Artificial Neural Networks

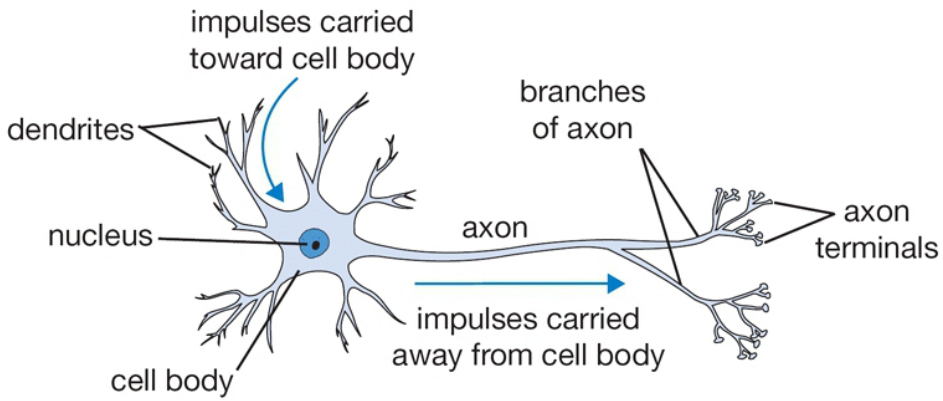
Introduction to Artificial Intelligence with Mathematics
Lecture Notes

Ganguk Hwang

Department of Mathematical Sciences
KAIST

Motivation from the brain

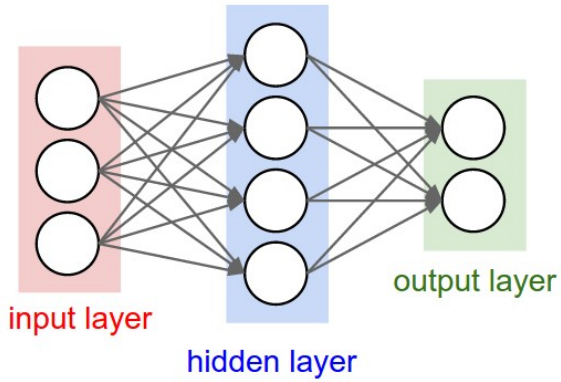
- The basic computational unit of the brain is a neuron.
- Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately $10^{14} \sim 10^{15}$ synapses.



source: https://cs231n.github.io/neural-networks-1/

Artificial Neural Networks

- Artificial neural networks are inspired by the human brain and mimic the process of transmitting biological neuron signals in the human brain.
- ANNs consist of an input layer, one or more hidden layers, and an output layer.
- ANNs define functions of the inputs, expressed by a collection of nodes/units (artificial neurons).
- Each node connects to another nodes and has associated weights and an activation function.

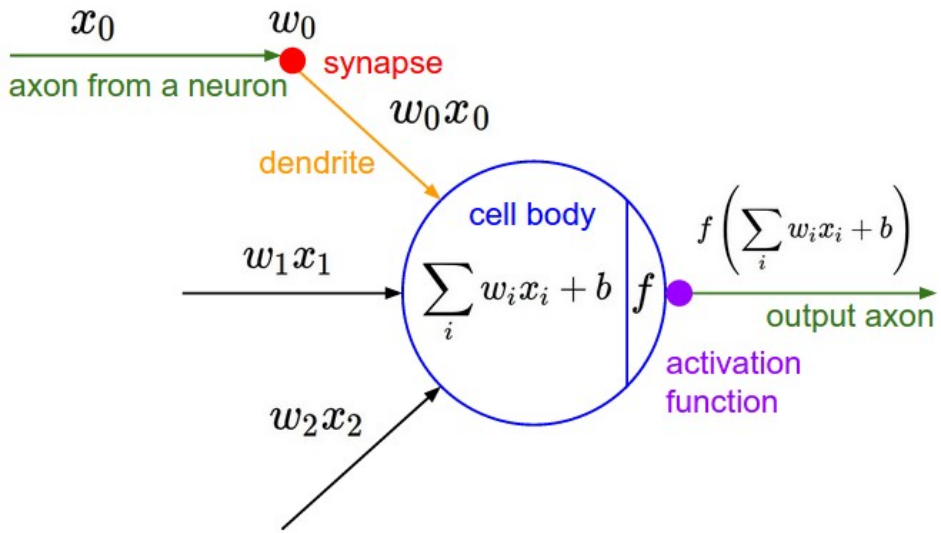


source: https://cs231n.github.io/neural-networks-1/

- ANNs rely on training data to learn and improve their accuracy over time.
- It takes time for ANNs to be trained, but after finishing the training they become powerful tools for given problems such as speech recognition or image recognition.
- Deep Learning and artificial neural networks are interchangeably used, but it is worth noting that the *deep* in deep learning is just referring to the depth of layers in an artificial neural network.
- An artificail neural network that consists of more than three layers is called a deep neural network and considered as a deep learning algorithm.
- An artificial neural network that only has two or three layers is just called a shallow neural network.
- Multi-Layer Perceptrons (MLPs) are also used interchangeably to refer to ANNs.

