

# Tutorial 4

## Supervised Learning using Neural Network

Luke Chang

The University of Auckland

Mar. 2021

# Objectives

- 1 Overview on Neural Network
- 2 Activation Functions
- 3 Loss Functions
- 4 Optimization
- 5 Different Types of Layers
- 6 Tutorial Questions

# Overview on Neural Network

Where does neural network (NN) shine?

- Commonly used in supervised learning where data has a lot of instances and feature space is large.
- It scales well when data size increases.

An (artificial) neural network is a direct graph.

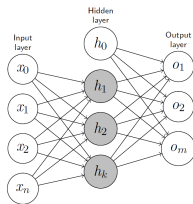
- **Feed-forward neural network (FFNN):** A directed acyclic graph, where nodes are arranged in layers from inputs to outputs.
- **Recurrent neural network (RNN):** A directed graph with cycles; nodes have additionally feedback into themselves or previous nodes.

## Motivation

To combat **gradient vanishing** problem, *feedback nodes* in the hidden layers are commonly used in large-scale deep neural networks.

# A basic NN with 1 hidden layer

- The data has  $n$  features, and the outputs have  $m$  classes.
- $k$  hidden neurons in the hidden layer.
- Let  $x$  be the input vector where  $x \in^n$ , hidden layer  $h$  is a vector where  $h \in^k$  and output  $o$  is a vector where  $o \in^m$ .
- Nodes are connected by weights. These weights are learnt during the training. The weight matrices for the hidden layer is  $W_0 \in^{k \times n}$ , and for the output layer is  $W_1 \in^{m \times k}$ .
- Biases are  $b_0 \in^k$  and  $b_1 \in^m$ , where  $x_0 = h_0 = 1$ .



$o = f(W_1 \cdot f(W_0 \cdot x + b_0) + b_1)$ , where  $f$  is the activation function, applied element-wise.

**Note:** The last activation function should be based on the desired outputs. (Eg: One-hot encoding, regression)

# Activation Functions

The activation function must be **non-linear**.

For a given non-input node  $h$ :

- Suppose there are  $n$  nodes connect to  $h$ .
- Let  $x$  be the column vector input to the node, that  $x = (x_0, x_1, \dots, x_n)^T$ , where  $x_0 = 1$ .
- Let  $w$  be the weights on the incident edges, that  $w = (w_0, w_1, \dots, w_n)^T$ , where  $w_0 = b$  is the bias.

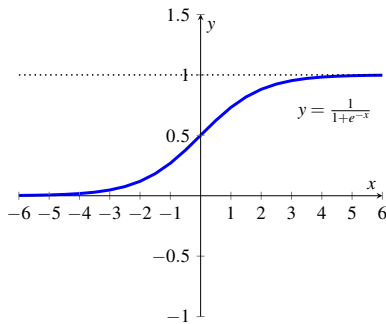
The output from this node is given by:

$$f(w^T x) = f\left(\sum_{i=0}^n w_i x_i\right) = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

Where  $f$  is the activation function for this node.

If the activation function is linear, we can merge multiple hidden layers into one layer. The structure is equivalent to linear regression.

# Sigmoid Function



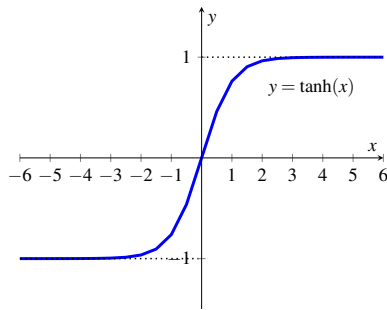
$$f(x) = \text{Sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Where  $f(x) \in (0, 1)$ , and the derivative is  $f'(x) = f(x)(1 - f(x))$ .

Rarely used in hidden layers for state-of-the-art models.

Often used in the last hidden layer to produce logistic outputs.

# Hyperbolic Tangent (tanh) Function

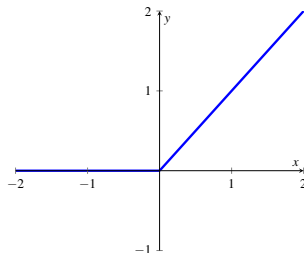


$$f(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Where  $f(x) \in (-1, 1)$ , and the derivative is  $f'(x) = 1 - f(x)^2$

Similar to Sigmoid function, but faster to converge.

