

# CS 584 – MACHINE LEARNING

## TOPIC: NEURAL NETWORKS



**Mustafa Bilgic**

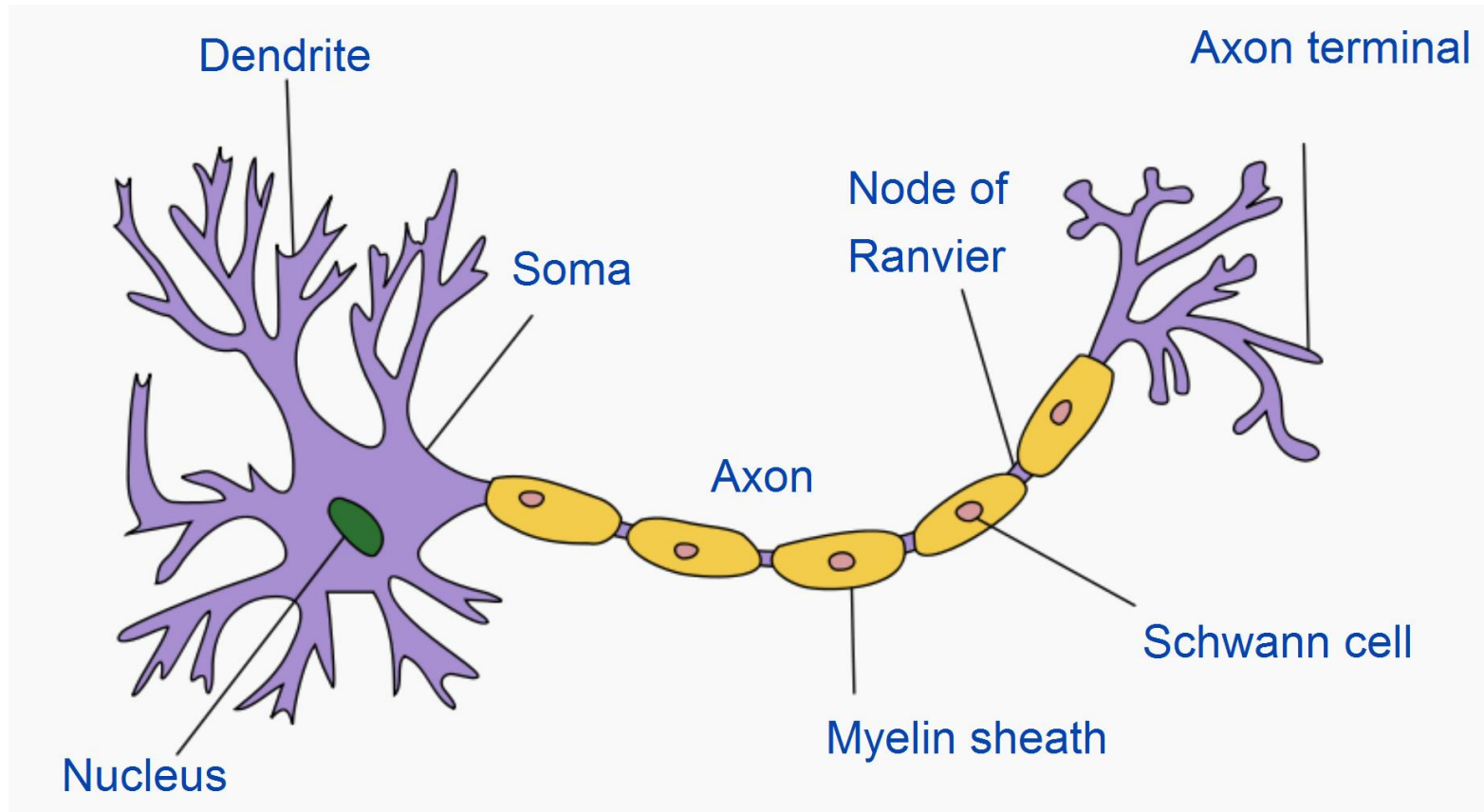


<http://www.cs.iit.edu/~mbilgic>



<https://twitter.com/bilgicm>

# NEURON



By Quasar Jarosz at English Wikipedia, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=7616130>

