrained model (for example, **hidden layer** (#hidden_layer) values in a **neural network** (#neural_network)) for use in another model as input.
- Synonym for **feature engineering** (#feature_engineering).

feature importances

Synonym for **variable importances** (#variable-importances).

feature set

The group of **features** (#feature) your machine learning **model** (#model) trains on. For example, postal code, property size, and property condition might comprise a simple feature set for a model that predicts housing prices.

feature spec

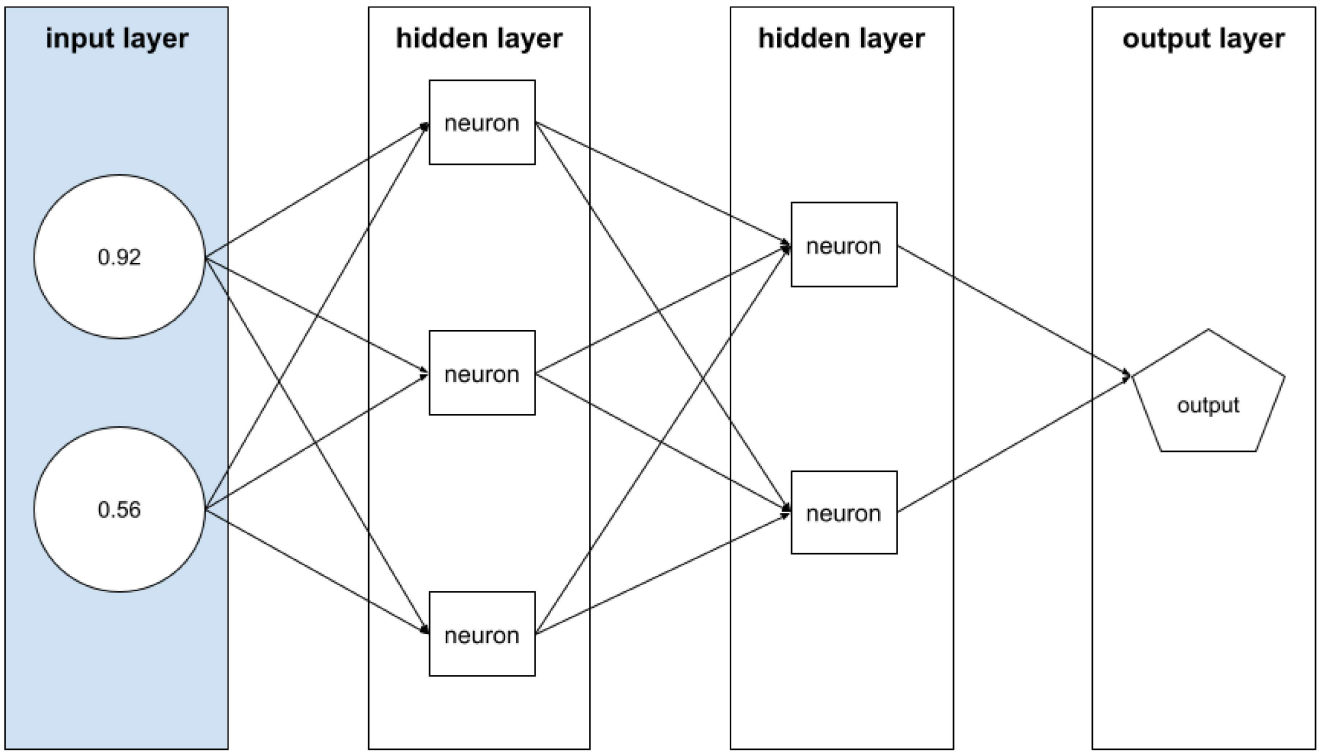
Describes the information required to extract **features** (#feature) data from the **tf.Example** (#tf.Example) protocol buffer. Because the tf.Example protocol buffer is just a container for data, you must specify the following:

- the data to extract (that is, the keys for the features)
- the data type (for example, float or int)
- The length (fixed or variable)

feature vector

The array of **feature** (#feature) values comprising an **example** (#example). The feature vector is input during **training** (#training) and during **inference** (#inference). For example, the feature vector for a model with two discrete features might be:

```
[0.92, 0.56]
```



Each example supplies different values for the feature vector, so the feature vector for the next example could be something like:

```
[0.73, 0.49]
```

Feature engineering (#feature_engineering) determines how to represent features in the feature vector. For example, a binary categorical feature with five possible values might be represented with **one-hot encoding** (#one-hot_encoding). In this case, the portion of the feature vector for a particular example would consist of four zeroes and a single 1.0 in the third position, as follows:

```
[0.0, 0.0, 1.0, 0.0, 0.0]
```

As another example, suppose your model consists of three features:

- a binary categorical feature with *five* possible values represented with one-hot encoding; for example: `[0.0, 1.0, 0.0, 0.0, 0.0]`
- another binary categorical feature with *three* possible values represented with one-hot encoding; for example: `[0.0, 0.0, 1.0]`
- a floating-point feature; for example: `8.3`.

In this case, the feature vector for each example would be represented by *nine* values. Given the example values in the preceding list, the feature vector would be:

```
0.0
1.0
0.0
0.0
0.0
0.0
0.0
1.0
8.3
```

federated learning

A distributed machine learning approach that **trains** (#training) machine learning **models** (#model) using decentralized **examples** (#example) residing on devices such as smartphones. In federated learning, a subset of devices downloads the current model from a central coordinating server. The devices use the examples stored on the devices to make improvements to the model. The devices then upload the model improvements (but not the training examples) to the coordinating server, where they are aggregated with other updates to yield an improved global model. After the aggregation, the model updates computed by devices are no longer needed, and can be discarded.

Since the training examples are never uploaded, federated learning follows the privacy principles of focused data collection and data minimization.

For more information about federated learning, see this tutorial (https://federated.withgoogle.com).

feedback loop

In machine learning, a situation in which a model's predictions influence the training data for the same model or another model. For example, a model that recommends movies will influence the movies that people see, which will then influence subsequent movie recommendation models.

feedforward neural network (FFN)

A neural network without cyclic or recursive connections. For example, traditional **deep neural networks** (#deep_neural_network) are feedforward neural networks. Contrast with **recurrent**

neural networks (#recurrent_neural_network), which are cyclic.

few-shot learning

A machine learning approach, often used for object classification, designed to train effective classifiers from only a small number of training examples.

See also **one-shot learning** (#one-shot_learning) and **zero-shot learning** (#zero-shot-learning).

Fiddle

A Python-first **configuration** (#configuration) library that sets the values of functions and classes without invasive code or infrastructure. In the case of **Pax** (#pax)—and other ML codebases—these functions and classes represent **models** (#model) and **training** (#training) **hyperparameters** (#hyperparameter).

Fiddle (https://github.com/google/fiddle) assumes that machine learning codebases are typically divided into:

- Library code, which defines the layers and optimizers.
- Dataset "glue" code, which calls the libraries and wires everything together.

Fiddle captures the call structure of the glue code in an unevaluated and mutable form.

fine tuning

Performing a secondary optimization to adjust the parameters of an already trained **model** (#model) to fit a new problem. Fine tuning often refers to refitting the weights of a trained **unsupervised** (#unsupervised_machine_learning) model to a **supervised** (#supervised_machine_learning) model.

Flax

A high-performance open-source library (https://github.com/google/flax) for deep learning built on top of **JAX** (#JAX). Flax provides functions for **training** (#training) **neural networks** (#neural-network), as well as methods for evaluating their performance.

Flaxformer

An open-source **Transformer** (#transformer) library (https://github.com/google/flaxformer), built on **Flax** (#flax), designed primarily for natural language processing and multimodal research.

forget gate

The portion of a **Long Short-Term Memory** (#Long_Short-Term_Memory) cell that regulates the flow of information through the cell. Forget gates maintain context by deciding which information to discard from the cell state.

full softmax

Synonym for **softmax** (#softmax).

Contrast with **candidate sampling** (#candidate_sampling).

fully connected layer

A **hidden layer** (#hidden_layer) in which each **node** (#node) is connected to *every* node in the subsequent hidden layer.

A fully connected layer is also known as a **dense layer** (#dense_layer).

G

GAN

Abbreviation for **generative adversarial network** (#generative_adversarial_network).

generalization

A **model's** (#model) ability to make correct predictions on new, previously unseen data. A model that can generalize is the opposite of a model that is **overfitting** (#overfitting).

Click the icon for additional notes.

You train a model on the examples in the training set. Consequently, the model learns the peculiarities of the data in the training set. Generalization essentially asks whether your model can make good predictions on examples that are *not* in the training set.

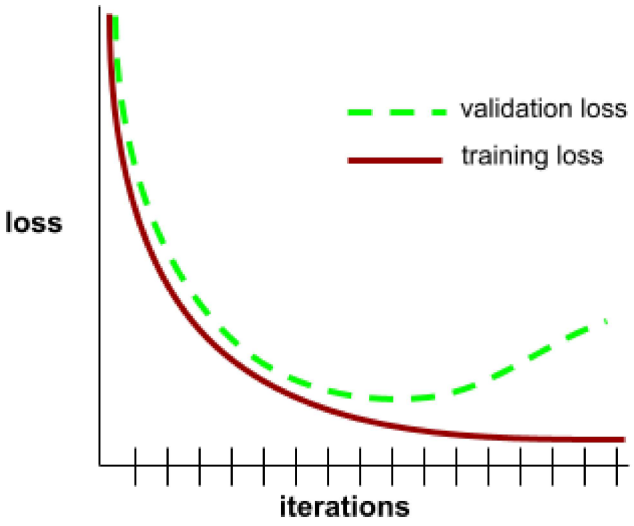
To encourage generalization, **regularization** (#regularization) helps a model train less exactly to the peculiarities of the data in the training set.

---

generalization curve

A plot of both **training loss** (#training-loss) and **validation loss** (#validation-loss) as a function of the number of **iterations** (#iteration).

A generalization curve can help you detect possible **overfitting** (#overfitting). For example, the following generalization curve suggests overfitting because validation loss ultimately becomes significantly higher than training loss.



generalized linear model

A generalization of **least squares regression** (#least_squares_regression) models, which are based on Gaussian noise (https://wikipedia.org/wiki/Gaussian_noise), to other types of models based on other types of noise, such as Poisson noise (https://wikipedia.org/wiki/Shot_noise) or categorical noise. Examples of generalized linear models include:

- **logistic regression** (#logistic_regression)
- multi-class regression
- least squares regression

The parameters of a generalized linear model can be found through **convex optimization** (#convex_optimization).

Generalized linear models exhibit the following properties:

- The average prediction of the optimal least squares regression model is equal to the average label on the training data.
- The average probability predicted by the optimal logistic regression model is equal to the average label on the training data.

The power of a generalized linear model is limited by its features. Unlike a deep model, a generalized linear model cannot "learn new features."

generative adversarial network (GAN)

A system to create new data in which a **generator** (#generator) creates data and a **discriminator** (#discriminator) determines whether that created data is valid or invalid.

generative model

Practically speaking, a model that does either of the following:

- Creates (generates) new examples from the training dataset. For example, a generative model could create poetry after training on a dataset of poems. The **generator** (#generator) part of a **generative adversarial network** (#generative_adversarial_network) falls into this category.
- Determines the probability that a new example comes from the training set, or was created from the same mechanism that created the training set. For example, after training on a dataset consisting of English sentences, a generative model could determine the probability that new input is a valid English sentence.

A generative model can theoretically discern the distribution of examples or particular features in a dataset. That is:

```
p(examples)
```

Unsupervised learning models are generative.

Contrast with **discriminative models** (#discriminative_model).

generator

The subsystem within a **generative adversarial network** (#generative_adversarial_network) that creates new **examples** (#example).

Contrast with **discriminative model** (#discriminative_model).

GPT (Generative Pre-trained Transformer)

A family of **Transformer** (#Transformer)-based **large language models** (#large-language-model) developed by OpenAI (https://openai.com/).

GPT variants can apply to multiple **modalities** (#modality), including:

- image generation (for example, ImageGPT)
- text-to-image generation (for example, DALL-E (https://openai.com/blog/dall-e/)).

gini impurity

♠♠♠

A metric similar to **entropy** (#entropy). **Splitters** (#splitter) use values derived from either gini impurity or entropy to compose **conditions** (#condition) for classification **decision trees** (#decision-tree). **Information gain** (#information-gain) is derived from entropy. There is no universally accepted equivalent term for the metric derived from gini impurity; however, this unnamed metric is just as important as information gain.

Gini impurity is also called **gini index**, or simply **gini**.

Click the icon for mathematical details about gini impurity.

Gini impurity is the probability of misclassifying a new piece of data taken from the same distribution. The gini impurity of a set with two possible values "0" and "1" (for example, the labels in a **binary classification** (#binary_classification) problem) is calculated from the following formula:

$$I = 1 - (p^2 + q^2) = 1 - (p^2 + (1-p)^2)$$

where:

- `I` is the gini impurity.
- `p` is the fraction of "1" examples.
- `q` is the fraction of "0" examples. Note that `q = 1-p`

For example, consider the following dataset:

- 100 labels (0.25 of the dataset) contain the value "1"
- 300 labels (0.75 of the dataset) contain the value "0"

Therefore, the gini impurity is:

- $p = 0.25$
- $q = 0.75$
- $I = 1 - (0.25^2 + 0.75^2) = \mathbf{0.375}$

Consequently, a random label from the same dataset would have a 37.5% chance of being misclassified, and a 62.5% chance of being properly classified.

A perfectly balanced label (for example, 200 "0"s and 200 "1"s) would have a gini impurity of 0.5. A highly **imbalanced** (#class_imbalanced_data_set) label would have a gini impurity close to 0.0.

---

gradient

The vector of **partial derivatives** (#partial_derivative) 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 boosting

A training algorithm where weak models are trained to iteratively improve the quality (reduce the loss) of a strong model. For example, a weak model could be a linear or small decision tree model. The strong model becomes the sum of all the previously trained weak models.

In the simplest form of gradient boosting, at each iteration, a weak model is trained to predict the loss gradient of the strong model. Then, the strong model's output is updated by subtracting the predicted gradient, similar to **gradient descent** (#gradient_descent).

$$F_0 = 0$$

$$F_{i+1} = F_i - \xi f_i$$

where:

- F_0 is the starting strong model.
- F_{i+1} is the next strong model.
- F_i is the current strong model.
- ξ is a value between 0.0 and 1.0 called **shrinkage** (#shrinkage), which is analogous to the **learning rate** (#learning_rate) in gradient descent.
- f_i is the weak model trained to predict the loss gradient of F_i.

Modern variations of gradient boosting also include the second derivative (Hessian) of the loss in their computation.

Decision trees (#decision-tree) are commonly used as weak models in gradient boosting. See **gradient boosted (decision) trees** (#gbt).

gradient boosted (decision) trees (GBT)

A type of **decision forest** (#decision-forest) in which:

- **Training** (#training) relies on **gradient boosting** (#gradient-boosting).
- The weak model is a **decision tree** (#decision-tree).

gradient clipping

A commonly used mechanism to mitigate the **exploding gradient problem** (#exploding_gradient_problem) by artificially limiting (clipping) the maximum value of gradients when using **gradient descent** (#gradient_descent) to **train** (#training) a model.

gradient descent

A mathematical technique to minimize **loss** (#loss). Gradient descent iteratively adjusts **weights** (#weight) and **biases** (#bias), gradually finding the best combination to minimize loss.

Gradient descent is older—much, much older—than machine learning.

graph

In TensorFlow, a computation specification. Nodes in the graph represent operations. Edges are directed and represent passing the result of an operation (a **Tensor** (#tensor)) as an operand to another operation. Use **TensorBoard** (#TensorBoard) to visualize a graph.

graph execution

A TensorFlow programming environment in which the program first constructs a **graph** (#graph) and then executes all or part of that graph. Graph execution is the default execution mode in TensorFlow 1.x.

Contrast with **eager execution** (#eager_execution).

greedy policy RL

In reinforcement learning, a **policy** (#policy) that always chooses the action with the highest expected **return** (#return).

ground truth

Reality.

The thing that actually happened.

For example, consider a **binary classification** (#binary_classification) model that predicts whether a student in their first year of university will graduate within six years. Ground truth for this model is whether or not that student actually graduated within six years.

Click the icon for additional notes.

We assess model quality against ground truth. However, ground truth is not always completely, well, truthful. For example, consider the following examples of potential imperfections in ground truth:

- In the graduation example, are we *certain* that the graduation records for each student are always correct? Is the university's record-keeping flawless?
- Suppose the label is a floating-point value measured by instruments (for instance, barometers). How can we be sure that each instrument is calibrated identically or that each reading was taken under the same circumstances?
- If the label is a matter of human opinion, how can we be sure that each human **rater** (#rater) is evaluating events in the same way? To improve consistency, *expert* human raters sometimes intervene.

group attribution bias

Assuming that what is true for an individual is also true for everyone in that group. The effects of group attribution bias can be exacerbated if a **convenience sampling** (#convenience_sampling) is used for data collection. In a non-representative sample, attributions may be made that do not reflect reality.

See also **out-group homogeneity bias** (#out-group_homogeneity_bias) and **in-group bias** (#in-group_bias).

H

hallucination

The production of plausible-seeming but factually incorrect output by a **generative model** (#generative_model) that purports to be making an assertion about the real world. For example, if a dialog agent claims that Barack Obama died in 1865, the agent is *hallucinating*.

hashing

In machine learning, a mechanism for bucketing **categorical data** (#categorical_data), particularly when the number of categories is large, but the number of categories actually appearing in the dataset is comparatively small.

For example, Earth is home to about 73,000 tree species. You could represent each of the 73,000 tree species in 73,000 separate categorical buckets. Alternatively, if only 200 of those tree species actually appear in a dataset, you could use hashing to divide tree species into perhaps 500 buckets.

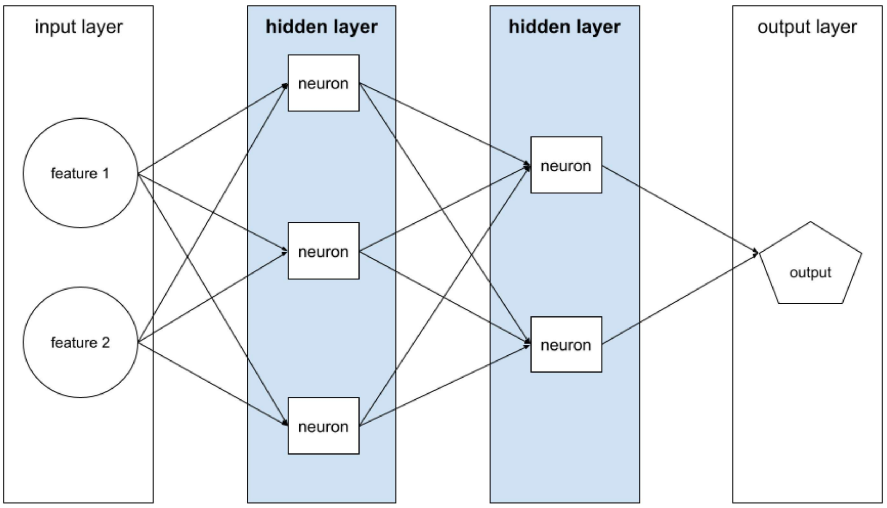
A single bucket could contain multiple tree species. For example, hashing could place *baobab* and *red maple*—two genetically dissimilar species—into the same bucket. Regardless, hashing is still a good way to map large categorical sets into the desired number of buckets. Hashing turns a categorical feature having a large number of possible values into a much smaller number of values by grouping values in a deterministic way.

heuristic

A simple and quickly implemented solution to a problem. For example, "With a heuristic, we achieved 86% accuracy. When we switched to a deep neural network, accuracy went up to 98%."

hidden layer

A layer in a **neural network** (#neural_network) between the **input layer** (#input_layer) (the features) and the **output layer** (#output_layer) (the prediction). Each hidden layer consists of one or more **neurons** (#neuron). For example, the following neural network contains two hidden layers, the first with three neurons and the second with two neurons:



A **deep neural network** (#deep_neural_network) contains more than one hidden layer. For example, the preceding illustration is a deep neural network because the model contains two hidden layers.

hierarchical clustering

A category of **clustering** (#clustering) algorithms that create a tree of clusters. Hierarchical clustering is well-suited to hierarchical data, such as botanical taxonomies. There are two types of hierarchical clustering algorithms:

- **Agglomerative clustering** first assigns every example to its own cluster, and iteratively merges the closest clusters to create a hierarchical tree.
- **Divisive clustering** first groups all examples into one cluster and then iteratively divides the cluster into a hierarchical tree.

Contrast with **centroid-based clustering** (#centroid_based_clustering).

hinge loss

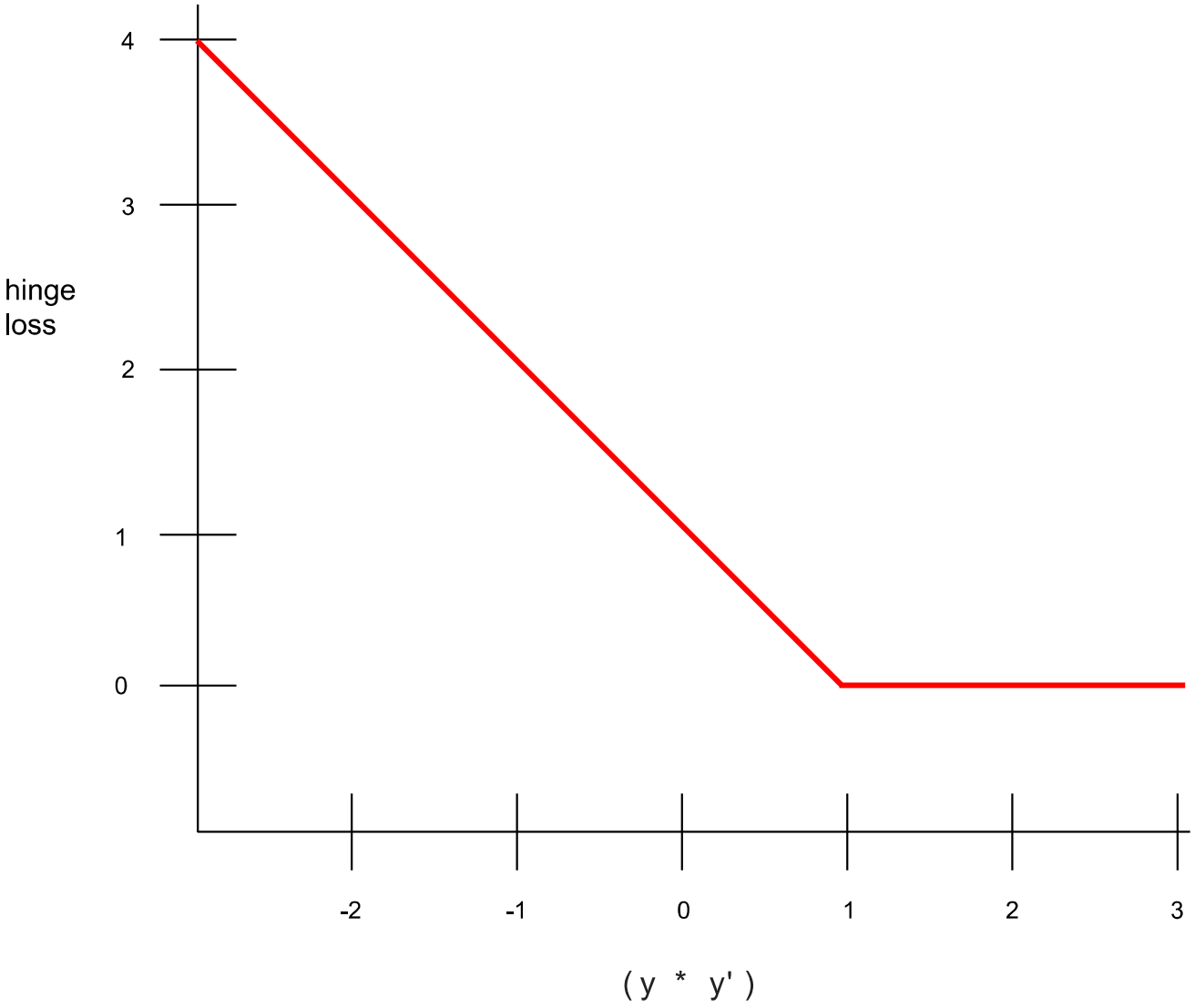
A family of **loss** (#loss) functions for **classification** (#classification_model) designed to find the **decision boundary** (#decision_boundary) as distant as possible from each training example, thus maximizing the margin between examples and the boundary. **KSVMs** (#KSVMs) use hinge loss (or a related function, such as squared hinge loss). For binary classification, the hinge loss function is defined as follows:

$$\text{loss} = \max(0, 1 - (y * y'))$$

where *y* is the true label, either -1 or +1, and *y′* is the raw output of the classifier model:

$$y' = b + w_1x_1 + w_2x_2 + \ldots w_nx_n$$

Consequently, a plot of hinge loss vs. (y * y') looks as follows:



holdout data

Examples (#example) intentionally not used ("held out") during training. The **validation dataset** (#validation_set) and **test dataset** (#test_set) are examples of holdout data. Holdout data helps evaluate your model's ability to generalize to data other than the data it was trained on. The loss on the holdout set provides a better estimate of the loss on an unseen dataset than does the loss on the training set.

hyperparameter

The variables that you or a hyperparameter tuning service adjust during successive runs of training a model. For example, **learning rate** (#learning_rate) is a hyperparameter. You could set the learning rate to 0.01 before one training session. If you determine that 0.01 is too high, you could perhaps set the learning rate to 0.003 for the next training session.