Mathematical Model of A Neuron in ANN

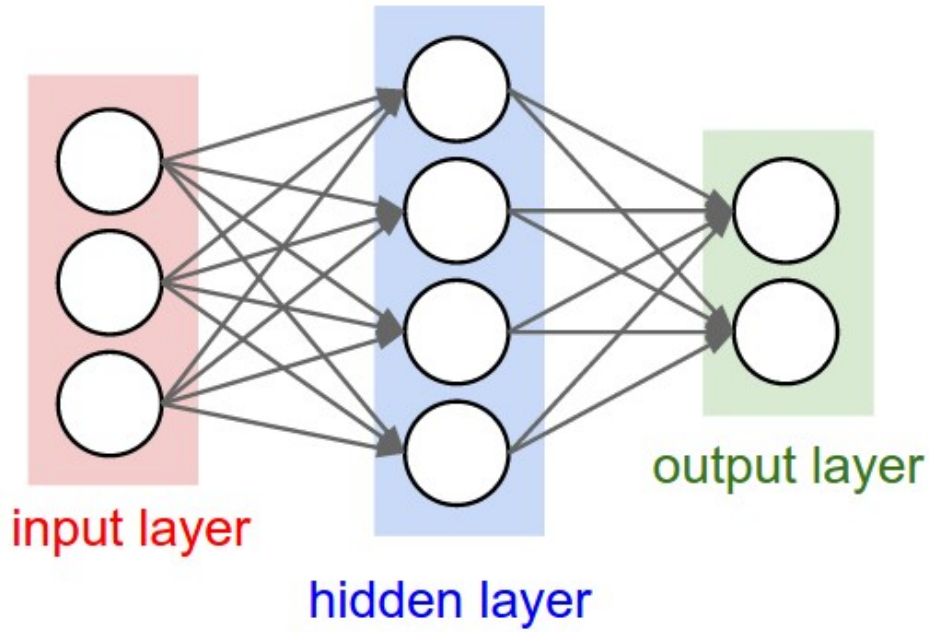
- Each node computes its output based on a linear combination of its inputs, bias, and an activation function.



source: https://cs231n.github.io/neural-networks-1/

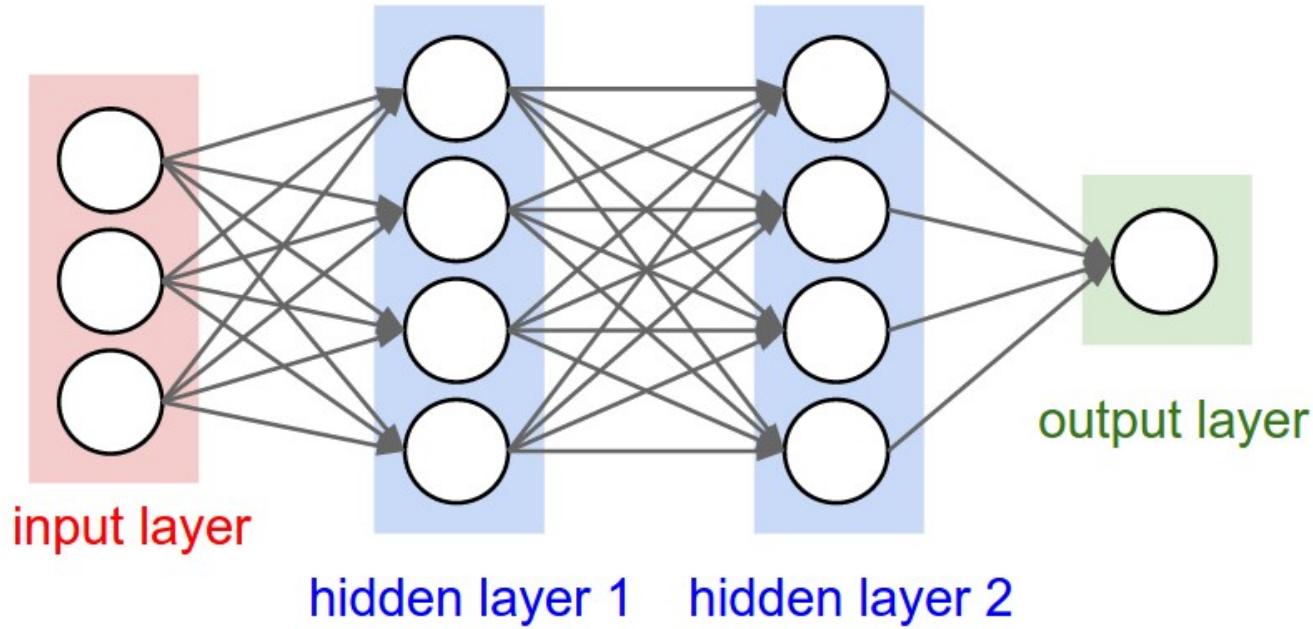
Neural Network Architecture

Example 1. A 2-layer Neural Network (one hidden layer of 4 nodes and one output layer with 2 nodes) and three inputs



source: https://cs231n.github.io/neural-networks-1/

Example 2. A 3-layer neural network with three inputs, two hidden layers of 4 nodes each and one output layer



source: https://cs231n.github.io/neural-networks-1/

Naming conventions

- When we say an N-layer neural network, we do not count the input layer.
- For instance, a 2-layer neural network has one layer of hidden nodes and one output layer.

Universal Approximation of ANNs

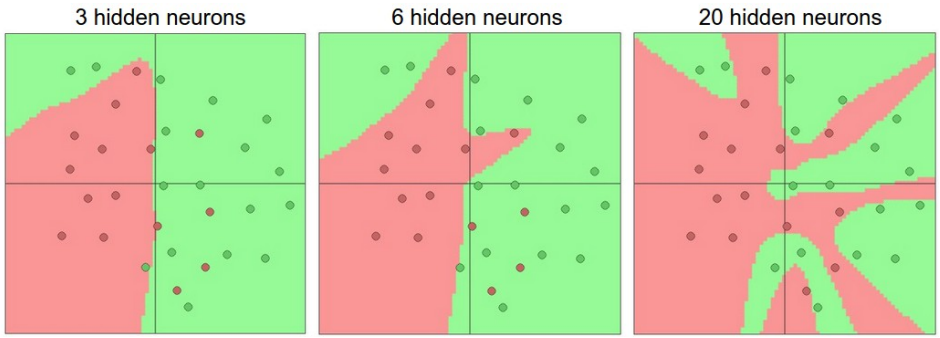
- It turns out that Neural Networks with at least one hidden layer are universal approximators, i.e.,

  for any given continuous function $f(x)$ and some $\epsilon > 0$, there exists a Neural Network $g(x)$ with one hidden layer (with a reasonable choice of non-linearity, e.g. sigmoid) such that $|f(x) - g(x)| < \epsilon$ for any x. In other words, the neural network can approximate any continuous function.

