Introducción a Redes Neuronales

Dr. Mauricio Toledo-Acosta

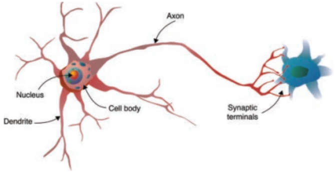
Diplomado Ciencia de Datos con Python

Table of Contents

Introduction

Neural Networks

Artificial neural networks are machine learning techniques that simulate the mechanism of learning in biological organisms. The human nervous system contains cells, referred to as neurons. The neurons are connected to one another by of axons and dendrites, and the connecting regions between axons and dendrites are called synapses. The strengths of synaptic connections often change in response to external stimuli. This change is how learning takes place in living organisms.



El modelo lineal de clasificación

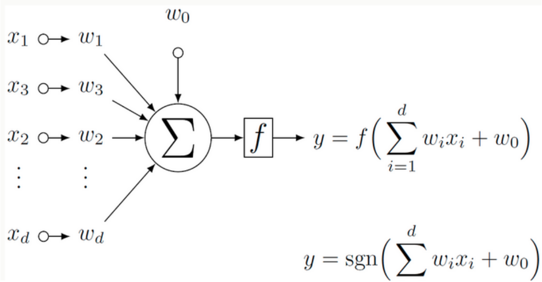
- The Perceptrón algorithm (Rosenblatt, 1961) played an important role in Machine Learning history. It was first simulated in a computer IBM 704 at Cornell in 1957. By the early 60s, a dedicated hardware was designed to implement the learning algorithm.

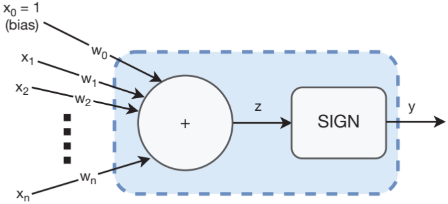
El modelo lineal de clasificación

- The Perceptrón algorithm (Rosenblatt, 1961) played an important role in Machine Learning history. It was first simulated in a computer IBM 704 at Cornell in 1957. By the early 60s, a dedicated hardware was designed to implement the learning algorithm.
- It was criticized by Marvin Minsky, who showed the limitations of the perceptron algorithm when dealing with a non linear separable set of points. This caused a void in the neural computation research lasting until the mid 80s.

Table of Contents

The Rosenblatt Perceptron





The Algorithm

We iterate over the training set

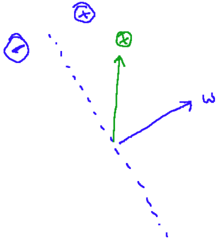
$$(x_1, y_1), (x_2, y_2) \cdots , (x_N, y_N).$$

The Algorithm

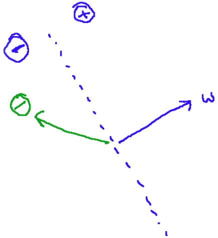
We iterate over the training set

$$(x_1, y_1), (x_2, y_2) \cdots , (x_N, y_N).$$

a) If the point is well classified:



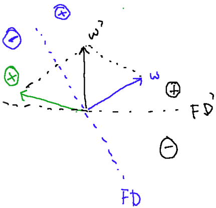
(a) If $y = +1$



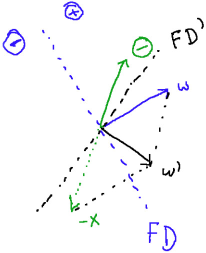
(b) If $y = -1$

The Algorithm

- ⓑ If the point is not well classified:



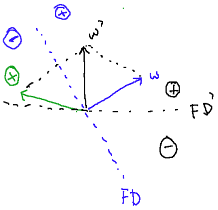
(a) If $y = +1$



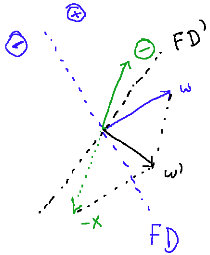
(b) If $y = -1$

The Algorithm

- If the point is not well classified:



(a) If $y = +1$



(b) If $y = -1$

We update the vector w with $w + yx$

The Algorithm

For each epoch, the algorithm is summarized as follows:

Data: A: training set of points, $X = \{x_1, ..., x_N\} \subset \mathbb{R}^{D+1}$,
$Y = \{y_1, ..., y_N\}$: set of labels
$\eta > 0$: learning rate.
Result: $\mathbf{w}$: weight vector defining the decision frontier.