In contrast, **parameters** (#parameter) are the various **weights** (#weight) and **bias** (#bias) that the model *learns* during training.

hyperplane

A boundary that separates a space into two subspaces. For example, a line is a hyperplane in two dimensions and a plane is a hyperplane in three dimensions. More typically in machine learning, a hyperplane is the boundary separating a high-dimensional space. **Kernel Support Vector Machines** (#KSVMs) use hyperplanes to separate positive classes from negative classes, often in a very high-dimensional space.

I

i.i.d.

Abbreviation for **independently and identically distributed** (#iid).

image recognition

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

For more information, see ML Practicum: Image Classification (/machine-learning/practica/image-classification).

imbalanced dataset

Synonym for **class-imbalanced dataset** (#class_imbalanced_data_set).

implicit bias

Automatically making an association or assumption based on one's mental models and memories. Implicit bias can affect the following:

- How data is collected and classified.
- How machine learning systems are designed and developed.

For example, when building a classifier to identify wedding photos, an engineer may use the presence of a white dress in a photo as a feature. However, white dresses have been customary only during certain eras and in certain cultures.

See also **confirmation bias** (#confirmation_bias).

incompatibility of fairness metrics ⚖️

The idea that some notions of fairness are mutually incompatible and cannot be satisfied simultaneously. As a result, there is no single universal **metric** (#fairness_metric) for quantifying fairness that can be applied to all ML problems.

While this may seem discouraging, incompatibility of fairness metrics doesn't imply that fairness efforts are fruitless. Instead, it suggests that fairness must be defined contextually for a given ML problem, with the goal of preventing harms specific to its use cases.

See "On the (im)possibility of fairness" (https://arxiv.org/pdf/1609.07236.pdf) for a more detailed discussion of this topic.

independently and identically distributed (i.i.d) 🐣

Data drawn from a distribution that doesn't change, and where each value drawn doesn't depend on values that have been drawn previously. An i.i.d. is the ideal gas

(https://wikipedia.org/wiki/Ideal_gas) of machine learning—a useful mathematical construct but almost never exactly found in the real world. For example, the distribution of visitors to a web page may be i.i.d. over a brief window of time; that is, the distribution doesn't change during that brief window and one person's visit is generally independent of another's visit. However, if you expand that window of time, seasonal differences in the web page's visitors may appear.

See also **nonstationarity** (#nonstationarity).

individual fairness

A fairness metric that checks whether similar individuals are classified similarly. For example, Brobdingnagian Academy might want to satisfy individual fairness by ensuring that two students with identical grades and standardized test scores are equally likely to gain admission.

Note that individual fairness relies entirely on how you define "similarity" (in this case, grades and test scores), and you can run the risk of introducing new fairness problems if your similarity metric misses important information (such as the rigor of a student's curriculum).

See "Fairness Through Awareness" (https://arxiv.org/pdf/1104.3913.pdf) for a more detailed discussion of individual fairness.

inference

In machine learning, the process of making predictions by applying a trained model to **unlabeled examples** (#unlabeled_example).

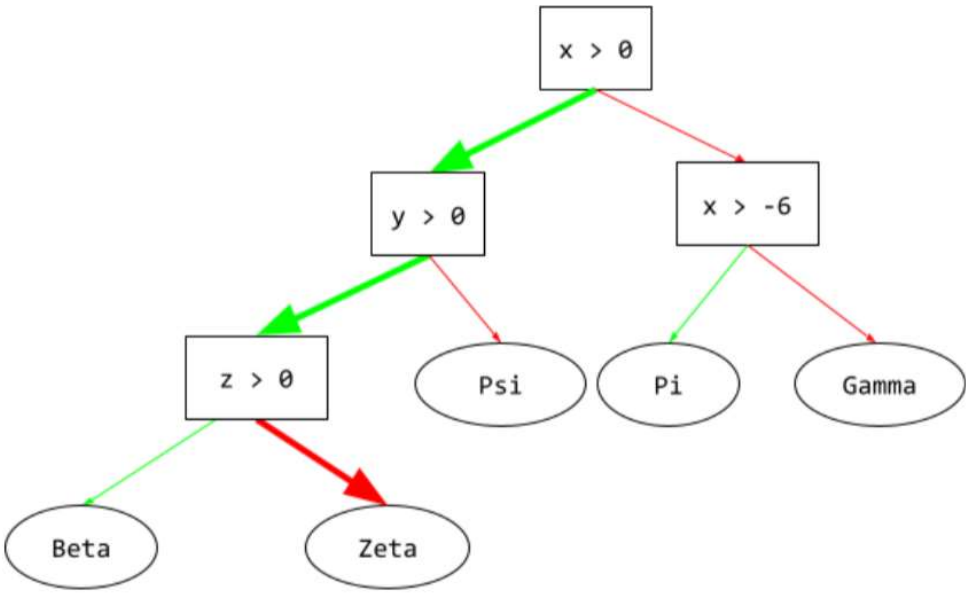
Inference has a somewhat different meaning in statistics. See the Wikipedia article on statistical inference (https://wikipedia.org/wiki/Statistical_inference) for details.

inference path

In a **decision tree** (#decision-tree), during **inference** (#inference), the route a particular **example** (#example) takes from the **root** (#root) to other **conditions** (#condition), terminating with a **leaf** (#leaf). For instance, in the following decision tree, the thicker arrows show the inference path for an example with the following feature values:

- x = 7
- y = 12
- z = -3

The inference path in the following illustration travels through three conditions before reaching the leaf (`Zeta`).



The three thick arrows show the inference path.

information gain

In **decision forests** (#decision-forest), the difference between a node's **entropy** (#entropy) and the weighted (by number of examples) sum of the entropy of its children nodes. A node's

entropy is the entropy of the examples in that node.

For example, consider the following entropy values:

- entropy of parent node = 0.6
- entropy of one child node with 16 relevant examples = 0.2
- entropy of another child node with 24 relevant examples = 0.1

So 40% of the examples are in one child node and 60% are in the other child node. Therefore:

- weighted entropy sum of child nodes = (0.4 * 0.2) + (0.6 * 0.1) = 0.14

So, the information gain is:

- information gain = entropy of parent node - weighted entropy sum of child nodes
- information gain = 0.6 - 0.14 = 0.46

Most **splitters** (#splitter) seek to create **conditions** (#condition) that maximize information gain.

in-group bias

Showing partiality to one's own group or own characteristics. If testers or raters consist of the machine learning developer's friends, family, or colleagues, then in-group bias may invalidate product testing or the dataset.

In-group bias is a form of **group attribution bias** (#group_attribution_bias). See also **out-group homogeneity bias** (#out-group_homogeneity_bias).

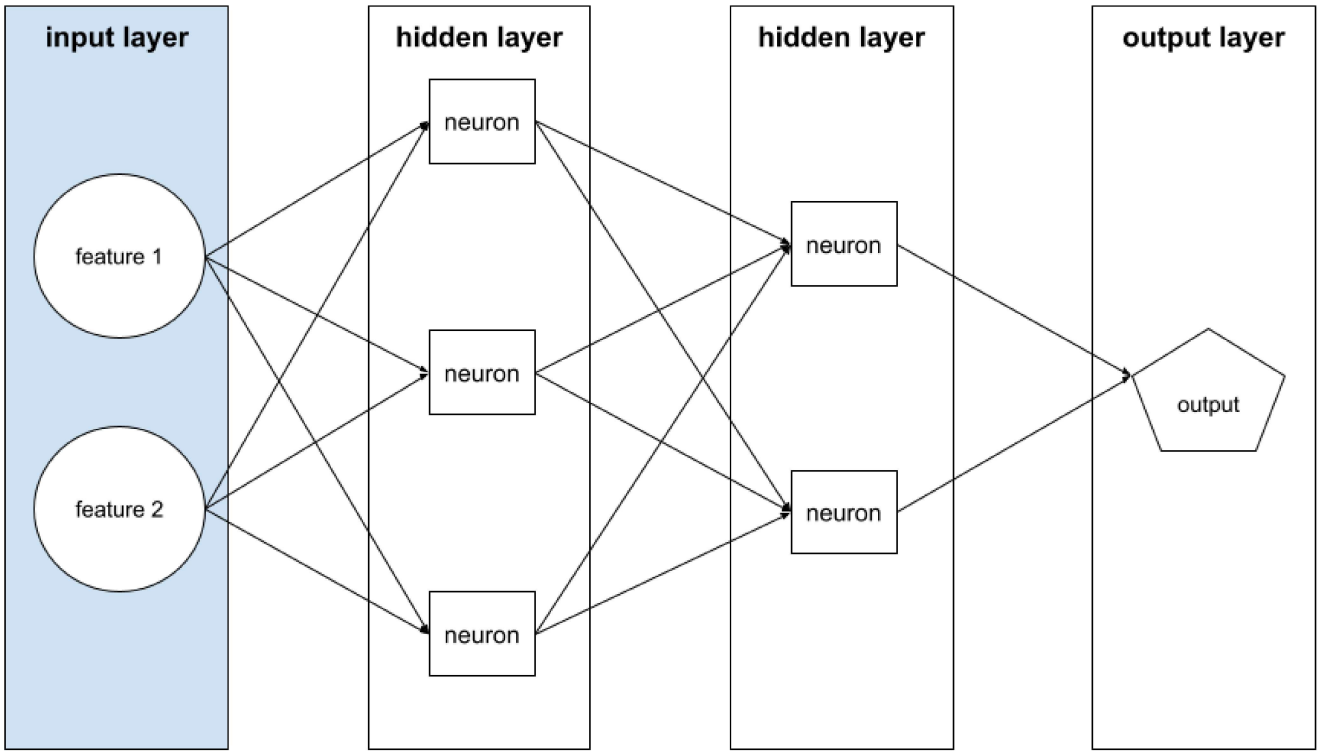
input generator

A mechanism by which data is loaded into a **neural network** (#neural-network).

An input generator can be thought of as a component responsible for processing raw data into tensors which are iterated over to generate batches for training, evaluation, and inference.

input layer

The **layer** (#layer) of a **neural network** (#neural_network) that holds the **feature vector** (#feature_vector). That is, the input layer provides **examples** (#example) for **training** (#training) or **inference** (#inference). For example, the input layer in the following neural network consists of two features:



in-set condition

In a **decision tree** (#decision-tree), a **condition** (#condition) that tests for the presence of one item in a set of items. For example, the following is an in-set condition:

```
house-style in [tudor, colonial, cape]
```

During inference, if the value of the house-style **feature** (#feature) is `tudor` or `colonial` or `cape`, then this condition evaluates to Yes. If the value of the house-style feature is something else (for example, `ranch`), then this condition evaluates to No.

In-set conditions usually lead to more efficient decision trees than conditions that test **one-hot encoded** (#one-hot_encoding) features.

instance

Synonym for **example** (#example).

interpretability

The ability to explain or to present an ML model's reasoning in understandable terms to a human.

Most linear regression models, for example, are highly interpretable. (You merely need to look at the trained weights for each feature.) Decision forests are also highly interpretable. Some models, however, require sophisticated visualization to become interpretable.

inter-rater agreement

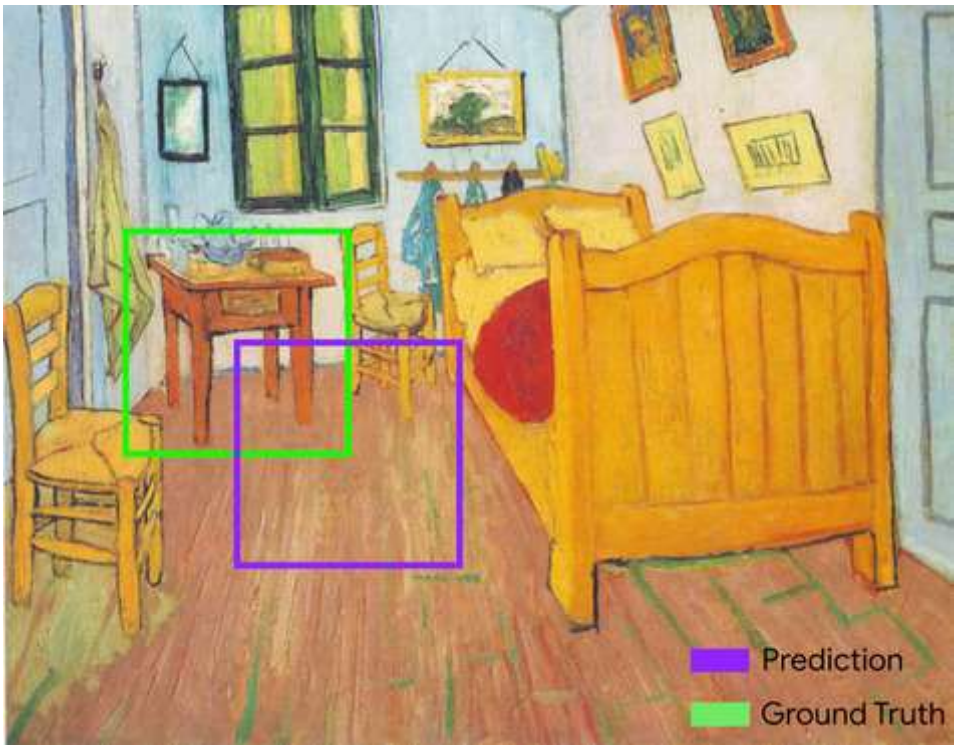
A measurement of how often human raters agree when doing a task. If raters disagree, the task instructions may need to be improved. Also sometimes called **inter-annotator agreement** or **inter-rater reliability**. See also Cohen's kappa (https://wikipedia.org/wiki/Cohen%27s_kappa), which is one of the most popular inter-rater agreement measurements.

intersection over union (IoU)

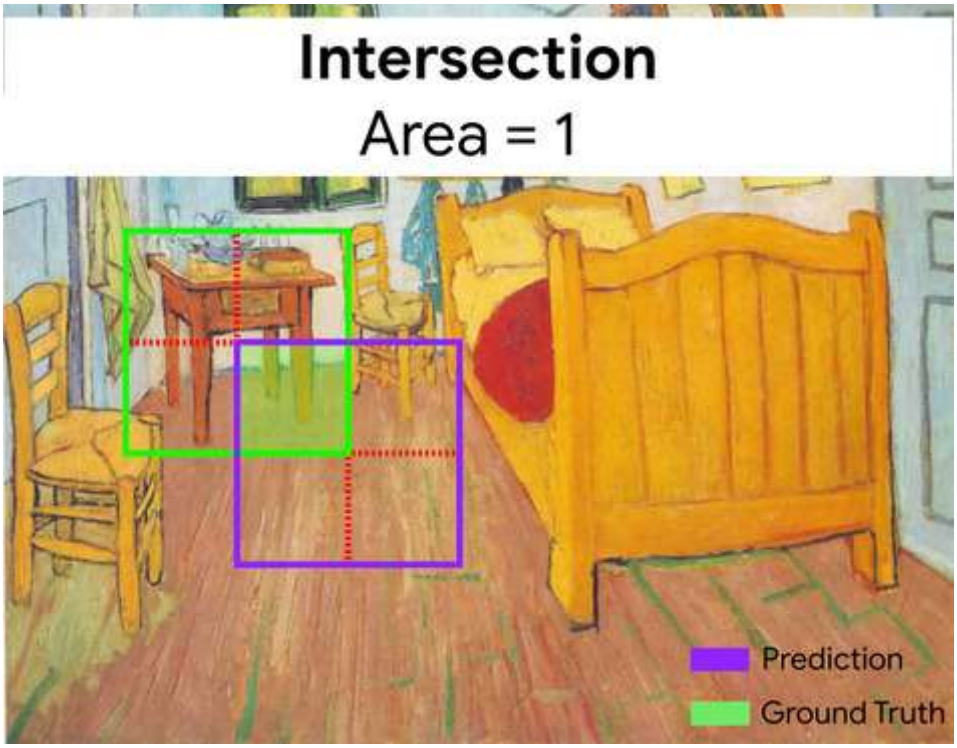
The intersection of two sets divided by their union. In machine-learning image-detection tasks, IoU is used to measure the accuracy of the model's predicted **bounding box** (#bounding_box) with respect to the **ground-truth** (#ground_truth) bounding box. In this case, the IoU for the two boxes is the ratio between the overlapping area and the total area, and its value ranges from 0 (no overlap of predicted bounding box and ground-truth bounding box) to 1 (predicted bounding box and ground-truth bounding box have the exact same coordinates).

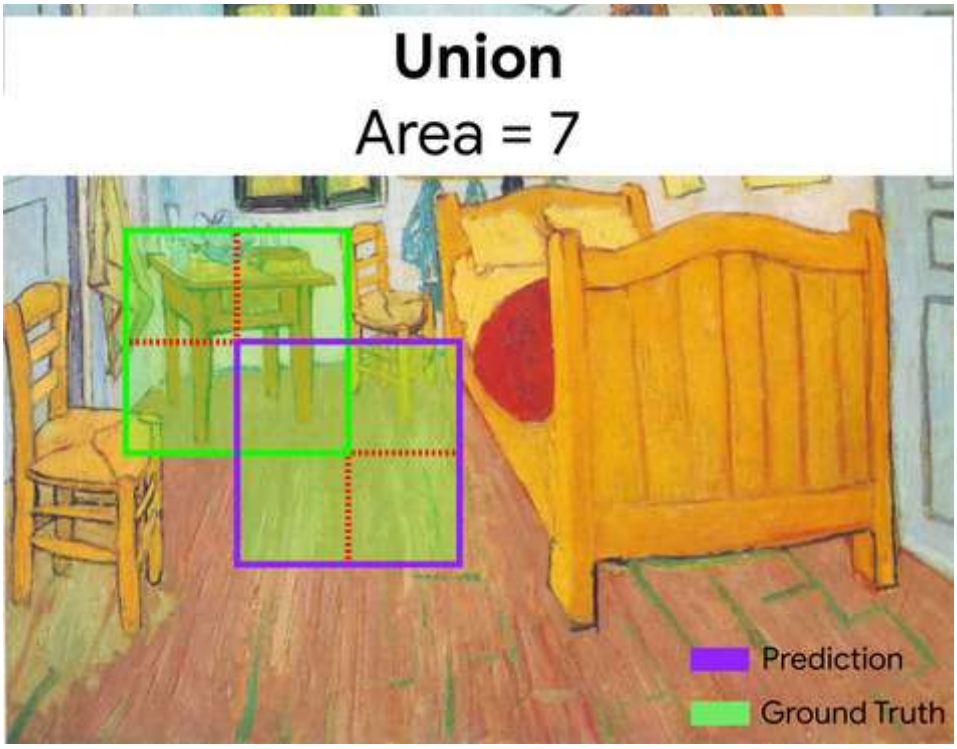
For example, in the image below:

- The predicted bounding box (the coordinates delimiting where the model predicts the night table in the painting is located) is outlined in purple.
- The ground-truth bounding box (the coordinates delimiting where the night table in the painting is actually located) is outlined in green.



Here, the intersection of the bounding boxes for prediction and ground truth (below left) is 1, and the union of the bounding boxes for prediction and ground truth (below right) is 7, so the IoU is $\frac{1}{7}$.





IoU

Abbreviation for **intersection over union** (#intersection_over_union).

item matrix

In **recommendation systems** (#recommendation_system), a matrix of **embedding vectors** (#embedding_vector) generated by **matrix factorization** (#matrix_factorization) that holds latent signals about each **item** (#items). Each row of the item matrix holds the value of a single latent feature for all items. For example, consider a movie recommendation system. Each column in the item matrix represents a single movie. The latent signals might represent genres, or might be harder-to-interpret signals that involve complex interactions among genre, stars, movie age, or other factors.

The item matrix has the same number of columns as the target matrix that is being factorized. For example, given a movie recommendation system that evaluates 10,000 movie titles, the item matrix will have 10,000 columns.

items

In a **recommendation system** (#recommendation_system), the entities that a system recommends. For example, videos are the items that a video store recommends, while books are the items that a bookstore recommends.

iteration

A single update of a **model's** (#model) parameters—the model's **weights** (#weight) and **biases** (#bias)—during **training** (#training). The **batch size** (#batch_size) determines how many examples the model processes in a single iteration. For instance, if the batch size is 20, then the model processes 20 examples before adjusting the parameters.

When training a **neural network** (#neural_network), a single iteration involves the following two passes:

1. A forward pass to evaluate loss on a single batch.
2. A backward pass (**backpropagation** (#backpropagation)) to adjust the model's parameters based on the loss and the learning rate.

J

JAX

An array computing library, bringing together **XLA (Accelerated Linear Algebra)** (#XLA) and automatic differentiation for high-performance numerical computing. JAX provides a simple and powerful API for writing accelerated numerical code with composable transformations. JAX provides features such as:

- `grad` (automatic differentiation)
- `jit` (just-in-time compilation)
- `vmap` (automatic vectorization or batching)
- `pmap` (parallelization)

JAX is a language for expressing and composing transformations of numerical code, analogous—but much larger in scope—to Python's **NumPy** (#numpy) library. (In fact, the .numpy library under JAX is a functionally equivalent, but entirely rewritten version of the Python NumPy library.)

JAX is particularly well-suited for speeding up many machine learning tasks by transforming the models and data into a form suitable for parallelism across GPU and **TPU** (#TPU) **accelerator chips** (#accelerator-chip).

Flax (#flax), **Optax** (#optax), **Pax** (#pax), and many other libraries are built on the JAX infrastructure.

K

Keras

A popular Python machine learning API. Keras (https://keras.io) runs on several deep learning frameworks, including TensorFlow, where it is made available as tf.keras (https://www.tensorflow.org/api_docs/python/tf/keras).

keypoints

The coordinates of particular features in an image. For example, for an **image recognition** (#image_recognition) model that distinguishes flower species, keypoints might be the center of each petal, the stem, the stamen, and so on.

Kernel Support Vector Machines (KSVMs)

A classification algorithm that seeks to maximize the margin between **positive** (#positive_class) and **negative classes** (#negative_class) by mapping input data vectors to a higher dimensional space. For example, consider a classification problem in which the input dataset has a hundred features. To maximize the margin between positive and negative classes, a KSVM could internally map those features into a million-dimension space. KSVMs uses a loss function called **hinge loss** (#hinge-loss).

k-fold cross validation

An algorithm for predicting a model's ability to **generalize** (#generalization) to new data. The *k* in k-fold refers to the number of equal groups you divide a dataset's examples into; that is, you

train and test your model k times. For each round of training and testing, a different group is the test set, and all remaining groups become the training set. After k rounds of training and testing, you calculate the mean and standard deviation of the desired test metric(s).

For example, suppose your dataset consists of 120 examples. Further suppose, you decide to set k to 4. Therefore, after shuffling the examples, you divide the dataset into four equal groups of 30 examples and conduct four training/testing rounds:

For example, **Mean Squared Error (MSE)** (#MSE) might be the most meaningful metric for a linear regression model. Therefore, you would find the mean and standard deviation of the MSE across all four rounds.

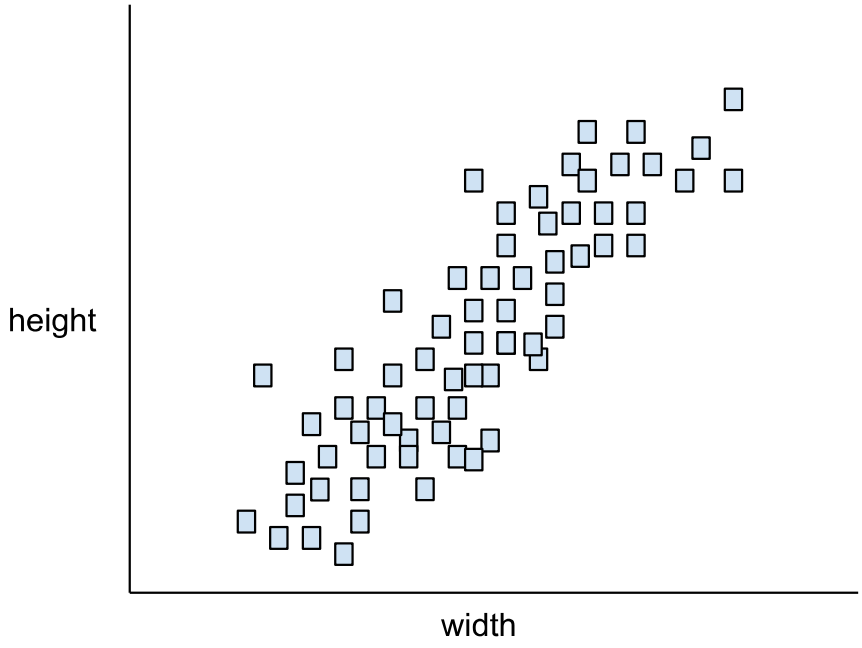
k-means

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

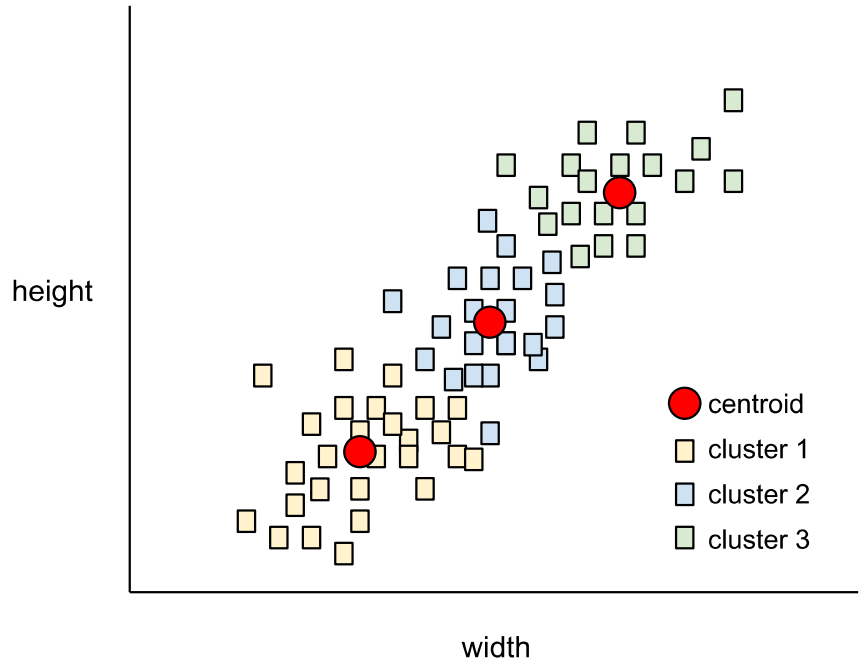
- Iteratively determines the best k center points (known as **centroids** (#centroid)).
- Assigns each example to the closest centroid. Those examples nearest the same centroid belong to the same group.

