Terminology

Mathematics

Tensor Mathematical object that generalizes the concepts of scalars, vectors, and matrices. Tensors can be thought of as multi-dimensional arrays of numbers. There are different types of tensors, characterized by their rank. Tensors of rank greater than 2 are referred to as higher-order tensors.

Shape Indication of the size of each dimension along which the tensor is defined. In general, the shape of a tensor is represented as a tuple of integers, where each integer represents the size of a particular dimension of the tensor. So, for an n-dimensional tensor, the shape would be represented by a tuple of n integers.

Categorical Data Type of data that represents categories or labels, rather than numerical values. It is used to classify items into groups or classes based on qualitative attributes or characteristics. There are two types of categorical data.

Ordinal data represents categories with a natural order or ranking. While the categories are still discrete, they have a meaningful sequence or hierarchy. This kind of data can easily be converted to numerical data with ordinal or integer encoding.

Nominal data represents categories that have no inherent order or ranking. Each category is distinct and unrelated to the others. One-hot encoding can be applied for the class labels.

One-Hot Encoding Technique used to represent categorical variables as binary vectors in machine learning and data processing. It is commonly used when dealing with categorical data, such as class labels, where the values are discrete and unordered.

In one-hot encoding, each category or class is represented as a binary vector where only one element is "hot" (set to 1) while all other elements are "cold" (set to 0).

One-hot encoding is useful because it allows machine learning algorithms to work with categorical data, which are typically represented as strings or integers, in a numerical format that can be fed into models for training and prediction. Additionally, it ensures that the encoded features are equidistant from each other, which can prevent certain models from assuming false ordinal relationships between categories.

Gradient Vector of partial derivatives for a multivariate function that points in the direction in which the function increases most quickly. Its components are the partial derivatives of that function with respect to each of its variables.

In optimization algorithms such as gradient descent, the gradient is used to update the parameters of a model in order to minimize a loss function. By iteratively following the direction of the negative gradient, one can descend to a local minimum of the function.

Jacobian Matrix Jacobian matrix is a matrix of (first-order) partial derivatives. It describes the rate of change of a multivariable function with respect to its input variables.

The Jacobian matrix provides important information about the local behavior of a function near a specific point. It describes how small changes in the input variables result in changes to the output variables.

Jacobian Determinant Measures the volume scaling factor associated with the transformation of a multivariate function near an input point. It provides important information about the local behavior (expansion or contraction) of functions and is closely related to volume scaling and change of variables.

Likelihood Function Represents the probability of the observed data given a set of parameters. It is a fundamental concept in likelihood-based methods, such as maximum likelihood estimation. The likelihood function plays a central role in statistical inference where the goal is to find the set of parameters that maximizes the likelihood function. It is often used to make inferences about these parameters, such as estimating their values or testing hypotheses about them.

Log-Likelihood The natural logarithm of the likelihood function. It is often used in statistical inference and parameter estimation, particularly in cases where the likelihood function involves products of

many probabilities, which can be computationally cumbersome to work with directly.

Taking the logarithm of the likelihood function has the advantage that maximizing the log-likelihood is equivalent to maximizing the likelihood itself, and that the log-likelihood function has a single global maximum (it is concave), which efficiently facilitates gradient descent.

Global and Local Minimum A local minimum is a point where the function takes on its smallest value in a small neighborhood around that point. A global minimum is a point where the function takes on its smallest value over its entire domain.

In graphical terms, you can visualize a local minimum as a "valley" where the function is lower than its neighboring points but may not be the absolute lowest point in the entire graph. On the other hand, a global minimum represents the lowest point on the entire graph.

The presence of multiple local minima can make optimization problems challenging, especially in non-convex functions, where gradient-based optimization methods may converge to local minima instead of the global minimum.

Moving/Rolling Average Statistical technique used to smooth out fluctuations in data over time by creating a series of averages of different subsets of the full dataset. It is commonly used in time series analysis to identify trends and patterns in data.

The basic idea behind a moving average is to calculate the average value of a fixed number of consecutive data points (referred to as the "window" or "period") and use this average to represent the underlying trend in the data. As new data becomes available, the moving average is recalculated by shifting the window along the dataset. They help to reduce noise and highlight underlying trends or patterns in the data.

Mean Squared Error Common metric used to measure the average squared difference between ground truth values and predicted values produced by a model. It's widely used in regression analysis to evaluate the performance of a predictive model.

The MSE quantifies the average magnitude of errors made by the model. A lower MSE indicates that the model's predictions are closer to the actual values, while a higher MSE indicates larger discrepancies between the predicted and actual values.

MSE is a useful metric because it penalizes larger errors more heavily than smaller errors due to the squaring operation. However, it has the drawback of being sensitive to outliers.