Function `Perceptron`(X, y, η)**:**
- $\mathbf{w} = 0$;
- converged $=$ `False`;
- **while** *converged* $==$ `False` **do**
  - **for** $i \in \{1, ..., N\}$ **do**
    - **if** $y_i \langle \mathbf{w}, x \rangle \leq 0$ **then**
      - $w = w + y_i \eta x_i$;
      - converged $=$ `False`
- **return** $\mathbf{w}$;

The Algorithm

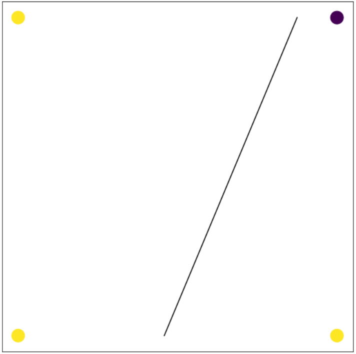
For each epoch, the algorithm is summarized as follows:

Data: A: training set of points, $X = \{x_1, \dots, x_N\} \subset \mathbb{R}^{D+1}$,
$Y = \{y_1, \dots, y_N\}$: set of labels
$\eta > 0$: learning rate.
Result: $\mathbf{w}$: weight vector defining the decision frontier.

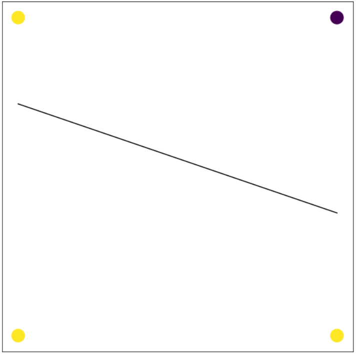
Function `Perceptron`(X, y, η)**:**
    $\mathbf{w} = 0$;
    converged $=$ `False`;
    while *converged* $==$ `False` **do**
        for $i \in \{1, \dots, N\}$ **do**
            if $y_i \langle \mathbf{w}, x \rangle \leq 0$ **then**
                $w = w + y_i \eta x_i$;
                converged $=$ `False`
    return $\mathbf{w}$;

$\mathbf{w}$ is a linear combination of $x_1, \dots, x_N$.

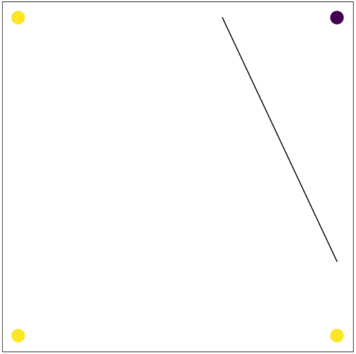
Example



Example

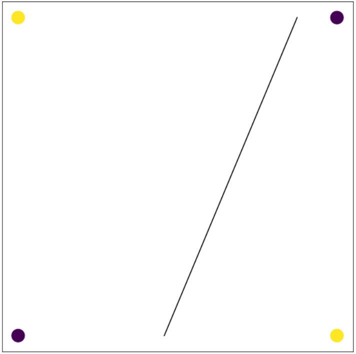


Example



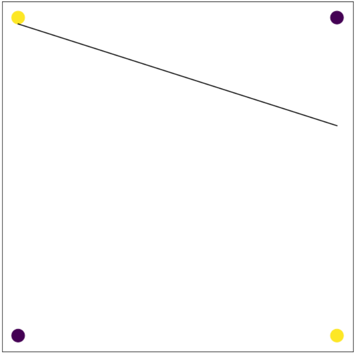
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