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.

k-median

A clustering algorithm closely related to **k-means** (#k-means). The practical difference between the two is as follows:

- In k-means, centroids are determined by minimizing the sum of the *squares* of the distance between a centroid candidate and each of its examples.
- In k-median, centroids are determined by minimizing the sum of the distance between a centroid candidate and each of its examples.

Note that the definitions of distance are also different:

- k-means relies on the Euclidean distance (https://wikipedia.org/wiki/Euclidean_distance) from the centroid to an example. (In two dimensions, the Euclidean distance means using the Pythagorean theorem to calculate the hypotenuse.) For example, the k-means distance between (2,2) and (5,-2) would be:

$$\textbf{Euclidean distance} = \sqrt{(2-5)^2 + (2--2)^2} = 5$$

- k-median relies on the Manhattan distance (https://wikipedia.org/wiki/Taxicab_geometry) from the centroid to an example. This distance is the sum of the absolute deltas in each dimension. For example, the k-median distance between (2,2) and (5,-2) would be:

$$\textbf{Manhattan distance} = |2-5| + |2--2| = 7$$

L

L_0 regularization

A type of **regularization** (#regularization) that penalizes the *total number* of nonzero **weights** (#weight) in a model. For example, a model having 11 nonzero weights would be penalized more than a similar model having 10 nonzero weights.

L_0 regularization is sometimes called *L0-norm regularization*.

Click the icon for additional notes.

L_0 regularization is generally impractical in large models because L_0 regularization turns training into a **convex** (#convex_function) optimization problem.

L_1 loss

A **loss function** (#loss-function) that calculates the absolute value of the difference between actual **label** (#label) values and the values that a **model** (#model) predicts. For example, here's the calculation of L_1 loss for a **batch** (#batch) of five **examples** (#example):

Actual value of example	Model's predicted value	Absolute value of delta
7	6	1
5	4	1
8	11	3
4	6	2
9	8	1

	8 = L_1 loss

L_1 loss is less sensitive to **outliers** (#outliers) than **L_2 loss** (#squared_loss).

The **Mean Absolute Error** (#MAE) is the average L_1 loss per example.

Click the icon to see the formal math.

$$L_1 loss = \sum_{i=0}^{n} |y_i - \hat{y}_i|$$

where:

- n is the number of examples.
- y is the actual value of the label.
- $\hat{y}$ is the value that the model predicts for y.

L_1 regularization

A type of **regularization** (#regularization) that penalizes **weights** (#weight) in proportion to the sum of the absolute value of the weights. L_1 regularization helps drive the weights of irrelevant or barely relevant features to *exactly 0*. A **feature** (#feature) with a weight of 0 is effectively removed from the model.

Contrast with **L_2 regularization** (#L2_regularization).

L_2 loss

A **loss function** (#loss-function) that calculates the square of the difference between actual **label** (#label) values and the values that a **model** (#model) predicts. For example, here's the calculation of L_2 loss for a **batch** (#batch) of five **examples** (#example):

Actual value of example	Model's predicted value	Square of delta
7	6	1
5	4	1
8	11	9
4	6	4
9	8	1
		16 = L_2 loss

Due to squaring, L_2 loss amplifies the influence of **outliers** (#outliers). That is, L_2 loss reacts more strongly to bad predictions than **L_1 loss** (#L1_loss). For example, the L_1 loss for the preceding batch would be 8 rather than 16. Notice that a single outlier accounts for 9 of the 16.

Regression models (#regression_model) typically use L_2 loss as the loss function.

The **Mean Squared Error** (#MSE) is the average L_2 loss per example. **Squared loss** is another name for L_2 loss.

Click the icon to see the formal math.

$$L_2 loss = \sum_{i=0}^{n} (y_i - \hat{y}_i)^2$$

where:

- n is the number of examples.
- y is the actual value of the label.
- $\hat{y}$ is the value that the model predicts for y.

L_2 regularization

A type of **regularization** (#regularization) that penalizes **weights** (#weight) in proportion to the sum of the *squares* of the weights. L_2 regularization helps drive **outlier** (#outliers) weights (those with high positive or low negative values) closer to 0 but *not quite to 0*. Features with values very close to 0 remain in the model but don't influence the model's prediction very much.

L_2 regularization always improves generalization in **linear models** (#linear_model).

Contrast with **L_1 regularization** (#L1_regularization).

label

In **supervised machine learning** (#supervised_machine_learning), the "answer" or "result" portion of an **example** (#example).

Each **labeled example** (#labeled_example) consists of one or more **features** (#feature) and a label. For instance, in a spam detection dataset, the label would probably be either "spam" or "not spam." In a rainfall dataset, the label might be the amount of rain that fell during a certain period.

labeled example

An example that contains one or more **features** (#feature) and a **label** (#label). For example, the following table shows three labeled examples from a house valuation model,

each with three features and one label:

Number of bedrooms	Number of bathrooms	House age	House price (label)
3	2	15	$345,000
2	1	72	$179,000
4	2	34	$392,000

In **supervised machine learning** (#supervised_machine_learning), models train on labeled examples and make predictions on **unlabeled examples** (#unlabeled_example).

Contrast labeled example with unlabeled examples.

label leakage

A model design flaw in which a **feature** (#feature) is a proxy for the **label** (#label). For example, consider a **binary classification** (#binary-classification) model that predicts whether or not a prospective customer will purchase a particular product. Suppose that one of the features for the model is a Boolean named `SpokeToCustomerAgent`. Further suppose that a customer agent is only assigned *after* the prospective customer has actually purchased the product. During training, the model will quickly learn the association between `SpokeToCustomerAgent` and the label.

LaMDA (Language Model for Dialogue Applications)

A **Transformer** (#Transformer)-based **large language model** (#large-language-model) developed by Google trained on a large dialogue dataset that can generate realistic conversational responses.

LaMDA: our breakthrough conversation technology (https://blog.google/technology/ai/lamda/) provides an overview.

lambda

Synonym for **regularization rate** (#regularization_rate).

Lambda is an overloaded term. Here we're focusing on the term's definition within **regularization** (#regularization).

landmarks

Synonym for **keypoints** (#keypoints).

language model

A **model** (#model) that estimates the probability of a **token** (#token) or sequence of tokens occurring in a longer sequence of tokens.

Click the icon for additional notes.

Though counterintuitive, many models that evaluate text are not **language models**. For example, text classification models and sentiment analysis models are not **language models**.

large language model

An informal term with no strict definition that usually means a **language model** (#language-model) that has a high number of **parameters** (#parameter). Some large language models contain over 100 billion parameters.

Click the icon for additional notes.

You might be wondering when a **language model** (#language-model) becomes large enough to be termed a **large language model**. Currently, there is no agreed-upon defining line for the number of parameters.

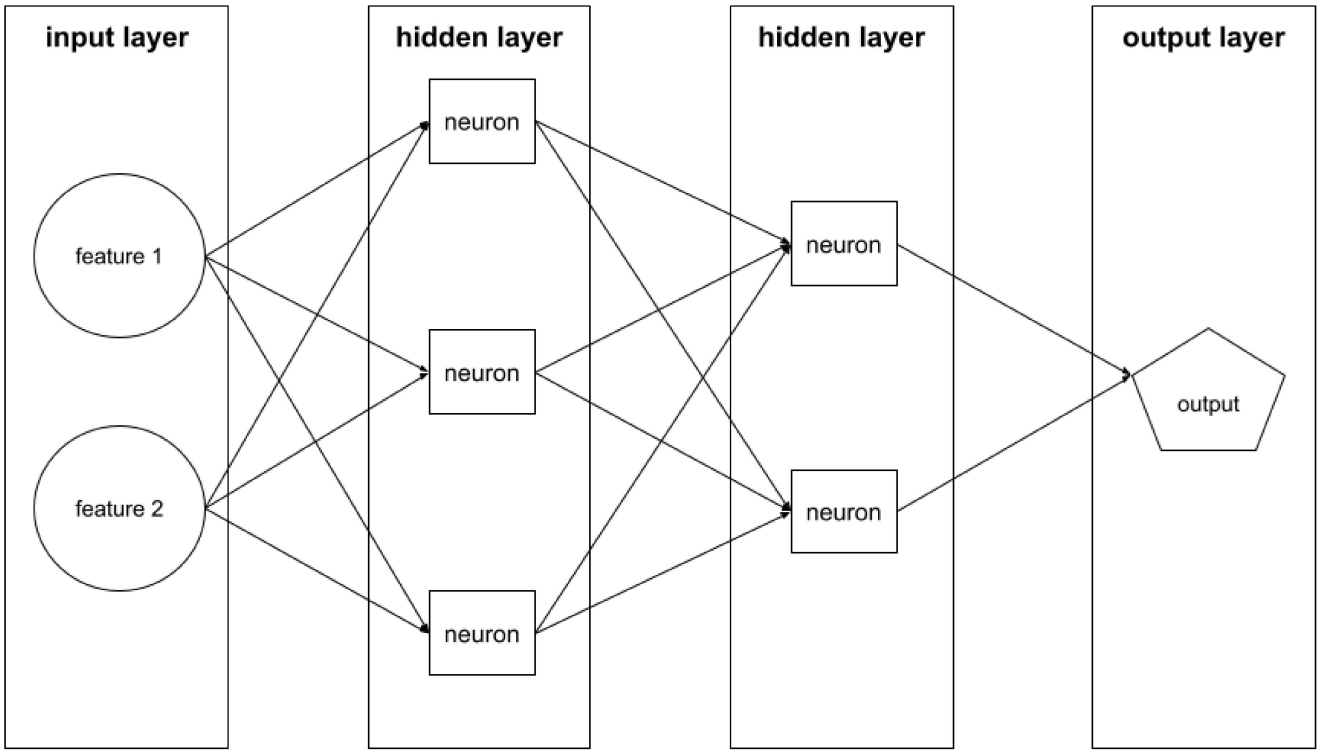
Most current large language models (for example, **GPT** (#GPT)) are based on **Transformer** (#Transformer) architecture.

layer

A set of **neurons** (#neuron) in a **neural network** (#neural_network). Three common types of layers are as follows:

- The **input layer** (#input_layer), which provides values for all the **features** (#feature).
- One or more **hidden layers** (#hidden_layer), which find nonlinear relationships between the features and the label.
- The **output layer** (#output_layer), which provides the prediction.

For example, the following illustration shows a neural network with one input layer, two hidden layers, and one output layer:



In **TensorFlow** (#TensorFlow), **layers** are also Python functions that take **Tensors** (#tensor) and configuration options as input and produce other tensors as output.

Layers API (tf.layers)

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

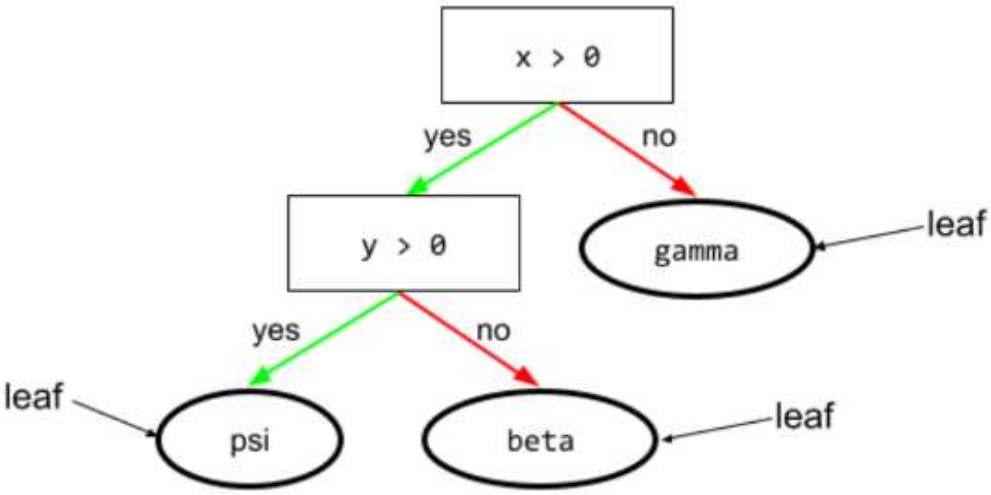
- `tf.layers.Dense` for a **fully-connected layer** (#fully_connected_layer).
- `tf.layers.Conv2D` for a convolutional layer.

The Layers API follows the **Keras** (#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.

leaf

Any endpoint in a **decision tree** (#decision-tree). Unlike a **condition** (#condition), a leaf does not perform a test. Rather, a leaf is a possible prediction. A leaf is also the terminal **node** (#node) of an **inference path** (#inference-path).

For example, the following decision tree contains three leaves:



learning rate

A floating-point number that tells the **gradient descent** (#gradient_descent) algorithm how strongly to adjust weights and biases on each **iteration** (#iteration). For example, a learning rate of 0.3 would adjust weights and biases three times more powerfully than a learning rate of 0.1.

Learning rate is a key **hyperparameter** (#hyperparameter). If you set the learning rate too low, training will take too long. If you set the learning rate too high, gradient descent often has trouble reaching **convergence** (#convergence).

Click the icon for a more mathematical explanation.

During each iteration, the **gradient descent** (#gradient_descent) algorithm multiplies the learning rate by the gradient. The resulting product is called the **gradient step**.

least squares regression

A linear regression model trained by minimizing **L_2 Loss** (#L2_loss).

linear model

A **model** (#model) that assigns one **weight** (#weight) per **feature** (#feature) to make **predictions** (#prediction). (Linear models also incorporate a **bias** (#bias).) In contrast, the relationship of features to predictions in **deep models** (#deep_model) is generally **nonlinear**.

Linear models are usually easier to train and more **interpretable** (#interpretability) than deep models. However, deep models can learn complex relationships *between* features.

Linear regression (#linear_regression) and **logistic regression** (#logistic_regression) are two types of linear models.

Click the icon to see the math.

A linear model follows this formula:

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

where:

- y' is the raw prediction. (In certain kinds of linear models, this raw prediction will be further modified. For example, see **logistic regression** (#logistic_regression).)
- b is the **bias** (#bias).
- w is a **weight** (#weight), so w_1 is the weight of the first feature, w_2 is the weight of the second feature, and so on.

- x is a **feature** (#feature), so x_1 is the value of the first feature, x_2 is the value of the second feature, and so on.

For example, suppose a linear model for three features learns the following bias and weights:

- b = 7
- w_1 = -2.5
- w_2 = -1.2
- w_3 = 1.4

Therefore, given three features (x_1, x_2, and x_3), the linear model uses the following equation to generate each prediction:

$$y' = 7 + (-2.5)(x_1) + (-1.2)(x_2) + (1.4)(x_3)$$

Suppose a particular example contains the following values:

- x_1 = 4
- x_2 = -10
- x_3 = 5

Plugging those values into the formula yields a prediction for this example:

$$y' = 7 + (-2.5)(4) + (-1.2)(-10) + (1.4)(5)$$
$$y' = 16$$

Linear models include not only models that use only a linear equation to make predictions but also a broader set of models that use a linear equation as just one component of the formula that makes predictions. For example, logistic regression post-processes the raw prediction (y') to produce a final prediction value between 0 and 1, exclusively.

---

linear

A relationship between two or more variables that can be represented solely through addition and multiplication.

The plot of a linear relationship is a line.

Contrast with **nonlinear** (#nonlinear).

linear regression

A type of machine learning model in which both of the following are true:

- The model is a **linear model** (#linear_model).
- The prediction is a floating-point value. (This is the **regression** (#regression_model) part of *linear regression*.)

Contrast linear regression with **logistic regression** (#logistic_regression). Also, contrast regression with **classification** (#classification_model).

logistic regression

A type of **regression model** (#regression_model) that predicts a probability. Logistic regression models have the following characteristics:

- The label is **categorical** (#categorical_data). The term logistic regression usually refers to **binary logistic regression**, that is, to a model that calculates probabilities for labels with two possible values. A less common variant, **multinomial logistic regression**, calculates probabilities for labels with more than two possible values.

- The loss function during training is **Log Loss** (#Log_Loss). (Multiple Log Loss units can be placed in parallel for labels with more than two possible values.)
- The model has a linear architecture, not a deep neural network. However, the remainder of this definition also applies to **deep models** (#deep_model) that predict probabilities for categorical labels.

For example, consider a logistic regression model that calculates the probability of an input email being either spam or not spam. During inference, suppose the model predicts 0.72. Therefore, the model is estimating:

- A 72% chance of the email being spam.
- A 28% chance of the email not being spam.

A logistic regression model uses the following two-step architecture:

1. The model generates a raw prediction (y') by applying a linear function of input features.
2. The model uses that raw prediction as input to a **sigmoid function** (#sigmoid-function), which converts the raw prediction to a value between 0 and 1, exclusive.

Like any regression model, a logistic regression model predicts a number. However, this number typically becomes part of a binary classification model as follows:

- If the predicted number is *greater* than the **classification threshold** (#classification_threshold), the binary classification model predicts the positive class.
- If the predicted number is *less* than the classification threshold, the binary classification model predicts the negative class.

logits

The vector of raw (non-normalized) predictions that a classification model generates, which is ordinarily then passed to a normalization function. If the model is solving a **multi-class classification** (#multi-class) problem, logits typically become an input to the **softmax** (#softmax) function. The softmax function then generates a vector of (normalized) probabilities with one value for each possible class.

tf.nn.sigmoid_cross_entropy_with_logits
(https://www.tensorflow.org/api_docs/python/tf/nn/sigmoid_cross_entropy_with_logits).

Log Loss

The **loss function** (#loss-function) used in binary **logistic regression** (#logistic_regression).

Click the icon to see the math.

The following formula calculates Log Loss:

$$\text{Log Loss} = \sum_{(x,y)\in D} -y \log(y') - (1-y) \log(1-y')$$

where:

- $(x, y) \in D$ is the data set containing many labeled examples, which are (x, y) pairs.
- y is the label in a labeled example. Since this is logistic regression, every value of y must either be 0 or 1.
- y' is the predicted value (somewhere between 0 and 1, exclusive), given the set of features in x.

log-odds

The logarithm of the odds of some event.

Click the icon to see the math.

If the event is a binary probability, then **odds** refers to the ratio of the probability of success (*p*) to the probability of failure (1-*p*). For example, suppose that a given event has a 90% probability of success and a 10% probability of failure. In this case, odds is calculated as follows:

$$\text{odds} = \frac{\text{p}}{\text{(1-p)}} = \frac{.9}{.1} = 9$$

The log-odds is simply the logarithm of the odds. By convention, "logarithm" refers to natural logarithm (https://wikipedia.org/wiki/Natural_logarithm), but logarithm could actually be any base greater than 1. Sticking to convention, the log-odds of our example is therefore:

$$\text{log-odds} = ln(9) = 2.2$$

The log-odds function is the inverse of the **sigmoid function** (#sigmoid-function).

Long Short-Term Memory (LSTM)

A type of cell in a **recurrent neural network** (#recurrent_neural_network) used to process sequences of data in applications such as handwriting recognition, machine translation, and image captioning. LSTMs address the **vanishing gradient problem** (#vanishing_gradient_problem) that occurs when training RNNs due to long data sequences by maintaining history in an internal memory state based on new input and context from previous cells in the RNN.

---

loss

During the **training** (#training) of a **supervised model** (#supervised_machine_learning), a measure of how far a model's **prediction** (#prediction) is from its **label** (#label).

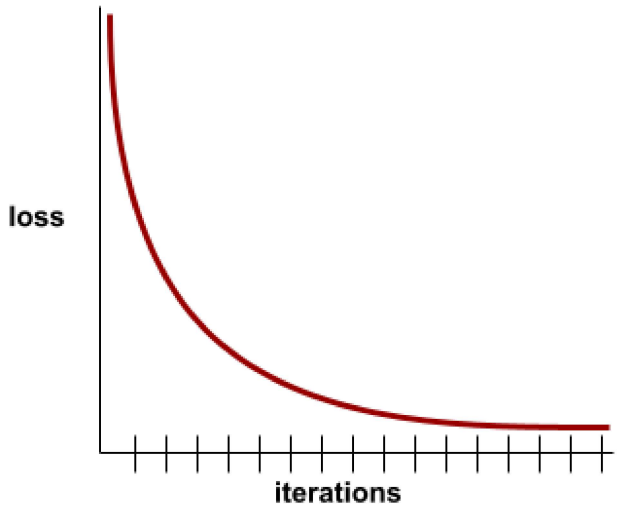
A **loss function** (#loss-function) calculates the loss.

loss aggregator

A type of **machine learning** (#machine_learning) algorithm that improves the **performance** (#performance) of a **model** (#model) by combining the **predictions** (#prediction) of multiple models and using those predictions to make a single prediction. As a result, a loss aggregator can reduce the variance of the predictions and improve the **accuracy** (#accuracy) of the predictions.

loss curve

A plot of **loss** (#loss) as a function of the number of training **iterations** (#iteration). The following plot shows a typical loss curve:



Loss curves can help you determine when your model is **converging** (#convergence) or **overfitting** (#overfitting).

Loss curves can plot all of the following types of loss:

- **training loss** (#training-loss)

- **validation loss** (#validation-loss)
- **test loss** (#test-loss)

See also **generalization curve** (#generalization_curve).

loss function

During **training** (#training) or testing, a mathematical function that calculates the loss on a **batch** (#batch) of examples. A loss function returns a lower loss for models that makes good predictions than for models that make bad predictions.

The goal of training is typically to minimize the loss that a loss function returns.

Many different kinds of loss functions exist. Pick the appropriate loss function for the kind of model you are building. For example:

- **L_2 loss** (#L2_loss) (or **Mean Squared Error** (#MSE)) is the loss function for **linear regression** (#linear_regression).
- **Log Loss** (#Log_Loss) is the loss function for **logistic regression** (#logistic_regression).

loss surface

A graph of weight(s) vs. loss. **Gradient descent** (#gradient_descent) aims to find the weight(s) for which the loss surface is at a local minimum.

LSTM

Abbreviation for **Long Short-Term Memory** (#Long_Short-Term_Memory).

M

machine learning

A program or system that **trains** (#training) a **model** (#model) from input data. The trained model can 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.

majority class

The more common label in a **class-imbalanced dataset** (#class_imbalanced_data_set). For example, given a dataset containing 99% negative labels and 1% positive labels, the negative labels are the majority class.

Contrast with **minority class** (#minority_class).

Markov decision process (MDP) RL

A graph representing the decision-making model where decisions (or **actions** (#action)) are taken to navigate a sequence of **states** (#state) under the assumption that the **Markov property** (#Markov_property) holds. In **reinforcement learning** (#reinforcement_learning), these transitions between states return a numerical **reward** (#reward).

Markov property RL

A property of certain **environments** (#environment), where state transitions are entirely determined by information implicit in the current **state** (#state) and the agent's **action** (#action).

masked language model

A **language model** (#language-model) that predicts the probability of candidate tokens to fill in blanks in a sequence. For instance, a masked language model can calculate probabilities for candidate word(s) to replace the underline in the following sentence:

The ____ in the hat came back.

The literature typically uses the string "MASK" instead of an underline. For example:

The "MASK" in the hat came back.

Most modern masked language models are **bidirectional** (#bidirectional).

matplotlib

An open-source Python 2D plotting library. matplotlib (https://matplotlib.org/) helps you visualize different aspects of machine learning.

matrix factorization

In math, a mechanism for finding the matrices whose dot product approximates a target matrix.

In **recommendation systems** (#recommendation_system), the target matrix often holds users' ratings on **items** (#items). For example, the target matrix for a movie recommendation system might look something like the following, where the positive integers are user ratings and 0 means that the user didn't rate the movie:

	Casablanca	The Philadelphia Story	Black Panther	Wonder Woman	Pulp Fiction
User 1	5.0	3.0	0.0	2.0	0.0
User 2	4.0	0.0	0.0	1.0	5.0
User 3	3.0	1.0	4.0	5.0	0.0

The movie recommendation system aims to predict user ratings for unrated movies. For example, will User 1 like *Black Panther*?

One approach for recommendation systems is to use matrix factorization to generate the following two matrices:

- A **user matrix** (#user_matrix), shaped as the number of users X the number of embedding dimensions.
- An **item matrix** (#item_matrix), shaped as the number of embedding dimensions X the number of items.

For example, using matrix factorization on our three users and five items could yield the following user matrix and item matrix:

```
User Matrix                 Item Matrix

1.1   2.3           0.9   0.2   1.4    2.0   1.2
0.6   2.0           1.7   1.2   1.2   -0.1   2.1
2.5   0.5
```

The dot product of the user matrix and item matrix yields a recommendation matrix that contains not only the original user ratings but also predictions for the movies that each user hasn't seen. For example, consider User 1's rating of *Casablanca*, which was 5.0. The dot product corresponding to that cell in the recommendation matrix should hopefully be around 5.0, and it is:

```
(1.1 * 0.9) + (2.3 * 1.7) = 4.9
```

More importantly, will User 1 like *Black Panther*? Taking the dot product corresponding to the first row and the third column yields a predicted rating of 4.3:

```
(1.1 * 1.4) + (2.3 * 1.2) = 4.3
```

Matrix factorization typically yields a user matrix and item matrix that, together, are significantly more compact than the target matrix.

Mean Absolute Error (MAE)

The average loss per example when **L_1 loss** (#L1_loss) is used. Calculate Mean Absolute Error as follows:

1. Calculate the L_1 loss for a batch.
2. Divide the L_1 loss by the number of examples in the batch.

Click the icon to see the formal math.