- G. Cybenko, Approximation by Superpositions of Sigmoidal Function, Mathematics of Control, Signals, and Systems, Vol. 2, pp. 303-314, 1989.

The impact of the number of layers and nodes

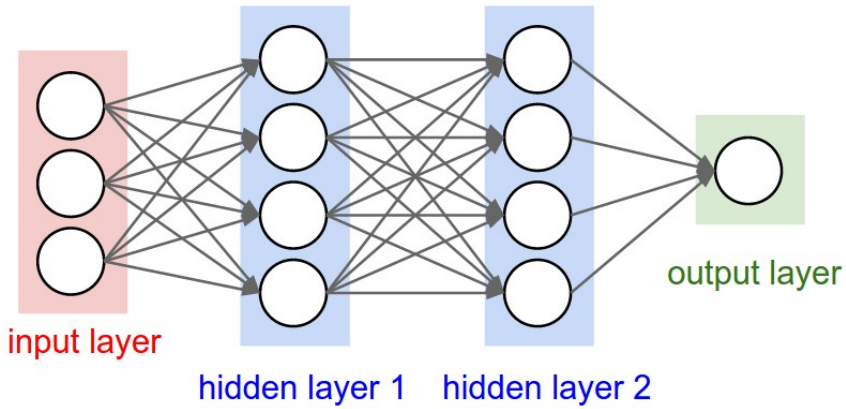
- The capacity of the network increases as we increase the numbers of hidden layers and nodes.
- Overfitting occurs when a neural network with high capacity fits the noise in the data instead of the (assumed) underlying relationship.



Neural Networks with one hidden layer, but different numbers of nodes
source: https://cs231n.github.io/neural-networks-1/

Feed-forward Network

- w^l_{jk}: the weight to connect from the k-th node in the $l-1$-th layer to the j-th node in the l-th layer
- b^l_j: the bias of the j-th node in the l-th layer
- y^l_j: the output of the j-th node in the l-th layer
- x^l_j: the input of the j-th node in the l-th layer



source: https://cs231n.github.io/neural-networks-1/

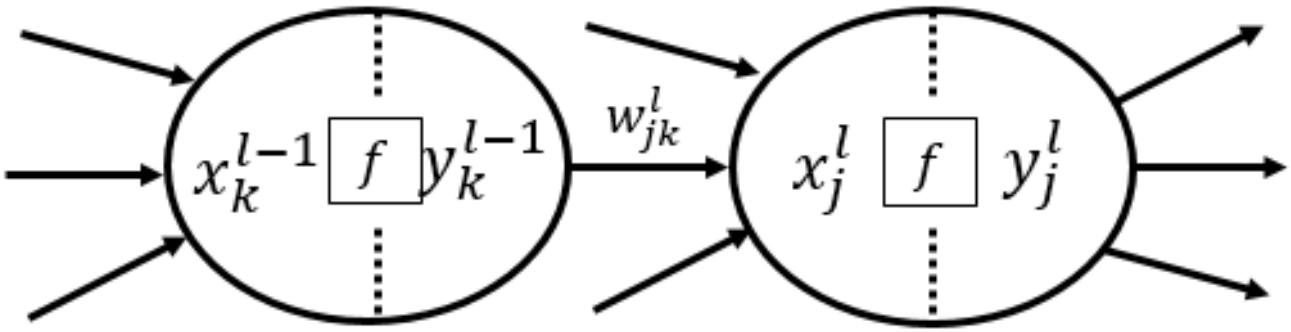
For layer l with an activation function f,

$$\mathbf{x}^l = (x_j^l), x_j^l = \sum_k w_{jk}^l y_k^{l-1} + b_j^l,$$

$$\mathbf{y}^l = (y_j^l), y_j^l = f\left(\sum_k w_{jk}^l y_k^{l-1} + b_j^l\right),$$

