Foundations of Machine Learning

Neural Networks

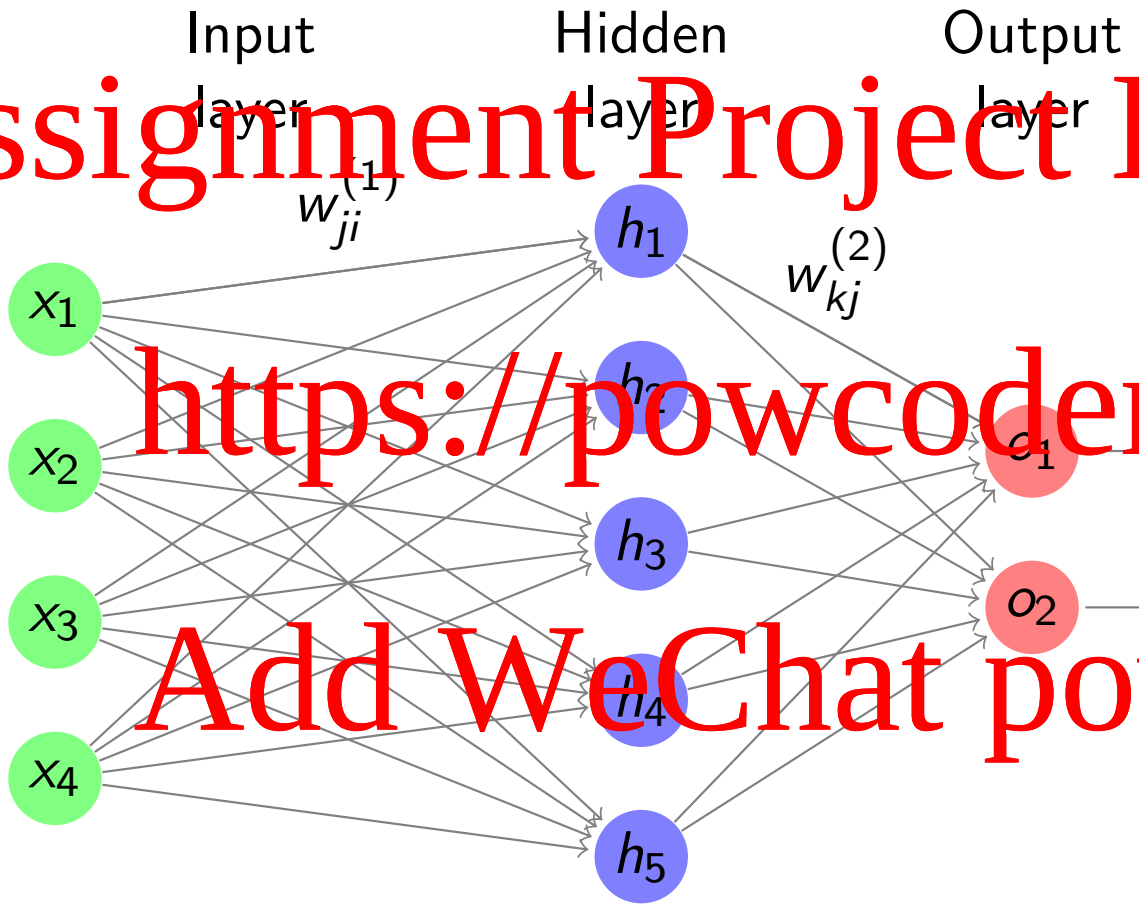
Kate Farrahi

ECS Southampton

November 23, 2020

The Multilayer Perceptron

Multilayer Perceptron



The Multilayer Perceptron

- MLPs are fully connected
- MLPs consist of three of more layers of nodes
- 1 input layer, 1 output layer, 1 or more hidden layers
- A 4 - 5 - 2 fully connected network is illustrated on the previous slide