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

where:

- n is the number of examples.
- y is the actual value of the label.
- $\hat{y}$ is the value that the model predicts for y.

For example, consider the calculation of L_1 loss on the following batch of five examples:

Actual value of example	Model's predicted value	Loss (difference between actual and predicted)
7	6	1
5	4	1
8	11	3
4	6	2
9	8	1
		8 = L_1 loss

So, L_1 loss is 8 and the number of examples is 5. Therefore, the Mean Absolute Error is:

```
Mean Absolute Error = L1 loss / Number of Examples
Mean Absolute Error = 8/5 = 1.6
```

Contrast Mean Absolute Error with **Mean Squared Error** (#MSE) and **Root Mean Squared Error** (#RMSE).

Mean Squared Error (MSE)

The average loss per example when **L_2 loss** (#L2_loss) is used. Calculate Mean Squared Error as follows:

1. Calculate the L_2 loss for a batch.
2. Divide the L_2 loss by the number of examples in the batch.

Click the icon to see the formal math.

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2$$

where:

- n is the number of examples.
- y is the actual value of the label.
- $\hat{y}$ is the model's prediction for y.

For example, consider the loss on the following batch of five examples:

Actual value	Model's prediction	Loss	Squared loss
7	6	1	1
5	4	1	1
8	11	3	9
4	6	2	4
9	8	1	1
			16 = L_2 loss

Therefore, the Mean Squared Error is:

```
Mean Squared Error = L2 loss / Number of Examples
Mean Squared Error = 16/5 = 3.2
```

Mean Squared Error is a popular training **optimizer** (#optimizer), particularly for **linear regression** (#linear_regression).

Contrast Mean Squared Error with **Mean Absolute Error** (#MAE) and **Root Mean Squared Error** (#RMSE).

TensorFlow Playground (#TensorFlow_Playground) uses Mean Squared Error to calculate loss values.

Click the icon to see more details about outliers.

Outliers (#outliers) strongly influence Mean Squared Error. For example, a loss of 1 is a squared loss of 1, but a loss of 3 is a squared loss of 9. In the preceding table, the example with a loss of 3 accounts for ~56% of the Mean Squared Error, while each of the examples with a loss of 1 accounts for only 6% of the Mean Squared Error.

Outliers do not influence Mean Absolute Error as strongly as Mean Squared Error. For example, a loss of 3 accounts for only ~38% of the Mean Absolute Error.

Clipping (#clipping) is one way to prevent extreme outliers from damaging your model's predictive ability.

metric

A statistic that you care about.

An **objective** (#objective) is a metric that a machine learning system tries to optimize.

meta-learning

A subset of machine learning that discovers or improves a learning algorithm. A meta-learning system can also aim to train a model to quickly learn a new task from a small amount of data or from experience gained in previous tasks. Meta-learning algorithms generally try to achieve the following:

- Improve/learn hand-engineered features (such as an initializer or an optimizer).
- Be more data-efficient and compute-efficient.
- Improve generalization.

Meta-learning is related to **few-shot learning** (#few-shot_learning).

Metrics API (tf.metrics)

A TensorFlow API for evaluating models. For example, `tf.metrics.accuracy` determines how often a model's predictions match labels.

mini-batch

A small, randomly selected subset of a **batch** (#batch) processed in one **iteration** (#iteration). The **batch size** (#batch_size) of a mini-batch is usually between 10 and 1,000 examples.

For example, suppose the entire training set (the full batch) consists of 1,000 examples. Further suppose that you set the **batch size** (#batch_size) of each mini-batch to 20. Therefore,

each iteration determines the loss on a random 20 of the 1,000 examples and then adjusts the **weights** (#weight) and **biases** (#bias) accordingly.

It is much more efficient to calculate the loss on a mini-batch than the loss on all the examples in the full batch.

mini-batch stochastic gradient descent

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

minimax loss

A loss function for **generative adversarial networks** (#generative_adversarial_network), based on the **cross-entropy** (#cross-entropy) between the distribution of generated data and real data.

Minimax loss is used in the first paper (https://arxiv.org/pdf/1406.2661.pdf) to describe generative adversarial networks.

minority class

The less common label in a **class-imbalanced dataset** (#class_imbalanced_data_set). For example, given a dataset containing 99% negative labels and 1% positive labels, the positive labels are the minority class.

Contrast with **majority class** (#majority_class).

Click the icon for additional notes.

A training set with a million examples (#example) sounds impressive. However, if the minority class is poorly represented, then even a very large training set might be insufficient. Focus less on the total number of examples in the dataset and more on the number of examples in the minority class.

If your dataset doesn't contain enough minority class examples, consider using **downsampling** (#downsampling) (the definition in the second bullet) to supplement the minority class.

ML

Abbreviation for **machine learning** (#machine_learning).

MNIST

A public-domain dataset compiled by LeCun, Cortes, and Burges containing 60,000 images, each image showing how a human manually wrote a particular digit from 0–9. Each image is stored as a 28x28 array of integers, where each integer is a grayscale value between 0 and 255, inclusive.

MNIST is a canonical dataset for machine learning, often used to test new machine learning approaches. For details, see The MNIST Database of Handwritten Digits (http://yann.lecun.com/exdb/mnist/).

modality

A high-level data category. For example, numbers, text, images, video, and audio are five different modalities.

model

In general, any mathematical construct that processes input data and returns output. Phrased differently, a model is the set of parameters and structure needed for a system to make predictions. In **supervised machine learning** (#supervised_machine_learning), a model takes an **example** (#example) as input and infers a **prediction** (#prediction) as output. Within supervised machine learning, models differ somewhat. For example:

- A linear regression model consists of a set of **weights** (#weight) and a **bias** (#bias).
- A **neural network** (#neural-network) model consists of:
  - A set of **hidden layers** (#hidden_layer), each containing one or more **neurons** (#neuron).
  - The weights and bias associated with each neuron.
- A **decision tree** (#decision-tree) model consists of:
  - The shape of the tree; that is, the pattern in which the conditions and leaves are connected.
  - The conditions and leaves.

You can save, restore, or make copies of a model.

Unsupervised machine learning (#unsupervised_machine_learning) also generates models, typically a function that can map an input example to the most appropriate **cluster** (#clustering).

Click the icon to compare algebraic and programming functions to ML models.

An algebraic function such as the following is a model:

$$f(x, y) = 3x - 5xy + y^2 + 17$$

The preceding function maps input values (x and y) to output.

Similarly, a programming function like the following is also a model:

```
def half_of_greater(x, y):
  if (x > y):
    return(x / 2)
  else
    return(y / 2)
```

A caller passes arguments to the preceding Python function, and the Python function generates output (via the `return` statement).

Although a **deep neural network** (#deep_neural_network) has a very different mathematical structure than an algebraic or programming function, a deep neural network still takes input (an example) and returns output (a prediction).

A human programmer codes a programming function manually. In contrast, a machine learning model gradually learns the optimal parameters during automated training.

model capacity

The complexity of problems that a model can learn. The more complex the problems that a model can learn, the higher the model's capacity. A model's capacity typically increases with the number of model parameters. For a formal definition of classifier capacity, see VC dimension (https://wikipedia.org/wiki/VC_dimension).

model parallelism

A way of scaling training or inference that puts different parts of one model on different devices. Model parallelism enables models that are too big to fit on a single device.

See also **data parallelism** (#data-parallelism).

model training

The process of determining the best **model** (#model).

Momentum

A sophisticated gradient descent algorithm in which a learning step depends not only on the derivative in the current step, but also on the derivatives of the step(s) that immediately preceded it. Momentum involves computing an exponentially weighted moving average of the gradients over time, analogous to momentum in physics. Momentum sometimes prevents learning from getting stuck in local minima.

multi-class classification

In supervised learning, a **classification** (#classification_model) problem in which the dataset contains *more than two* **classes** (#class) of labels. For example, the labels in the Iris dataset must be one of the following three classes:

- Iris setosa

- Iris virginica
- Iris versicolor

A model trained on the Iris dataset that predicts Iris type on new examples is performing multi-class classification.

In contrast, classification problems that distinguish between exactly two classes are **binary classification models** (#binary_classification). For example, an email model that predicts either *spam* or *not spam* is a binary classification model.

In clustering problems, multi-class classification refers to more than two clusters.

multi-class logistic regression

Using **logistic regression** (#logistic_regression) in **multi-class classification** (#multi-class) problems.

multi-head self-attention

An extension of **self-attention** (#self-attention) that applies the self-attention mechanism multiple times for each position in the input sequence.

Transformers (#Transformer) introduced multi-head self-attention.

multimodal model

A model whose inputs and/or outputs include more than one **modality** (#modality). For example, consider a model that takes both an image and a text caption (two modalities) as **features** (#feature), and outputs a score indicating how appropriate the text caption is for the image. So, this model's inputs are multimodal and the output is unimodal.

multinomial classification

Synonym for **multi-class classification** (#multi-class).

multinomial regression

Synonym for **multi-class logistic regression** (#multi-class_logistic_regression).

multitask

A machine learning technique in which a single **model** (#model) is trained to perform multiple **tasks** (#task).

Multitask models are created by training on data that is appropriate for each of the different tasks. This allows the model to learn to share information across the tasks, which helps the model learn more effectively.

A model trained for multiple tasks often has improved generalization abilities and can be more robust at handling different types of data.

N

NaN trap

When one number in your model becomes a NaN (https://wikipedia.org/wiki/NaN) during training, which causes many or all other numbers in your model to eventually become a NaN.

NaN is an abbreviation for **N**ot **a** **N**umber.

natural language understanding

Determining a user's intentions based on what the user typed or said. For example, a search engine uses natural language understanding to determine what the user is searching for based on what the user typed or said.

negative class

In **binary classification** (#binary_classification), one class is termed *positive* and the other is termed *negative*. The positive class is the thing or event that the model is testing for and the negative class is the other possibility. For example:

- The negative class in a medical test might be "not tumor."
- The negative class in an email classifier might be "not spam."

Contrast with **positive class** (#positive_class).

negative sampling

Synonym for **candidate sampling** (#candidate_sampling).

Neural Architecture Search (NAS)

A technique for automatically designing the architecture of a **neural network** (#neural-network). NAS algorithms can reduce the amount of time and resources required to train a neural network.

NAS typically uses:

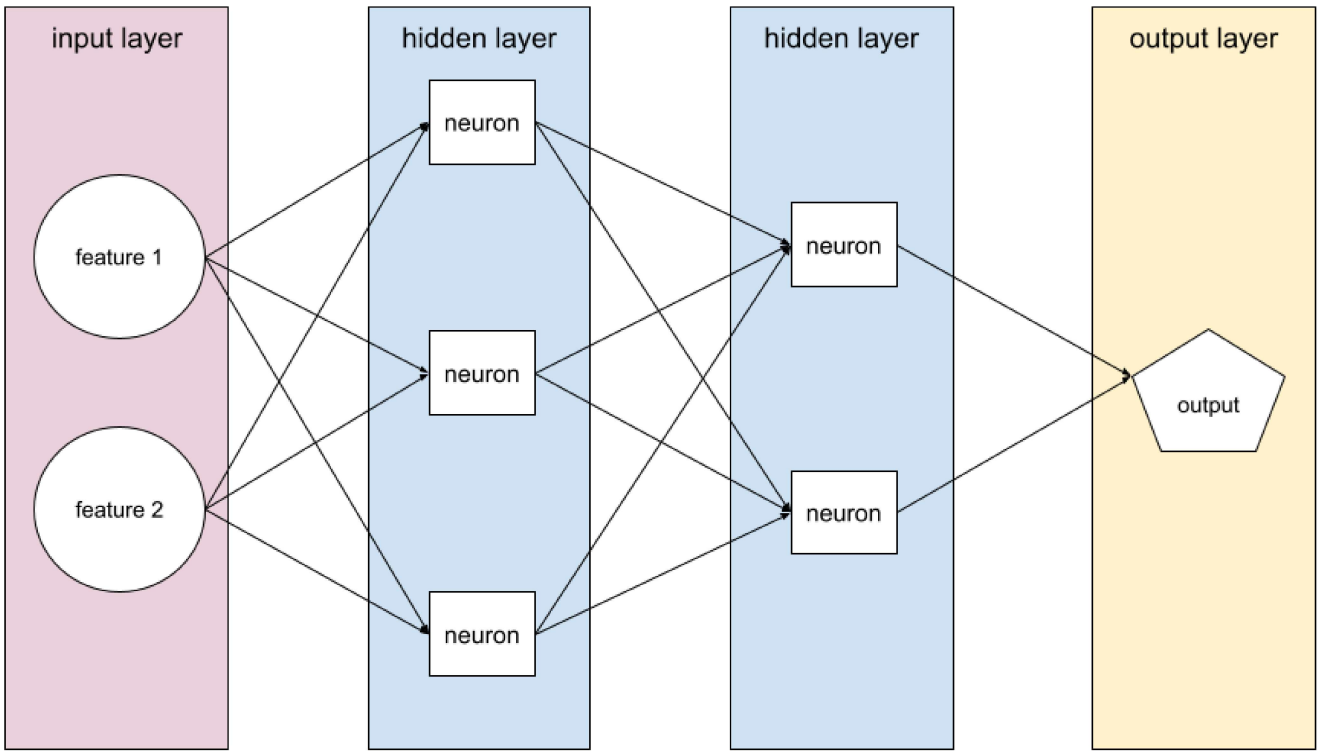
- A search space, which is a set of possible architectures.
- A fitness function, which is a measure of how well a particular architecture performs on a given task.

NAS algorithms often start with a small set of possible architectures and gradually expand the search space as the algorithm learns more about what architectures are effective. The fitness function is typically based on the performance of the architecture on a training set, and the algorithm is typically trained using a **reinforcement learning** (#reinforcement_learning) technique.

NAS algorithms have proven effective in finding high-performing architectures for a variety of tasks, including image **classification** (#classification_model), text classification, and machine translation.

neural network

A **model** (#model) containing at least one **hidden layer** (#hidden_layer). A **deep neural network** (#deep_neural_network) is a type of neural network containing more than one hidden layer. For example, the following diagram shows a deep neural network containing two hidden layers.



Each neuron in a neural network connects to all of the nodes in the next layer. For example, in the preceding diagram, notice that each of the three neurons in the first hidden layer separately connect to both of the two neurons in the second hidden layer.

Neural networks implemented on computers are sometimes called **artificial neural networks** to differentiate them from neural networks found in brains and other nervous systems.

Some neural networks can mimic extremely complex nonlinear relationships between different features and the label.

See also **convolutional neural network** (#convolutional_neural_network) and **recurrent neural network** (#recurrent_neural_network).

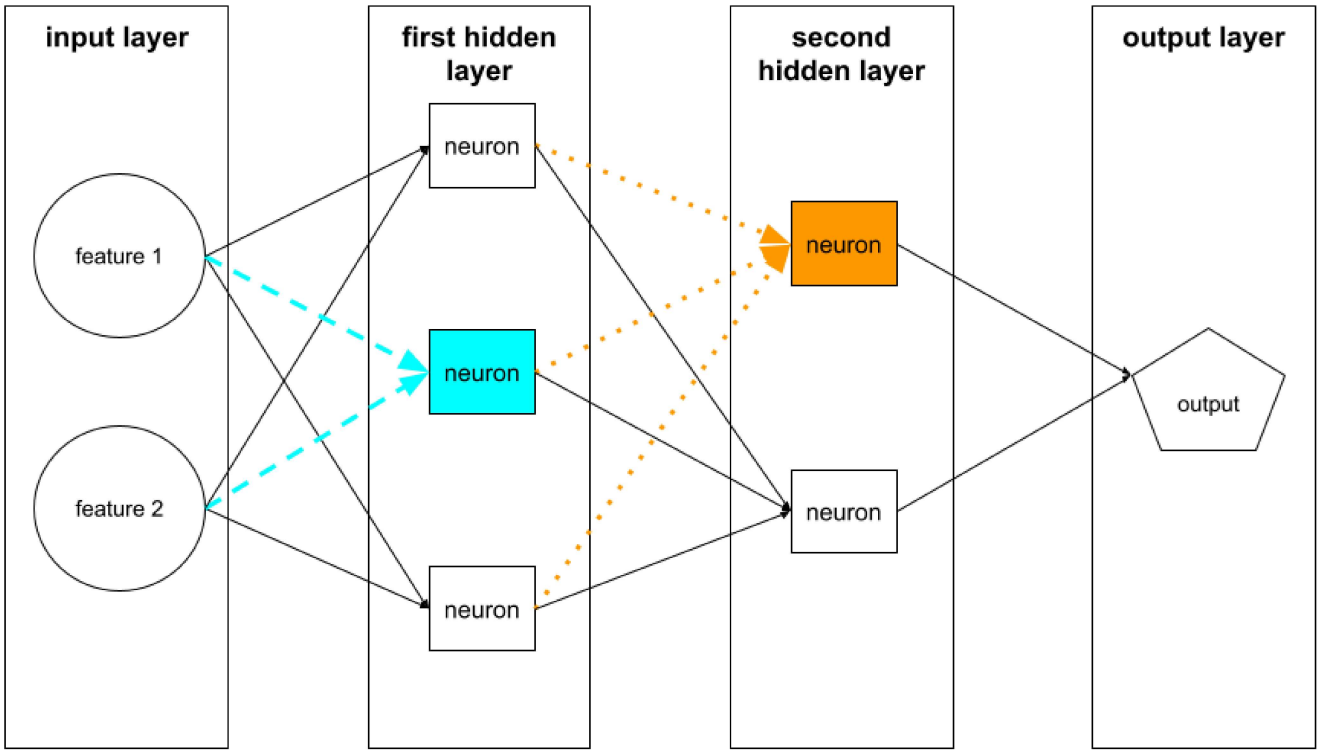
neuron

In machine learning, a distinct unit within a **hidden layer** (#hidden_layer) of a **neural network** (#neural_network). Each neuron performs the following two-step action:

1. Calculates the **weighted sum** (#weighted_sum) of input values multiplied by their corresponding weights.
2. Passes the weighted sum as input to an **activation function** (#activation_function).

A neuron in the first hidden layer accepts inputs from the feature values in the **input layer** (#input_layer). A neuron in any hidden layer beyond the first accepts inputs from the neurons in the preceding hidden layer. For example, a neuron in the second hidden layer accepts inputs from the neurons in the first hidden layer.

The following illustration highlights two neurons and their inputs.



A neuron in a neural network mimics the behavior of neurons in brains and other parts of nervous systems.

N-gram

An ordered sequence of N words. For example, *truly madly* is a 2-gram. Because order is relevant, *madly truly* is a different 2-gram than *truly madly*.

N	Name(s) for this kind of N-gram	Examples
2	bigram or 2-gram	*to go, go to, eat lunch, eat dinner*
3	trigram or 3-gram	*ate too much, three blind mice, the bell tolls*
4	4-gram	*walk in the park, dust in the wind, the boy ate lentils*

Many **natural language understanding** (#natural_language_understanding) models rely on N-grams to predict the next word that the user will type or say. For example, suppose a user typed *three blind*. An NLU model based on trigrams would likely predict that the user will next type *mice*.

Contrast N-grams with **bag of words** (#bag_of_words), which are unordered sets of words.

NLU

Abbreviation for **natural language understanding** (#natural_language_understanding).

node (neural network)

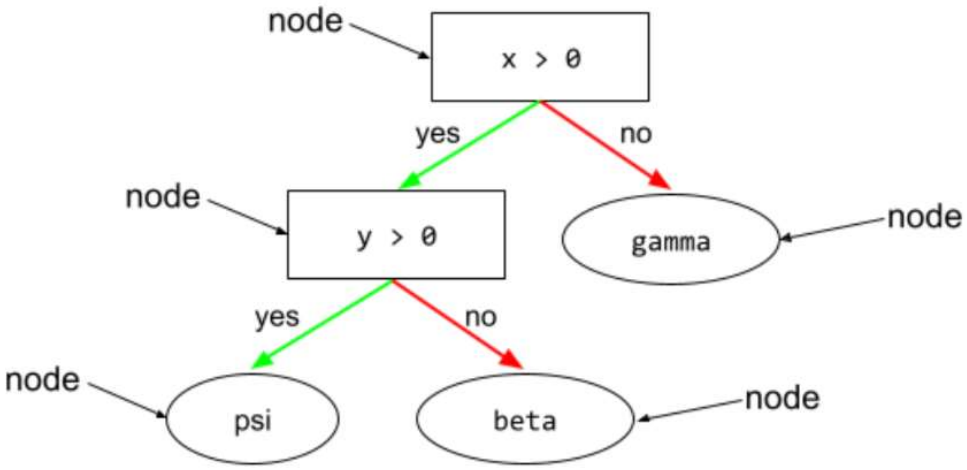
A **neuron** (#neuron) in a **hidden layer** (#hidden_layer).

node (TensorFlow graph)

An operation in a TensorFlow **graph** (#graph).

node (decision tree)

In a **decision tree** (#decision-tree), any **condition** (#condition) or **leaf** (#leaf).



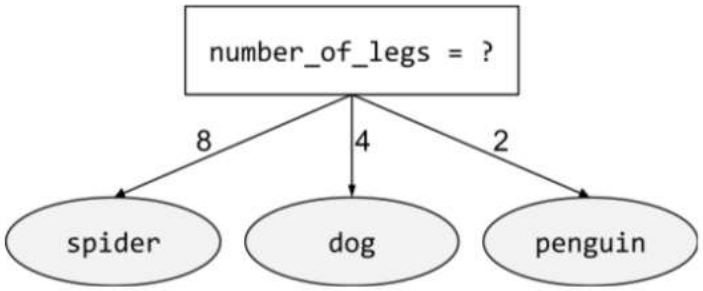
noise

Broadly speaking, anything that obscures the signal in a dataset. Noise can be introduced into data in a variety of ways. For example:

- Human raters make mistakes in labeling.
- Humans and instruments mis-record or omit feature values.

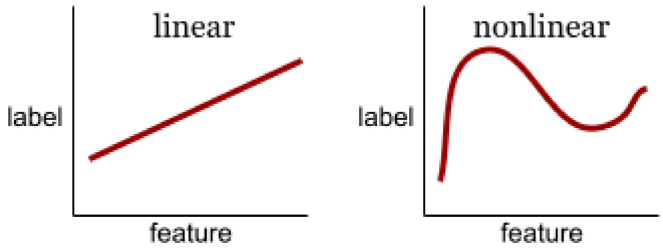
non-binary condition

A **condition** (#condition) containing more than two possible outcomes. For example, the following non-binary condition contains three possible outcomes:



nonlinear

A relationship between two or more variables that can't be represented solely through addition and multiplication. A *linear* relationship can be represented as a line; a *nonlinear* relationship can't be represented as a line. For example, consider two models that each relate a single feature to a single label. The model on the left is linear and the model on the right is nonlinear:



non-response bias

See **selection bias** (#selection_bias).

nonstationarity

A feature whose values change across one or more dimensions, usually time. For example, consider the following examples of nonstationarity:

- The number of swimsuits sold at a particular store varies with the season.
- The quantity of a particular fruit harvested in a particular region is zero for much of the year but large for a brief period.
- Due to climate change, annual mean temperatures are shifting.

Contrast with **stationarity** (#stationarity).

normalization

Broadly speaking, the process of converting a variable's actual range of values into a standard range of values, such as:

- -1 to +1
- 0 to 1
- the normal distribution

For example, suppose the actual range of values of a certain feature is 800 to 2,400. As part of **feature engineering** (#feature_engineering), you could normalize the actual values down to a standard range, such as -1 to +1.

Normalization is a common task in **feature engineering** (#feature_engineering). Models usually train faster (and produce better predictions) when every numerical feature in the **feature vector** (#feature_vector) has roughly the same range.

novelty detection

The process of determining whether a new (novel) example comes from the same distribution as the **training set** (#training_set). In other words, after training on the training set, novelty detection determines whether a *new* example (during inference or during additional training) is an **outlier** (#outliers).

Contrast with **outlier detection** (#outlier-detection).

numerical data

Features (#feature) represented as integers or real-valued numbers. For example, a house valuation model would probably represent the size of a house (in square feet or square meters) as numerical data. Representing a feature as numerical data indicates that the feature's values have a *mathematical* relationship to the label. That is, the number of square meters in a house probably has some mathematical relationship to the value of the house.

Not all integer data should be represented as numerical data. For example, postal codes in some parts of the world are integers; however, integer postal codes should not be represented as numerical data in models. That's because a postal code of `20000` is not twice (or half) as potent as a postal code of 10000. Furthermore, although different postal codes *do* correlate to different real estate values, we can't assume that real estate values at postal code 20000 are twice as valuable as real estate values at postal code 10000. Postal codes should be represented as **categorical data** (#categorical_data) instead.

Numerical features are sometimes called **continuous features** (#continuous_feature).

NumPy

An open-source math library (http://www.numpy.org/) that provides efficient array operations in Python. **pandas** (#pandas) is built on NumPy.

O

objective

A metric that your algorithm is trying to optimize.

objective function

The mathematical formula or **metric** (#metric) that a model aims to optimize. For example, the objective function for **linear regression** (#linear_regression) is usually **Mean Squared Loss** (#MSE). Therefore, when training a linear regression model, training aims to minimize Mean Squared Loss.

In some cases, the goal is to *maximize* the objective function. For example, if the objective function is accuracy, the goal is to maximize accuracy.

See also **loss** (#loss).

oblique condition

In a **decision tree** (#decision-tree), a **condition** (#condition) that involves more than one **feature** (#feature). For example, if height and width are both features, then the following is an oblique condition:

```
height > width
```

Contrast with **axis-aligned condition** (#axis-aligned-condition).

offline

Synonym for **static** (#static).

offline inference

The process of a model generating a batch of **predictions** (#prediction) and then caching (saving) those predictions. Apps can then access the desired prediction from the cache rather than rerunning the model.