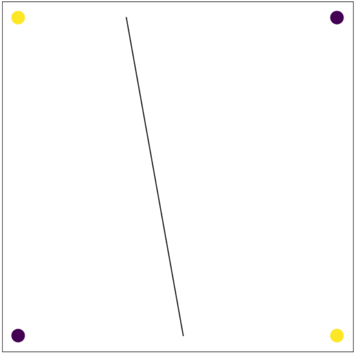
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



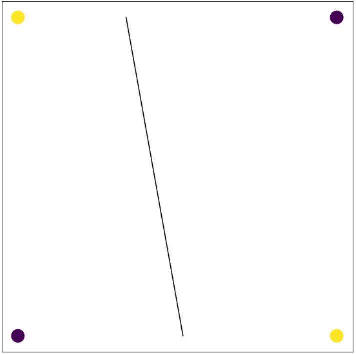
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



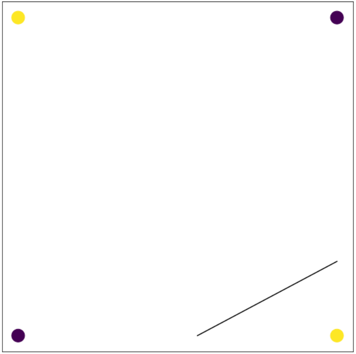
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



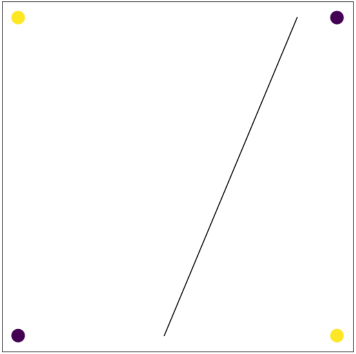
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



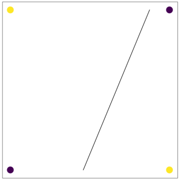
Limitations of the Perceptron

What happens if a straight line cannot separate the data points?



Limitations of the Perceptron

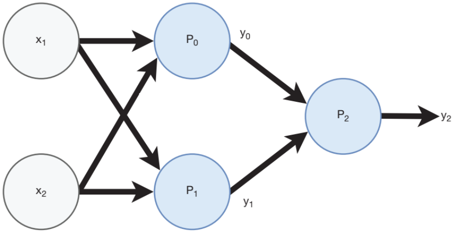
What happens if a straight line cannot separate the data points?



Alternatives:

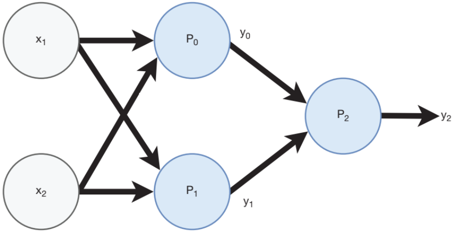
- We change the model of a neuron or
- We combine multiple of them to solve this limitation.

Combining Perceptrons



$$
\begin{array}{r|c}
-1 & (-1,-1) \\
+1 & (1,-1) \\
+1 & (-1,1) \\
-1 & (1,1)
\end{array}
$$

Combining Perceptrons



This neural network is one of the simplest examples of a fully connected feedforward network. Fully connected means that the output of each neuron in one layer is connected to all neurons in the next layer. Feedforward means that there are no backward connections. A multilevel neural network has an input layer, one or more hidden layers, and an output layer. The input layer does not contain neurons but contains only the inputs themselves.

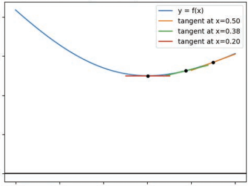
Table of Contents

How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.

How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.
- The derivative at the point x that minimizes the value of f is 0.



How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.
- The derivative at the point x that minimizes the value of f is 0.
- Given an initial value x, the sign of the derivative $f'(x)$ indicates in what direction to adjust x to reduce the value of $f(x)$.