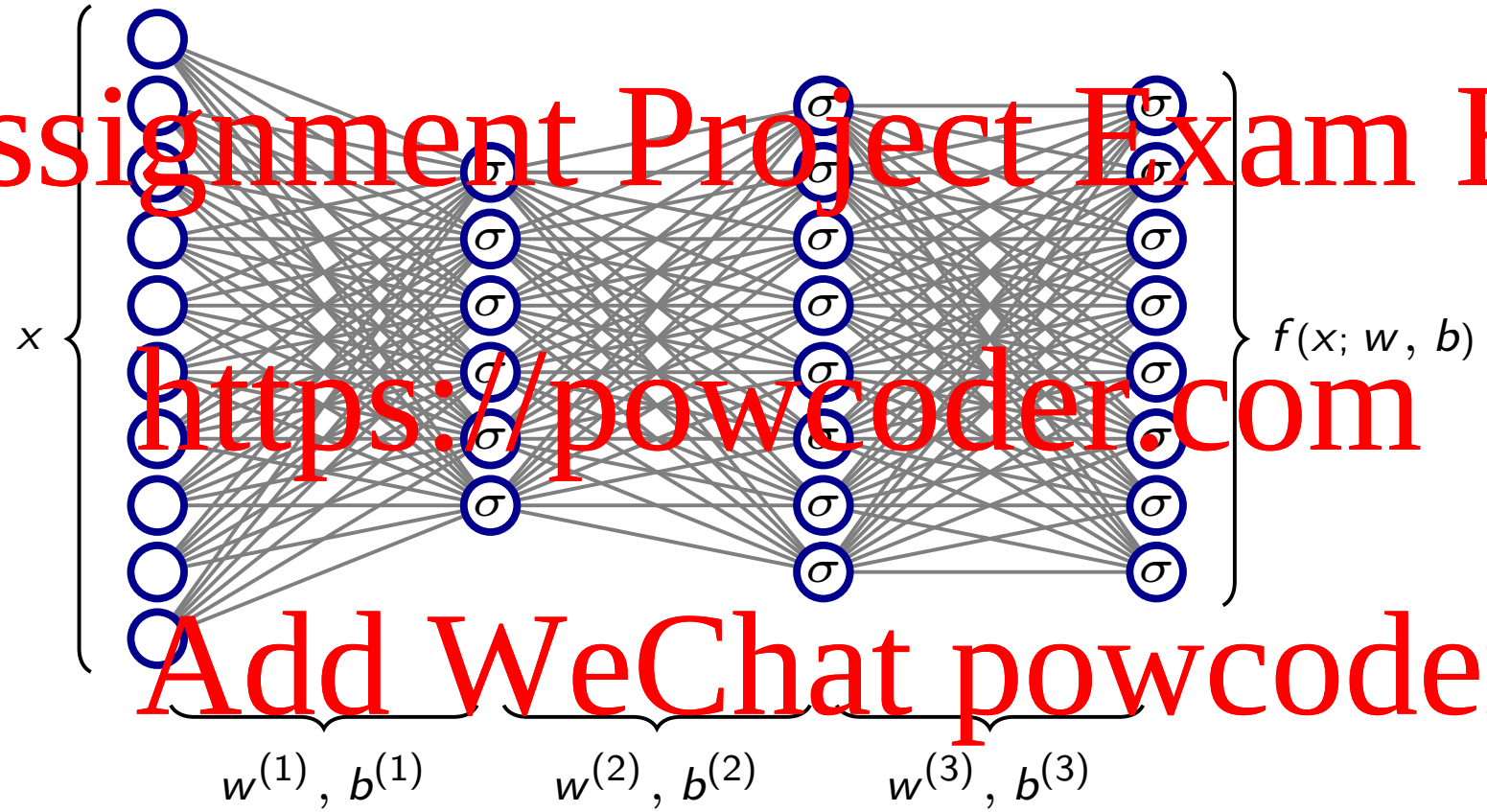
The Multilayer Perceptron – Input Layer

- d - dimensional input x
- no neurons at the input layer, simply input units
- each input unit simply emits the input x_i

The Multilayer Perceptron – Output Layer

- c neurons in the output layer
- each neuron in the output layer also uses a non-linear activation function
- c and the activation function at the output layer relate to the problem that is being solved – more details later