For example, consider a model that generates local weather forecasts (predictions) once every four hours. After each model run, the system caches all the local weather forecasts. Weather apps retrieve the forecasts from the cache.

Offline inference is also called **static inference**.

Contrast with **online inference** (#online_inference).

one-hot encoding

Representing categorical data as a vector in which:

- One element is set to 1.
- All other elements are set to 0.

One-hot encoding is commonly used to represent strings or identifiers that have a finite set of possible values. For example, suppose a certain categorical feature named `Scandinavia` has five possible values:

- "Denmark"
- "Sweden"
- "Norway"
- "Finland"
- "Iceland"

One-hot encoding could represent each of the five values as follows:

country	Vector				
"Denmark"	1	0	0	0	0
"Sweden"	0	1	0	0	0
"Norway"	0	0	1	0	0
"Finland"	0	0	0	1	0
"Iceland"	0	0	0	0	1

Thanks to one-hot encoding, a model can learn different connections based on each of the five countries.

Representing a feature as **numerical data** (#numerical_data) is an alternative to one-hot encoding. Unfortunately, representing the Scandinavian countries numerically is not a good choice. For example, consider the following numeric representation:

- "Denmark" is 0

- "Sweden" is 1
- "Norway" is 2
- "Finland" is 3
- "Iceland" is 4

With numeric encoding, a model would interpret the raw numbers mathematically and would try to train on those numbers. However, Iceland isn't actually twice as much (or half as much) of something as Norway, so the model would come to some strange conclusions.

one-shot learning

A machine learning approach, often used for object classification, designed to learn effective classifiers from a single training example.

See also **few-shot learning** (#few-shot_learning) and **zero-shot learning** (#zero-shot-learning).

one-vs.-all

Given a classification problem with N classes, a solution consisting of N separate **binary classifiers** (#binary_classification)—one binary classifier for each possible outcome. For example, given a model that classifies examples as animal, vegetable, or mineral, a one-vs.-all solution would provide the following three separate binary classifiers:

- animal vs. not animal
- vegetable vs. not vegetable
- mineral vs. not mineral

online

Synonym for **dynamic** (#dynamic).

online inference

Generating **predictions** (#prediction) on demand. For example, suppose an app passes input to a model and issues a request for a prediction. A system using online inference responds to the request by running the model (and returning the prediction to the app).

Contrast with **offline inference** (#offline_inference).

operation (op)

In TensorFlow, any procedure that creates, manipulates, or destroys a **Tensor** (#tensor). For example, a matrix multiply is an operation that takes two Tensors as input and generates one Tensor as output.

Optax

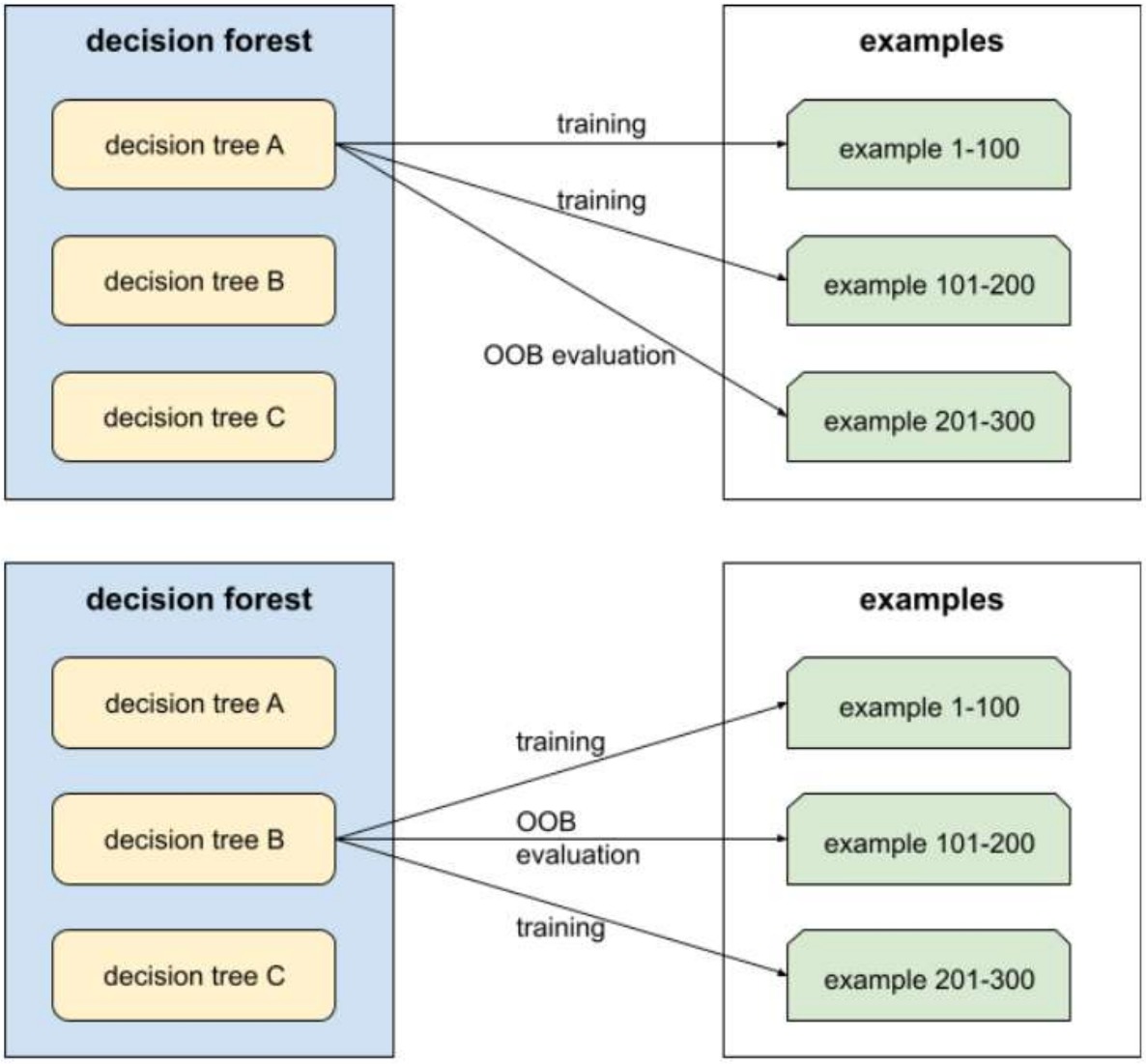
A gradient processing and optimization library for **JAX** (#JAX). Optax facilitates research by providing building blocks that can be recombined in custom ways to optimize parametric models such as deep neural networks. Other goals include:

- Providing readable, well-tested, efficient implementations of core components.

- Improving productivity by making it possible to combine low level ingredients into custom optimizers (or other gradient processing components).
- Accelerating adoption of new ideas by making it easy for anyone to contribute.

out-of-bag evaluation (OOB evaluation)

A mechanism for evaluating the quality of a **decision forest** (#decision-forest) by testing each **decision tree** (#decision-tree) against the **examples** (#example) *not* used during **training** (#training) of that decision tree. For example, in the following diagram, notice that the system trains each decision tree on about two-thirds of the examples and then evaluates against the remaining one-third of the examples.



Out-of-bag evaluation is a computationally efficient and conservative approximation of the **cross-validation** (#cross-validation) mechanism. In cross-validation, one model is trained for each cross-validation round (for example, 10 models are trained in a 10-fold cross-validation). With OOB evaluation, a single model is trained. Because **bagging** (#bagging) withholds some data from each tree during training, OOB evaluation can use that data to approximate cross-validation.

optimizer

A specific implementation of the **gradient descent** (#gradient_descent) algorithm. Popular optimizers include:

- **AdaGrad** (#AdaGrad), which stands for ADAptive GRADient descent.
- Adam, which stands for ADAptive with Momentum.

out-group homogeneity bias

The tendency to see out-group members as more alike than in-group members when comparing attitudes, values, personality traits, and other characteristics. **In-group** refers to people you interact with regularly; **out-group** refers to people you do not interact with regularly. If you create a dataset by asking people to provide attributes about out-groups, those attributes may be less nuanced and more stereotyped than attributes that participants list for people in their in-group.

For example, Lilliputians might describe the houses of other Lilliputians in great detail, citing small differences in architectural styles, windows, doors, and sizes. However, the same Lilliputians might simply declare that Brobdingnagians all live in identical houses.

Out-group homogeneity bias is a form of **group attribution bias** (#group_attribution_bias).

See also **in-group bias** (#in-group_bias).

outlier detection

The process of identifying **outliers** (#outliers) in a **training set** (#training_set).

Contrast with **novelty detection** (#novelty-detection).

outliers

Values distant from most other values. In machine learning, any of the following are outliers:

- Input data whose values are more than roughly 3 standard deviations from the mean.
- **Weights** (#weight) with high absolute values.
- Predicted values relatively far away from the actual values.

For example, suppose that `widget-price` is a feature of a certain model. Assume that the mean `widget-price` is 7 Euros with a standard deviation of 1 Euro. Examples containing a `widget-price` of 12 Euros or 2 Euros would therefore be considered outliers because each of those prices is five standard deviations from the mean.

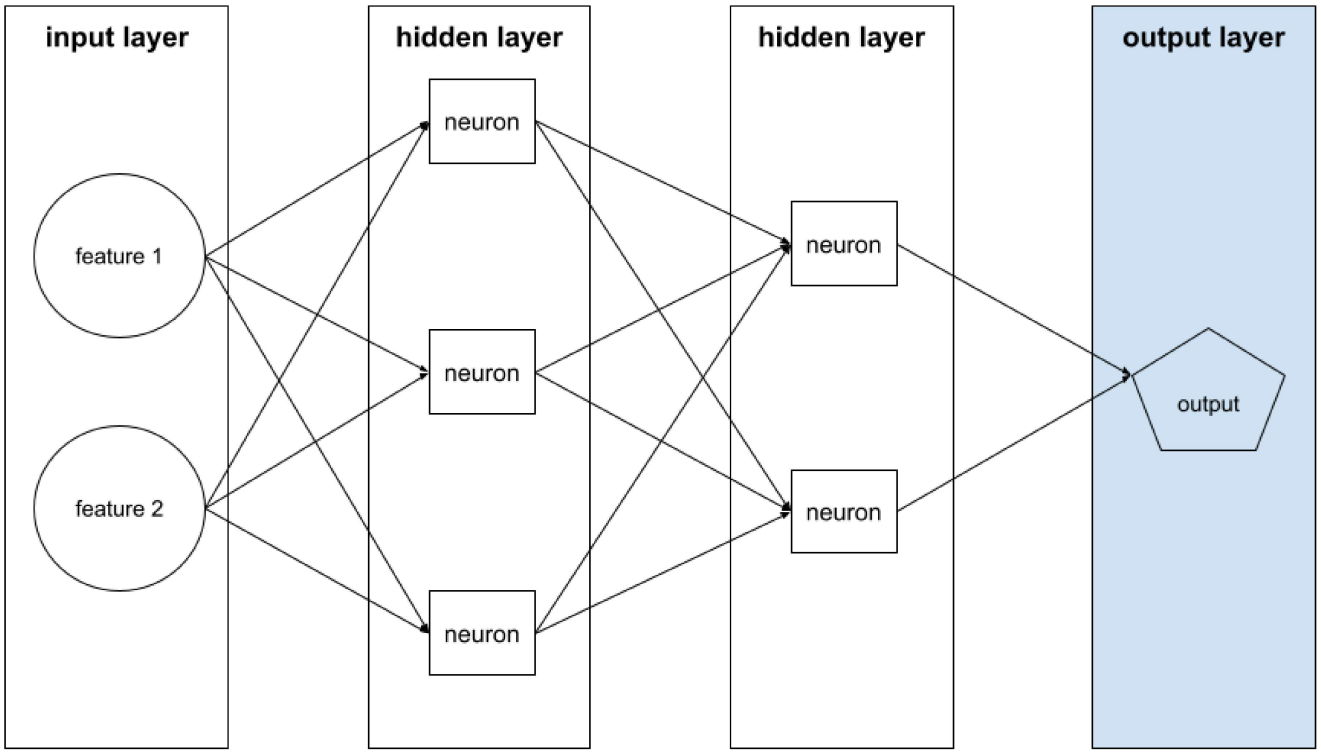
Outliers are often caused by typos or other input mistakes. In other cases, outliers aren't mistakes; after all, values five standard deviations away from the mean are rare but hardly impossible.

Outliers often cause problems in model training. **Clipping** (#clipping) is one way of managing outliers.

output layer

The "final" layer of a neural network. The output layer contains the prediction.

The following illustration shows a small deep neural network with an input layer, two hidden layers, and an output layer:



overfitting

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

Regularization (#regularization) can reduce overfitting. Training on a large and diverse training set can also reduce overfitting.

Click the icon for additional notes.

Overfitting is like strictly following advice from only your favorite teacher. You'll probably be successful in that teacher's class, but you might "overfit" to that teacher's ideas and be unsuccessful in other classes. Following advice from a mixture of teachers will enable you to adapt better to new situations.

oversampling

Reusing the **examples** (#example) of a **minority class** (#minority_class) in a **class-imbalanced dataset** (#class_imbalanced_data_set) in order to create a more balanced **training set** (#training_set).

For example, consider a **binary classification** (#binary_classification) problem in which the ratio of the **majority class** (#majority_class) to the minority class is 5,000:1. If the dataset contains a million examples, then the dataset contains only about 200 examples of the minority class, which might be too few examples for effective training. To overcome this deficiency, you might oversample (reuse) those 200 examples multiple times, possibly yielding sufficient examples for useful training.

You need to be careful about over **overfitting** (#overfitting) when oversampling.

Contrast with **undersampling** (#undersampling).

P

packed data

An approach for storing data more efficiently.

Packed data stores data either by using a compressed format or in some other way that allows it to be accessed more efficiently. Packed data minimizes the amount of memory and computation required to access it, leading to faster training and more efficient model inference.

Packed data is often used with other techniques, such as **data augmentation** (#data_augmentation) and **regularization** (#regularization), further improving the performance of **models** (#model).

pandas

A column-oriented data analysis API built on top of **numpy** (#numpy). Many machine learning frameworks, including TensorFlow, support pandas data structures as inputs. See the pandas documentation (http://pandas.pydata.org/) for details.

parameter

The **weights** (#weight) and **biases** (#bias) that a model learns during **training** (#training). For example, in a **linear regression** (#linear_regression) model, the parameters consist of the bias (b) and all the weights (w_1, w_2, and so on) in the following formula:

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

In contrast, **hyperparameter** (#hyperparameter) are the values that *you* (or a hyperparameter turning service) supply to the model. For example, **learning rate** (#learning_rate) is a hyperparameter.

Parameter Server (PS)

A job that keeps track of a model's **parameters** (#parameter) in a distributed setting.

parameter update

The operation of adjusting a model's **parameters** (#parameter) during training, typically within a single iteration of **gradient descent** (#gradient_descent).

partial derivative

A derivative in which all but one of the variables is considered a constant. For example, the partial derivative of *f(x, y)* with respect to *x* is the derivative of *f* considered as a function of *x* alone (that is, keeping *y* constant). The partial derivative of *f* with respect to *x* focuses only on how *x* is changing and ignores all other variables in the equation.

participation bias

Synonym for non-response bias. See **selection bias** (#selection_bias).

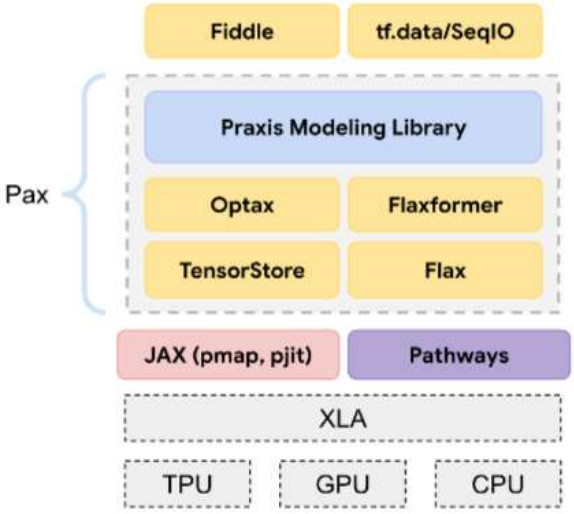
partitioning strategy

The algorithm by which variables are divided across **parameter servers** (#Parameter_Server).

Pax

A programming framework designed for training large-scale **neural network** (#neural-network) **models** (#model) so large that they span multiple **TPU** (#TPU) **accelerator chip** (#accelerator-chip) **slices** (#TPU_slice) or **pods** (#TPU_Pod).

Pax is built on **Flax** (#flax), which is built on **JAX** (#JAX).

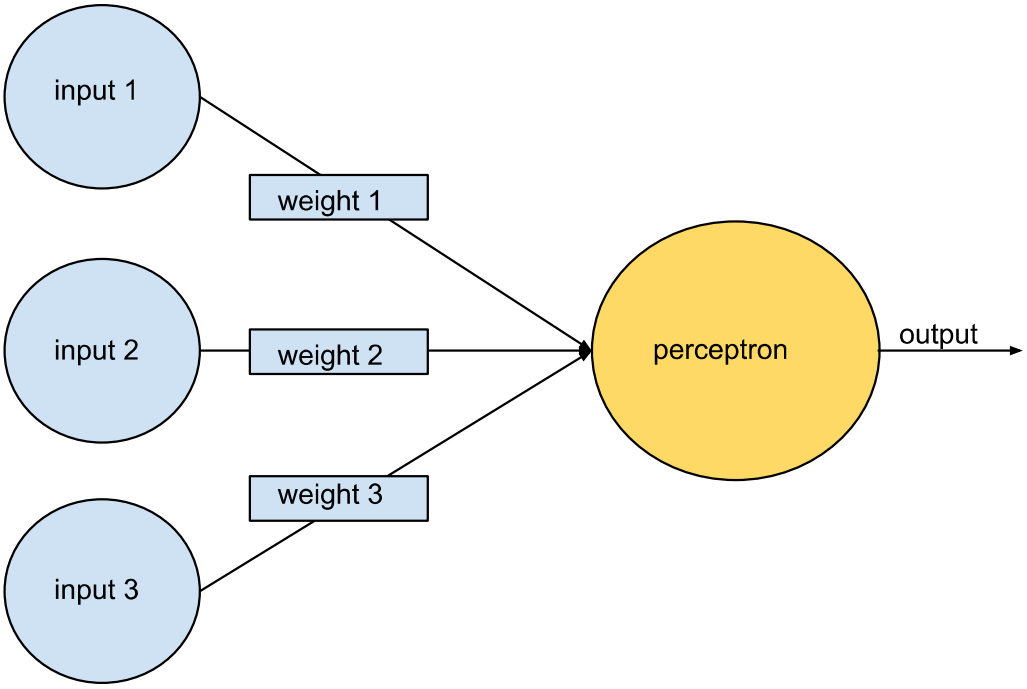


perceptron

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** (#ReLU), **sigmoid** (#sigmoid-function), or tanh (https://wikipedia.org/wiki/Hyperbolic_functions). 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 **neurons** (#neuron) in **neural networks** (#neural-network).

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** (#model)? That is, how good are the model's predictions?

permutation variable importances

A type of **variable importance** (#variable-importances) that evaluates the increase in the prediction error of a model *after* permuting the feature's values. Permutation variable importance is a model agnostic metric.

perplexity

One measure of how well a **model** (#model) is accomplishing its task. For example, suppose your task is to read the first few letters of a word a user is typing on a smartphone keyboard, and to offer a list of possible completion words. Perplexity, P, for this task is approximately the number of guesses you need to offer in order for your list to contain the actual word the user is trying to type.

Perplexity is related to **cross-entropy** (#cross-entropy) as follows:

$$P = 2^{-\text{cross entropy}}$$

pipeline

The infrastructure surrounding a machine learning algorithm. A pipeline includes gathering the data, putting the data into training data files, training one or more models, and exporting the models to production.

pipelining

A form of **model parallelism** (#model-parallelism) in which a model's processing is divided into consecutive stages and each stage is executed on a different device. While a stage is processing one batch, the preceding stage can work on the next batch.

See also **staged training** (#staged-training).

pjit

A **JAX** (#JAX) function that splits code to run across multiple **accelerator chips** (#accelerator-chip). The user passes a function to pjit, which returns a function that has the equivalent semantics but is compiled into an **XLA** (#XLA) computation that runs across multiple devices (such as GPUs or **TPU** (#TPU) cores).

pjit enables users to shard computations without rewriting them by using the **SPMD** (#single-program) partitioner.

As of March 2023, `pjit` has been merged with `jit`. Refer to Distributed arrays and automatic parallelization (https://jax.readthedocs.io/en/latest/notebooks/Distributed_arrays_and_automatic_parallelization.html) for more details.

pmap

A **JAX** (#JAX) function that executes copies of an input function on multiple underlying hardware devices (CPUs, GPUs, or **TPUs** (#TPU)), with different input values. pmap relies on **SPMD** (#single-program).

policy RL

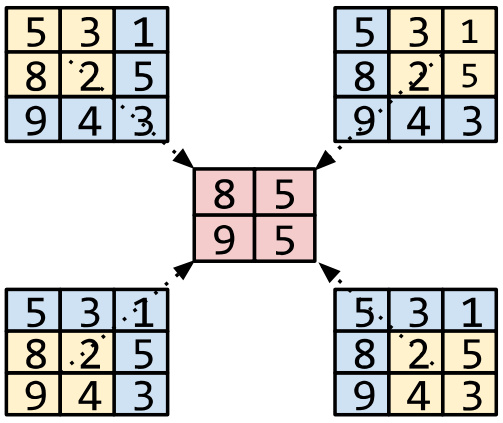
In reinforcement learning, an **agent's** (#agent) probabilistic mapping from **states** (#state) to **actions** (#action).

pooling

Reducing a matrix (or matrices) created by an earlier **convolutional layer** (#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** (#stride). 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** (#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**.

positional encoding

A technique to add information about the *position* of a token in a sequence to the token's embedding. **Transformer models** (#transformer) use positional encoding to better understand the relationship between different parts of the sequence.

A common implementation of positional encoding uses a sinusoidal function. (Specifically, the frequency and amplitude of the sinusoidal function are determined by the position of the token in the sequence.) This technique enables a Transformer model to learn to attend to different parts of the sequence based on their position.

positive class

The class you are testing for.

For example, the positive class in a cancer model might be "tumor." The positive class in an email classifier might be "spam."

Contrast with **negative class** (#negative_class).

Click the icon for additional notes.

The term **positive class** can be confusing because the "positive" outcome of many tests is often an undesirable result. For example, the positive class in many medical tests corresponds to tumors or diseases. In general, you want a doctor to tell you, "Congratulations! Your test results were negative." Regardless, the positive class is the event that the test is seeking to find.

Admittedly, you're simultaneously testing for both the positive and negative classes.

post-processing

Adjusting 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** (#equality_of_opportunity) is maintained for some attribute by checking that the **true positive rate** (#TP_rate) is the same for all values of that attribute.

PR AUC (area under the PR curve)

Area under the interpolated **precision-recall curve** (#precision-recall_curve), obtained by plotting (recall, precision) points for different values of the **classification threshold** (#classification_threshold). Depending on how it's calculated, PR AUC may be equivalent to the **average precision** (#average_precision) of the model.

Praxis

A core, high-performance ML library of **Pax** (#pax). Praxis is often called the "Layer library".

Praxis contains not just the definitions for the Layer class, but most of its supporting components as well, including:

- data inputs
- configuration libraries (HParam and **Fiddle** (#fiddle))
- **optimizers** (#optimizer)

Praxis provides the definitions for the Model class.

---

precision

A metric for **classification models** (#classification_model) that answers the following question:

> When the model predicted the **positive class** (#positive_class), what percentage of the predictions were correct?

Here is the formula:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

where:

- true positive means the model *correctly* predicted the positive class.
- false positive means the model *mistakenly* predicted the positive class.

For example, suppose a model made 200 positive predictions. Of these 200 positive predictions:

- 150 were true positives.
- 50 were false positives.

In this case:

$$\text{Precision} = \frac{150}{150 + 50} = 0.75$$

Contrast with **accuracy** (#accuracy) and **recall** (#recall).

precision-recall curve

A curve of **precision** (#precision) vs. **recall** (#recall) at different **classification thresholds** (#classification_threshold).

prediction

A model's output. For example:

- The prediction of a binary classification model is either the positive class or the negative class.
- The prediction of a multi-class classification model is one class.
- The prediction of a linear regression model is a number.

prediction bias

A value indicating how far apart the average of **predictions** (#prediction) is from the average of **labels** (#label) in the dataset.

Not to be confused with the **bias term** (#bias) in machine learning models or with **bias in ethics and fairness** (#bias_ethics).

predictive parity

A **fairness metric** (#fairness_metric) that checks whether, for a given classifier, the **precision** (#precision) rates are equivalent for subgroups under consideration.

For example, a model that predicts college acceptance would satisfy predictive parity for nationality if its precision rate is the same for Lilliputians and Brobdingnagians.

Predictive parity is sometime also called *predictive rate parity*.

See "Fairness Definitions Explained" (http://fairware.cs.umass.edu/papers/Verma.pdf) (section 3.2.1) for a more detailed discussion of predictive parity.

predictive rate parity

Another name for **predictive parity** (#predictive_parity).

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** (#sensitive_attribute) as possible. Preprocessing can help satisfy **fairness constraints** (#fairness_constraint).

pre-trained model

Models or model components (such as **embedding vector** (#embedding_vector)) that have been already been trained. Sometimes, you'll feed pre-trained embedding vectors into a **neural network** (#neural_network). Other times, your model will train the embedding vectors itself rather than rely on the pre-trained embeddings.

prior belief

What you believe about the data before you begin training on it. For example, **L_2 regularization** (#L2_regularization) relies on a prior belief that **weights** (#weight) should be small and normally distributed around zero.

probabilistic regression model

A **regression model** (#regression_model) that uses not only the **weights** (#weight) for each **feature** (#feature), but also the uncertainty of those weights. A probabilistic regression model generates a prediction and the uncertainty of that prediction. For example, a probabilistic regression model might yield a prediction of 325 with a standard deviation of 12. For more information about probabilistic regression models, see this Colab on tensorflow.org (https://www.tensorflow.org/probability/examples/Probabilistic_Layers_Regression).

proxy (sensitive attributes) ⚖

An attribute used as a stand-in for a **sensitive attribute** (#sensitive_attribute). For example, an individual's postal code might be used as a proxy for their income, race, or ethnicity.

proxy labels

Data used to approximate labels not directly available in a dataset.