How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.
- The derivative at the point x that minimizes the value of f is 0.
- Given an initial value x, the sign of the derivative $f'(x)$ indicates in what direction to adjust x to reduce the value of $f(x)$.
- Gradient descent uses the value of the derivative to decide how much to adjust x.

$$x_{n+1} = x_n - \eta f'(x_n).$$

How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.
- The derivative at the point x that minimizes the value of f is 0.
- Given an initial value x, the sign of the derivative $f'(x)$ indicates in what direction to adjust x to reduce the value of $f(x)$.
- Gradient descent uses the value of the derivative to decide how much to adjust x.

$$x_{n+1} = x_n - \eta f'(x_n).$$

- If the learning rate η is to too large, gradient descent can overshoot the solution and fail to converge.

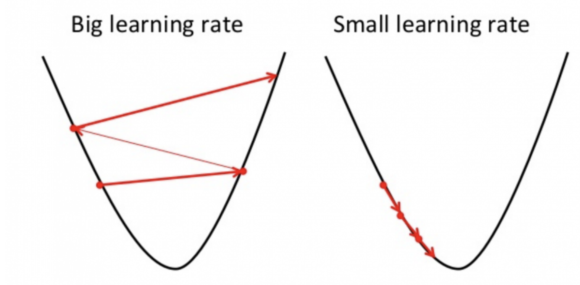
How to find the minimum of a function

- Let $f(x)$ be a function, we want to find the minimum of f.
- The derivative at the point x that minimizes the value of f is 0.
- Given an initial value x, the sign of the derivative $f'(x)$ indicates in what direction to adjust x to reduce the value of $f(x)$.
- Gradient descent uses the value of the derivative to decide how much to adjust x.

$$x_{n+1} = x_n - \eta f'(x_n).$$

- If the learning rate η is to too large, gradient descent can overshoot the solution and fail to converge.
- The algorithm is not guaranteed to find the global minimum because it can get stuck in a local minimum.

Learning Rate

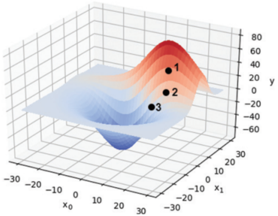


The high-dimensional case

- Now, consider the function $f(x_0, x_1)$.

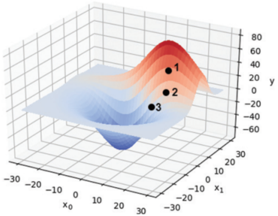
The high-dimensional case

- Now, consider the function $f(x_0, x_1)$.
- The gradient is a vector consisting of partial derivatives and indicates the direction in the input space that results in the steepest ascent for the value of f.



The high-dimensional case

- Now, consider the function $f(x_0, x_1)$.
- The gradient is a vector consisting of partial derivatives and indicates the direction in the input space that results in the steepest ascent for the value of f.



- If we are at the point $\mathbf{x_n} = (x_0^{(n)}, x_1^{(n)})$ and want to minimize y, then we choose our next point as

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \eta \nabla f(\mathbf{x_n})$$

Gradient-based Learning

When applying gradient descent to our neural network, we consider input values $\mathbf{x}$ to be constants, with our goal being to adjust the weights $\mathbf{w}$.

- Which function we want to minimize? A loss function.

Gradient-based Learning

When applying gradient descent to our neural network, we consider input values $\mathbf{x}$ to be constants, with our goal being to adjust the weights $\mathbf{w}$.

- Which function we want to minimize? A loss function.
- In the context of an optimization algorithm, the function used to evaluate a candidate solution is referred to as the objective function. With neural networks, we seek to minimize the error. As such, the objective function is often referred to as a cost function or a loss function.