Mean Absolute Error Metric used to measure the average absolute difference between ground truth values and predicted values. It is widely used in regression analysis to evaluate the performance of a predictive model.

The MAE measures the average absolute difference between the predicted and actual values, and therefore, it quantifies the average magnitude of errors made by the model without considering their direction. A lower MAE indicates that the model's predictions are closer to the actual values.

MAE is less sensitive to outliers compared to Mean Squared Error (MSE) because it does not square the errors. However, it treats positive and negative errors equally, which may not be desirable.

Kronecker Delta The Kronecker delta is a mathematical tool that is defined as equal to 1 if indices given i and j are equal, and it is equal to 0 if the indices i and j are different.

Bernoulli Distribution Discrete probability distribution that models a single random experiment with two possible outcomes: success and failure. Each trial has only two possible outcomes, often denoted as "success" and "failure." These outcomes are mutually exclusive and exhaustive, meaning that one and only one of them must occur in each trial, and the probabilities of success and of failure remain constant from trial to trial.

The Bernoulli distribution is often used as a building block for more complex probability distributions and statistical models. It serves as the foundation for the binomial distribution, which describes the number of successes in a fixed number of independent Bernoulli trials. Examples of Bernoulli experiments include flipping a coin.

Moments The first and second moments are two fundamental moments often used to characterize a probability distribution. The first moment (mean) provides information about the center of the distribution, while the second moment (variance) provides information about the spread or

dispersion of the distribution around the mean. Together, these moments offer key insights into the shape and characteristics of the probability distribution of a random variable.

The mean (μ) is calculated as the weighted average of all possible values of the random variable, where each value is weighted by its respective probability.

The variance (σ^2) is calculated as the weighted average of the squared deviations of all possible values of the random variable from the mean, where each squared deviation is weighted by its respective probability.

Note that a standard deviation (σ) is the square root of the variance. It is a more intuitive measure of dispersion compared to variance because it is expressed in the same units as the original data.

k-Fold Cross-Validation Technique used in machine learning for evaluating the performance of a predictive model. It helps to assess how well a model generalizes to new data by partitioning the available dataset into multiple subsets, called folds, and iteratively training and evaluating the model on different combinations of these folds.

The original dataset is randomly partitioned into K equal-sized subsets or folds. The model is trained on $K-1$ folds (the training set), and evaluated on the left-out fold (the validation set) to compute a loss metric. After K iterations, all metrics are combined to obtain a single estimate of the model's performance.

Machine Learning

Machine learning is a subfield of Artificial Intelligence that focuses on the development of algorithms and models that allow computers to learn from data and make predictions or decisions without being explicitly programmed for every task. In essence, machine learning algorithms enable computers to learn patterns and relationships from data and use this knowledge to make decisions or predictions on new, unseen data.

The three main categories of machine learning are supervised learning, unsupervised learning, and reinforcement learning.

Pattern Recognition Pattern recognition is a field within machine learning that focuses on the automated identification of patterns or regularities in data. It involves the development of algorithms and techniques to recognize and classify patterns in various types of data, such as images, text, audio, or time series.

The goal of pattern recognition is to extract meaningful information from raw data and use it to make decisions or predictions.

Neural Network Class of machine learning models inspired by the structure and function of the human brain. They consist of interconnected layers of nodes (neurons) and are capable of learning complex patterns and relationships in data. Examples include feedforward neural networks, convolutional neural networks, and recurrent neural networks.

A model refers to the representation of a system or a process that is learned from data. It encapsulates the relationship between input data and the corresponding output, allowing the model to make predictions or decisions about new, unseen data. A neural network is one type of model.

Once a neural network is trained on a dataset, it can be used to make predictions or decisions about new, unseen data. The process of training a model involves feeding it with labeled data (in supervised learning) and adjusting the model parameters to minimize a loss function, which quantifies the difference between the model's predictions and the true values.

Deep neural networks can have various architectures, including fully connected (or dense) networks, convolutional neural networks for image data, or recurrent neural networks for sequential data. The choice of architecture depends on the nature of the input data and the specific task to be performed.

Deep Learning Deep learning is a subfield of machine learning that focuses on the development and training of artificial neural networks with multiple layers.

Deep learning models automatically learn hierarchical representations of data, starting from low-level features (such as edges in images or phonemes in speech) and progressing to higher-level

features (such as object shapes or spoken words). Deep learning models can scale effectively to large datasets and complex problems and learn to automatically extract useful representations of data, which can be transferred to other tasks or domains with minimal adaptation. This property makes deep learning particularly well-suited for transfer learning and multitask learning.

Feedforward Neural Network Also known as a multilayer perceptron, a feedforward neural network one of the simplest and most common types of artificial neural networks used in machine learning and deep learning. It consists of multiple layers of neurons, where each neuron in one layer is connected to every neuron in the next layer, and information flows in one direction, from the input layer to the output layer, without any feedback loops.