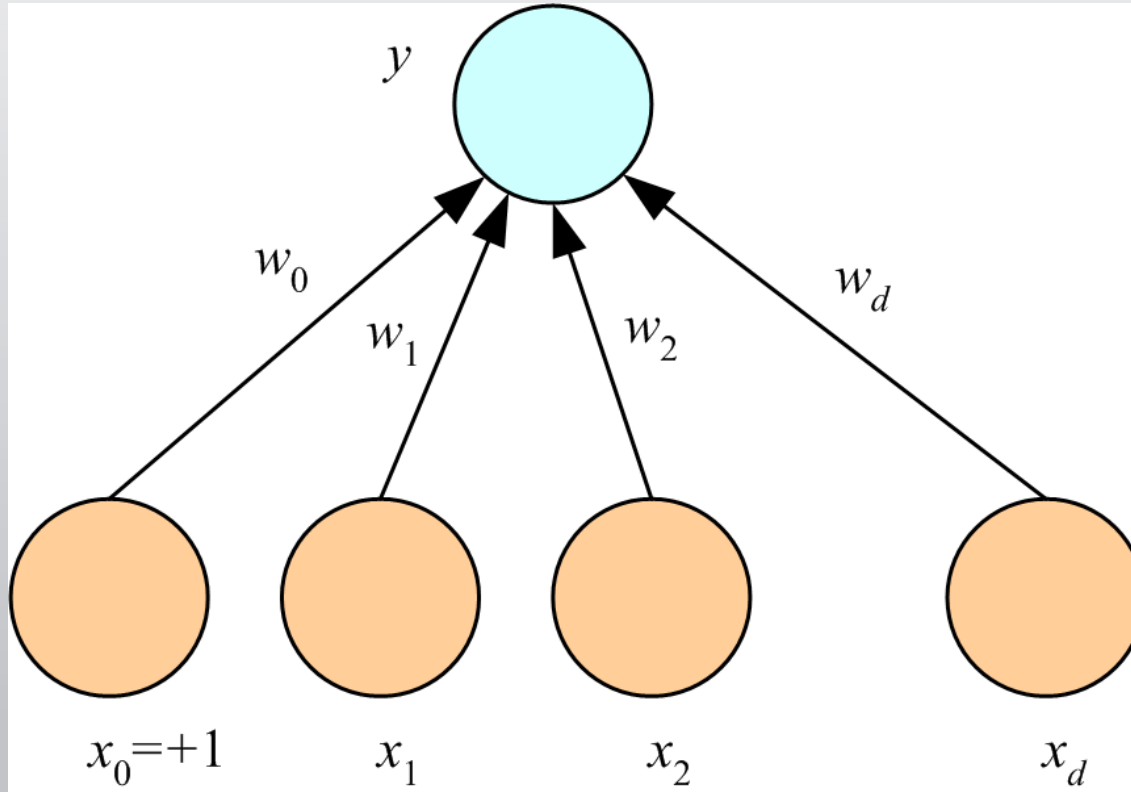
# NEURON

- Neurons can have multiple dendrites and at most one axon
- Typical connections are from an axon of a neuron to dendrites of other neurons
- Synaptic signals are received through dendrites and somas; signals are transmitted through axons
- Signals can excite or inhibit the receiving neuron
- A neuron fires when the excitement is above a threshold
- Note: these are general statements and simplifications, and there are many exceptions!

# ARTIFICIAL NEURAL NETWORKS

- Artificial neural networks are *inspired* by real neurons
- 1943 – One of the first neural computational models was proposed by McCulloch and Pitts
- 1958 – Rosenblatt proposed perceptron
- 1969 – A paper by Minsky and Papert almost killed the entire field
  - Perceptrons are incapable of representing XOR
  - Computational resources are too great
- 1975 – Backpropagation algorithm renewed interest in neural networks
- 1980s – parallel architectures were popular
- Late 1990s and 2000s – other methods, such as support vector machines, became more popular
- 2010s – neural networks of several hidden layers are back with the new name “deep learning”

# PERCEPTRON



$$y = \text{sign}(w_0 + \sum w_i x_i)$$

# WHAT AN ARTIFICIAL NEURON DOES

- Takes a weighted sum of its inputs
  - $w_0 + \sum_{i=1}^k w_i x_i$
  - Assume that there is always a constant input 1, that is,  $x_0 = 1$ . Then,
  - $\sum_{i=0}^k w_i x_i$
- Passes this sum through its activation function
  - $f(\sum_{i=0}^k w_i x_i)$