Multilayer Perceptrons



Multiple layers of units

Multilayer Perceptron (MLP)

We can define the MLP formally as,

$$\forall l = 1, 2, \ldots, L, \qquad a^{(l)} = \sigma(w^{(l)} a^{(l-1)} + b^{(l)})$$

where $a^{(0)} \equiv x$, and $f(x; w, b) = a^{(L)} = \hat{y}$

Note, we will define the weighted input term,
$z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}$, for the backpropagation derivation

Multilayer Perceptron (MLP)

Define the following expressions:

1. $z^{(1)}$
2. $z^{(2)}$
3. $a^{(1)}$
4. $a^{(2)}$
5. $a^{(3)}$

Activation Functions

Activation Functions

- The activation function in a neural network is a function used to transform the activation level of a unit (neuron) into an output signal.
- The activation function essentially divides the original space into typically two partitions, having a "squashing" effect.

- The activation function is usually required to be a non-linear function.
- The input space is mapped to a different space in the output.

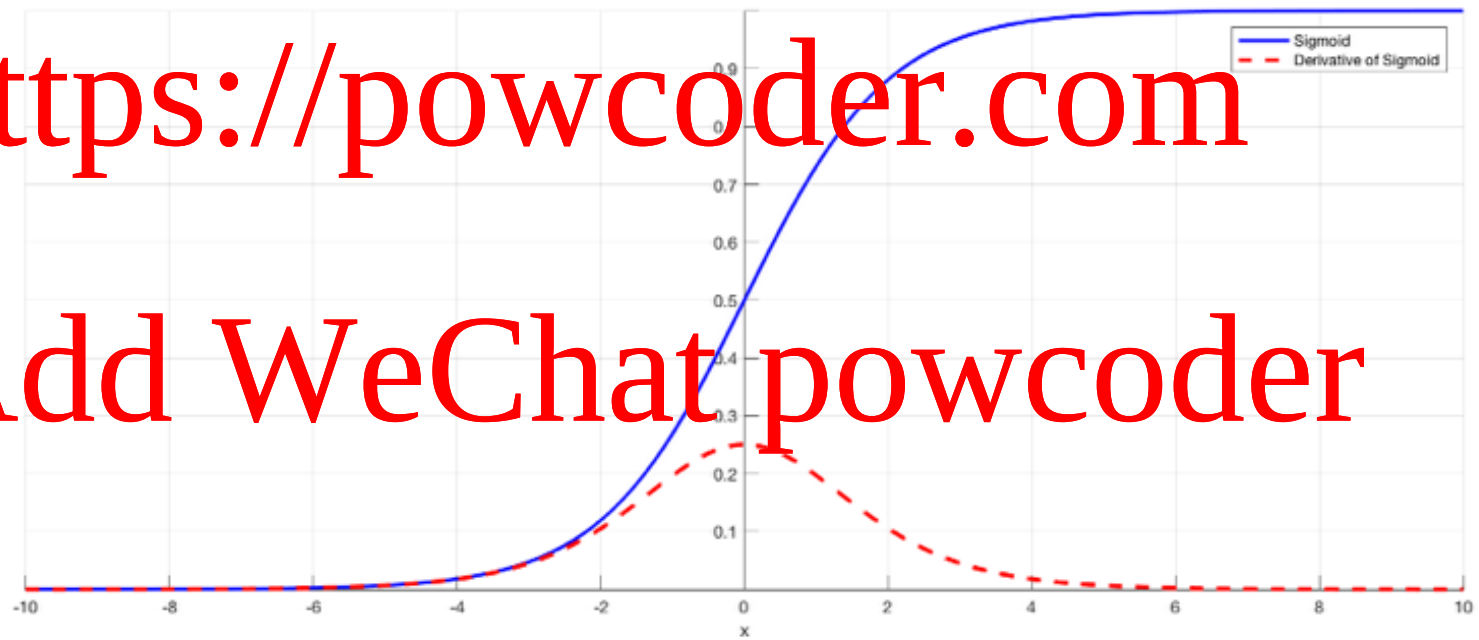
- There have been many kinds of activation functions proposed over the years (640+), however, the most commonly used are the Sigmoid, Tanh, ReLU, and Softmax