During the training process, the weights and biases of the neurons in the network are iteratively adjusted using optimization algorithms such as gradient descent, so that the network learns to produce the correct outputs for a given set of inputs. This process involves forward propagation (computing a prediction) and backward propagation (computing the gradients of the loss function with respect to the weights and biases), hence the name "feedforward" neural network.

Feedforward neural networks are versatile and can be applied to a wide range of tasks, including classification, regression, and function approximation. However, they may struggle with capturing long-range dependencies in sequential data, such as natural language or time-series data.

Convolutional Neural Network A specialized type of deep neural network designed to process and analyze spatial data, such as images and videos. CNNs are widely used in computer vision tasks, including image classification, object detection, and segmentation. They consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

Recurrent Neural Network Type of neural network designed to process and analyze sequential data, such as time series, text, audio, and video. RNNs are characterized by their ability to maintain a memory of previous inputs through hidden states, allowing them to capture temporal dependencies and patterns in the data.

Supervised Learning

Type of machine learning where the goal of is to learn a mapping from input features to output labels or values based on the examples provided in the training data. Each training example consists of an input feature vector and a corresponding output label or value. The algorithm learns from these examples by adjusting its internal parameters to minimize the error between its predictions and the true labels in the training data.

Supervised learning algorithms learn to generalize from the training data to make predictions or decisions on new, unseen data. The quality of the predictions depends on factors such as the choice of algorithm, the quality and quantity of the training data, the selection of features, and the model's capacity to capture complex relationships in the data.

The two main types of supervised learning tasks are classification and regression.

Regression Supervised learning task where the goal is to predict a continuous numerical value based on input features. The input data consists of a set of features or attributes, and each example is associated with a continuous target value. A regressor's task is to learn a mapping from the input features to the output target values based on the examples provided in the training data. The output variable is continuous and represents a quantity.

Classification Supervised learning task where the goal is to predict the categorical label or class of an input data point based on its features. The input data consists of a set of features or attributes, and each example is associated with a class label indicating its category or class. A classifier's task is to learn a mapping from the input features to the output labels based on the examples provided in the training data. The output variable is discrete and represents a category or class.

Classification can be binary or multiclass. Confusingly, binary classification is also called binary logistic regression, where the output is a probability score between 0 and 1, representing the likelihood that an observation belongs to the target class.

Confidence In the context of classification tasks, a confidence score represents the model's confidence

in its prediction of the class label for a given input data point. It typically reflects the probability that the predicted class is correct.

In regression tasks a confidence score may represent the level of uncertainty associated with the predicted numerical value. It can indicate the range of values within which the true value is likely to fall.

Inference Once a classification model has been trained on a labeled dataset, it learns the relationship between the input features and the corresponding class labels. During inference, the trained model takes the features of a new data point as input and predicts the class label or category that the data point belongs to.

The process of inference typically involves the following: input preprocessing, prediction, decision, and (optionally) evaluation.

Unsupervised Learning

Unsupervised learning is a type of machine learning where the algorithm learns patterns, structures, or relationships in the data without explicit supervision or labeled output. In unsupervised learning, the algorithm is given input data without corresponding output labels, and its goal is to discover hidden patterns or structure in the data.

The primary objective of unsupervised learning is to find inherent structures within the data, such as clusters, subgroups, or underlying distributions. Unlike supervised learning, where the algorithm is provided with labeled examples and learns to map input data to output labels, unsupervised learning algorithms operate on raw, unlabeled data and are tasked with extracting meaningful information from it.

Common unsupervised learning tasks are clustering and anomaly detection.

Neuron

A neuron is a fundamental building block in a neural network that mimics the behavior of biological neurons in the human brain. Neurons process information by receiving input signals, performing computations, and producing output signals. A neuron typically has weights, a bias, and an activation function.

Neurons are organized into layers, with each layer consisting of one or more neurons. The input layer receives the input data, hidden layers perform intermediate computations, and the output layer produces the final output of the network. The connections between neurons, along with their associated weights, form the structure of the neural network, allowing it to learn complex patterns and relationships in the data.

During the training process, the weights of the neurons are adjusted iteratively using optimization algorithms such as gradient descent, so that the network learns to produce the correct outputs for a given set of inputs. This process of learning from data is known as training the neural network, and it involves adjusting the weights to minimize a loss function that measures the difference between the predicted outputs and the true outputs.

Weights Parameters that determine the strength of the connections between neurons in adjacent layers. Each neuron connected to neurons in the subsequent layer has an associated weight. These weights are adjusted during the training process to minimize the difference between the predicted outputs of the network and the true labels of the training data.