# EXAMPLES

- Logical AND
- Logical OR
- Logical XOR
- See OneNote and Notebook

# SIMPLE MULTILAYER NETWORK FOR XOR

- $XOR(A, B) = (A \wedge \neg B) \vee (\neg A \wedge B)$
- One perceptron for  $(A \wedge \neg B)$
- One perceptron for  $(\neg A \wedge B)$
- One perceptron for combining the outputs, through OR, of the two previous perceptrons
- See Notebook



# VARIOUS ACTIVATION FUNCTIONS

- Identity function
- Bipolar step function
- Binary sigmoid
- Bipolar sigmoid
- Hyperbolic tangent

# BIPOLAR STEP FUNCTION

- $f(\sum_{i=0}^k w_i x_i) = \text{sign}(\sum_{i=0}^k w_i x_i)$
- Returns either +1 or -1 (except right on the decision boundary)
- Useful for both hidden layers and output layer
- However, its discontinuous and it is problematic for learning algorithms that require taking its derivative

# IDENTITY FUNCTION

- $f(\sum_{i=0}^k w_i x_i) = \sum_{i=0}^k w_i x_i$
- Typically used for the output neurons, when the task is regression
- The identity function should not be used in the hidden layers
  - Linear combination of linear functions is another linear function, and hence using the identity function in the hidden layers do not increase representative power of the neural network

# BINARY SIGMOID

- $f(\sum_{i=0}^k w_i x_i) = \frac{1}{1 + e^{-\sum_{i=0}^k w_i x_i}}$

$$\frac{1}{1 + e^{-s}}$$

- This is the logistic function that we used
  - Except, notice the minus sign in front of the sum
    - This is only a convention and does not change much
- The output of the binary sigmoid is between 0 and 1
  - Useful for output layer when the task is classification
  - The output can be interpreted as a probability

# HYPERBOLIC TANGENT

$$\frac{e^s - e^{-s}}{e^s + e^{-s}}$$

- $f\left(\sum_{i=0}^k w_i x_i\right) = \frac{e^{\sum_{i=0}^k w_i x_i} - e^{-\sum_{i=0}^k w_i x_i}}{e^{\sum_{i=0}^k w_i x_i} + e^{-\sum_{i=0}^k w_i x_i}}$
- The output of tanh is between -1 and +1
  - Useful for output layer when the task is classification
  - Useful for both hidden and output layers