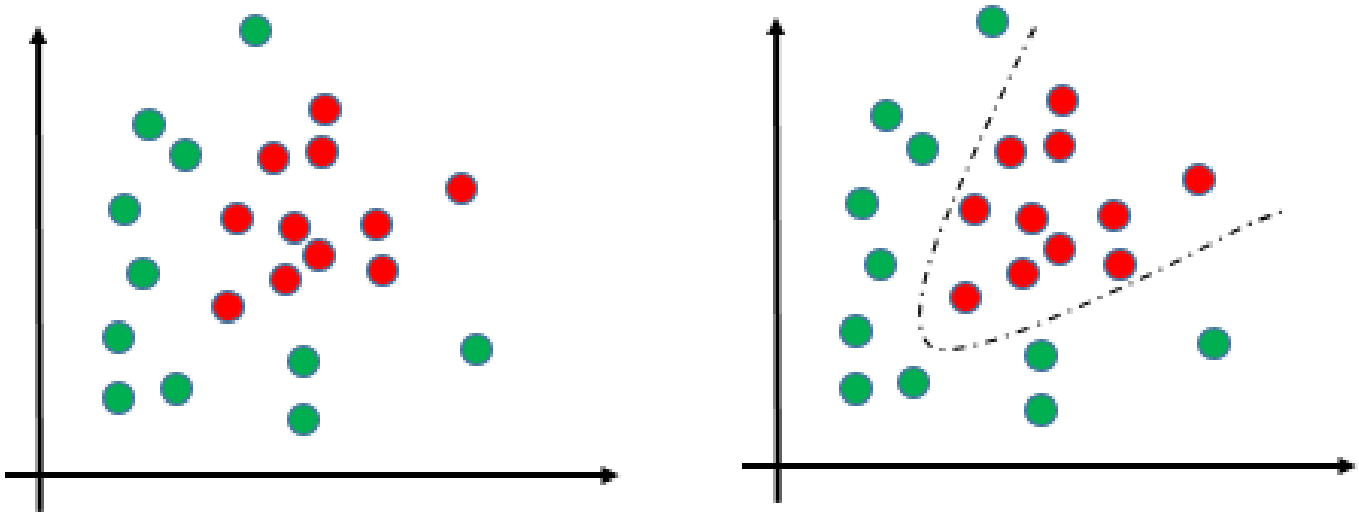
$$\mathbf{y}^l = f(\mathbf{W}^l \mathbf{y}^{l-1} + \mathbf{b}^l)$$

$\mathbf{W}^l = (w_{jk}^l)$: weight matrix for layer l



Activation functions

The purpose of activation functions is to introduce non-linearities into the network.



Linear activation functions always produce linear decisions

Useful activation functions include

- sigmoid/logistic function: $\sigma(x) = \frac{1}{1+\exp(-x)}$
- tanh function: $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
- ReLU (Rectified Linear Unit) function: $\text{ReLu}(x) = \max(0, x)$

Training Neural Networks

- Training neural networks implies adjusting all weights and biases in neural networks.
- For training a neural network, we first define a loss function $J = J(\mathbf{y}, \mathbf{y}^L)$ where $\mathbf{y}$ denotes the target and $\mathbf{y}^L$ denotes the output from the neural network and consider

$$\mathbf{W}^*, \mathbf{b}^* = \arg\min_{\mathbf{W},\mathbf{b}} \sum_{n=1}^{N} J(\mathbf{y}_n, \mathbf{y}_n^L).$$

- We use (stochastic) gradient descent with learning rate η.

$$\mathbf{W}^{t+1} = \mathbf{W}^t - \eta \frac{\partial J}{\partial \mathbf{W}}(\mathbf{W}^t, \mathbf{b}^t), \quad \mathbf{b}^{t+1} = \mathbf{b}^t - \eta \frac{\partial J}{\partial \mathbf{b}}(\mathbf{W}^t, \mathbf{b}^t).$$

Derivatives of Activation Functions

Sigmoid function: $\sigma(x) = \frac{1}{1+\exp(-x)}$

$$\frac{d}{dx}\sigma(x) = \sigma(x) \cdot (1 - \sigma(x))$$

Tanh function: $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$

$$\frac{d}{dx}\tanh(x) = \frac{1}{\cosh^2(x)}$$

ReLU: $\mathsf{ReLu}(x) = \max(0, x)$

$$\frac{d}{dx}\mathsf{ReLu}(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x \leq 0 \end{cases}$$

Backpropagation

Backpropagation is an efficient way of computing the gradients of the error function w.r.t. weights.

- First, feed the input $\mathbf{x}$ forward through the network and store all $\mathbf{y}^l$ for all layers l.
- Compute the output error and propagate the error backward from output nodes.
- Backpropagation is based on chain rule.

First, the error function is given by

$$J = \frac{1}{2}\|\mathbf{y} - \mathbf{y}^L\|^2 = \frac{1}{2}\sum_j (y_j - y_j^L)^2,$$

which we would like to minimize with a given training set.

We first define the error δ_j^l of node j in layer l by

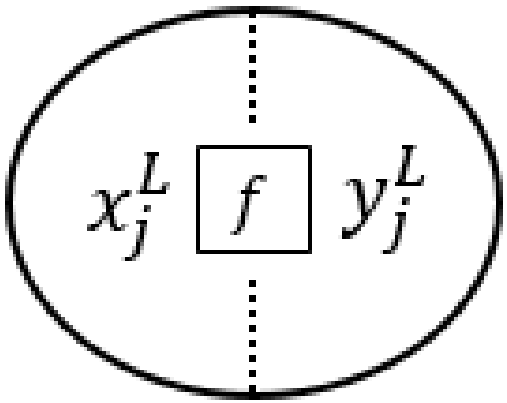
$$\delta_j^l = \frac{\partial J}{\partial x_j^l}, \boldsymbol{\delta}^l = (\delta_j^l).$$

We will derive a recursive expression for $\boldsymbol{\delta}^l$.

Equations for Output Layer L:

Noting that J is a function of y_j^L, we obtain

$$\begin{aligned}\delta_j^L &= \frac{\partial J}{\partial x_j^L} = \sum_k \frac{\partial J}{\partial y_k^L}\frac{\partial y_k^L}{\partial x_j^L} \\ &= \frac{\partial J}{\partial y_j^L}\frac{\partial y_j^L}{\partial x_j^L} = \frac{\partial J}{\partial y_j^L} f'(x_j^L)\end{aligned}$$



From $J = \frac{1}{2}\sum_j (y_j - y_j^L)^2$, for layer L we obtain

$$\frac{\partial J}{\partial y_j^L} = y_j^L - y_j.$$

From $\delta_j^L = \frac{\partial J}{\partial y_j^L} f'(x_j^L)$, we obtain

$$\begin{aligned} \boldsymbol{\delta}^L &= \nabla_{\mathbf{y}^L} J \odot f'(\mathbf{x}^L) \\ &= (\mathbf{y}^L - \mathbf{y}) \odot f'(\mathbf{x}^L) \end{aligned}$$

Here, $\odot$ denotes the element-wise product (Hadamad product).