Biases Additional parameters added to each neuron in the network, which allow the network to learn more complex functions. Biases shift the output of each neuron to the left or right, affecting the activation function's decision boundary. Like weights, biases are learned during training to improve the performance of the network.

Glorot Uniform Also known as Xavier uniform initialization, Glorot Uniform is a method used to initialize the weights of neural network layers in a principled way.

The idea is to scale the initial weights of each layer in such a way that the variance of the activations

remains approximately constant across different layers of the network. This helps to ensure that the gradients neither explode nor vanish during the training process, improving training efficiency.

Another initialization technique is Kaiming Initialization or He Initialization.

Model Fitting

Model fitting, also known as model training or model estimation, is the process of adjusting the parameters of a machine learning model to minimize the difference between the model's predictions and the actual target values in the training data. The objective of model fitting is to create a model that accurately captures the underlying patterns or relationships in the data and can generalize well to new, unseen data.

In model fitting, the algorithm iteratively adjusts the parameters of the model using an optimization algorithm to minimize a predefined loss function. The loss function measures the discrepancy between the predicted values of the model and the actual target values in the training data. The goal is to find the values of the model parameters that minimize this discrepancy, effectively "fitting" the model to the training data.

Epoch Represents one complete forward, backward, and optimization pass through the entire training dataset during the training of a machine learning model. Multiple epochs are typically required to train a model effectively, with each epoch allowing the model to learn from and adjust to the training data incrementally.

Choosing the appropriate number of epochs depends on various factors, including the complexity of the model, the size of the dataset, and the desired level of convergence.

After completing all epochs, the training process is complete, and the model can be evaluated on a separate validation or test dataset to assess its performance.

Hyperparameters Hyperparameters are configuration settings that govern the learning process of a model but are not learned from the data. These parameters are set before the learning process begins and remain fixed throughout training. Unlike model parameters, which are learned during training (such as weights in a neural network), hyperparameters control aspects of the learning algorithm itself, such as its complexity, capacity, or optimization behavior.

Some common examples of hyperparameters include the number of epochs, learning rate, learning rate decay, regularization strength, number of hidden layers, activation functions, batch size, and dropout rate.

Convergence Refers to the process by which the parameters of the network, such as weights and biases, gradually adjust during training to minimize the loss function and improve the model's performance on the training data. In other words, convergence occurs when the optimization algorithm used to train the neural network finds parameter values that result in a satisfactory level of performance on the training dataset.

The goal of training a neural network is to find the optimal set of parameters that minimize a predefined loss function, which measures the discrepancy between the model's predictions and the true labels in the training data. Convergence is achieved when the optimization algorithm successfully finds parameter values that lead to a sufficiently low value of the loss function.

Generalization The ability of a trained model to perform well on new, unseen data that it hasn't been exposed to during training. In other words, a model that generalizes well can make accurate predictions or classifications on data it hasn't encountered before, beyond the examples it was trained on.

The ultimate goal of machine learning is to develop models that can generalize well to real-world scenarios and make reliable predictions on unseen data. Generalization is a crucial property of machine learning models because it ensures that they are able to capture underlying patterns and relationships in the data, rather than simply memorizing the training examples.

Memorization Refers to a phenomenon where a model learns to memorize the training data instead of learning the underlying patterns or relationships within the data. Memorization occurs when a model becomes overly complex and captures noise or specific characteristics of the training data that are not representative of the underlying distribution.

Memorization is typically associated with overfitting, which is a situation where a model performs well on the training data but fails to generalize to new, unseen data. Instead of learning the true underlying patterns, an overfitted model has essentially "memorized" the training examples and their corresponding labels.

Overfitting Common problem in machine learning where a model learns to capture noise or random fluctuations in the training data rather than the underlying patterns or relationships. Overfitting occurs when a model becomes overly complex and learns to fit the training data too closely, to the extent that it performs poorly on new, unseen data.

Memorization and overfitting can be mitigated by techniques such as regularization, cross validation, reducing model/parameter complexity, early stopping, or data augmentation.

Underfitting Problem where a model is too simple to capture the underlying patterns or relationships in the data. Underfitting occurs when a model is not able to learn from the training data effectively, resulting in poor performance on both the training data and new, unseen data.

An underfitted model fails to capture the complexity of the underlying data and is unable to make accurate predictions or classifications. It typically exhibits high bias and low variance.

Noise Random variations or disturbances in the input data, model parameters, or the training process itself. By effectively managing noise, neural networks can achieve better performance, robustness, and generalization to new data.

Data noise can include measurement errors, missing values, outliers, or irrelevant features. Data preprocessing techniques such as filtering, smoothing, and outlier detection can help mitigate data noise.

Model noise are random fluctuations or uncertainty in the model parameters or architecture. This can occur due to initialization randomness, numerical instability, or stochastic optimization algorithms. Regularization techniques such as weight decay or dropout can help reduce model noise by promoting smoother or more stable parameter values.