For example, suppose you must train a model to predict employee stress level. Your dataset contains a lot of predictive features but doesn't contain a label named *stress level.* Undaunted, you pick "workplace accidents" as a proxy label for stress level. After all, employees under high stress get into more accidents than calm employees. Or do they? Maybe workplace accidents actually rise and fall for multiple reasons.

As a second example, suppose you want *is it raining?* to be a Boolean label for your dataset, but your dataset doesn't contain rain data. If photographs are available, you might establish pictures of people carrying umbrellas as a proxy label for *is it raining?* Is that a good proxy label? Possibly, but people in some cultures may be more likely to carry umbrellas to protect against sun than the rain.

Proxy labels are often imperfect. When possible, choose actual labels over proxy labels. That said, when an actual label is absent, pick the proxy label very carefully, choosing the least horrible proxy label candidate.

pure function

A function whose outputs are based only on its inputs, and that has no side effects. Specifically, a pure function does not use or change any global state, such as the contents of a file or the value of a variable outside the function.

Pure functions can be used to create thread-safe code, which is beneficial when sharding **model** (#model) code across multiple **accelerator chips** (#accelerator-chip).

JAX's (#JAX) function transformation methods require that the input functions are pure functions.

Q

Q-function RL

In **reinforcement learning** (#reinforcement_learning), the function that predicts the expected **return** (#return) from taking an **action** (#action) in a **state** (#state) and then following a given **policy** (#policy).

Q-function is also known as **state-action value function**.

Q-learning RL

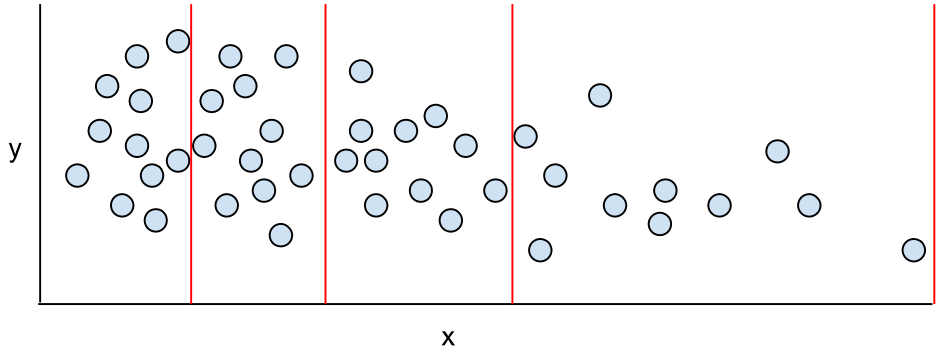
In **reinforcement learning** (#reinforcement_learning), an algorithm that allows an **agent** (#agent) to learn the optimal **Q-function** (#q-function) of a **Markov decision process** (#markov_decision_process) by applying the **Bellman equation** (#bellman_equation). The Markov decision process models an **environment** (#environment).

quantile

Each bucket in **quantile bucketing** (#quantile_bucketing).

quantile bucketing

Distributing a feature's values into **buckets** (#bucketing) so that each bucket contains the same (or almost the same) number of examples. For example, the following figure divides 44 points into 4 buckets, each of which contains 11 points. In order for each bucket in the figure to contain the same number of points, some buckets span a different width of x-values.



quantization

Overloaded term that could be used in either of two ways:

- Implementing **quantile bucketing** (#quantile_bucketing) on a particular **feature** (#feature).
- Transforming data into zeroes and ones for quicker storing, training, and inferring. As Boolean data is more robust to noise and errors than other formats, quantization can improve model correctness. Quantization techniques include rounding, truncating, and **binning** (#binning).

queue

A TensorFlow **Operation** (#Operation) that implements a queue data structure. Typically used in I/O.

R

random forest

An **ensemble** (#ensemble) of **decision trees** (#decision-tree) in which each decision tree is trained with a specific random noise, such as **bagging** (#bagging).

Random forests are a type of **decision forest** (#decision-forest).

random policy

RL

In **reinforcement learning** (#reinforcement_learning), a **policy** (#policy) that chooses an **action** (#action) at random.

ranking

A type of **supervised learning** (#supervised_machine_learning) whose objective is to order a list of items.

rank (ordinality)

The ordinal position of a class in a machine learning problem that categorizes classes from highest to lowest. For example, a behavior ranking system could rank a dog's rewards from highest (a steak) to lowest (wilted kale).

rank (Tensor)

The number of dimensions in a **Tensor** (#tensor). For instance, a scalar has rank 0, a vector has rank 1, and a matrix has rank 2.

Not to be confused with **rank (ordinality)** (#rank_ordinality).

rater

A human who provides **labels** (#label) for **examples** (#example). "Annotator" is another name for rater.

recall

A metric for **classification models** (#classification_model) that answers the following question:

> When **ground truth** (#ground_truth) was the **positive class** (#positive_class), what percentage of predictions did the model correctly identify as the positive class?

Here is the formula:

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

where:

- true positive means the model *correctly* predicted the positive class.
- false negative means that the model *mistakenly* predicted the **negative class** (#negative_class).

For instance, suppose your model made 200 predictions on examples for which ground truth was the positive class. Of these 200 predictions:

- 180 were true positives.
- 20 were false negatives.

In this case:

$$\text{Recall} = \frac{180}{180 + 20} = 0.9$$

Click the icon for notes about class-imbalanced datasets.

Recall is particularly useful for determining the predictive power of classification models in which the positive class is rare. For example, consider a **class-imbalanced dataset** (#class_imbalanced_data_set) in which the positive class for a certain disease occurs in only 10 patients out of a million. Suppose your model makes five million predictions that yield the following outcomes:

- 30 True Positives
- 20 False Negatives
- 4,999,000 True Negatives
- 950 False Positives

The recall of this model is therefore:

```
recall = TP / (TP + FN)
recall = 30 / (30 + 20) = 0.6 = 60%
```

By contrast, the accuracy (#accuracy) of this model is:

```
accuracy = (TP + TN) / (TP + TN + FP + FN)
accuracy = (30 + 4,999,000) / (30 + 4,999,000 + 950 + 20) = 99.98%
```

That high value of accuracy looks impressive but is essentially meaningless. Recall is a much more useful metric for class-imbalanced datasets than accuracy.

---

recommendation system

A system that selects for each user a relatively small set of desirable **items** (#items) from a large corpus. For example, a video recommendation system might recommend two videos from a corpus of 100,000 videos, selecting *Casablanca* and *The Philadelphia Story* for one user, and *Wonder Woman* and *Black Panther* for another. A video recommendation system might base its recommendations on factors such as:

- Movies that similar users have rated or watched.
- Genre, directors, actors, target demographic...

Rectified Linear Unit (ReLU)

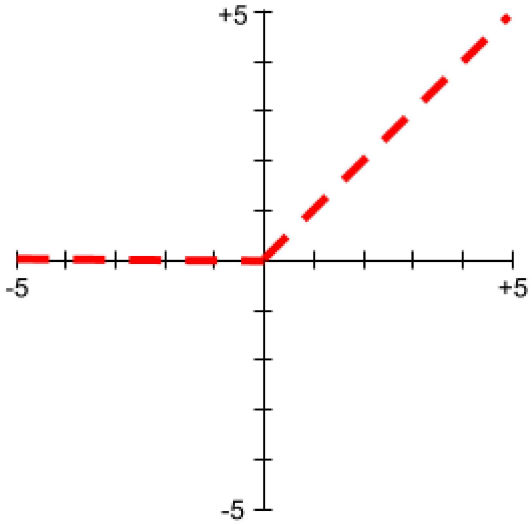
An **activation function** (#activation_function) with the following behavior:

- If input is negative or zero, then the output is 0.
- If input is positive, then the output is equal to the input.

For example:

- If the input is -3, then the output is 0.
- If the input is +3, then the output is 3.0.

Here is a plot of ReLU:

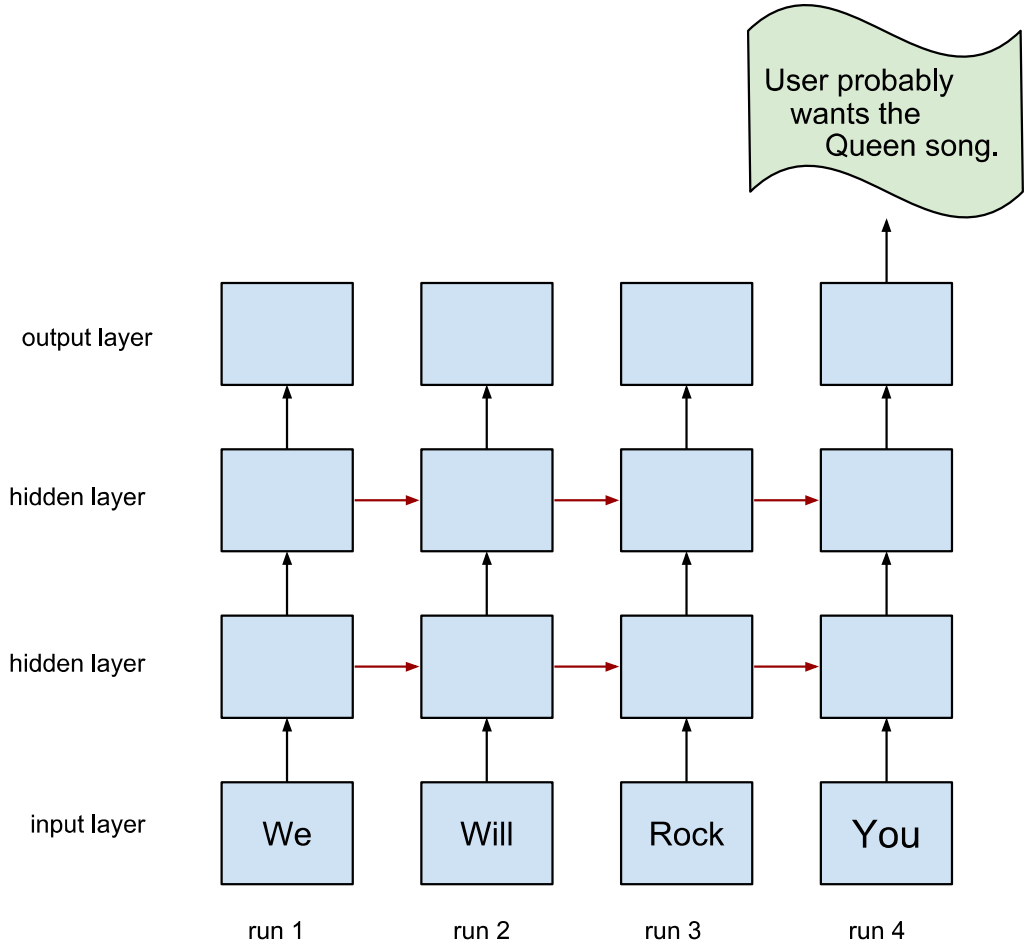


ReLU is a very popular activation function. Despite its simple behavior, ReLU still enables a neural network to learn **nonlinear** (#nonlinear) relationships between **features** (#feature) and the **label** (#label).

recurrent neural network

A **neural network** (#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.



regression model

Informally, a model that generates a numerical prediction. (In contrast, a **classification model** (#classification_model) generates a class prediction.) For example, the following are all regression models:

- A model that predicts a certain house's value, such as 423,000 Euros.
- A model that predicts a certain tree's life expectancy, such as 23.2 years.
- A model that predicts the amount of rain that will fall in a certain city over the next six hours, such as 0.18 inches.

Two common types of regression models are:

- **Linear regression** (#linear_regression), which finds the line that best fits label values to features.
- **Logistic regression** (#logistic_regression), which generates a probability between 0.0 and 1.0 that a system typically then maps to a class prediction.

Not every model that outputs numerical predictions is a regression model. In some cases, a numeric prediction is really just a classification model that happens to have numeric class names. For example, a model that predicts a numeric postal code is a classification model, not a regression model.

regularization

Any mechanism that reduces **overfitting** (#overfitting). Popular types of regularization include:

- **L_1 regularization** (#L1_regularization)
- **L_2 regularization** (#L2_regularization)
- **dropout regularization** (#dropout_regularization)
- **early stopping** (#early_stopping) (this is not a formal regularization method, but can effectively limit overfitting)

Regularization can also be defined as the penalty on a model's complexity.

Click the icon for additional notes.

Regularization is counterintuitive. Increasing regularization usually *increases* training loss, which is confusing because, well, isn't the goal to *minimize* training loss?

Actually, no. The goal isn't to minimize training loss. The goal is to make excellent predictions on real-world examples. Remarkably, even though increasing regularization increases training loss, it usually helps models make better predictions on real-world examples.

---

regularization rate

A number that specifies the relative importance of **regularization** (#regularization) during training. Raising the regularization rate reduces **overfitting** (#overfitting) but may reduce the model's predictive power. Conversely, reducing or omitting the regularization rate increases overfitting.

Click the icon to see the math.

The regularization rate is usually represented as the Greek letter lambda. The following simplified **loss** (#loss) equation shows lambda's influence:

$$\text{minimize}(\text{loss function} + \lambda(\text{regularization}))$$

where *regularization* is any regularization mechanism, including;

- **L_1 regularization** (#L1_regularization)
- **L_2 regularization** (#L2_regularization)

---

reinforcement learning (RL) RL

A family of algorithms that learn an optimal **policy** (#policy), whose goal is to maximize **return** (#return) when interacting with an **environment** (#environment). For example, the ultimate reward of most games is victory. Reinforcement learning systems can become expert at playing complex games by evaluating sequences of previous game moves that ultimately led to wins and sequences that ultimately led to losses.

ReLU

Abbreviation for **Rectified Linear Unit** (#ReLU).

replay buffer

RL

In **DQN** (#deep_q-network)-like algorithms, the memory used by the agent to store state transitions for use in **experience replay** (#experience_replay).

reporting bias

The fact that the frequency with which people write about actions, outcomes, or properties is not a reflection of their real-world frequencies or the degree to which a property is characteristic of a class of individuals. Reporting bias can influence the composition of data that machine learning systems learn from.

For example, in books, the word *laughed* is more prevalent than *breathed*. A machine learning model that estimates the relative frequency of laughing and breathing from a book corpus would probably determine that laughing is more common than breathing.

representation

The process of mapping data to useful **features** (#feature).

re-ranking

The final stage of a **recommendation system** (#recommendation_system), during which scored items may be re-graded according to some other (typically, non-ML) algorithm. Re-ranking evaluates the list of items generated by the **scoring** (#scoring) phase, taking actions such as:

- Eliminating items that the user has already purchased.
- Boosting the score of fresher items.

return RL

In reinforcement learning, given a certain policy and a certain state, the return is the sum of all **rewards** (#reward) that the **agent** (#agent) expects to receive when following the **policy** (#policy) from the **state** (#state) to the end of the **episode** (#episode). The agent accounts for the delayed nature of expected rewards by discounting rewards according to the state transitions required to obtain the reward.

Therefore, if the discount factor is γ, and $r_0, \ldots, r_N$ denote the rewards until the end of the episode, then the return calculation is as follows:

$$\text{Return} = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^{N-1} r_{N-1}$$

reward RL

In reinforcement learning, the numerical result of taking an **action** (#action) in a **state** (#state), as defined by the **environment** (#environment).

ridge regularization

Synonym for **L_2 regularization** (#L2_regularization). The term **ridge regularization** is more frequently used in pure statistics contexts, whereas **L_2 regularization** is used more often in machine learning.

RNN

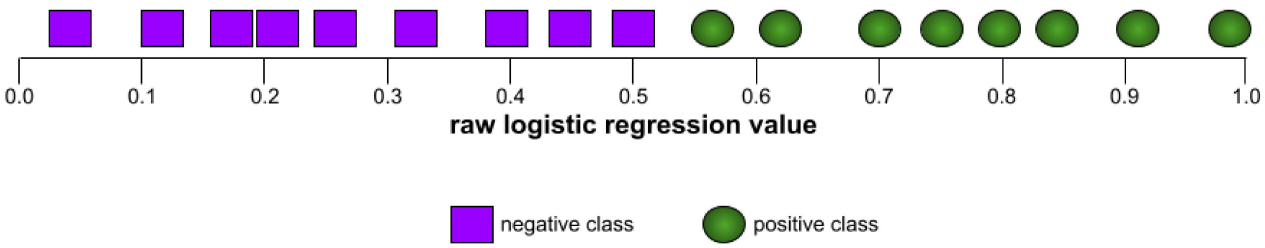
Abbreviation for **recurrent neural networks** (#recurrent_neural_network).

ROC (receiver operating characteristic) Curve

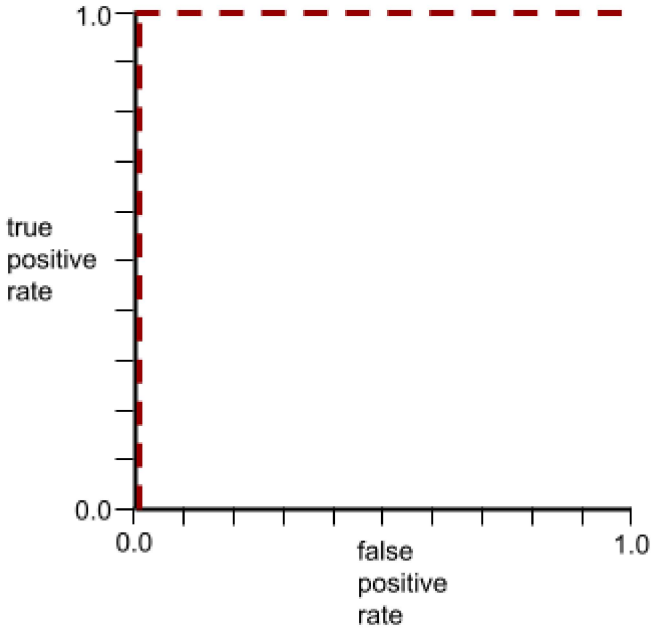
A graph of **true positive rate** (#TP_rate) vs. **false positive rate** (#FP_rate) for different **classification thresholds** (#classification_threshold) in binary classification.

The shape of an ROC curve suggests a binary classification model's ability to separate positive classes from negative classes. Suppose, for example, that a binary classification model

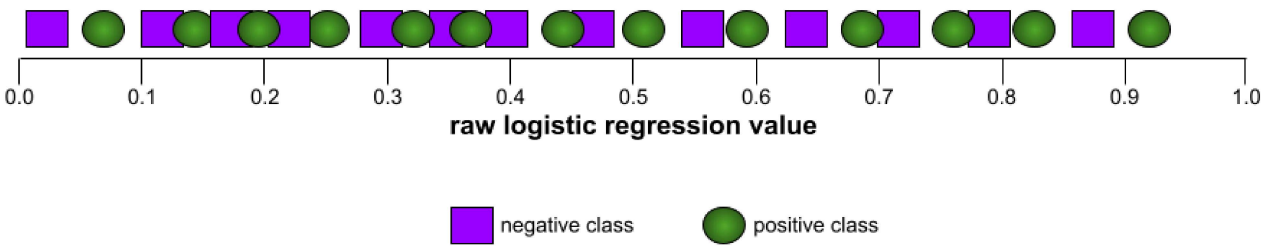
perfectly separates all the negative classes from all the positive classes:



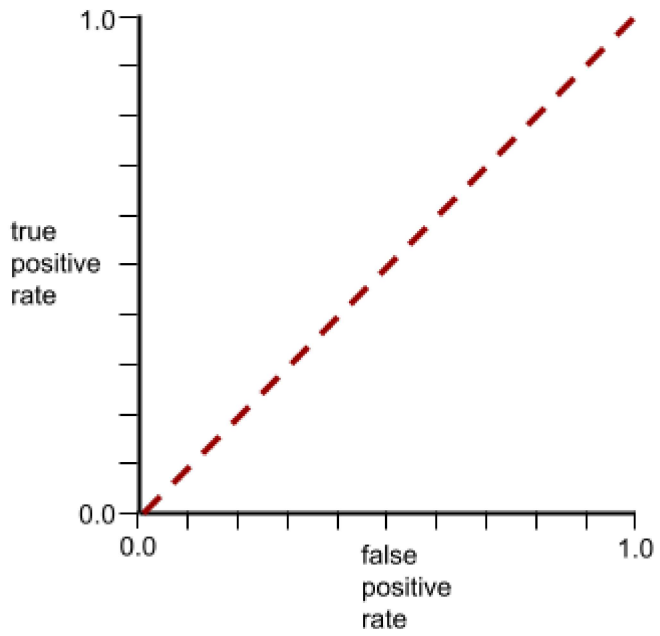
The ROC curve for the preceding model looks as follows:



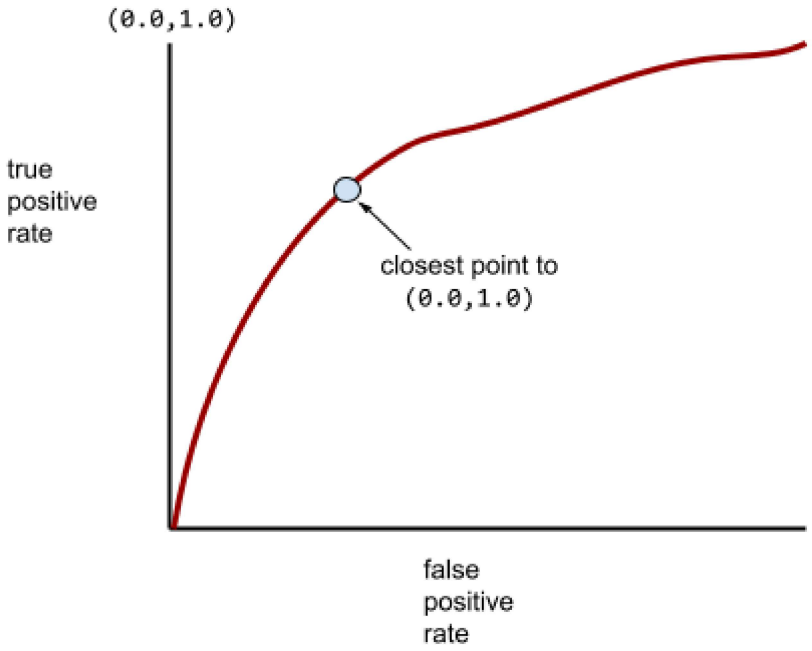
In contrast, the following illustration graphs the raw logistic regression values for a terrible model that can't separate negative classes from positive classes at all:



The ROC curve for this model looks as follows:



Meanwhile, back in the real world, most binary classification models separate positive and negative classes to some degree, but usually not perfectly. So, a typical ROC curve falls somewhere between the two extremes:



The point on an ROC curve closest to (0.0,1.0) theoretically identifies the ideal classification threshold. However, several other real-world issues influence the selection of the ideal

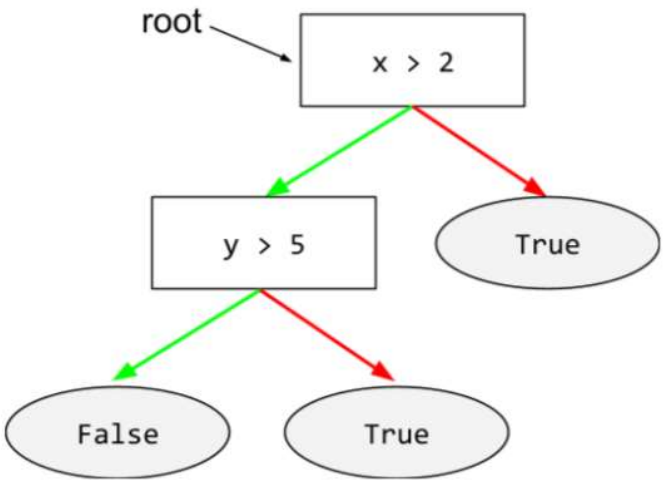
classification threshold. For example, perhaps false negatives cause far more pain than false positives.

A numerical metric called **AUC** (#AUC) summarizes the ROC curve into a single floating-point value.

root

♠♠♠

The starting **node** (#node-decision-tree) (the first **condition** (#condition)) in a **decision tree** (#decision-tree). By convention, diagrams put the root at the top of the decision tree. For example:



root directory

The directory you specify for hosting subdirectories of the TensorFlow checkpoint and events files of multiple models.

Root Mean Squared Error (RMSE)

The square root of the **Mean Squared Error** (#MSE).

rotational invariance

In an image classification problem, an algorithm's ability to successfully classify images even when the orientation of the image changes. For example, the algorithm can still identify a tennis racket whether it is pointing up, sideways, or down. Note that rotational invariance is not always desirable; for example, an upside-down 9 should not be classified as a 9.

See also **translational invariance** (#translational_invariance) and **size invariance** (#size_invariance).

R-squared

A **regression** (#regression_model) metric indicating how much variation in a **label** (#label) is due to an individual feature or to a feature set. R-squared is a value between 0 and 1, which you can interpret as follows:

- An R-squared of 0 means that none of a label's variation is due to the feature set.
- An R-squared of 1 means that all of a label's variation is due to the feature set.
- An R-squared between 0 and 1 indicates the extent to which the label's variation can be predicted from a particular feature or the feature set. For example, an R-squared of 0.10 means that 10 percent of the variance in the label is due to the feature set, an R-squared of 0.20 means that 20 percent is due to the feature set, and so on.