Gradient-based Learning

- In the case of the Perceptron, the loss function is given by

$$L^{(0/1)}(\mathbf{w}) = (y_i - \mathrm{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle)^2$$

Gradient-based Learning

- In the case of the Perceptron, the loss function is given by

$$L^{(0/1)}(\mathbf{w}) = \frac{1}{2}(y_i - \text{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle)^2 = 1 - y_i \text{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle$$

- This function is not smooth, we use the smooth surrogate loss function

$$L(\mathbf{w}) = \max\{-y_i \text{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle, 0\}.$$

Gradient-based Learning

- In the case of the Perceptron, the loss function is given by
- This function is not smooth, we use the smooth surrogate loss function

$$L(\mathbf{w}) = \max\{-y_i \mathrm{sign}\langle \mathbf{w}, \mathbf{x}_i \rangle, 0\}.$$

- Applying gradient descent

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n - \eta \nabla L(\mathbf{w}) \\ &= \begin{cases} \mathbf{w}_n + \eta y_i \mathbf{x}_i, & \text{well classified} \\ \mathbf{w}_n, & \text{misclassified} \end{cases} \end{aligned}$$

Loss Functions

Other examples of loss functions

Keras Losses

Loss Functions: MSE

Mean Square Error (MSE)

$$L(y, t) = (t - y)^2.$$

- L2 loss.

Loss Functions: MSE

Mean Square Error (MSE)

$$L(y, t) = (t - y)^2.$$

- L2 loss.
- Good for regression tasks.

Loss Functions: MSE

Mean Square Error (MSE)

$$L(y, t) = (t - y)^2.$$

- L2 loss.
- Good for regression tasks.
- Trivial derivative for gradient descent.

Loss Functions: MAE

Mean Absolute Error (MAE)

$$L = |t - y|.$$

- L1 loss.
- More robust to outliers than mse.
- Good for regression tasks.
- Discontinuity in its derivative.

Loss Functions: MAE

Mean Absolute Error (MAE)

$$L = |t - y|.$$

- L1 loss.
- More robust to outliers than mse.
- Good for regression tasks.
- Discontinuity in its derivative.

Loss Functions: Hinge

Hinge Loss

$$L = \max\{-y_i \hat{y}_i, 0\}.$$

- Used in SVMs and Perceptron.
- Penalizes errors, but also correct predictions of low confidence (probabilities).
- Good for binary classification tasks.

Loss Functions: Categorical cross entropy

Categorical cross entropy

$$L = \sum_{i}^{n} y_i \log\left(\hat{y}_i\right).$$

- Good for multi-class classification problems.
- Considers y to be a one-hot encoding vector in n classes.

Optimizers: Gradient Descent

Gradient descent is the most basic but most used optimization algorithm.

Optimizers: Gradient Descent

Gradient descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms.

Optimizers: Gradient Descent

Gradient descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms. Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function.

Optimizers: Gradient Descent

Advantages:

- Easy computation.
- Easy to implement.
- Easy to understand.

Optimizers: Gradient Descent

Advantages:

- Easy computation.
- Easy to implement.
- Easy to understand.

Disadvantages:

- May trap at local minima.
- Weights are changed after calculating gradient on the whole dataset. May take a long time to converge.
- Requires large memory to calculate gradient on the whole dataset.

Optimizers

- Gradient Descent
- Stochastic Gradient Descent
- Stochastic Gradient descent with momentum
- Mini-Batch Gradient Descent
- Adagrad
- RMSProp
- AdaDelta
- Adam

Keras Optimizers

Stochastic Gradient Descent

- Instead of taking the whole dataset for each iteration in each iteration, we randomly shuffle the data and take a batch.

Stochastic Gradient Descent

- Instead of taking the whole dataset for each iteration in each iteration, we randomly shuffle the data and take a batch.
- The path took by the algorithm is full of noise as compared to the gradient descent algorithm.

Stochastic Gradient Descent

- Instead of taking the whole dataset for each iteration in each iteration, we randomly shuffle the data and take a batch.
- The path took by the algorithm is full of noise as compared to the gradient descent algorithm.
- Due to an increase in the number of iterations, the overall computation time increases. Still, the computation cost is still less than that of the gradient descent optimizer.