Training noise refers to variations or randomness introduced during the training process. This can include mini-batch sampling noise, learning rate scheduling, or early stopping. Techniques such as batch normalization or adaptive learning rate methods can help stabilize the training process and reduce training noise.

Co-Adaptation Phenomenon where certain neurons or groups of neurons within the network become highly interdependent during training. This interdependence can result in the network relying too heavily on specific subsets of neurons to make predictions, potentially leading to inefficiencies, reduced generalization performance, or overfitting.

Co-adaptation occurs when the optimization process of training the neural network encourages certain neurons to specialize in representing specific patterns or features in the training data. While specialization can be beneficial to some extent, excessive co-adaptation can hinder the network's ability to generalize to new, unseen data because it may become overly reliant on these specialized neurons.

One common example of co-adaptation is the phenomenon known as "dead neurons," where certain neurons become inactive or do not contribute meaningfully to the network's predictions. Dead neurons can occur when a neuron's weights are initialized in such a way that they are unable to effectively update during training, or when the neuron becomes saturated due to the activation function used.

Co-adaptation can be mitigated with regularization, early stopping, cross-validation, or redesigning a network's architecture.

Dead Neurons Neurons within a neural network that do not contribute to the learning process and remain inactive throughout training and inference. Dead neurons can occur due to various reasons and are typically associated with vanishing gradients, which result in the neuron's weights not being updated during training.

Some potential causes include: initialization issues, zero activation, saturated activation, low learning rates. These can be combated by choosing proper activation functions, initialization techniques (like Glorot or He), and applying regularization (such as dropout or batch normalization).

By addressing dead neurons and ensuring that all neurons actively contribute to the learning process, the overall performance and effectiveness of the neural network can be enhanced.

Expoding Gradients Phenomenon that can occur during the training of neural networks, particularly deep neural networks, where the gradients of the loss function with respect to the model parameters become extremely large. This can lead to numerical instability during optimization, causing the model's weights to grow exponentially and the training process to diverge.

Gradient explosion is the opposite of gradient vanishing, where gradients become extremely small, and it tends to occur in deep networks with a large number of layers, particularly during backpropagation, the process of computing gradients and updating the weights of the network.

Several factors can contribute to this explosion: poor weight initialization, unstable architecture, high learning rates, or poorly scaled features.

They can be mitigated with proper weight initialization, gradient clipping, and lowering learning rates.

Transfer Learning Learning technique where knowledge gained from training a model on one task is leveraged to improve performance on a related but different task. Instead of training a model from scratch on the target task, transfer learning involves using a pre-trained model as a starting point and fine-tuning it on the new task.

The core idea behind transfer learning is that features learned by a model on one task may be useful for solving a related task. By transferring knowledge from the source task to the target task, transfer learning can help improve model performance, reduce the amount of labeled data needed for training, and speed up the training process.

Activation Function

An activation function is applied to the output of neurons to introduce nonlinearity into a neural network. They aid in learning complex patterns and relationships in the data by enabling them to model nonlinear mappings between inputs and outputs. Activation functions play a crucial role in determining the output of a neuron and the overall behavior of the network.

The choice of activation function depends on the specific requirements of the task, the characteristics of the data, and the architecture of the neural network.

Linear Also known as the identity activation function, the linear activation function is a simple mathematical function that computes the output as a linear transformation of the input. In other words, the output of a neuron with a linear activation function is directly proportional to its input, with no non-linear transformation applied.

Sigmoid Also known as the logistic sigmoid function, it is a non-linear mathematical function that maps any real-valued number to a value between 0 and 1. It is commonly used in binary classification tasks, where the goal is to predict probabilities of belonging to one of two classes.

The sigmoid function has some limitations. First, the gradients of the sigmoid function become very small for large positive and negative inputs, leading to the vanishing gradient problem. This can make training deep neural networks with sigmoid activations slow and difficult. Second, it saturates when the input is very large or very small, leading to gradients close to zero. This can cause the model to learn slowly and exhibit poor convergence properties.

Rectifier Also known as rectified linear unit (ReLU), a rectifier function is a type of activation function commonly used in deep neural networks. It introduces non-linearity into the network by outputting the input directly if it is positive, and zero otherwise.

The rectifier function has become one of the most widely used activation functions in deep learning due to its simplicity, efficiency, and effectiveness in practice. It is often used in hidden layers of neural networks, especially in convolutional neural networks (CNNs) for computer vision tasks, and in feedforward neural networks for various other applications.

It has several advantageous properties. First, it is non-linear, which allows neural networks to learn complex mappings between inputs and outputs. Non-linear activation functions are essential for modeling non-linear relationships in data. Second, it encourages sparsity in activations, meaning

that only a subset of neurons in a layer will be active (i.e., have non-zero outputs) at any given time. This property can lead to more efficient computation and better generalization. Third, the rectifier is also computationally efficient to evaluate and differentiate.