R-squared is the square of the Pearson correlation coefficient (https://wikipedia.org/wiki/Correlation_coefficient) between the values that a model predicted and **ground truth** (#ground_truth).

S

sampling bias

See **selection bias** (#selection_bias).

sampling with replacement

A method of picking items from a set of candidate items in which the same item can be picked multiple times. The phrase "with replacement" means that after each selection, the selected item is returned to the pool of candidate items. The inverse method, **sampling without replacement**, means that a candidate item can only be picked once.

For example, consider the following fruit set:

```
fruit = {kiwi, apple, pear, fig, cherry, lime, mango}
```

Suppose that the system randomly picks `fig` as the first item. If using sampling with replacement, then the system picks the second item from the following set:

```
fruit = {kiwi, apple, pear, fig, cherry, lime, mango}
```

Yes, that's the same set as before, so the system could potentially pick `fig` again.

If using sampling without replacement, once picked, a sample can't be picked again. For example, if the system randomly picks `fig` as the first sample, then `fig` can't be picked again. Therefore, the system picks the second sample from the following (reduced) set:

```
fruit = {kiwi, apple, pear, cherry, lime, mango}
```

Click the icon for additional notes.

The word *replacement* in **sampling with replacement** confuses many people. In English, *replacement* means "substitution." However, **sampling with replacement** actually uses the French definition for *replacement*, which means "putting something back." The English word *replacement* is translated as the French word *remplacement*.

SavedModel

The recommended format for saving and recovering TensorFlow models. SavedModel is a language-neutral, recoverable serialization format, which enables higher-level systems and tools to produce, consume, and transform TensorFlow models.

See the Saving and Restoring chapter (https://www.tensorflow.org/guide/saved_model) in the TensorFlow Programmer's Guide for complete details.

Saver

A TensorFlow object (https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/Saver) responsible for saving model checkpoints.

scalar

A single number or a single string that can be represented as a **tensor** (#tensor) of **rank** (#rank) 0. For example, the following lines of code each create one scalar in TensorFlow:

```
breed = tf.Variable("poodle", tf.string)
temperature = tf.Variable(27, tf.int16)
precision = tf.Variable(0.982375101275, tf.float64)
```

scaling

Any mathematical transform or technique that shifts the range of a label and/or feature value. Some forms of scaling are very useful for transformations like **normalization** (#normalization).

Common forms of scaling useful in Machine Learning include:

- linear scaling, which typically uses a combination of subtraction and division to replace the original value with a number between -1 and +1 or between 0 and 1.
- logarithmic scaling, which replaces the original value with its logarithm.
- **Z-score normalization** (#Z-score-normalization), which replaces the original value with a floating-point value representing the number of standard deviations from that feature's mean.

scikit-learn

A popular open-source machine learning platform. See scikit-learn.org (http://scikit-learn.org/).

scoring

The part of a **recommendation system** (#recommendation_system) that provides a value or ranking for each item produced by the **candidate generation** (#candidate_generation) phase.

selection bias

Errors in conclusions drawn from sampled data due to a selection process that generates systematic differences between samples observed in the data and those not observed. The following forms of selection bias exist:

- **coverage bias**: The population represented in the dataset does not match the population that the machine learning model is making predictions about.
- **sampling bias**: Data is not collected randomly from the target group.
- **non-response bias** (also called **participation bias**): Users from certain groups opt-out of surveys at different rates than users from other groups.

For example, suppose you are creating a machine learning model that predicts people's enjoyment of a movie. To collect training data, you hand out a survey to everyone in the front row of a theater showing the movie. Offhand, this may sound like a reasonable way to gather a dataset; however, this form of data collection may introduce the following forms of selection bias:

- coverage bias: By sampling from a population who chose to see the movie, your model's predictions may not generalize to people who did not already express that level of interest in the movie.
- sampling bias: Rather than randomly sampling from the intended population (all the people at the movie), you sampled only the people in the front row. It is possible that the people sitting in the front row were more interested in the movie than those in other rows.
- non-response bias: In general, people with strong opinions tend to respond to optional surveys more frequently than people with mild opinions. Since the movie survey is

optional, the responses are more likely to form a bimodal distribution (https://wikipedia.org/wiki/Multimodal_distribution) than a normal (bell-shaped) distribution.

self-attention (also called self-attention layer)

A neural network layer that transforms a sequence of embeddings (for instance, **token** (#token) embeddings) into another sequence of embeddings. Each embedding in the output sequence is constructed by integrating information from the elements of the input sequence through an **attention** (#attention) mechanism.

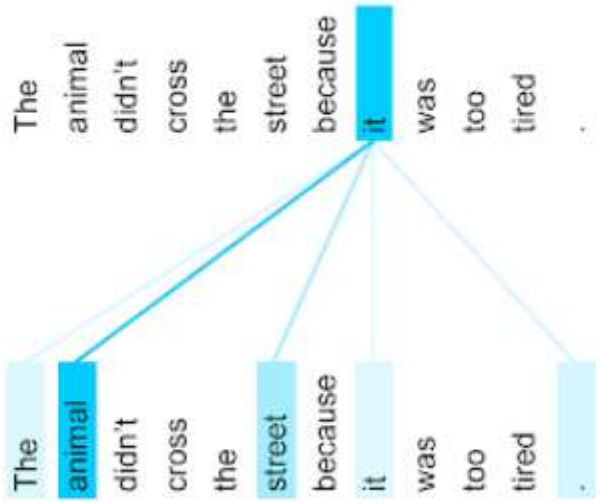
The **self** part of **self-attention** refers to the sequence attending to itself rather than to some other context. Self-attention is one of the main building blocks for **Transformers** (#Transformer) and uses dictionary lookup terminology, such as “query”, “key”, and “value”.

A self-attention layer starts with a sequence of input representations, one for each word. The input representation for a word can be a simple embedding. For each word in an input sequence, the network scores the relevance of the word to every element in the whole sequence of words. The relevance scores determine how much the word's final representation incorporates the representations of other words.

For example, consider the following sentence:

> The animal didn't cross the street because it was too tired.

The following illustration (from Transformer: A Novel Neural Network Architecture for Language Understanding (https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html)) shows a self-attention layer's attention pattern for the pronoun **it**, with the darkness of each line indicating how much each word contributes to the representation:



The self-attention layer highlights words that are relevant to "it". In this case, the attention layer has learned to highlight words that **it** might refer to, assigning the highest weight to **animal**.

For a sequence of *n* **tokens** (#token), self-attention transforms a sequence of embeddings *n* separate times, once at each position in the sequence.

Refer also to **attention** (#attention) and **multi-head self-attention** (#multi-head-self-attention).

self-supervised learning

A family of techniques for converting an **unsupervised machine learning** (#unsupervised_machine_learning) problem into a **supervised machine learning** (#supervised_machine_learning) problem by creating surrogate **labels** (#label) from **unlabeled examples** (#unlabeled_example).

Some **Transformer** (#Transformer)-based models such as **BERT** (#BERT) use self-supervised learning.

Self-supervised training is a **semi-supervised learning** (#semi-supervised_learning) approach.

self-training

A variant of **self-supervised learning** (#self-supervised-learning) that is particularly useful when all of the following conditions are true:

- The ratio of **unlabeled examples** (#unlabeled_example) to **labeled examples** (#labeled_example) in the dataset is high.
- This is a **classification** (#classification_model) problem.

Self-training works by iterating over the following two steps until the model stops improving:

1. Use **supervised machine learning** (#supervised_machine_learning) to train a model on the labeled examples.
2. Use the model created in Step 1 to generate predictions (labels) on the unlabeled examples, moving those in which there is high confidence into the labeled examples with the predicted label.

Notice that each iteration of Step 2 adds more labeled examples for Step 1 to train on.

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.

Self-training (#self-training) is one technique for semi-supervised learning.

sensitive attribute

A human attribute that may be given special consideration for legal, ethical, social, or personal reasons.

sentiment analysis

abc

Using statistical or machine learning algorithms to determine a group's overall attitude—positive or negative—toward a service, product, organization, or topic. For example, using **natural language understanding** (#natural_language_understanding), an algorithm could perform sentiment analysis on the textual feedback from a university course to determine the degree to which students generally liked or disliked the course.

sequence model

A model whose inputs have a sequential dependence. For example, predicting the next video watched from a sequence of previously watched videos.

sequence-to-sequence task

abc

A task that converts an input sequence of **tokens** (#token) to an output sequence of tokens. For example, two popular kinds of sequence-to-sequence tasks are:

- Translators:
  - Sample input sequence: "I love you."
  - Sample output sequence: "Je t'aime."
- Question answering:

- Sample input sequence: "Do I need my car in New York City?"
- Sample output sequence: "No. Please keep your car at home."

serving

A synonym for **inferring** (#inference).

shape (Tensor)

The number of elements in each **dimension** (#dimensions) of a tensor. The shape is represented as a list of integers. For example, the following two-dimensional tensor has a shape of [3,4]:

```
[[5, 7, 6, 4],
 [2, 9, 4, 8],
 [3, 6, 5, 1]]
```

TensorFlow uses row-major (C-style) format to represent the order of dimensions, which is why the shape in TensorFlow is [3,4] rather than [4,3]. In other words, in a two-dimensional TensorFlow Tensor, the shape is [*number of rows*, *number of columns*].

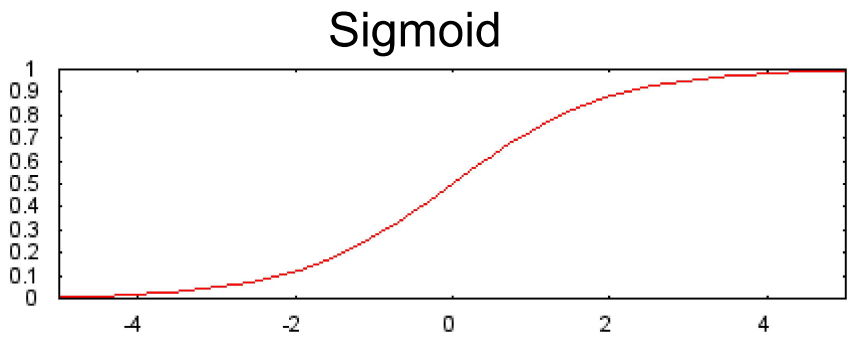
shrinkage

A **hyperparameter** (#hyperparameter) in **gradient boosting** (#gradient-boosting) that controls **overfitting** (#overfitting). Shrinkage in gradient boosting is analogous to **learning rate**

(#learning_rate) in **gradient descent** (#gradient_descent). Shrinkage is a decimal value between 0.0 and 1.0. A lower shrinkage value reduces overfitting more than a larger shrinkage value.

sigmoid function

A mathematical function that "squishes" an input value into a constrained range, typically 0 to 1 or -1 to +1. That is, you can pass any number (two, a million, negative billion, whatever) to a sigmoid and the output will still be in the constrained range. A plot of the sigmoid activation function looks as follows:



The sigmoid function has several uses in machine learning, including:

- Converting the raw output of a **logistic regression** (#logistic_regression) or **multinomial regression** (#multinomial-regression) model to a probability.
- Acting as an **activation function** (#activation_function) in some neural networks.

Click the icon to see the math.

The sigmoid function over an input number *x* has the following formula:

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

In machine learning, *x* is generally a **weighted sum** (#weighted_sum).

similarity measure

In **clustering** (#clustering) algorithms, the metric used to determine how alike (how similar) any two examples are.

single program / multiple data (SPMD)

A parallelism technique where the same computation is run on different input data in parallel on different devices. The goal of SPMD is to obtain results more quickly. It is the most common style of parallel programming.

size invariance

In an image classification problem, an algorithm's ability to successfully classify images even when the size of the image changes. For example, the algorithm can still identify a cat whether it consumes 2M pixels or 200K pixels. Note that even the best image classification algorithms still have practical limits on size invariance. For example, an algorithm (or human) is unlikely to correctly classify a cat image consuming only 20 pixels.

See also **translational invariance** (#translational_invariance) and **rotational invariance** (#rotational_invariance).

sketching

In **unsupervised machine learning** (#unsupervised_machine_learning), a category of algorithms that perform a preliminary similarity analysis on examples. Sketching algorithms use a locality-sensitive hash function (https://wikipedia.org/wiki/Locality-sensitive_hashing) to identify points that are likely to be similar, and then group them into buckets.

Sketching decreases the computation required for similarity calculations on large datasets. Instead of calculating similarity for every single pair of examples in the dataset, we calculate similarity only for each pair of points within each bucket.

softmax

A function that determines probabilities for each possible class in a **multi-class classification model** (#multi-class). The probabilities add up to exactly 1.0. For example, the following table shows how softmax distributes various probabilities:

Image is a...	Probability
dog	.85
cat	.13
horse	.02

Softmax is also called **full softmax**.

Contrast with **candidate sampling** (#candidate_sampling).

Click the icon to see the math.

The softmax equation is as follows:

$$\sigma_i = \frac{e^{z_i}}{\sum_{j=1}^{j=K} e^{z_j}}$$

where:

- σ_i is the output vector. Each element of the output vector specifies the probability of this element. The sum of all the elements in the output vector is 1.0. The output vector contains the same number of elements as the input vector, z.
- z is the input vector. Each element of the input vector contains a floating-point value.
- K is the number of elements in the input vector (and the output vector).

For example, suppose the input vector is:

```
[1.2, 2.5, 1.8]
```

Therefore, softmax calculates the denominator as follows:

$$\text{denominator} = e^{1.2} + e^{2.5} + e^{1.8} = 21.552$$

The softmax probability of each element is therefore:

$$\sigma_1 = \frac{e^{1.2}}{21.552} = 0.154$$

$$\sigma_2 = \frac{e^{2.5}}{21.552} = 0.565$$

$$\sigma_1 = \frac{e^{1.8}}{21.552} = 0.281$$

So, the output vector is therefore:

$$\sigma = [0.154, 0.565, 0.281]$$

The sum of the three elements in σ is 1.0. Phew!

---

sparse feature

A **feature** (#feature) whose values are predominately zero or empty. For example, a feature containing a single 1 value and a million 0 values is sparse. In contrast, a **dense feature** (#dense_feature) has values that are predominantly not zero or empty.

In machine learning, a surprising number of features are sparse features. Categorical features are usually sparse features. For example, of the 300 possible tree species in a forest, a single example might identify just a *maple tree*. Or, of the millions of possible videos in a video library, a single example might identify just "Casablanca."

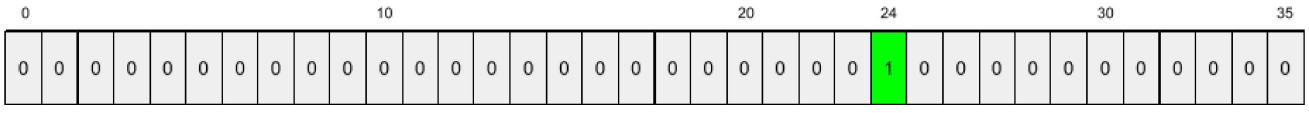
In a model, you typically represent sparse features with **one-hot encoding** (#one-hot_encoding). If the one-hot encoding is big, you might put an **embedding layer** (#embedding_layer) on top of the one-hot encoding for greater efficiency.

sparse representation

Storing only the *position(s)* of nonzero elements in a sparse feature.

For example, suppose a categorical feature named `species` identifies the 36 tree species in a particular forest. Further assume that each **example** (#example) identifies only a single species.

You could use a one-hot vector to represent the tree species in each example. A one-hot vector would contain a single `1` (to represent the particular tree species in that example) and 35 `0`s (to represent the 35 tree species *not* in that example). So, the one-hot representation of `maple` might look something like the following:



Alternatively, sparse representation would simply identify the position of the particular species. If `maple` is at position 24, then the sparse representation of `maple` would simply be:

24

Notice that the sparse representation is much more compact than the one-hot representation.

Note: You shouldn't pass a sparse representation as a direct feature input to a model. Instead, you should convert the sparse representation into a one-hot representation before training on it.

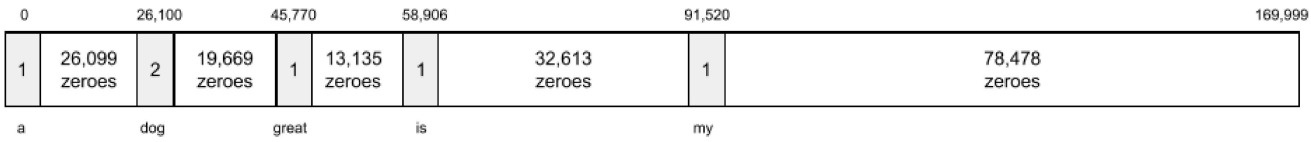
Click the icon for a slightly more complex example.

Suppose each example in your model must represent the words—but not the order of those words—in an English sentence. English consists of about 170,000 words, so English is a categorical feature with about 170,000 elements. Most English sentences use an extremely tiny fraction of those 170,000 words, so the set of words in a single example is almost certainly going to be sparse data.

Consider the following sentence:

```
My dog is a great dog
```

You could use a variant of one-hot vector to represent the words in this sentence. In this variant, multiple cells in the vector can contain a nonzero value. Furthermore, in this variant, a cell can contain an integer other than one. Although the words "my", "is", "a", and "great" appear only once in the sentence, the word "dog" appears twice. Using this variant of one-hot vectors to represent the words in this sentence yields the following 170,000-element vector:



A sparse representation of the same sentence would simply be:

```
0: 1
26100: 2
45770: 1
58906: 1
91520: 1
```

Click the icon if you are confused.

The term "sparse representation" confuses a lot of people because sparse representation is itself *not a sparse vector*. Rather, sparse representation is actually a *dense representation of a sparse vector*. The synonym **index representation** is a little clearer than "sparse representation."

---

sparse vector

A vector whose values are mostly zeroes. See also **sparse feature** (#sparse_features) and **sparsity** (#sparsity).

---

sparsity

The number of elements set to zero (or null) in a vector or matrix divided by the total number of entries in that vector or matrix. For example, consider a 100-element matrix in which 98 cells contain zero. The calculation of sparsity is as follows:

$$\text{sparsity} = \frac{98}{100} = 0.98$$

Feature sparsity refers to the sparsity of a feature vector; **model sparsity** refers to the sparsity of the model weights.

---

spatial pooling

See **pooling** (#pooling).

split

♠♠♠

In a **decision tree** (#decision-tree), another name for a **condition** (#condition).

splitter

♠♠♠

While training a **decision tree** (#decision-tree), the routine (and algorithm) responsible for finding the best **condition** (#condition) at each **node** (#node-decision-tree).

SPMD

Abbreviation for **single program / multiple data** (#single-program).

squared hinge loss

The square of the **hinge loss** (#hinge-loss). Squared hinge loss penalizes outliers more harshly than regular hinge loss.

squared loss

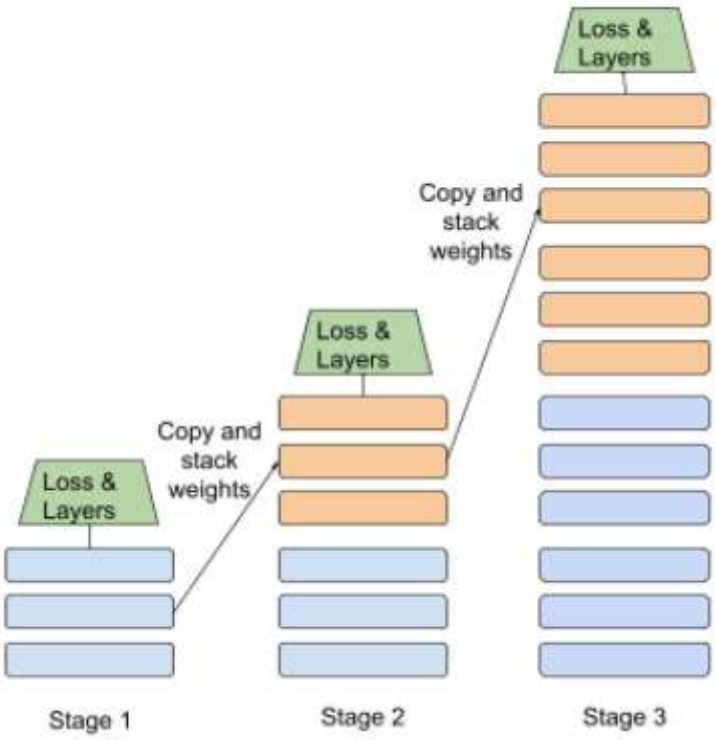
Synonym for **L_2 loss** (#L2_loss).

staged training

A tactic of training a model in a sequence of discrete stages. The goal can be either to speed up the training process, or to achieve better model quality.

An illustration of the progressive stacking approach is shown below:

- Stage 1 contains 3 hidden layers, stage 2 contains 6 hidden layers, and stage 3 contains 12 hidden layers.
- Stage 2 begins training with the weights learned in the 3 hidden layers of Stage 1. Stage 3 begins training with the weights learned in the 6 hidden layers of Stage 2.



See also **pipelining** (#pipelining).

state RL

In reinforcement learning, the parameter values that describe the current configuration of the environment, which the **agent** (#agent) uses to choose an **action** (#action).

state-action value function RL

Synonym for **Q-function** (#q-function).

static

Something done once rather than continuously. The terms **static** and **offline** are synonyms. The following are common uses of **static** and **offline** in machine learning:

- **static model** (or **offline model**) is a model trained once and then used for a while.
- **static training** (or **offline training**) is the process of training a static model.
- **static inference** (or **offline inference**) is a process in which a model generates a batch of predictions at a time.

Contrast with **dynamic** (#dynamic).

static inference

Synonym for **offline inference** (#offline_inference).

stationarity

A feature whose values don't change across one or more dimensions, usually time. For example, a feature whose values look about the same in 2021 and 2023 exhibits stationarity.

In the real world, very few features exhibit stationarity. Even features synonymous with stability (like sea level) change over time.

Contrast with **nonstationarity** (#nonstationarity).

step

A forward pass and backward pass of one **batch** (#batch).

See **backpropagation** (#backpropagation) for more information on the forward pass and backward pass.

step size

Synonym for **learning rate** (#learning_rate).

stochastic gradient descent (SGD)

A **gradient descent** (#gradient_descent) algorithm in which the **batch size** (#batch_size) is one. In other words, SGD trains on a single example chosen uniformly at random from a **training set** (#training_set).

stride

In a convolutional operation or pooling, the delta in each dimension of the next series of input slices. For example, the following animation demonstrates a (1,1) stride during a convolutional operation. Therefore, the next input slice starts one position to the right of the previous input slice. When the operation reaches the right edge, the next slice is all the way over to the left but one position down.

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		

The preceding example demonstrates a two-dimensional stride. If the input matrix is three-dimensional, the stride would also be three-dimensional.

structural risk minimization (SRM)

An algorithm that balances two goals:

- The desire to build the most predictive model (for example, lowest loss).
- The desire to keep the model as simple as possible (for example, strong regularization).

For example, a function that minimizes loss+regularization on the training set is a structural risk minimization algorithm.

Contrast with **empirical risk minimization** (#ERM).

subsampling

See **pooling** (#pooling).

summary

In TensorFlow, a value or set of values calculated at a particular **step** (#step), usually used for tracking model metrics during training.

supervised machine learning

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

Compare with **unsupervised machine learning** (#unsupervised_machine_learning).

synthetic feature

A **feature** (#feature) not present among the input features, but assembled from one or more of them. Methods for creating synthetic features include the following:

- **Bucketing** (#bucketing) a continuous feature into range bins.
- Creating a **feature cross** (#feature_cross).
- Multiplying (or dividing) one feature value by other feature value(s) or by itself. For example, if `a` and `b` are input features, then the following are examples of synthetic features:
  - `ab`
  - a^2
- Applying a transcendental function to a feature value. For example, if `c` is an input feature, then the following are examples of synthetic features:
  - `sin(c)`
  - `ln(c)`

Features created by **normalizing** (#normalization) or **scaling** (#scaling) alone are not considered synthetic features.

T

T5

A text-to-text **transfer learning** (#transfer_learning) **model** (#model) introduced by Google AI in 2020 (https://arxiv.org/pdf/1910.10683.pdf). T5 is an **encoder** (#encoder)-**decoder** (#decoder) model, based on the **Transformer** (#transformer) architecture, trained on an extremely large

dataset. It is effective at a variety of natural language processing tasks, such as generating text, translating languages, and answering questions in a conversational manner.

T5 gets its name from the five T's in "Text-to-Text Transfer Transformer."

T5X

abc

An open-source, **machine learning** (#machine_learning) framework designed to build and **train** (#training) large-scale natural language processing (NLP) models. **T5** (#T5) is implemented on the T5X codebase (which is built on **JAX** (#JAX) and **Flax** (#flax)).

tabular Q-learning

RL

In **reinforcement learning** (#reinforcement_learning), implementing **Q-learning** (#q-learning) by using a table to store the **Q-functions** (#q-function) for every combination of **state** (#state) and **action** (#action).

target

Synonym for **label** (#label).

target network

RL

In **Deep Q-learning** (#q-learning), a neural network that is a stable approximation of the main neural network, where the main neural network implements either a **Q-function** (#q-function) or a **policy** (#policy). Then, you can train the main network on the Q-values predicted by the target network. Therefore, you prevent the feedback loop that occurs when the main network trains on Q-values predicted by itself. By avoiding this feedback, training stability increases.

task

A problem that can be solved using machine learning techniques, such as:

- **classification** (#classification_model)
- **regression** (#regression_model)
- **clustering** (#clustering)
- **anomaly detection** (#anomaly-detection)

temporal data

Data recorded at different points in time. For example, winter coat sales recorded for each day of the year would be temporal data.

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.

TensorBoard

The dashboard that displays the summaries saved during the execution of one or more TensorFlow programs.

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.

TensorFlow Playground

A program that visualizes how different **hyperparameters** (#hyperparameter) influence model (primarily neural network) training. Go to http://playground.tensorflow.org (http://playground.tensorflow.org) to experiment with TensorFlow Playground.