Stochastic Gradient Descent

- Instead of taking the whole dataset for each iteration in each iteration, we randomly shuffle the data and take a batch.
- The path took by the algorithm is full of noise as compared to the gradient descent algorithm.
- Due to an increase in the number of iterations, the overall computation time increases. Still, the computation cost is still less than that of the gradient descent optimizer.
- If the data is enormous and computational time is an essential factor, stochastic gradient descent should be preferred over batch gradient descent algorithm.

Adagrad: Adaptive gradient descent

- The adaptive gradient descent algorithm uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training.

Adagrad: Adaptive gradient descent

- The adaptive gradient descent algorithm uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training.
- The more the weights change, the least the learning rate changes.

Adagrad: Adaptive gradient descent

- The adaptive gradient descent algorithm uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training.
- The more the weights change, the least the learning rate changes.
- The benefit of using Adagrad is that it abolishes the need to modify the learning rate manually. It is more reliable than gradient descent algorithmss, as it reaches convergence at a higher speed.

Adagrad: Adaptive gradient descent

- The adaptive gradient descent algorithm uses different learning rates for each iteration. The change in learning rate depends upon the difference in the parameters during training.
- The more the weights change, the least the learning rate changes.
- The benefit of using Adagrad is that it abolishes the need to modify the learning rate manually. It is more reliable than gradient descent algorithmss, as it reaches convergence at a higher speed.
- One downside of AdaGrad optimizer is that it decreases the learning rate aggressively and monotonically. There might be a point when the learning rate becomes extremely small.

Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.

Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.
- Adam optimizer updates the learning rate for each network weight individually.

Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.
- Adam optimizer updates the learning rate for each network weight individually.
- The Adam optimizers inherit the features of both Adagrad and RMSProp algorithms.

Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.
- Adam optimizer updates the learning rate for each network weight individually.
- The Adam optimizers inherit the features of both Adagrad and RMSProp algorithms.
- Instead of adapting learning rates based upon the first moment (mean), it also uses the second moment of the gradients (variance).

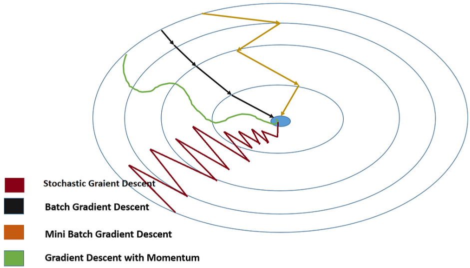
Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.
- Adam optimizer updates the learning rate for each network weight individually.
- The Adam optimizers inherit the features of both Adagrad and RMSProp algorithms.
- Instead of adapting learning rates based upon the first moment (mean), it also uses the second moment of the gradients (variance).
- It is often used as a default optimization algorithm. It has faster running time, low memory requirements, and requires less tuning than any other optimization algorithm.

Adam: ADAptive Moment estimation

- It is an extension of stochastic gradient descent.
- Adam optimizer updates the learning rate for each network weight individually.
- The Adam optimizers inherit the features of both Adagrad and RMSProp algorithms.
- Instead of adapting learning rates based upon the first moment (mean), it also uses the second moment of the gradients (variance).
- It is often used as a default optimization algorithm. It has faster running time, low memory requirements, and requires less tuning than any other optimization algorithm.
- It tends to focus on faster computation time, it might not generalize the data well enough.