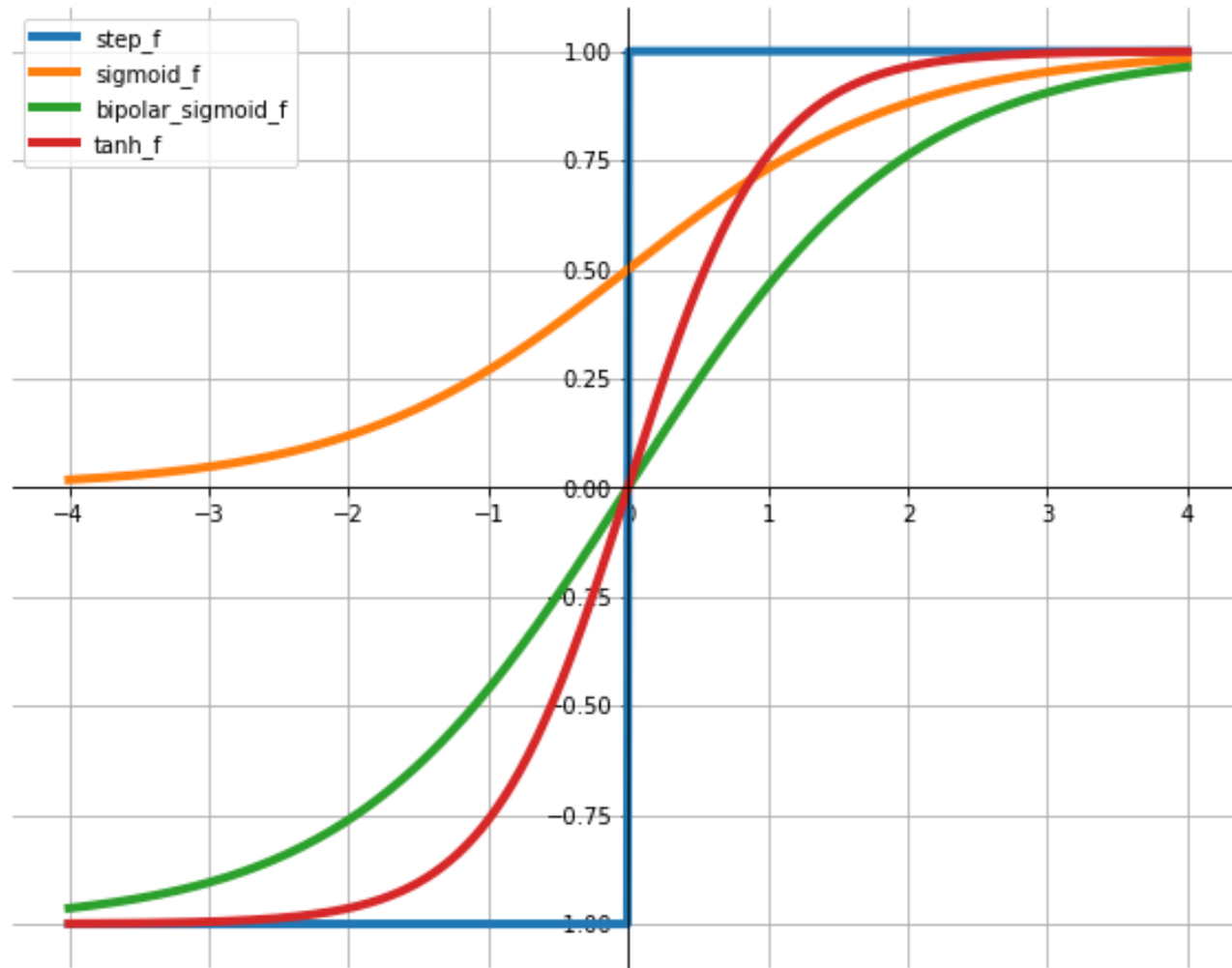
# RELU

- $f(\sum_{i=0}^k w_i x_i) = \sum_{i=0}^k w_i x_i$  if  $\sum_{i=0}^k w_i x_i > 0$ ;
  - 0 otherwise
- Typically used for hidden layers, especially for computer vision tasks

# BIPOLAR SIGMOID

- $f\left(\sum_{i=0}^k w_i x_i\right) = \frac{2}{1+e^{-\sum_{i=0}^k w_i x_i}} - 1$
- This is a rescaled version of the binary sigmoid
- The output of the bipolar sigmoid is between -1 and +1
  - Useful for output layer when the task is classification
  - Useful for both hidden and output layers

# ACTIVATION FUNCTION PLOTS





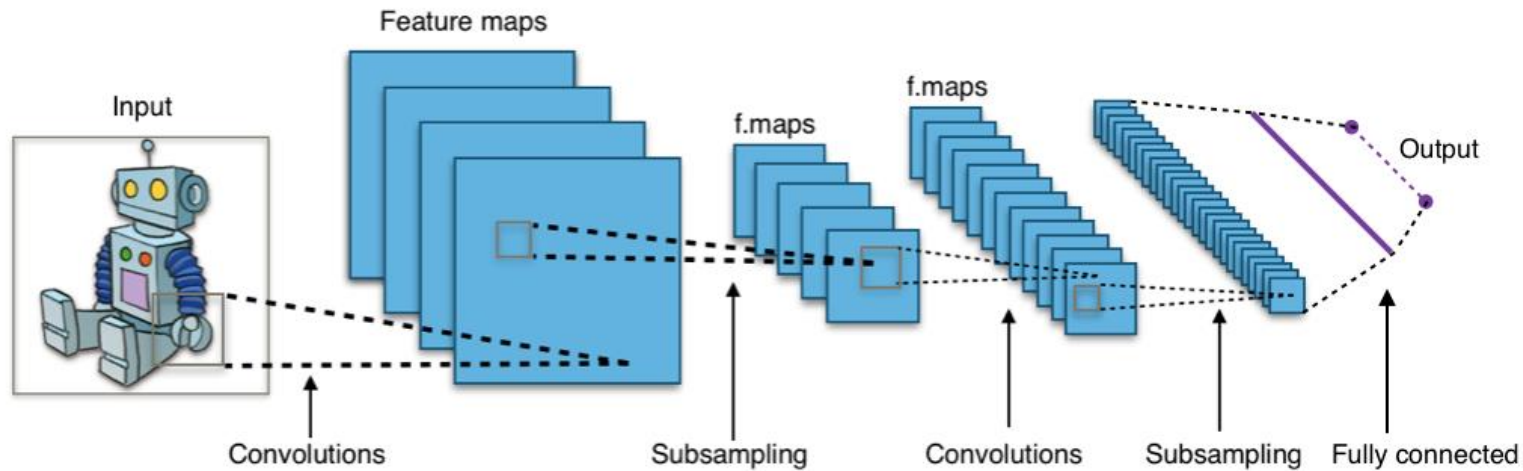
# DEEP LEARNING

- Several hidden layers
  - Millions of parameters
- Big data, big computation
- If a neural network with a single hidden layer is a universal approximator, why go deep?
  - “Why and When Can Deep -- but Not Shallow -- Networks Avoid the Curse of Dimensionality: a Review” <https://arxiv.org/abs/1611.00740>