TensorFlow Serving

A platform to deploy trained models in production.

Tensor Processing Unit (TPU)

An application-specific integrated circuit (ASIC) that optimizes the performance of machine learning workloads. These ASICs are deployed as multiple **TPU chips** (#TPU_chip) on a **TPU device** (#TPU_device).

Tensor rank

See **rank (Tensor)** (#rank_Tensor).

Tensor shape

The number of elements a **Tensor** (#tensor) contains in various dimensions. For example, a [5, 10] Tensor has a shape of 5 in one dimension and 10 in another.

Tensor size

The total number of scalars a **Tensor** (#tensor) contains. For example, a [5, 10] Tensor has a size of 50.

TensorStore

A library (https://google.github.io/tensorstore/) for efficiently reading and writing large multi-dimensional arrays.

termination condition RL

In **reinforcement learning** (#reinforcement_learning), the conditions that determine when an **episode** (#episode) ends, such as when the agent reaches a certain state or exceeds a threshold number of state transitions. For example, in tic-tac-toe (https://wikipedia.org/wiki/Tic-tac-toe) (also known as noughts and crosses), an episode terminates either when a player marks three consecutive spaces or when all spaces are marked.

test

In a **decision tree** (#decision-tree), another name for a **condition** (#condition).

test loss

A **metric** (#metric) representing a model's **loss** (#loss) against the **test set** (#test_set).
When building a **model** (#model), you typically try to minimize test loss. That's because a low test loss is a stronger quality signal than a low **training loss** (#training-loss) or low **validation loss** (#validation-loss).

A large gap between test loss and training loss or validation loss sometimes suggests that you need to increase the **regularization rate** (#regularization_rate).

test set

A subset of the **dataset** (#dataset) reserved for testing a trained **model** (#model).

Traditionally, you divide examples in the dataset into the following three distinct subsets:

- a **training set** (#training_set)
- a **validation set** (#validation_set)
- a test set

Each example in a dataset should belong to only one of the preceding subsets. For instance, a single example should not belong to both the training set and the test set.

The training set and validation set are both closely tied to training a model. Because the test set is only indirectly associated with training, **test loss** (#test-loss) is a less biased, higher quality metric than **training loss** (#training-loss) or **validation loss** (#validation-loss).

text span

The array index span associated with a specific subsection of a text string. For example, the word `good` in the Python string `s="Be good now"` occupies the text span from 3 to 6.

tf.Example

A standard protocol buffer (https://developers.google.com/protocol-buffers/) for describing input data for machine learning model training or inference.

tf.keras

An implementation of **Keras** (#Keras) integrated into **TensorFlow** (#TensorFlow).

threshold (for decision trees)

In an **axis-aligned condition** (#axis-aligned-condition), the value that a **feature** (#feature) is being compared against. For example, 75 is the threshold value in the following condition:

```
grade >= 75
```

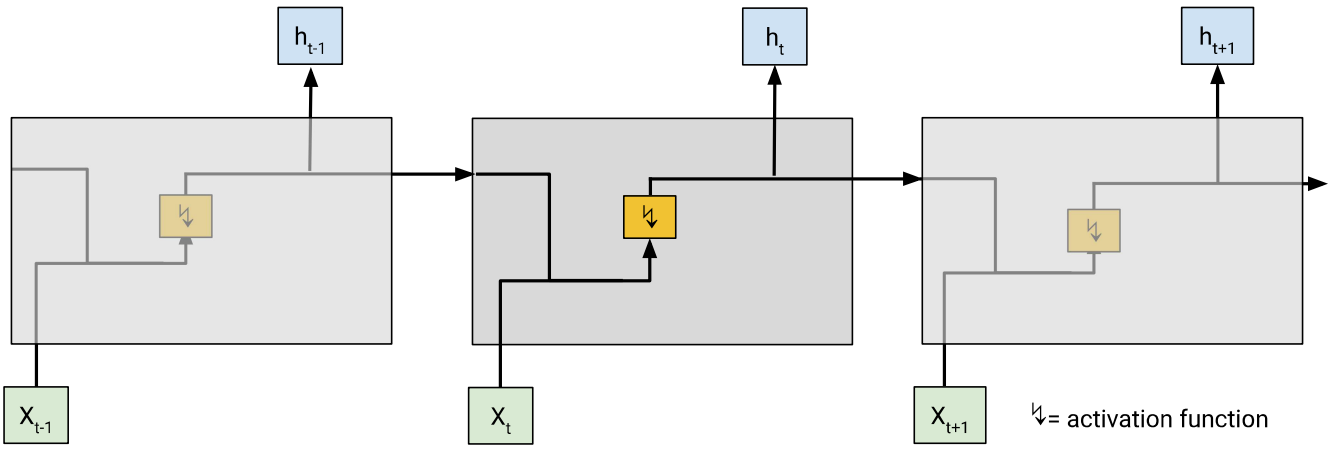
This form of the term **threshold** is different than **classification threshold** (#classification_threshold).

time series analysis

A subfield of machine learning and statistics that analyzes **temporal data** (#temporal_data). Many types of machine learning problems require time series analysis, including classification, clustering, forecasting, and anomaly detection. For example, you could use time series analysis to forecast the future sales of winter coats by month based on historical sales data.

timestep

One "unrolled" cell within a **recurrent neural network** (#recurrent_neural_network). For example, the following figure shows three timesteps (labeled with the subscripts t-1, t, and t+1):



token

In a **language model** (#language-model), the atomic unit that the model is training on and making predictions on. A token is typically one of the following:

- a word—for example, the phrase "dogs like cats" consists of three word tokens: "dogs", "like", and "cats".
- a character—for example, the phrase "bike fish" consists of nine character tokens. (Note that the blank space counts as one of the tokens.)
- subwords—in which a single word can be a single token or multiple tokens. A subword consists of a root word, a prefix, or a suffix. For example, a language model that uses subwords as tokens might view the word "dogs" as two tokens (the root word "dog" and the plural suffix "s"). That same language model might view the single word "taller" as two subwords (the root word "tall" and the suffix "er").

In domains outside of language models, tokens can represent other kinds of atomic units. For example, in computer vision, a token might be a subset of an image.

tower

A component of a **deep neural network** (#deep_neural_network) that is itself a deep neural network without an output layer. Typically, each tower reads from an independent data source. Towers are independent until their output is combined in a final layer.

TPU

Abbreviation for **Tensor Processing Unit** (#TPU).

TPU chip

A programmable linear algebra accelerator with on-chip high bandwidth memory that is optimized for machine learning workloads. Multiple TPU chips are deployed on a **TPU device** (#TPU_device).

TPU device

A printed circuit board (PCB) with multiple **TPU chips** (#TPU_chip), high bandwidth network interfaces, and system cooling hardware.

TPU master

The central coordination process running on a host machine that sends and receives data, results, programs, performance, and system health information to the **TPU workers** (#TPU_worker). The TPU master also manages the setup and shutdown of **TPU devices** (#TPU_device).

TPU node

A TPU resource on Google Cloud Platform with a specific **TPU type** (#TPU_type). The TPU node connects to your VPC Network (https://cloud.google.com/vpc/docs/) from a peer VPC network (https://cloud.google.com/vpc/docs/vpc-peering). TPU nodes are a resource defined in the Cloud TPU API (https://cloud.google.com/tpu/docs/reference/rest/v1/projects.locations.nodes).

TPU Pod

A specific configuration of **TPU devices** (#TPU_device) in a Google data center. All of the devices in a TPU pod are connected to one another over a dedicated high-speed network. A TPU Pod is the largest configuration of **TPU devices** (#TPU_device) available for a specific TPU version.

TPU resource

A TPU entity on Google Cloud Platform that you create, manage, or consume. For example, **TPU nodes** (#TPU_node) and **TPU types** (#TPU_type) are TPU resources.

TPU slice

A TPU slice is a fractional portion of the **TPU devices** (#TPU_device) in a **TPU Pod** (#TPU_Pod). All of the devices in a TPU slice are connected to one another over a dedicated high-speed network.

TPU type

A configuration of one or more **TPU devices** (#TPU_device) with a specific TPU hardware version. You select a TPU type when you create a **TPU node** (#TPU_node) on Google Cloud Platform. For example, a `v2-8` TPU type is a single TPU v2 device with 8 cores. A `v3-2048` TPU type has 256 networked TPU v3 devices and a total of 2048 cores. TPU types are a resource defined in the Cloud TPU API (https://cloud.google.com/tpu/docs/reference/rest/v1/projects.locations.acceleratorTypes).

TPU worker

A process that runs on a host machine and executes machine learning programs on **TPU devices** (#TPU_device).

training

The process of determining the ideal **parameters** (#parameter) (weights and biases) comprising a **model** (#model). During training, a system reads in **examples** (#example) and gradually adjusts parameters. Training uses each example anywhere from a few times to billions of times.

training loss

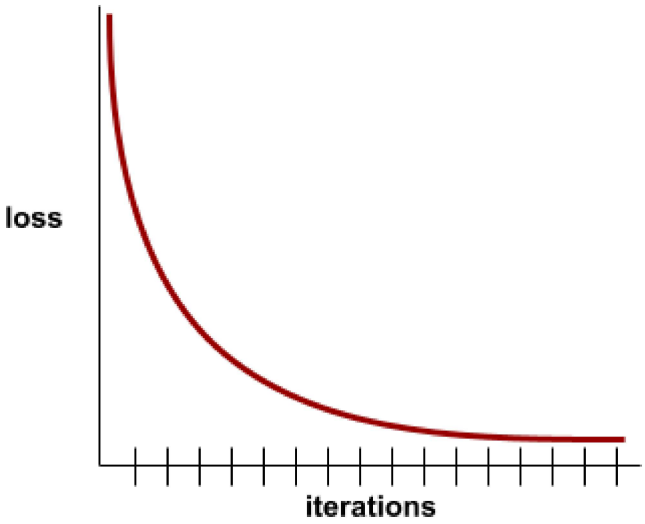
A **metric** (#metric) representing a model's **loss** (#loss) during a particular training iteration. For example, suppose the loss function is **Mean Squared Error** (#MSE). Perhaps the training loss (the Mean Squared Error) for the 10th iteration is 2.2, and the training loss for the 100th iteration is 1.9.

A **loss curve** (#loss_curve) plots training loss vs. the number of iterations. A loss curve provides the following hints about training:

- A downward slope implies that the model is improving.
- An upward slope implies that the model is getting worse.
- A flat slope implies that the model has reached **convergence** (#convergence).

For example, the following somewhat idealized **loss curve** (#loss_curve) shows:

- A steep downward slope during the initial iterations, which implies rapid model improvement.
- A gradually flattening (but still downward) slope until close to the end of training, which implies continued model improvement at a somewhat slower pace then during the initial iterations.
- A flat slope towards the end of training, which suggests convergence.



Although training loss is important, see also **generalization** (#generalization).

training-serving skew

The difference between a model's performance during **training** (#training) and that same model's performance during **serving** (#serving).

training set

The subset of the **dataset** (#dataset) used to train a **model** (#model).

Traditionally, examples in the dataset are divided into the following three distinct subsets:

- a training set
- a **validation set** (#validation_set)
- a **test set** (#test_set)

Ideally, each example in the dataset should belong to only one of the preceding subsets. For example, a single example should not belong to both the training set and the validation set.

trajectory RL

In **reinforcement learning** (#reinforcement_learning), a sequence of tuples (https://wikipedia.org/wiki/Tuple) that represent a sequence of **state** (#state) transitions of the **agent** (#agent), where each tuple corresponds to the state, **action** (#action), **reward** (#reward), and next state for a given state transition.

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** (#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.

Transformer abc

A **neural network** (#neural_network) architecture developed at Google that relies on **self-attention** (#self-attention) mechanisms to transform a sequence of input embeddings into a

sequence of output embeddings without relying on **convolutions** (#convolution) or **recurrent neural networks** (#recurrent_neural_network). A Transformer can be viewed as a stack of self-attention layers.

A Transformer can include any of the following:

- an **encoder** (#encoder)
- a **decoder** (#decoder)
- both an encoder and decoder

An **encoder** transforms a sequence of embeddings into a new sequence of the same length. An encoder includes N identical layers, each of which contains two sub-layers. These two sub-layers are applied at each position of the input embedding sequence, transforming each element of the sequence into a new embedding. The first encoder sub-layer aggregates information from across the input sequence. The second encoder sub-layer transforms the aggregated information into an output embedding.

A **decoder** transforms a sequence of input embeddings into a sequence of output embeddings, possibly with a different length. A decoder also includes N identical layers with three sub-layers, two of which are similar to the encoder sub-layers. The third decoder sub-layer takes the output of the encoder and applies the **self-attention** (#self-attention) mechanism to gather information from it.

The blog post Transformer: A Novel Neural Network Architecture for Language Understanding (https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html) provides a good introduction to Transformers.

translational invariance

In an image classification problem, an algorithm's ability to successfully classify images even when the position of objects within the image changes. For example, the algorithm can still identify a dog, whether it is in the center of the frame or at the left end of the frame.

See also **size invariance** (#size_invariance) and **rotational invariance** (#rotational_invariance).

trigram

An **N-gram** (#N-gram) in which N=3.

true negative (TN)

An example in which the model *correctly* predicts the **negative class** (#negative_class). For example, the model infers that a particular email message is *not spam*, and that email message really is *not spam*.

true positive (TP)

An example in which the model *correctly* predicts the **positive class** (#positive_class). For example, the model infers that a particular email message is spam, and that email message really is spam.

true positive rate (TPR)

Synonym for **recall** (#recall). That is:

$$\textbf{true positive rate} = \frac{\textbf{true positives}}{\textbf{true positives} + \textbf{false negatives}}$$

True positive rate is the y-axis in an **ROC curve** (#ROC).

U

unawareness (to a sensitive attribute)

A situation in which **sensitive attributes** (#sensitive_attribute) are present, but not included in the training data. Because sensitive attributes are often correlated with other attributes of one's data, a model trained with unawareness about a sensitive attribute could still have **disparate impact** (#disparate_impact) with respect to that attribute, or violate other **fairness constraints** (#fairness_constraint).

underfitting

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

- Training on the wrong set of **features** (#feature).
- Training for too few **epochs** (#epoch) or at too low a **learning rate** (#learning_rate).
- Training with too high a **regularization rate** (#regularization_rate).
- Providing too few **hidden layers** (#hidden_layer) in a deep neural network.

undersampling

Removing **examples** (#example) from the **majority class** (#majority_class) in a **class-imbalanced dataset** (#class_imbalanced_data_set) in order to create a more balanced **training set** (#training_set).

For example, consider a dataset in which the ratio of the majority class to the **minority class** (#minority_class) is 20:1. To overcome this class imbalance, you could create a training set consisting of *all* of the minority class examples but only a *tenth* of the majority class examples, which would create a training-set class ratio of 2:1. Thanks to undersampling, this more balanced training set might produce a better model. Alternatively, this more balanced training set might contain insufficient examples to train an effective model.

Contrast with **oversampling** (#oversampling).

unidirectional

A system that only evaluates the text that *precedes* a target section of text. In contrast, a bidirectional system evaluates both the text that *precedes* and *follows* a target section of text. See **bidirectional** (#bidirectional) for more details.

unidirectional language model

A **language model** (#language-model) that bases its probabilities only on the **tokens** (#token) appearing *before*, not *after*, the target token(s). Contrast with **bidirectional language model** (#bidirectional-language-model).

unlabeled example

An example that contains **features** (#feature) but no **label** (#label). For example, the following table shows three unlabeled examples from a house valuation model, each with three features but no house value:

Number of bedrooms	Number of bathrooms	House age
3	2	15
2	1	72
4	2	34

In **supervised machine learning** (#supervised_machine_learning), models train on labeled examples and make predictions on **unlabeled examples** (#unlabeled_example).

In **semi-supervised** (#semi-supervised_learning) and **unsupervised** (#unsupervised_machine_learning) learning, unlabeled examples are used during training.

Contrast unlabeled example with **labeled example** (#labeled_example).

unsupervised machine learning

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

The most common use of unsupervised machine learning is to **cluster** (#clustering) data into groups of similar examples. For example, an unsupervised machine learning algorithm can cluster songs 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 help when useful labels are scarce or absent. For example, in domains such as anti-abuse and fraud, clusters can help humans better understand the data.

Contrast with **supervised machine learning** (#supervised_machine_learning).

Click the icon for additional notes.

Another example of unsupervised machine learning is principal component analysis (PCA) (https://wikipedia.org/wiki/Principal_component_analysis). 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.

---

uplift modeling

A modeling technique, commonly used in marketing, that models the "causal effect" (also known as the "incremental impact") of a "treatment" on an "individual." Here are two examples:

- Doctors might use uplift modeling to predict the mortality decrease (causal effect) of a medical procedure (treatment) depending on the age and medical history of a patient (individual).
- Marketers might use uplift modeling to predict the increase in probability of a purchase (causal effect) due to an advertisement (treatment) on a person (individual).

Uplift modeling differs from **classification** (#classification_model) or **regression** (#regression_model) in that some labels (for example, half of the labels in binary treatments) are always missing in uplift modeling. For example, a patient can either receive or not receive a treatment; therefore, we can only observe whether the patient is going to heal or not heal in only one of these two situations (but never both). The main advantage of an uplift model is that it can generate predictions for the unobserved situation (the counterfactual) and use it to compute the causal effect.

---

upweighting

Applying a weight to the **downsampled** (#downsampling) class equal to the factor by which you downsampled.

user matrix

In **recommendation systems** (#recommendation_system), an **embedding vector** (#embedding_vector) generated by **matrix factorization** (#matrix_factorization) that holds latent signals about user preferences. Each row of the user matrix holds information about the relative strength of various latent signals for a single user. For example, consider a movie recommendation system. In this system, the latent signals in the user matrix might represent each user's interest in particular genres, or might be harder-to-interpret signals that involve complex interactions across multiple factors.

The user matrix has a column for each latent feature and a row for each user. That is, the user matrix has the same number of rows as the target matrix that is being factorized. For example, given a movie recommendation system for 1,000,000 users, the user matrix will have 1,000,000 rows.

V

validation

The initial evaluation of a model's quality. Validation checks the quality of a model's predictions against the **validation set** (#validation_set).

Because the validation set differs from the **training set** (#training_set), validation helps guard against **overfitting** (#overfitting).

You might think of evaluating the model against the validation set as the first round of testing and evaluating the model against the **test set** (#test_set) as the second round of testing.

validation loss

A **metric** (#metric) representing a model's **loss** (#loss) on the **validation set** (#validation_set) during a particular **iteration** (#iteration) of training.

See also **generalization curve** (#generalization_curve).

validation set

The subset of the **dataset** (#dataset) that performs initial evaluation against a trained **model** (#model). Typically, you evaluate the trained model against the **validation set** (#validation_set) several times before evaluating the model against the **test set** (#test_set).

Traditionally, you divide the examples in the dataset into the following three distinct subsets:

- a **training set** (#training_set)
- a validation set
- a **test set** (#test_set)

Ideally, each example in the dataset should belong to only one of the preceding subsets. For example, a single example should not belong to both the training set and the validation set.

value imputation

The process of replacing a missing value with an acceptable substitute. When a value is missing, you can either discard the entire example or you can use value imputation to salvage the example.

For example, consider a dataset containing a `temperature` feature that is supposed to be recorded every hour. However, the temperature reading was unavailable for a particular hour. Here is a section of the dataset:

Timestamp	Temperature
1680561000	10
1680564600	12
1680568200	missing
1680571800	20
1680575400	21
1680579000	21

A system could either delete the missing example or impute the missing temperature as 12, 16, 18, or 20, depending on the imputation algorithm.

vanishing gradient problem

The tendency for the gradients of early **hidden layers** (#hidden_layer) of some **deep neural networks** (#deep_neural_network) to become surprisingly flat (low). Increasingly lower gradients result in increasingly smaller changes to the weights on nodes in a deep neural network, leading to little or no learning. Models suffering from the vanishing gradient problem become difficult or impossible to train. **Long Short-Term Memory** (#Long_Short-Term_Memory) cells address this issue.

Compare to **exploding gradient problem** (#exploding_gradient_problem).

variable importances

A set of scores that indicates the relative importance of each **feature** (#feature) to the model.

For example, consider a **decision tree** (#decision-tree) that estimates house prices. Suppose this decision tree uses three features: size, age, and style. If a set of variable importances for the three features are calculated to be {size=5.8, age=2.5, style=4.7}, then size is more important to the decision tree than age or style.

Different variable importance metrics exist, which can inform ML experts about different aspects of models.

variational autoencoder (VAE)

A type of deep learning **model** (#model) that is used for **dimension reduction** (#dimension_reduction) and data compression. VAEs are based on variational inference: a technique for estimating the parameters of a probability model.

VAEs are composed of two **neural networks** (#neural-network): an **encoder** (#encoder) and a **decoder** (#decoder). The encoder takes input (such as an image, sound, or text) and compresses it into a vector of latent variables. The decoder takes the vector of latent variables and reconstructs the original data. A loss is computed by comparing the original data to the reconstructed data, and the weights are updated via **backpropagation** (#backpropagation) through both networks.

W

Wasserstein loss

One of the loss functions commonly used in **generative adversarial networks** (#generative_adversarial_network), based on the **earth mover's distance** (#earth-movers-distance) between the distribution of generated data and real data.

weight

A value that a model multiplies by another value. **Training** (#training) is the process of determining a model's ideal weights; **inference** (#inference) is the process of using those learned weights to make predictions.

Click the icon to see an example of weights in a linear model.

Imagine a **linear model** (#linear_model) with two features. Suppose that training determines the following weights (and bias (#bias)):

- The bias, b, has a value of 2.2
- The weight, w_1 associated with one feature is 1.5.
- The weight, w_2 associated with the other feature is 0.4.

Now imagine an example (#example) with the following feature values:

- The value of one feature, x_1, is 6.
- The value of the other feature, x_2, is 10.

This linear model uses the following formula to generate a prediction, y':

$$y' = b + w_1 x_1 + w_2 x_2$$

Therefore, the prediction is:

$$y' = 2.2 + (1.5)(6) + (0.4)(10) = 15.2$$

If a weight is 0, then the corresponding feature does not contribute to the model. For example, if w_1 is 0, then the value of x_1 is irrelevant.

Weighted Alternating Least Squares (WALS)

An algorithm for minimizing the objective function during **matrix factorization** (#matrix_factorization) in **recommendation systems** (#recommendation_system), which allows a downweighting of the missing examples. WALS minimizes the weighted squared error between the original matrix and the reconstruction by alternating between fixing the row factorization and column factorization. Each of these optimizations can be solved by least squares **convex optimization** (#convex_optimization). For details, see the Recommendation Systems course (/machine-learning/recommendation/collaborative/matrix).

weighted sum

The sum of all the relevant input values multiplied by their corresponding weights. For example, suppose the relevant inputs consist of the following:

input value	input weight
2	-1.3
-1	0.6
3	0.4

The weighted sum is therefore:

```
weighted sum = (2)(-1.3) + (-1)(0.6) + (3)(0.4) = -2.0
```

A weighted sum is the input argument to an **activation function** (#activation_function).

wide model

A linear model that typically has many **sparse input features** (#sparse_features). We refer to it as "wide" since such a model is a special type of **neural network** (#neural_network) with a large number of inputs that connect directly to the output node. Wide models are often easier to debug and inspect than **deep models** (#deep_model). Although wide models cannot express nonlinearities through **hidden layers** (#hidden_layer), wide models can use transformations such as **feature crossing** (#feature_cross) and **bucketization** (#bucketing) to model nonlinearities in different ways.

Contrast with **deep model** (#deep_model).

width

The number of **neurons** (#neuron) in a particular **layer** (#layer) of a **neural network** (#neural_network).

wisdom of the crowd

The idea that averaging the opinions or estimates of a large group of people ("the crowd") often produces surprisingly good results. For example, consider a game in which people guess the number of jelly beans packed into a large jar. Although most individual guesses will be inaccurate, the average of all the guesses has been empirically shown to be surprisingly close to the actual number of jelly beans in the jar.

Ensembles (#ensemble) are a software analog of wisdom of the crowd. Even if individual models make wildly inaccurate predictions, averaging the predictions of many models often generates surprisingly good predictions. For example, although an individual **decision tree** (#decision-tree) might make poor predictions, a **decision forest** (#decision-forest) often makes very good predictions.

word embedding

Representing (#representation) each word in a word set within an **embedding vector** (#embedding_vector); that is, representing each word as a vector of floating-point values between 0.0 and 1.0. Words with similar meanings have more-similar representations than words with different meanings. For example, *carrots*, *celery*, and *cucumbers* would all have relatively similar representations, which would be very different from the representations of *airplane*, *sunglasses*, and *toothpaste*.

X

XLA (Accelerated Linear Algebra)

An open-source machine learning compiler for GPUs, CPUs, and ML accelerators.

The XLA compiler takes models from popular ML frameworks such as PyTorch (https://pytorch.org), **TensorFlow** (#TensorFlow), and **JAX** (#JAX), and optimizes them for high-performance execution across different hardware platforms including GPUs, CPUs, and ML **accelerators** (#accelerator-chip).

Z

zero-shot learning

A type of machine learning **training** (#training) where the **model** (#model) infers a **prediction** (#prediction) for a task that it was not specifically already trained on. In other words, the model is given zero task-specific training **examples** (#example) but asked to do **inference** (#inference) for that task.

Z-score normalization

A **scaling** (#scaling) technique that replaces a raw **feature** (#feature) value with a floating-point value representing the number of standard deviations from that feature's mean. For example, consider a feature whose mean is 800 and whose standard deviation is 100. The following table shows how Z-score normalization would map the raw value to its Z-score:

Raw value	Z-score
800	0
950	+1.5
575	-2.25

The machine learning model then trains on the Z-scores for that feature instead of on the raw values.

Last updated 2023-05-01 UTC.