Optimizers


Stochastic Graient Descent
Batch Gradient Descent
Mini Batch Gradient Descent
Gradient Descent with Momentum

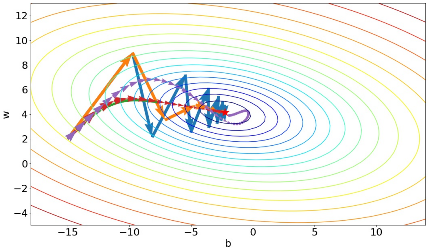
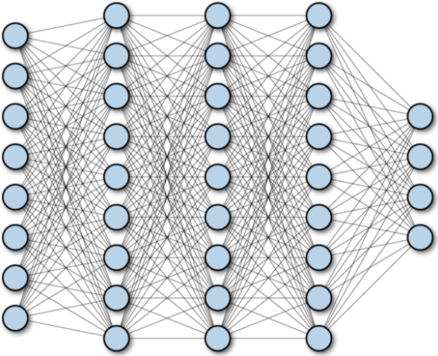

12
10
8
6
4
2
0
−2
−4
w
−15
−10
−5
0
5
10
b

Table of Contents

Fully Connected Networks



A fully connected neural network consists of a series of fully connected layers that connect every neuron in one layer to every neuron in the other layer.

Advantages and Disadvantages

- The major advantage of fully connected networks is that they are no special assumptions needed to be made about the input.

Advantages and Disadvantages

- The major advantage of fully connected networks is that they are no special assumptions needed to be made about the input.
- While being structure agnostic makes fully connected networks very broadly applicable, such networks do tend to have weaker performance than special-purpose networks tuned to the structure of a problem space.

The Algorithm

The algorithm consists of three simple steps:

- First, present one or more training examples to the neural network: Feed-Forward.

The Algorithm

The algorithm consists of three simple steps:

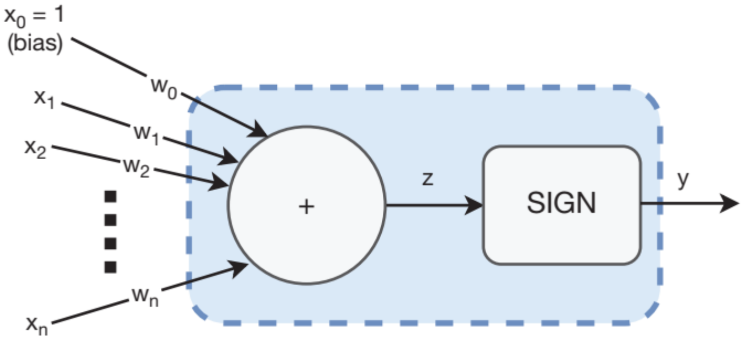
- First, present one or more training examples to the neural network: Feed-Forward.
- Second, compare the output of the neural network to the desired value: Loss function.

The Algorithm

The algorithm consists of three simple steps:

- First, present one or more training examples to the neural network: Feed-Forward.
- Second, compare the output of the neural network to the desired value: Loss function.
- Finally, adjust the weights to make the output get closer to the desired value using gradient descent: Back propagation.

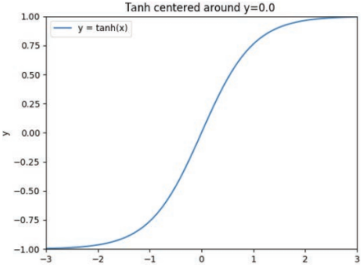
Feed-Forward



The sign function is an activation function of the neuron. It is not smooth because of the discontinuity in 0.

Other Activation Functions

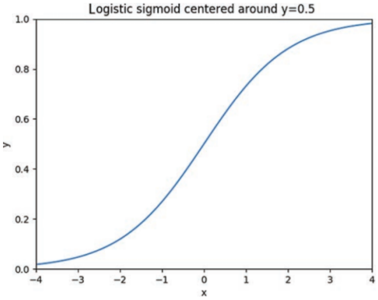
$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



$$\tanh'(x) = 1 - \tanh^2(x)$$

Other Activation Functions

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

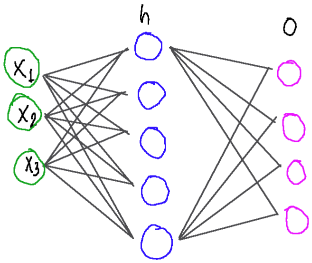


$$S'(x) = S(x)(1 - S(x))$$

Choice of Activation Functions

There exist a large number of activation functions. Two popular choices are tanh and the logistic sigmoid function. When picking between the two, choose tanh for hidden layers and logistic sigmoid for the output layer.