We now derive δ_j^l in terms of δ^{l+1}.

$$\begin{aligned}\delta_j^l &= \frac{\partial J}{\partial x_j^l} = \sum_k \frac{\partial J}{\partial x_k^{l+1}} \frac{\partial x_k^{l+1}}{\partial x_j^l} \\ &= \sum_k \delta_k^{l+1} \frac{\partial x_k^{l+1}}{\partial x_j^l}.\end{aligned}$$

Recall that

$$x_k^{l+1} = \sum_j w_{kj}^{l+1} y_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f(x_j^l) + b_k^{l+1}.$$

Differentiating x_k^{l+1} w.r.t. x_k^l yields

$$\frac{\partial x_k^{l+1}}{\partial x_j^l} = w_{kj}^{l+1} f'(x_j^l).$$

Combinig our results, we get

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'(x_j^l).$$

Similarly, we know

$$\frac{\partial J}{\partial b_j^l} = \sum_k \frac{\partial J}{\partial x_k^l} \frac{\partial x_k^l}{\partial b_j^l} = \sum_k \delta_k^l \frac{\partial x_k^l}{\partial b_j^l}.$$

Again from $x_k^l = \sum_j w_{kj}^l y_j^{l-1} + b_k^l = \sum_j w_{kj}^l f(x_j^{l-1}) + b_k^l$, we get

$$\frac{\partial x_j^l}{\partial b_j^l} = 1, \quad \frac{\partial x_k^l}{\partial b_j^l} = 0 \text{ for } k \neq j.$$

Consequently, we get

$$\frac{\partial J}{\partial b_j^l} = \delta_j^l.$$

We now derive the rate of change of the cost function w.r.t. weight w_{kj}^l. Note that

$$\frac{\partial J}{\partial w_{kj}^l} = \frac{\partial J}{\partial x_k^l}\frac{\partial x_k^l}{\partial w_{kj}^l} = \delta_k^l \frac{\partial x_k^l}{\partial w_{kj}^l}.$$

Again from $x_k^l = \sum_j w_{kj}^l y_j^{l-1} + b_k^l$, we know that

$$\frac{\partial x_k^l}{\partial w_{kj}^l} = y_j^{l-1},$$

and consequently

$$\frac{\partial J}{\partial w_{kj}^l} = \delta_k^l y_j^{l-1}.$$

We summarize all the results.

Equation for the error δ_j^l in terms of δ^{l+1}:

$$\boldsymbol{\delta}^l = ((\mathbf{W}^{l+1})^\top \boldsymbol{\delta}^{l+1}) \odot f'(\mathbf{x}^l)$$

Equation for the rate of change of the cost function w.r.t. bias:

$$\frac{\partial J}{\partial b_j^l} = \delta_j^l, \text{ equiv., } \frac{\partial J}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l$$

Equation for the rate of change of the cost function w.r.t. any weight:

$$\frac{\partial J}{\partial w_{kj}^l} = \delta_k^l y_j^{l-1}, \text{ equiv., } \frac{\partial J}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l (\mathbf{y}^{l-1})^\top.$$

Backpropagation Algorithm for a single input $\mathbf{x}$

- Input: For an input $\mathbf{x}$, let $\mathbf{y}^0 = \mathbf{x}$.
- Feedforward: For each $l = 1, 2, 3, \cdots, L$, compute

$$\mathbf{x}^l = \mathbf{W}^l \mathbf{y}^{l-1} + \mathbf{b}^l, \mathbf{y}^l = f(\mathbf{x}^l).$$

- Output error $\boldsymbol{\delta}^L$: Compute

$$\boldsymbol{\delta}^L = \nabla_{\mathbf{y}^L} J \odot f'(\mathbf{x}^L)$$

- Backpropagation of the error: For each $l = L-1, L-2, \cdots, 1$ compute

$$\boldsymbol{\delta}^l = ((\mathbf{W}^{l+1})^T \boldsymbol{\delta}^{l+1}) \odot f'(\mathbf{x}^l).$$

- Output: The gradients of the cost function are given by

$$\frac{\partial J}{\partial \mathbf{b}^l} = \boldsymbol{\delta}^l, \frac{\partial J}{\partial \mathbf{W}^l} = \boldsymbol{\delta}^l (\mathbf{y}^{l-1})^\top.$$

Backpropagation Algorithm for a mini-batch

- Input a mini-batch of size m from the training set
- For each training input $\mathbf{z}$, compute the following:
  - $\mathbf{y}^{\mathbf{z},0}$
  - $\mathbf{x}^{\mathbf{z},l} = \mathbf{W}^l \mathbf{y}^{\mathbf{z},l-1} + \mathbf{b}^l$ and $\mathbf{y}^{\mathbf{z},l} = f(\mathbf{x}^{\mathbf{z},l})$ for $l = 1, 2, 3, \cdots, L$
  - $\boldsymbol{\delta}^{\mathbf{z},L} = \nabla_{\mathbf{y}^L} J^{\mathbf{z}} \odot f'(\mathbf{x}^{\mathbf{z},L})$
  - $\boldsymbol{\delta}^{\mathbf{z},l} = ((\mathbf{W}^{l+1})^\top \boldsymbol{\delta}^{\mathbf{z},l+1}) \odot f'(\mathbf{x}^{\mathbf{z},l})$ for $l = L-1, L-2, \cdots, 1$
- For $l = L, L-1, \cdots, 1$, update the weights and biases as follows:

$$\mathbf{W}^l \to \mathbf{W}^l - \frac{\eta}{m} \sum_{\mathbf{z}} \boldsymbol{\delta}^{\mathbf{z},l} (\mathbf{y}^{\mathbf{z},l-1})^\top,$$

$$\mathbf{b}^l \to \mathbf{b}^l - \frac{\eta}{m} \sum_{\mathbf{z}} \boldsymbol{\delta}^{\mathbf{z},l}.$$

Example:

We're going to see the impact of the artificial neural network structure on performance in a regression problem. The training data is generated from

$$y = 10\sin(x) + 0.1 \times \mathcal{N}(0,1),\ x \in (0,10).$$

The number of training data points is 200.

