Artificial Neural Network

Enrique Alifio Ditya

What is an Artificial Neural Network?

Artificial Neural Network (ANN) is a predictive model that is inspired by the biological neural network. It is a collection of connected nodes called neurons. Each neuron has a set of weights and biases that are adjusted during the training process. The weights and biases are used to calculate the output of the neuron. The output of the neuron is then passed to the next layer of neurons. The process is repeated until the output layer is reached. The output layer is the final layer of the network and it is where the prediction is made.

What is the difference between a Neural Network and a Deep Neural Network?

A Neural Network is a network that has one or more hidden layers. A Deep Neural Network is a network that has more than one hidden layer. A Deep Neural Network is also called a Deep Learning model.

General Components of a Neural Network

Input Layer

The input layer is the first layer of the network. It is where the input is fed into the network. The input layer usually comprises of the same number of neurons as the number of features in the input data.

Output Layer

The output layer is the last layer of the network. It is where the output is calculated. The output layer usually comprises of the same number of neurons as the number of classes in the output data.

Hidden Layer

The hidden layer is the layer between the input layer and the output layer. It is where the weights and biases are adjusted during the training process. The number of neurons on each layers are arbitrary, and it is a cause of hyperparameter tuning and research.

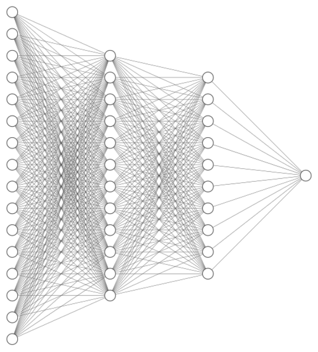

Figure 1: Deep Neural Network

How does it work?

Any kind of Neural Network generally works in two main phases: the forward pass and the backward pass. The forward pass is where the input is fed into the network and the output is calculated. The backward pass is where the weights and biases are adjusted based on the error of the output. The process is repeated until the error is minimized. The error is calculated using a loss function. The loss function is a function that calculates the error of the output. The loss function is usually a function that is differentiable. The loss function is used to calculate the gradient of the error. The gradient is then used to adjust the weights and biases. For a Multi-Layer Perceptron (MLP), it algorithmically works as follows:

1. Initialize the weights and biases of the network with random values.
2. Feed the input into the network (Forward Pass).
3. Calculate the error of the output using a loss function.
4. Calculate the gradient of the error.
5. Adjust the weights and biases using the gradient (Backward Pass).
6. Repeat steps 2-5 until the error is minimized for a given number of epochs, or until the performance converges.

The Forward Pass

The forward pass, also known as forward propagation, works by feeding the input into the network and calculating the output. The output is calculated by multiplying the input with the weights and adding the biases. The output is then passed to the next layer of neurons. The process is repeated until the output layer is reached. The output layer is the final layer of the network and it is where the prediction is made.

he Backward Pass

The backward pass, also known as backward propagation, works by propagating the error of the output, calculated by a selected loss function, backwards through the layer. This is done to refine the weights and biases such that the error from the loss function is minimized. This works by calculating the gradient of the error and adjusting the weights and biases using the negative gradient. The process is repeated until the error is minimized for a given number of epochs, or until the performance converges.

Terminologies

Activation Function

Activation functions are functions that introduces non-linearity into the neural network architecture. This is done by applying a non-linear function to the output of the neuron. Non-linearity is essential to the neural network architecture because it allows the network to learn complex patterns in the data. Without non-linearity, the neural network architecture would be a linear regression model. There are many types of activation functions, but the most common ones are:

Sigmoid

The Sigmoid function is a function that is bounded between 0 and 1. It is used in the output layer of a binary classification problem.

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

Tanh

The Tanh function is a function that is bounded between -1 and 1. It is used in the output layer of a multi-class classification problem.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU

The ReLU function is a function that is bounded between 0 and infinity. It is used in the hidden layers of a neural network.

$$ReLU(x) = \max(0, x)$$

Leaky ReLU

Leaky ReLU is similar to ReLU, but it has a small slope for negative values. It is used in the hidden layers of a neural network. This is designed to solve the dying ReLU problem.

$$LeakyReLU(x) = \max(0.01x, x)$$

Softmax

The Softmax function is a function that is bounded between 0 and 1. It is used in the output layer of a multi-class classification problem.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Loss Function

The loss function is a differentiable function that is used to calculate the errors of the output. Different loss functions generates different convergencies, meaning one loss function's performance may converge into a better result than another. The most common loss functions are:

Mean Squared Error (MSE)

The Mean Squared Error (MSE) is a loss function that calculates the mean of the squared errors of the output. It is most commonly used in regression problems.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

Binary Cross Entropy (BCE)

The Binary Cross Entropy (BCE) is a loss function that calculates the cross entropy of the output. It is used in binary classification problems. The idea of the Binary Cross Entropy is to penalize the model more if the prediction is further away from the actual value. Cross entropy is a function that is used to measure the difference between two probability distributions. It is calculated by taking the negative sum of the actual value multiplied by the logarithm of the predicted value.

$$BCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})$$

Categorical Cross Entropy (CCE)

The Categorical Cross Entropy (CCE) is a loss function that calculates the cross entropy of the output. It is used in multi-class classification problems. It differs from the Binary Cross Entropy in that it uses the Softmax function to calculate the cross entropy.

$$CCE = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y_i})$$

Optimizer

The optimizer is an algorithm that is used to adjust the weights and biases of the network. It is used to minimize the error of the output. The most common optimizers are:

Stochastic Gradient Descent (SGD)

SGD works by adjusting the weights and biases using the negative gradient of the error. It is the most basic optimizer and it is the most commonly used optimizer. It is also the most unstable optimizer because it is prone to getting stuck in local minima.