Feed-Forward: An Example



$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ & \cdots & & \\ 1 & x_1^{(N)} & x_2^{(N)} & x_3^{(N)} \end{pmatrix}$$

$$h = \Phi_1 (X \cdot W_1^T)$$

$$o = \Phi_2 (\bar{h} \cdot W_2^T)$$

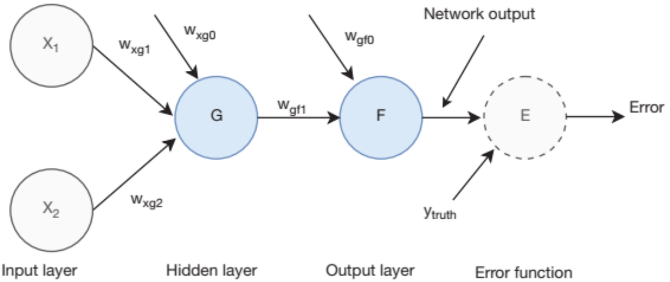
$W_1 \in M_{5 \times 4}$ $\quad$ $W_2 \in M_{4 \times 6}$

$X \in M_{N \times 4}$

$h \in M_{N \times 5}$

$\bar{h} \in M_{N \times 6}$

Example: Back-propagation



This neural network implements

$$\hat{y} = S\left(w_{gf0} + w_{gf1}\tanh(w_{xg0} + w_{xg1}x_1 + w_{xg2}x_2)\right)$$

We use the MSE:

$$e(f) = \frac{(y-f)^2}{2}.$$

Example: Back-propagation

- The error is given by

$$\text{Error} = \frac{1}{2}\left(y - S\left(w_{gf0} + w_{gf1}\tanh(w_{xg0} + w_{xg1}x_1 + w_{xg2}x_2\right)\right)$$

- We write it as:

$$\begin{aligned} e(f) &= \frac{1}{2}(y - f)^2 \\ f(z_f) &= S(z_f) \\ z_f(w_{gf0}, w_{gf1}, g) &= w_{gf0} + w_{gf1}g \\ g(z_g) &= \tanh(z_g) \\ z_g(w_{xg0}, w_{xg1}, w_{xg2}) &= w_{xg0} + w_{xg1}x_1 + w_{xg2}x_2 \end{aligned}$$

Example: Back-propagation

$$\frac{\partial e}{\partial w_{gf0}} = -(y-f)\cdot S'(z_f)$$

$$\frac{\partial e}{\partial w_{gf1}} = -(y-f)\cdot S'(z_f)\cdot g$$

$$\frac{\partial e}{\partial w_{xg0}} = -(y-f)\cdot S'(z_f)\cdot w_{gf1}\cdot tanh'(z_g)$$

$$\frac{\partial e}{\partial w_{xg1}} = -(y-f)\cdot S'(z_f)\cdot w_{gf1}\cdot tanh'(z_g)\cdot x_1$$

$$\frac{\partial e}{\partial w_{xg2}} = -(y-f)\cdot S'(z_f)\cdot w_{gf1}\cdot tanh'(z_g)\cdot x_2$$

Example: Back-propagation

Finally, we update the weights, via gradient descent

$$w_{gf0} \leftarrow w_{gf0} + \eta \frac{\partial e}{\partial w_{gf0}}$$
$$w_{gf1} \leftarrow w_{gf1} + \eta \frac{\partial e}{\partial w_{gf1}}$$
$$w_{xg0} \leftarrow w_{xg0} + \eta \frac{\partial e}{\partial w_{xg0}}$$
$$w_{xg1} \leftarrow w_{xg1} + \eta \frac{\partial e}{\partial w_{xg1}}$$
$$w_{xg2} \leftarrow w_{xg2} + \eta \frac{\partial e}{\partial w_{xg2}}$$