# EXAMPLE NETWORK ARCHITECTURES

- Convolutional neural networks (CNN)
- Recurrent neural networks (RNN)
- Long Short-Term Memory networks (LSTM)

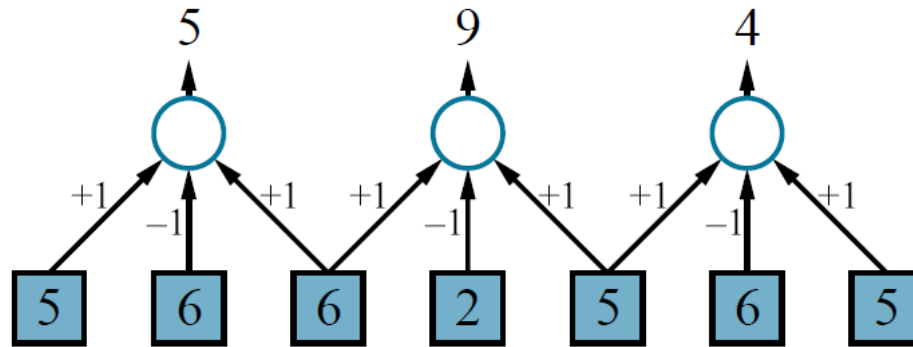
# CONVOLUTIONAL NEURAL NETWORKS



[https://commons.wikimedia.org/wiki/File:Typical\\_cnn.png](https://commons.wikimedia.org/wiki/File:Typical_cnn.png)

# CONVOLUTION

---



**Figure 21.4** An example of a one-dimensional convolution operation with a kernel of size  $l = 3$  and a stride  $s = 2$ . The peak response is centered on the darker (lower intensity) input pixel. The results would usually be fed through a nonlinear activation function (not shown) before going to the next hidden layer.

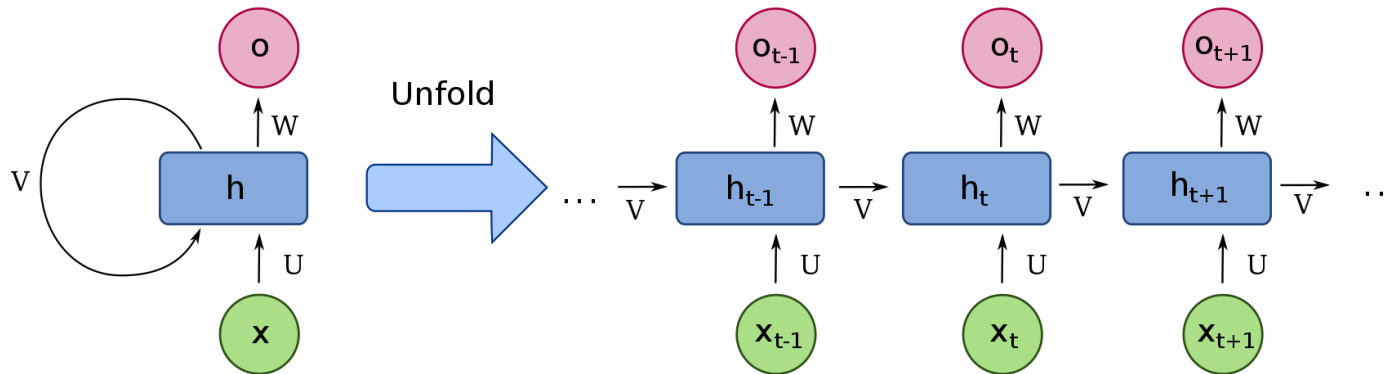
---

Figure from <http://aima.cs.berkeley.edu/figures.pdf>

# POOLING

- Aggregates a set of adjacent units
- Like convolution, has a kernel size and a stride size
- Unlike convolution, weights are fixed (not learned)
- Examples
  - Average pooling
  - Max pooling