# Rectified Linear Unit (ReLU) Function



$$f(x) = \text{ReLU}(x) = (x)^+ = \max(0, x)$$

Where  $f(x) \in [0, +\infty)$ . The derivative is equal to:

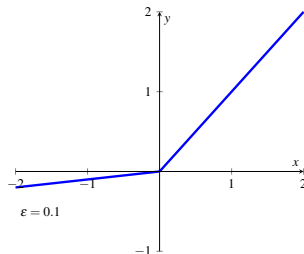
$$f'(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \end{cases}$$

Where  $x = 0$  is not differentiable.

ReLU is the most common activation function for hidden layers in recent deep learning.



# Leaky ReLU



Let  $\epsilon$  be the negative slope:

$$f(x) = \text{LeakyReLU}(x) = \max(0, x) + \epsilon \times \min(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ \epsilon \times x, & \text{otherwise} \end{cases}$$

Where  $f(x) \in \mathbb{R}$ , and  $\epsilon \in [0, 1)$  The derivative is equal to ( $x = 0$  is not differentiable):

$$f'(x) = \begin{cases} \epsilon, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \end{cases}$$

Similar to ReLU, but overcomes the “dead neuron” problem.

# Loss/Cost Functions - MSE

The goal of training is to minimize a loss function. An objective function is a loss function.

## Mean Squared Error (MSE)

$$\ell(X, y|W) = \text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

- Squared L2-norm error;
- Loss is computed forward,
- $W$  are the weights.
- $X$  are the data.
- $y$  are the target labels.
- Minimizing  $\ell$  using backpropagation with stochastic gradient descent (SGD) algorithm.

# Mean Absolute Error (MAE)

$$\ell(X, y|W) = \text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$

- Minimizing L1-norm error
- Compare with MSE, L1-norm is more robust to outliers.
- Compare with MSE, the solution from L1-norm is unstable, so it is harder to optimize.

When use as regularization:

- L1-regularization prefers sparse outputs.
- L2-regularization prefers smaller weights and distributes the weights more evenly.

# Softmax Function

Softmax function is defined as:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- The outputs from a NN can be negative or above 1.
- Softmax function rescales them so that the elements of a n-dimensional output lie in the range  $[0, 1]$  and sum to 1.

# Negative Log Likelihood (NLL) Loss

NLL Loss is useful to train a classification problem with multiple output classes.

Let  $p$  be a vector of probabilities (after Softmax function).

$$p_k = \frac{\exp(f_k)}{\sum_j \exp(f_j)}$$

The Loss for one example is:

$$L_i = -\log(p_{y_i})$$

The partial derivative is extremely simple:

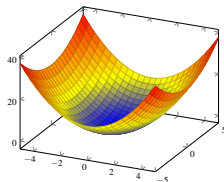
$$\frac{\partial L_i}{\partial f_k} = p_k - 1(y_i = k)$$

Example:

	Class	Prediction	df
Alpaca	0	0.2	<b>0.2</b>
Cat	1	0.3	<b>-0.7</b>
Dog	0	0.5	<b>0.5</b>

# Optimization - Stochastic Gradient Descent (SGD)

- Gradient Descent: Minimizing the loss function by computing the gradient of the error and adjust the weights to descend the landscape into an error minima.
- Iterative algorithm
- Starts from a random point and travels down its slope with **learning rate**  $\lambda$  until it reaches the lowest point.
- Stochastic Gradient Descent (SGD) - Stochastic means **random**; Randomly select one instance at each iteration to reduce the computation cost;
- SGD works well with **mini-batch**, where the entire training data is divided into multiple random batches (without replacement, usually between 32 and 256).

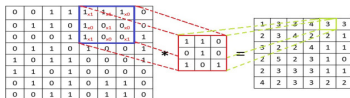


$$\text{step\_size} = \text{gradient} \times \lambda$$

$$W^{[i+1]} = W^{[i]} - \text{step\_size}$$

# Different Types of Layers - Convolutional Layers

- Convolutional Neural Network (CNN) is commonly applied to image datasets.
- CNN is a type of DNN.
- Convolutional layers is the fundamental building blocks for CNN.
- Convolutional layers convolve the input and pass its result to the next layer. (Convolution operation:  $f * g$ )
- Similar to apply an image filter, but all weights in the kernel are learnt during the training.

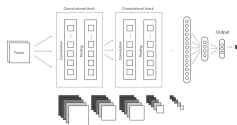


The height of the output is:

$$H_{\text{out}} =$$

The same formula is also applied to the output width.

# Pooling Layers





# Recurrent Layers

# Tutorial Questions