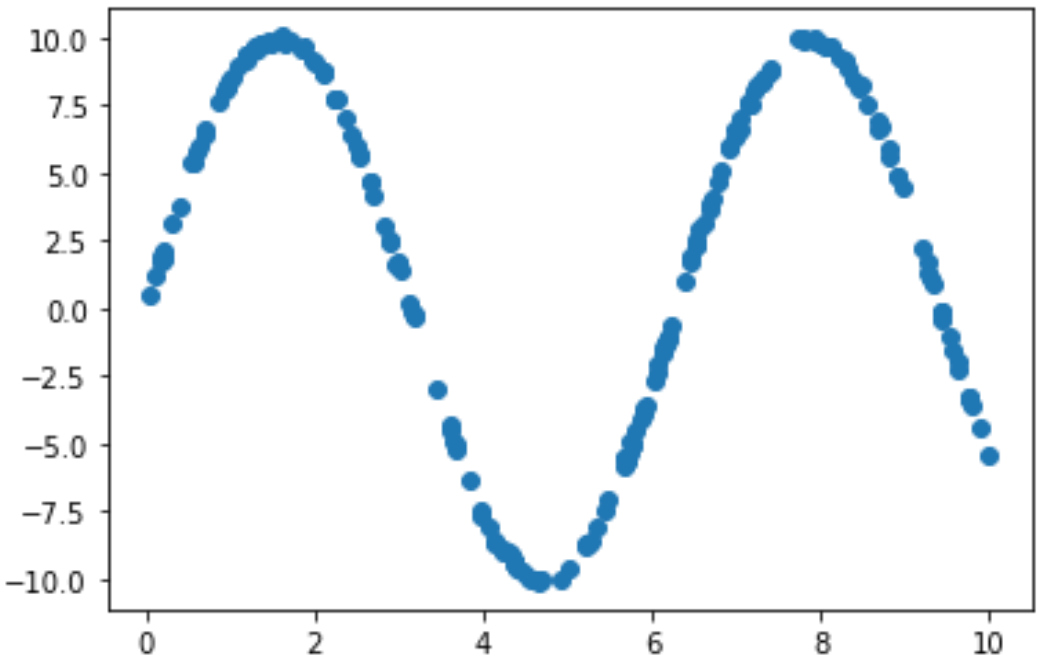


Figure: Training data

Impact of the number of hidden nodes

- The artificial neural network with one hidden layer is used for regression.
- The activation function for the hidden layer is relu.
- The total number of iterations is 10000 (this is for convergence) and the batch size is 10.
- The loss is the mean squared error.
- Let H denote the number of hidden nodes.