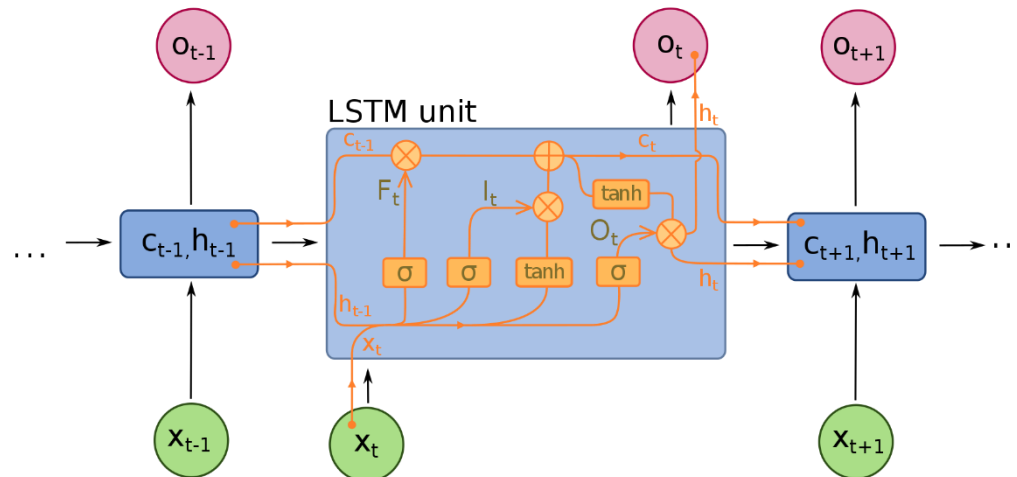
# RECURRENT NEURAL NETWORKS



[https://commons.wikimedia.org/wiki/File:Recurrent\\_neural\\_network\\_unfold.svg](https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg)

# LONG SHORT-TERM MEMORY (LSTM)

- A specialized RNN
- Works better than vanilla RNN for “remembering” long sequences
- Has additional units
  - Cell, forget gate, input gate, output gate



[https://commons.wikimedia.org/wiki/File:Long\\_Short-Term\\_Memory.svg](https://commons.wikimedia.org/wiki/File:Long_Short-Term_Memory.svg)

# OTHER NETWORKS/CONCEPTS

- Autoencoder
  - Input and output are the same
- Deep autoregressive model
  - Predict an element of the data using the other elements
- Generative adversarial networks (GAN)
  - A pair of generator and discriminator networks



# LEARNING THE WEIGHTS

- Define an error (loss) function
- Take its derivative with respect to the weights
- Perform gradient descent

# SOME ERROR/LOSS FUNCTIONS

- Classification: log-loss, cross entropy, negative CLL
  - $-(1 - t) \times \ln(1 - y) - t \times \ln(y)$
  - $t$ : the true target value (0/1)
  - $y$ : probability of class 1
- Regression: squared error
  - $\frac{1}{2}(t - y)^2$
  - $t$ : the true target value
  - $y$ : the predicted value

# BACKPROPAGATION ALGORITHM

- See OneNote

# DERIVATIVES OF THE ACTIVATION FUNCTIONS

- $f$  is the activation function,  $h$  is the weighted sum of the incoming signals
- $f(h(x)) \equiv$  binary sigmoid
  - $\frac{\partial f(h(x))}{\partial x} = f(h(x)) \times (1 - f(h(x))) \times \frac{\partial h(x)}{\partial x}$
- $f(h(x)) \equiv \tanh$ 
  - $\frac{\partial f(h(x))}{\partial x} = (1 + f(h(x))) \times (1 - f(h(x))) \times \frac{\partial h(x)}{\partial x}$
- See OneNote

# OVERFITTING

- Neural networks are powerful tools
- Even with a single hidden layer, they are “universal approximators”, i.e., they can approximate arbitrary functions arbitrarily close
- Therefore, it is very easy to overfit them
- To prevent overfitting, utilize
  - Domain knowledge
  - Shared parameters
  - Validation data
  - Regularization
  - Dropout

# SOME LIBRARIES

- Scikit-learn

- [https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html](https://scikit-learn.org/stable/modules/neural_networks_supervised.html)
- Only MLP; no GPU support

- Keras

- <https://keras.io/>

- Tensorflow

- <https://www.tensorflow.org/>

- PyTorch

- <https://pytorch.org/>