The Logistic (or Sigmoid) Activation Function

- The sigmoid function is a special case of a logistic function given by $f(x) = \frac{1}{1+e^{-x}}$
- non-linear (slope varies)
- continuously differentiable
- monotonically increasing
- NB: e is the natural logarithm

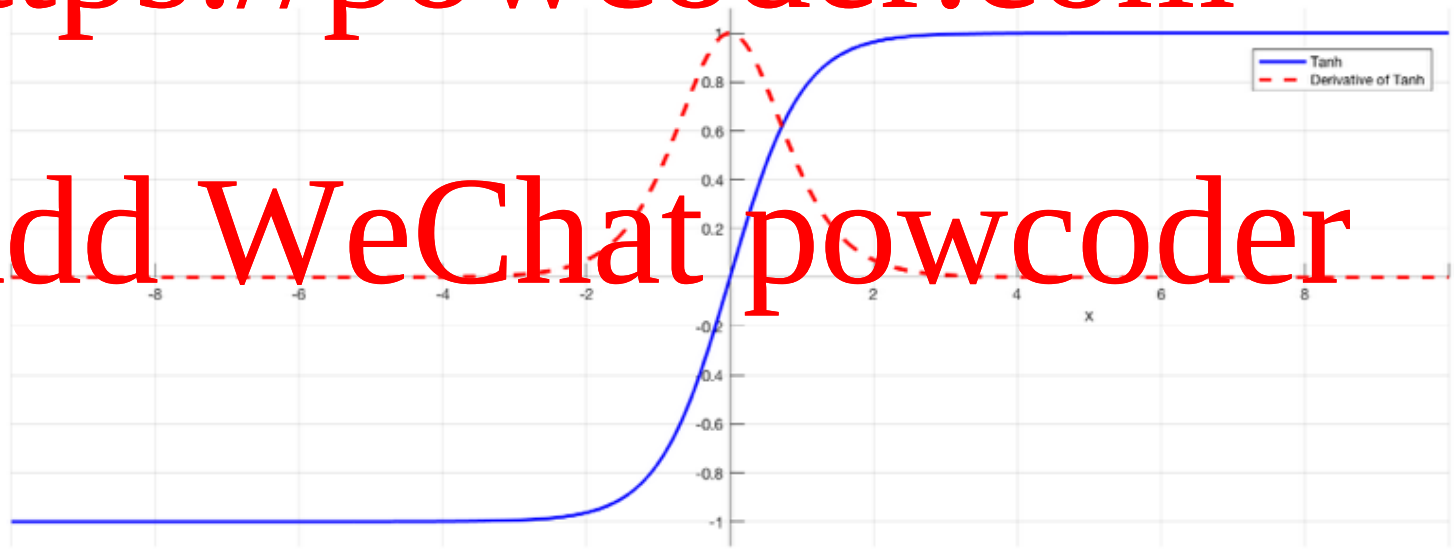
Sigmoid Function – Derivative

- The sigmoid function has an easily calculated derivative which is usesd in the back propagation algorithm