Impact of the number of hidden nodes

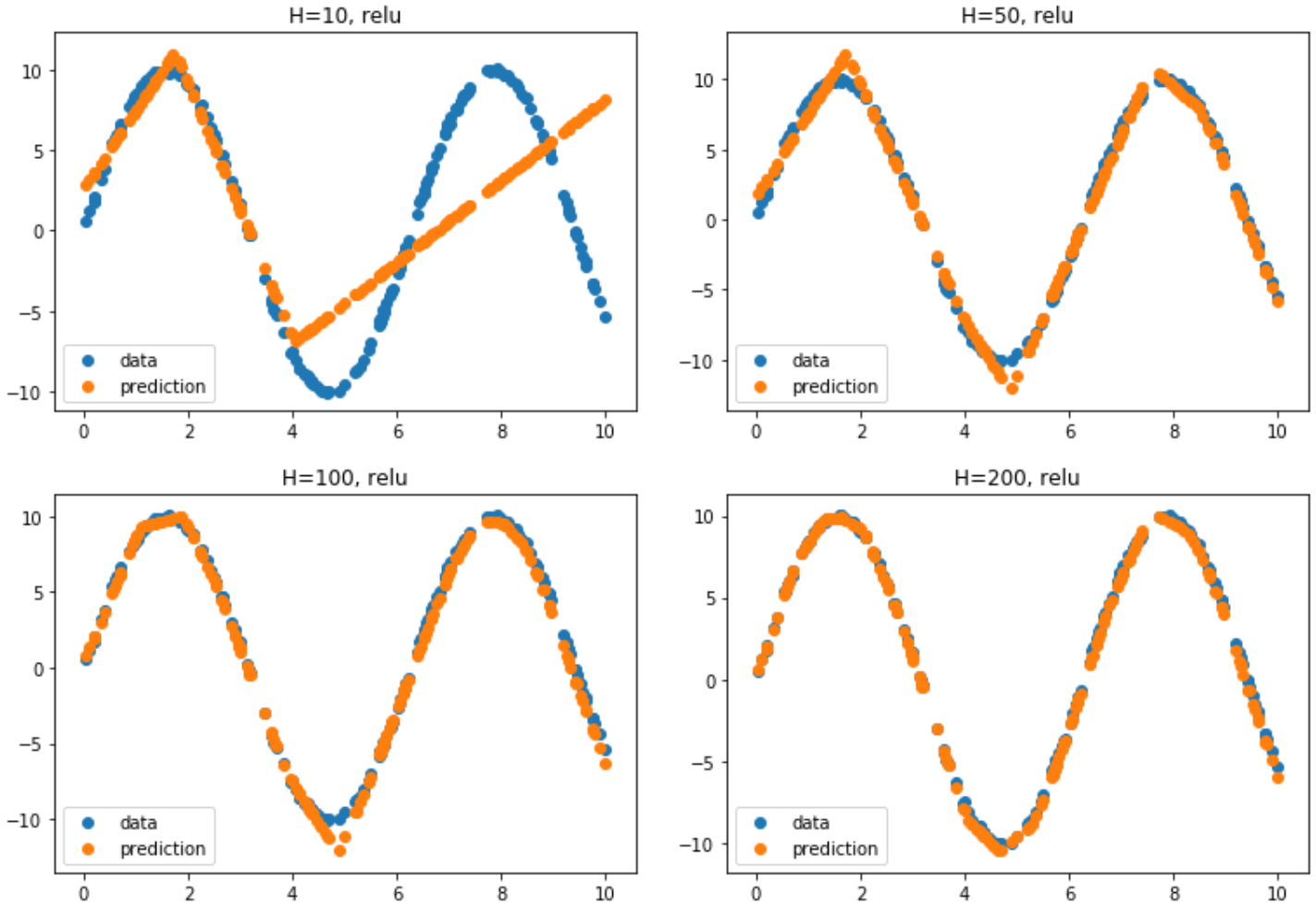


Figure: loss $= 17.3987, 0.3597.0.1369, 0.0700$

Impact of the number of hidden layers

- We next see the impact of the number of hidden layers.
- The total number of hidden nodes is fixed to be 24.
- The number of hidden layers is changed from 1 to 4.
- $H = (12, 12)$ means that the artificial neural network has two hidden layers and each layer has 12 hidden nodes.

Impact of number of hidden layers

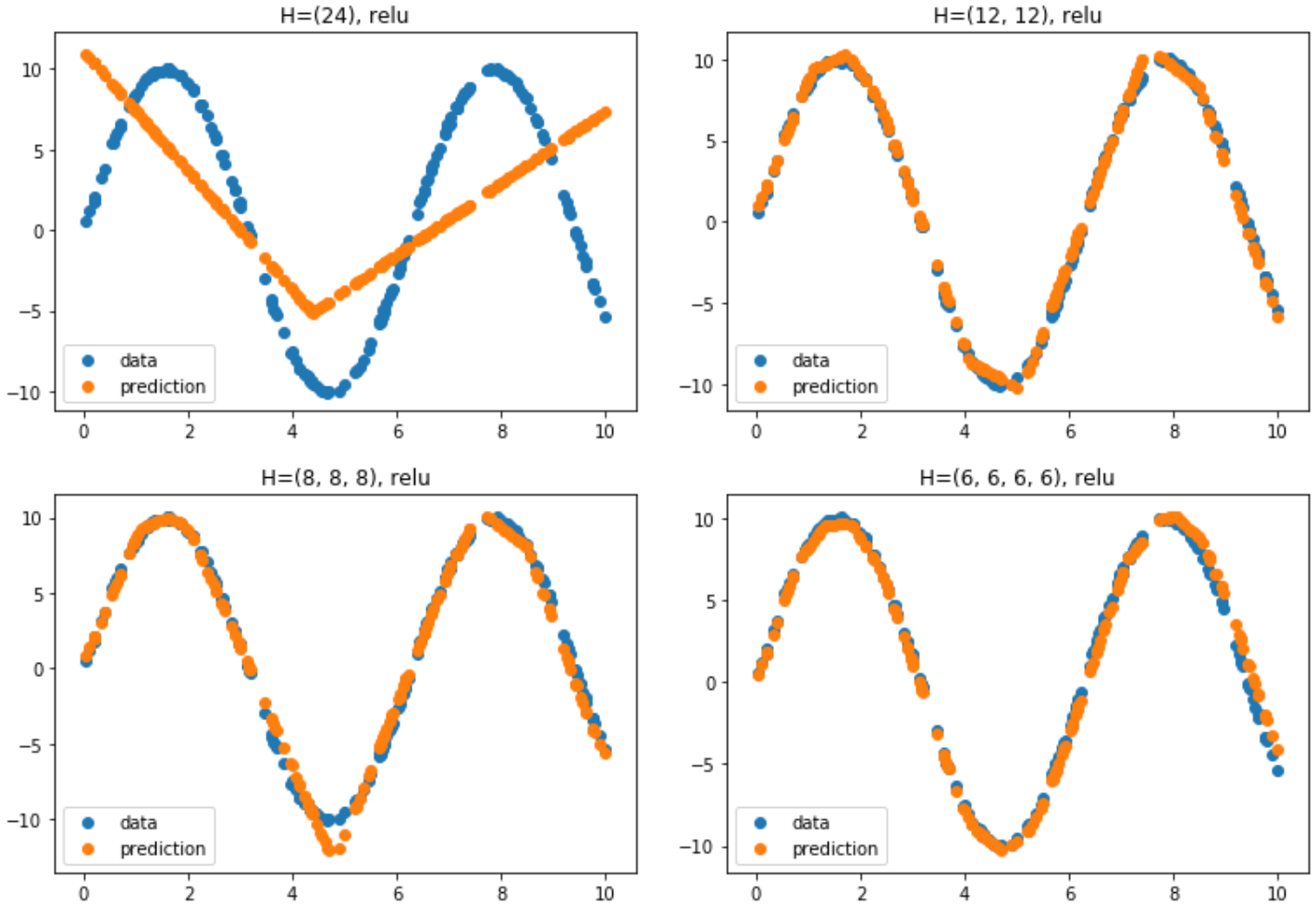


Figure: loss=24.6575, 0.1138, 0.3263, 0.0537

Impact of activation functions

- We can also investigate the impact of the activation function by changing it for the same number of hidden nodes and hidden layers.
- The upper 3 figures are for one hidden layer with 10 nodes, and the lower 3 figures are for one hidden layer with 100 nodes.