Softmax Function that converts a vector of arbitrary real values into a probability distribution. It is commonly used as the activation function in the output layer of a neural network for multiclass classification tasks, where the goal is to predict the probability that an input belongs to each class.

The softmax function essentially exponentiates each score and normalizes them so that they sum up to 1, thus producing a valid probability distribution. This enables the model to output probabilities for each class, allowing for easier interpretation and comparison of predictions.

Softmax is commonly used as the final activation function in neural networks for tasks such as image classification, natural language processing, and other classification problems with multiple classes. It is often paired with the categorical cross-entropy loss function, which measures the difference between the predicted probabilities and the true labels, to train the model.

Loss Function

Also known as the cost function or objective function, it quantifies the difference between the predicted values of a model and the true values of the target variable. The loss function measures how well the model is performing on a given dataset by evaluating the discrepancy or error between its predictions and the actual outcomes.

A loss function must be differentiable (has a derivative for each point in its domain) and convex (does not have local minima). Different types of machine learning problems, such as classification, regression, and clustering, may require different loss functions.

Logarithmic Loss Also known as log-loss or cross-entropy loss, this loss function is used in classification tasks to measure the accuracy of a probabilistic classifier's predictions. It quantifies the difference between predicted probabilities and the actual binary or multiclass labels.

Logarithmic loss penalizes confident and incorrect predictions more heavily than confident and correct predictions. It is a strictly proper scoring rule, meaning that minimizing log-loss during training directly optimizes the model's ability to predict probabilities. Therefore, it is commonly used as the loss function when training probabilistic classifiers, particularly in binary and multiclass classification tasks.

Binary Cross-Entropy Loss Specific form of the logarithmic loss function used in binary classification tasks. It measures the difference between predicted probabilities and the actual binary labels (0 or 1) for each instance.

Binary cross-entropy loss is particularly used when training probabilistic classifiers such as logistic regression or neural networks with a sigmoid activation function in the output layer. It is a strictly proper scoring rule, meaning that minimizing binary cross-entropy loss during training directly optimizes the model's ability to predict probabilities.

Categorical Cross-Entropy Loss Also known as multiclass cross-entropy loss, this is a specific form of the logarithmic loss function used in multiclass classification tasks. It measures the difference between predicted probabilities and the actual class labels for each instance.

The categorical cross-entropy loss computes the loss for each instance and class pair and then averages them over all instances. It penalizes confident and incorrect predictions more heavily than confident and correct predictions.

Categorical cross-entropy loss is particularly used when training probabilistic classifiers with the softmax activation function in the output layer. It is a strictly proper scoring rule, meaning that minimizing categorical cross-entropy loss during training directly optimizes the model's ability to predict probabilities for each class.

Mean Absolute Error Also known as L1 loss, mean absolute error (MAE) is a commonly used loss function in regression tasks. It measures the average absolute difference between the predicted values and the actual values.

MAE loss is less sensitive to outliers because it penalizes each error linearly. This means that large errors have the same impact on the loss regardless of their sign.

Mean Squared Error Also known as L2 loss, mean squared error (MSE) is a commonly used loss function in regression tasks. It measures the average squared difference between the predicted values and the actual values.

MSE loss penalizes larger errors more heavily than smaller errors due to the squaring operation. As a result, MSE loss is more sensitive to outliers compared to other loss functions like Mean Absolute Error (MAE) loss.

Optimization

Optimization refers to the process of adjusting the parameters (weights and biases) of the network to minimize a defined loss function. The goal of optimization is to find the set of parameters that results in the best performance of the neural network on a given task, such as classification or regression.

Optimization in neural networks typically involves iterative algorithms that update the parameters based on the gradients of the loss function with respect to those parameters. The gradients indicate the direction and magnitude of the steepest ascent of the loss function, and optimization algorithms adjust the parameters in the opposite direction of the gradients to descend towards the minimum of the loss function.

Gradient Descent Optimization algorithm used to find a local minimum of a loss function that takes repeated steps in the opposite direction of the (approximate) gradient of the function at the current point, because this is the direction of steepest descent.

Stochastic Gradient Descent Optimizer that picks a random sample from the dataset and runs a training epoch and it updates each training example's parameters one at a time. While frequent updates can offer more detail and speed, it can result in noisy gradients, but this can sometimes be helpful to escape local minima.

Batch Gradient Descent Optimizer that sums error for each point in a training set, updating the model only after a training epoch; after all training examples have been evaluated. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn't the most ideal, finding the local minimum versus the global one.

Mini-Batch Gradient Descent Optimizer that combines concepts from both stochastic gradient descent and batch gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches.