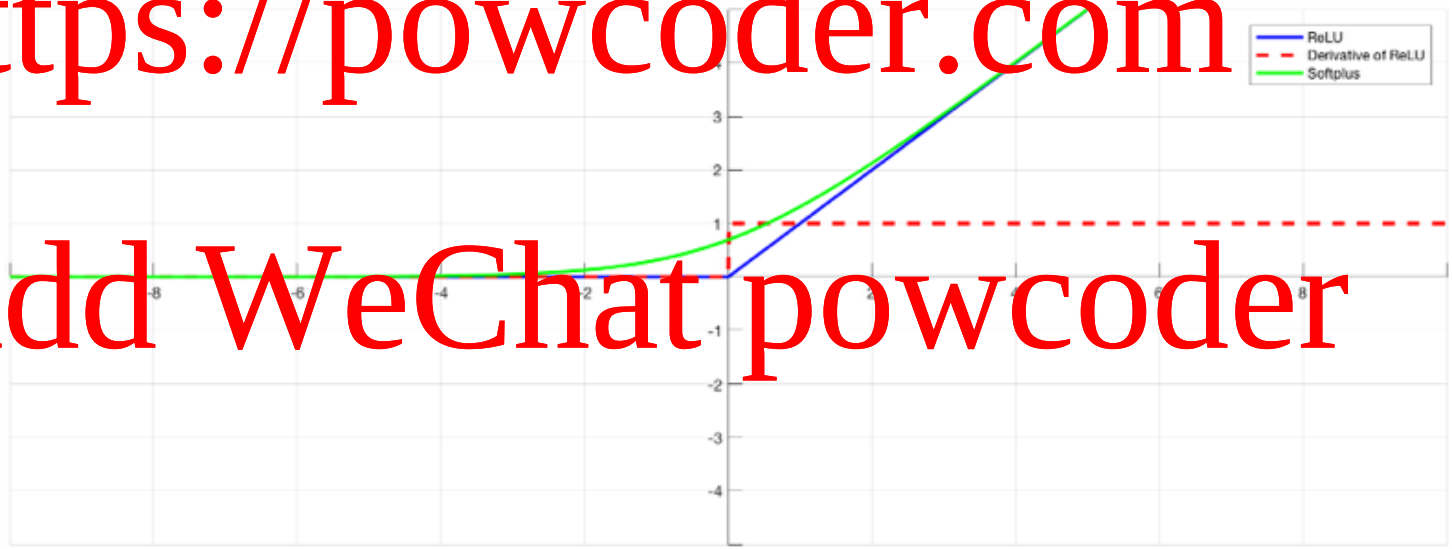
The Hyperbolic Tangent Activation Function

- The tanh function is also "s"-shaped like the sigmoidal function, but the output range is (-1, 1)
- $tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
- $tanh'(x) = 1 - tanh^2(x)$



Rectified Linear Units (ReLU)

- The ReLU (used for hidden layer neurons) is defined as: $f(x) = max(0, x)$
- The range of the ReLU is between 0 to ∞



Softmax

The softmax is an activation function used at the output layer of a neural network that forces the outputs to sum to 1 so that they can represent a probability distribution across a discrete mutually exclusive alternatives.

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for } j = 1, 2, \ldots, K$$

Note $\frac{\partial y_i}{z_i} = y_i(1 - y_i)$

- The output of a softmax layer is a set of positive numbers which sum up to 1 and can be thought of as a probability distribution

Question

- Given a binary classification problem, can you have 1 neuron at the output layer of an MLP?
- If so, what is the neuron's activation function?

Question

- Given a binary classification problem, can you have 2 neurons at the output layer of an MLP?
- If so, what are the neuron's activation functions?

The Cost Function (measure of discrepancy)

- Mean Squared Error (MSE) for M data points is given by $MSE = \frac{1}{2*M} \sum_{i=1}^{M} (\hat{y}_i - y_i)^2$
- $\frac{1}{2*M}$ just a constant so can be replaced by $\frac{1}{2}$ or $\frac{1}{M}$
- Other cost functions can be used as well, for example the KL divergence or Hellinger distance [1]
- MSE can be slow to learn, especially if the predictions are very far off the targets [2]
- Cross-Entropy Cost function is generally a better choice of cost function (discussed next)

[1] https://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

[2] http://neuralnetworksanddeeplearning.com/chap3.html

Cross-Entropy Cost Function

- $J = -\frac{1}{M}\sum_{i=1}^{M}[y_i ln(\hat{y}_i) + (1 - y_i) ln(1 - \hat{y}_i)]$ where M is the number of training examples
- The cross-entropy cost function is non-negative, $J > 0$
- $J \approx 0$ when the prediction and targets are equal (i.e. $y = 0$ and $\hat{y} = 0$ or when $y = 1$ and $\hat{y} = 1$)
- $\frac{\partial J}{\partial w_{ij}}$ is proportional to the error in the output $(\hat{y} - y)$ and therefore, the larger the error, the faster the neuron will learn!