Impact of activation functions

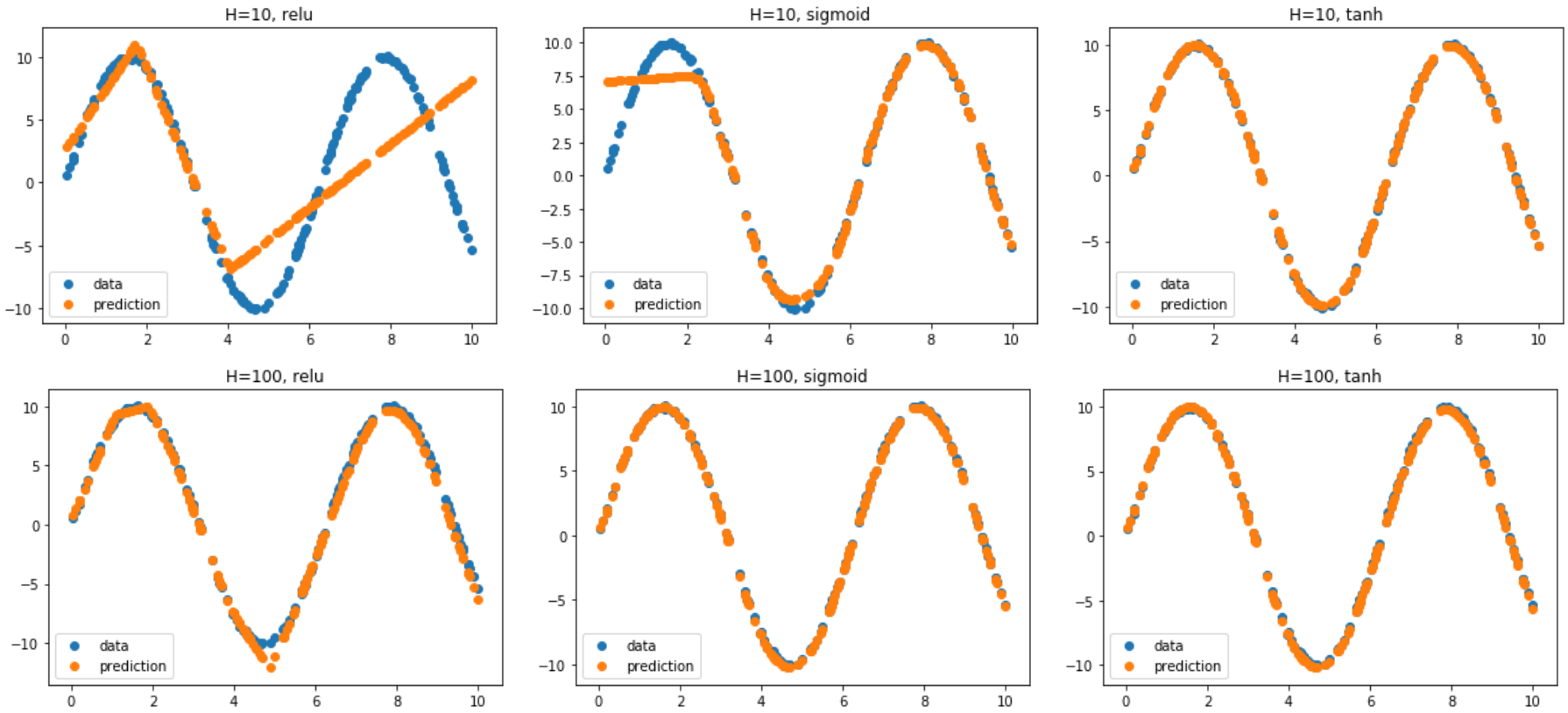


Figure: loss = 17.3987, 1.8532, 0.0131 (1st row) and loss = 0.1369, 0.0148, 0.0232 (2nd row)

Design Issues of ANNs

- Network Topology
  - It refers to the numer of layers and the number of nodes in each layer.
  - A large number of nodes may result in overfitting, but a less number of nodes may cause underfitting.
  - The increase in number of hidden layers results in decreasing of robustness of ANNs.
  - To find an optimum network topology, two approaches are suggested: network growing and network pruning. However, there are no generalized rules!
- Weight Initialization
  - Weight initialization can have great impact on the performance and convergence rate of a network.
  - Improperly initialized weights in a network may lead the output of the network towards local minima and degradation in performance of the network.
  - data dependent initialization, principal component analysis, and statistics based initialization

- Activation Functions
  - Different activation functions provide difference performance results.
  - Recently, the ReLU (Rectified Linear Units) activation function is gaining popularity specially due to the emergence of deep learning. It improves the training of the deep neural network by solving the vanishing gradient problem.
- Learning Rate
  - The learning rate represents the rate of the steps to be taken forward to achieve the minimimum of a given error.
  - It significantly affects the accuracy and convergence speed of a network.
  - If the learning rate is small, then it will take more time for the optimization. If it is high, then it will take less time for the optimization, but the training may diverge and skip the optimal point which may lead to high loss.
- Regularization
  - Regularization refers to the methods used as a remedy to overfitting of the networks.