Stochastic Gradient Descent with Momentum Optimizer that extends the basic SGD algorithm by incorporating a momentum term, which helps accelerate convergence and reduce oscillations during training. In it, the update rule is modified to include a momentum term accumulates a fraction of the previous gradients to determine the update direction.

The momentum term determines how much of the previous gradients to retain when computing the current update direction. A higher momentum value means that the algorithm will rely more on the accumulated gradient history, leading to smoother and more stable updates that help the optimization process escape from local minima and plateaus more effectively.

SGD with momentum helps overcome some of the limitations of standard SGD, such as slow convergence and oscillations in the parameter space. It is particularly useful for training deep neural networks with high-dimensional parameter spaces, where the optimization landscape may be complex and non-convex.

AdaGrad Also known as Adaptive Gradient, this optimizer adapts the learning rate of each parameter during training based on the historical gradients observed for that parameter.

In other words, the learning rate of each parameter for each step will be based on the magnitude of the gradients that have been observed for that parameter in the past. Parameters that have received large gradients in the past will have their learning rates decreased, while parameters that have received small gradients will have their learning rates increased. This allows AdaGrad to

automatically adapt the learning rates of different parameters based on their individual behavior during training.

AdaGrad effectively reduces the learning rate for frequently occurring parameters and increases the learning rate for infrequently occurring parameters, which can help improve convergence and stability during training.

However, one drawback of AdaGrad is that the learning rates can become too small over time, leading to premature convergence or slow progress in later stages of training. This issue has led to the development of variations of AdaGrad, such as RMSprop and Adam, which address some of its limitations while retaining its adaptive learning rate capabilities.

AdaDelta Adaptive learning rate optimization algorithm that is designed to address some of the limitations of AdaGrad, particularly the diminishing learning rates problem.

Like AdaGrad, AdaDelta (Adaptive Delta) adapts the learning rate for each parameter based on the historical gradients observed for that parameter. However, AdaDelta addresses the issue of diminishing learning rates by dynamically adapting the learning rate based on a moving average of the past gradients, rather than accumulating all past squared gradients.

The key idea behind AdaDelta is to compute an exponentially decaying average of past squared gradients, referred to as the root mean square (RMS) of the gradients, and use this average to scale the learning rate for each parameter.

AdaDelta effectively adapts the learning rate for each parameter based on the magnitude of the gradients observed for that parameter, allowing it to converge more quickly and robustly compared to fixed learning rate methods like AdaGrad. It also eliminates the need for manually tuning the learning rate or choosing a global learning rate schedule, making it easier to use in practice.

RMSProp Root Mean Square Propagation is an optimization algorithm used for training neural networks. It addresses the limitation of AdaGrad's diminishing learning rates by using a moving average of the squared gradients to scale the learning rates of the model parameters.

In RMSprop, the learning rate is adaptively adjusted for each parameter based on the magnitude of the gradients observed for that parameter. Parameters that receive large gradients will have their learning rates decreased, and parameters that receive small gradients will have their learning rates increased.

While similar to AdaDelta, the key differences is in the way they compute the moving average of past squared gradients and in the update rule for adjusting the model parameters.

Adam Short for Adaptive Moment Estimation, this popular optimizer combines ideas from two other optimization algorithms, RMSprop and momentum, to achieve efficient and robust optimization.

Adam maintains two moving averages: the first moment (the mean) of the gradients and the second moment (the variance) of the gradients. These moving averages are used to adaptively adjust the learning rates for each parameter during training.

Adam adapts the learning rates for each parameter based on the magnitude of the gradients and the past gradients' history. It effectively combines the benefits of momentum (which helps accelerate convergence) and RMSprop (which adapts the learning rates based on the magnitude of the gradients) to optimize deep neural networks efficiently.

Regularization

Regularization refers to techniques used to prevent overfitting, which occurs when a model learns to fit the training data too closely, capturing noise and irrelevant patterns that do not generalize well to new, unseen data. Regularization methods impose constraints on the model's parameters during training, discouraging overly complex models that may memorize the training data rather than learning underlying patterns.

Some regularization techniques are: L1 and L2 regularization, dropout, and early stopping.

L1 Regularization Regularization technique also called a lasso in which a penalty term proportional to the absolute values of the model's parameters is added to the loss function. This encourages

sparsity in the parameter values, effectively selecting only the most important features and reducing the overall complexity of the model.

L1 regularization is rarely used on its own.

L2 Regularization Regularization technique also known as a ridge in which a penalty term proportional to the squared values of the model's parameters is added to the loss function. This encourages smaller parameter values, effectively preventing any single parameter from becoming too large and dominating the model's behavior. L2 regularization is particularly effective at preventing weights from growing too large and helps to improve the generalization performance of the model.