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta)$$

Momentum

Momentum is an optimizer that is designed to solve the problem of SGD getting stuck in local minima. It works by adding a momentum term to the gradient of the error. This momentum term is a fraction of the previous gradient. This allows the optimizer to escape local minima.

$$v_{t+1} = \gamma v_t + \alpha \nabla_\theta J(\theta)$$

$$\theta_{t+1} = \theta_t - v_{t+1}$$

Nesterov Accelerated Gradient (NAG)

NAG is an optimizer that is designed to solve the problem of Momentum overshooting the global minima. It works by calculating the gradient of the error using the momentum term. This allows the optimizer to converge faster than Momentum.

$$v_{t+1} = \gamma v_t + \alpha \nabla_\theta J(\theta - \gamma v_t)$$
$$\theta_{t+1} = \theta_t - v_{t+1}$$

Adagrad

Adagrad is an optimizer that is designed to solve the problem of SGD having a slow convergence rate. It works by adjusting the learning rate of the optimizer based on the previous gradients. This allows the optimizer to converge faster than SGD.

$$g_{t+1} = g_t + \nabla_\theta J(\theta)^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{g_{t+1} + \epsilon}} \nabla_\theta J(\theta)$$

RMSProp

RMSProp is a form of improvement of Adagrad. It works by adjusting the learning rate of the optimizer based on the previous gradients. This allows the optimizer to converge even faster.

$$g_{t+1} = \gamma g_t + (1 - \gamma) \nabla_\theta J(\theta)^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{g_{t+1} + \epsilon}} \nabla_\theta J(\theta)$$

Adam

Adam is one of the most commonly used optimizer in recent years. It is a form of improvement of RMSProp. It works by adjusting the learning rate of the optimizer based on the previous gradients and the previous squared gradients. The reason it is popular and successful is because it combines the best of both worlds of Momentum and RMSProp.

$$m_{t+1} = \gamma_1 m_t + (1 - \gamma_1) \nabla_\theta J(\theta)$$

$$g_{t+1} = \gamma_2 g_t + (1 - \gamma_2) \nabla_\theta J(\theta)^2$$
$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \gamma_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1 - \gamma_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{g}_{t+1} + \epsilon}} \hat{m}_{t+1}$$

Other Types of Neural Networks

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a specialized type of deep learning architecture designed to process and recognize patterns in visual data, such as images and videos. It is used as a feature extractor in computer vision tasks, due to its ability to perceive locality and spatial relationships between pixels.

A typical CNN comprises multiple layers, each serving a specific purpose in feature extraction, transformation, and classification. The most commonly found types of layers in a CNN are:

Convolutional Layer

This is the core building block of a CNN. It consists of a set of learnable filters (also called kernels) that slide over the input image and perform element-wise multiplications followed by summation (a convolution operation). These filters act as feature detectors, capturing local patterns in the input, such as edges, textures, or other visual attributes. As the network progresses through successive convolutional layers, it learns more abstract and complex features.

Activation Layer

After each convolutional layer, a non-linear activation function is applied element-wise to introduce non-linearity into the network. ReLU (Rectified Linear Unit) is the most commonly used activation function as it helps alleviate the vanishing gradient problem and speeds up convergence.

Pooling Layer

The pooling layers serve to downsample the spatial dimensions of the feature maps, reducing computational complexity and making the network more robust to small spatial translations. Max-pooling is a popular pooling technique, where the maximum value from a specific region of the feature map is retained while discarding the rest

Batch Normalization Layer

Batch normalization is a technique for improving the speed, performance, and stability of artificial neural networks. It is used to normalize the input layer by adjusting and scaling the activations. This is done to make the network more robust to small changes in the input data.

Dropout Layer

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a form of regularization that works by randomly setting the outputs of neurons to zero during the training process. This is done to prevent overfitting.

Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) are a type of artificial neural network designed to process sequential data, such as time series, natural language, and audio. Unlike traditional feedforward neural networks, RNNs introduce a feedback loop, allowing information to persist across time steps, making them capable of modeling temporal dependencies in data. In other words, RNNs are able to remember past information in order to make predictions. This is essential especially to Markovian processes, where the current state is dependent on the previous state.

$$RNN(x_t, h_{t-1}) = h_t$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

At the core of an RNN is a recurrent cell, typically represented as a directed cycle in the network graph. This cell takes an input and the previous hidden state, processes them through a set of learnable parameters, and produces an output and a new hidden state. During training, the network "unrolls" through time, creating a chain of interconnected cells, one for each time step in the input sequence.

RNNs face challenges with long-term dependencies due to the vanishing gradient problem. When training, gradients can diminish exponentially as they propagate backward through time, making it difficult for the network to learn long-range dependencies. This issue hinders the effectiveness of traditional RNNs in capturing long-term patterns in sequential data. This is why, in most cases of forecasting, a problem of extrapolation (predicting values outside the range of the training data) is encountered. Through personal research, I found that in most cases of showcasing RNNs capability in predicting random walk data such as stock prices, the model is only acceptable to predict the next value, but not the next n values.

References

Personal research and projects (Private repositories).