Neural Networks

目录

1 Bases

1.1 Architectures

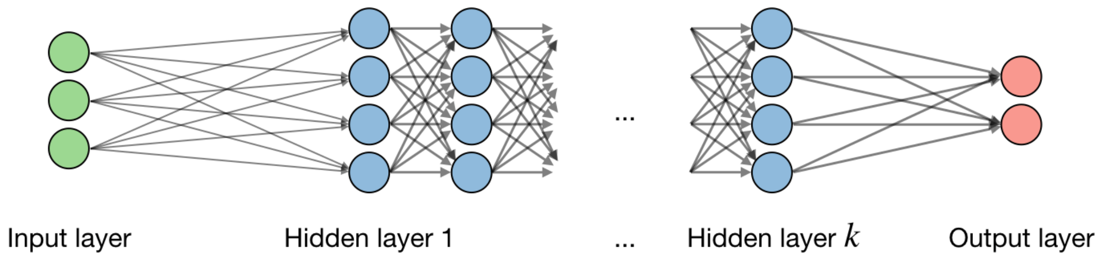


图 1: neural-networks-architectures

We will focus on one node, by noting:

- i : the i-th layer of the network
- j : the j-th hidden units of the layer
- $(\vec{x}, y)$: datasets, where $\vec{x}$ is the input and y is the desired output. $\vec{x} \in \mathcal{R}^{n_x}$ has n_x variables
- $\vec{w} \in \mathcal{R}^{n_x}$: weight, each w corresponds to one x
- $b \in \mathcal{R}$: bias
- z : output

We have:

Forward propagation (before using the activation function):

$$z_j^{[i]} = (\vec{w}_j^{[i]})^T \cdot \vec{x} + b_j^{[i]}$$

And then, **activation functions** $\hat{y} = g(z)$ are used at the end of a hidden unit to introduce non-linear complexities to the model.

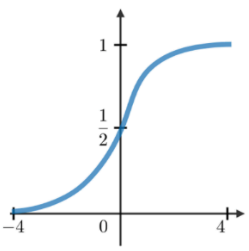
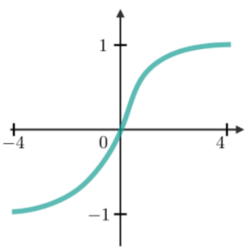
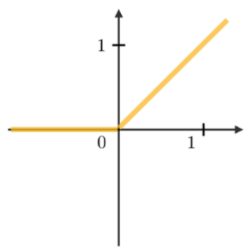
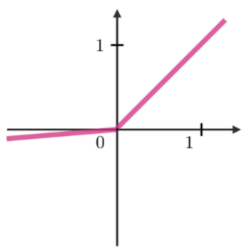
Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

图 2: activation-functions

Suming up:

> We have dataset $(\vec{x}, y)$. With input $x \in \mathcal{R}^{n_x}$, and the help of $\vec{w}, b$ and g,
>
> $$\vec{x} \mapsto z \mapsto \hat{y} = g(z)$$
>
> In this way having $\hat{y}$ in our model and y in reality.

1.2 Cross-entropy loss

Here, we are going to measure the difference between y and $\hat{y}$, by calculating the **Cross-entropy loss** $L(\hat{y}, y)$.

> **Cross-entropy loss**: to measure how well our algorithm is doing
>
> $$L(\hat{y}, y) = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})]$$

Also, we have the **cost function** based on a number of samples:

> **Cost function**: to measure how well you're doing in the entire training set. Here, n means the n-th sample, and m is the total number of the samples.
>
> $$J(w, b) = \frac{1}{m} \sum_{n=1}^{m} L(\hat{y}^{[n]}, y^{[n]})$$

1.3 Gradient Descent

Gradient Descent is based on a convex function and tweaks its parameters iteratively to minimize a given function (here, the cost function) to its local minimum.

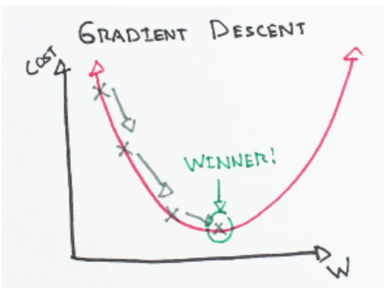


图 3: gradient-descent-1D

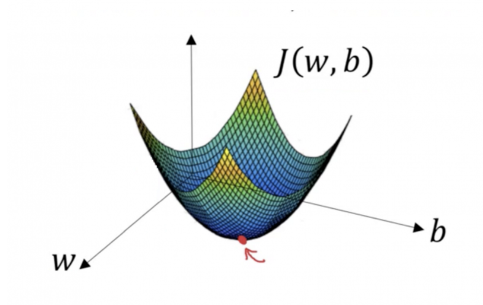


图 4: gradient-descent-3D

We note :

- a : current position
- b : the next position
- α : a waiting factor
- $\nabla f(a)$: the direction of the steepest descent at a

What **gradient descent** does:

$$b = a - \alpha \cdot \nabla f(a)$$

We define α as the **learning rate**, which indicates at which space the weights get updated.

It is very important for us to select a appropriate value of α.

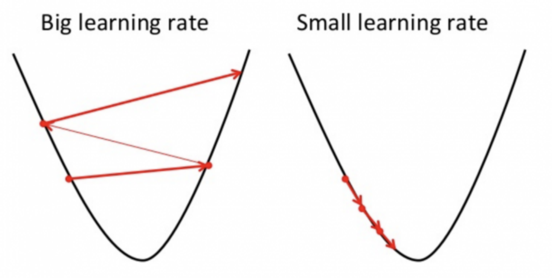


图 5: gradient-descent-learning-rate

After several iterations, $|\nabla f| < \varepsilon$, which means it has converged.

1.4 Backpropogation

Backpropogation is an algorithm for supervised learning of artificial neural networks using **gradient descent**.

The weights w are updated as follows :

1. Take a batch of training data
2. Perform **forward propagation** to obtain the corresponding loss
3. **Backpropagate** the loss to get the gradients
4. Use the gradients to update the weights of the network