Elastic Net Regularization Technique that combines L1 and L2 regularization by adding a penalty term that is a linear combination of both L1 and L2 penalties. This allows for the benefits of both L1 and L2 regularization, effectively promoting sparsity while also preventing overly large parameter values.

Dropout Technique where randomly selected neurons are ignored or "dropped out" during training with a certain probability. This forces the network to learn more robust features and prevents individual neurons from becoming too specialized or co-adapting to each other.

Early Stopping Involves monitoring the model's performance on a separate validation dataset during training and stopping the training process when the validation performance starts to degrade. This prevents the model from overfitting by halting training before it starts to memorize the training data too closely.

Dataset

Collections of data points used for training, validation, and testing the models. These datasets are typically organized into features (input variables) and labels (target variables) and are used to train the neural network to learn patterns and relationships between the features and labels.

It is important to use high-quality datasets that are representative of the problem domain and provide sufficient diversity and coverage to train robust and generalizable models.

Features Input variables or attributes that are used to represent the data being processed by the network. Features are essentially the characteristics or properties of the data that are relevant to the task at hand. They provide the raw information that the neural network processes to make predictions or perform a specific task.

Features can take many forms, depending on the nature of the data and the problem being solved. In image classification tasks features may represent pixel values or higher-level visual patterns extracted from images. In natural language processing tasks, features may represent words, phrases, or other linguistic elements extracted from text data. In tabular data analysis, features may represent numerical or categorical variables describing different aspects of the data.

Training Data Dataset used to train a machine learning model. It consists of a collection of input-output pairs, where each input is associated with a corresponding target output. The purpose of training data is to teach the model to learn patterns or relationships between the input features and the target outputs, so that it can make accurate predictions or classifications on new, unseen data.

In supervised learning, the training data consists of labeled examples, where each input is paired with the correct output. The model learns to map inputs to outputs by observing these examples and adjusting its parameters to minimize the difference between its predictions and the true labels in the training data.

Testing Data Also known as validation or evaluation data, it is a separate dataset used to assess the performance of a trained machine learning model. Unlike training data which is used to train the model's parameters, testing data is used to evaluate how well the trained model generalizes to new, unseen data.

The primary purpose of testing data is to estimate the model's performance on real-world data that it has not been exposed to during training. By evaluating the model on a separate dataset,

practitioners can assess its generalization ability and identify any issues such as overfitting or underfitting.

Testing data should be representative of the same distribution as the training data to ensure fair evaluation.

Batch Subset of the training data that is used together to compute the gradients of the model parameters during the optimization process. Instead of updating the model's parameters based on the gradients computed from individual training samples, batches allow for more efficient computation by aggregating gradients over multiple samples.

The size of the batch, known as the batch size, is a hyperparameter that can be adjusted based on factors such as the computational resources available, the size of the dataset, and the characteristics of the optimization problem. Common batch sizes range from small values (e.g., 32 or 64) to larger values (e.g., 128, 256, or even larger) depending on the specifics of the training process.

Augmentation Technique used to artificially increase the size of a training dataset by applying various transformations to the existing data samples. These transformations include operations such as rotation, translation, scaling, flipping, cropping, and changing brightness or contrast. By applying such transformations, data augmentation generates new training samples that are variations of the original data, thus providing more diverse examples for the neural network to learn from.

Data augmentation is particularly useful in scenarios where the size of the training dataset is limited or when the dataset lacks diversity. It helps to improve the generalization performance of the neural network by exposing it to a wider range of variations and reducing overfitting to the training data. Additionally, data augmentation can help to make the model more robust to variations in input data that it may encounter during deployment.

Preprocessing The steps taken to prepare a raw dataset before feeding it into the neural network for training. Preprocessing is an essential part of the machine learning pipeline and can have a significant impact on the performance of the model.

Some common steps include normalization, balancing, scaling, flattening, and shuffling.

Normalization typically scales the features to have a mean of 0 and a standard deviation of 1, while standardization scales the features to have a mean of 0 and a variance of 1.

Balancing addresses the issue of class imbalance in classification, where where certain classes may occur much less frequently than others in the dataset.

Scaling constraints input features to a specific range, such as [0, 1] or [-1, 1]. Feature scaling can help improve the convergence of optimization algorithms and prevent numerical instabilities.

Flattening is the conversion of a multidimensional tensor into a one-dimensional vector. For image data flattening is used to convert the two-dimensional pixel grid of an image into a one-dimensional vector that can be fed into a neural network's input layer.

Shuffling involves randomly reordering the data samples in a dataset, typically to improve the performance and generalization of machine learning models. It helps to reduce bias, avoid overfitting, and improving optimization. It is typically performed randomly before each epoch of training so that the model sees a different order of data samples in each epoch, helping to prevent it from memorizing the training data order.