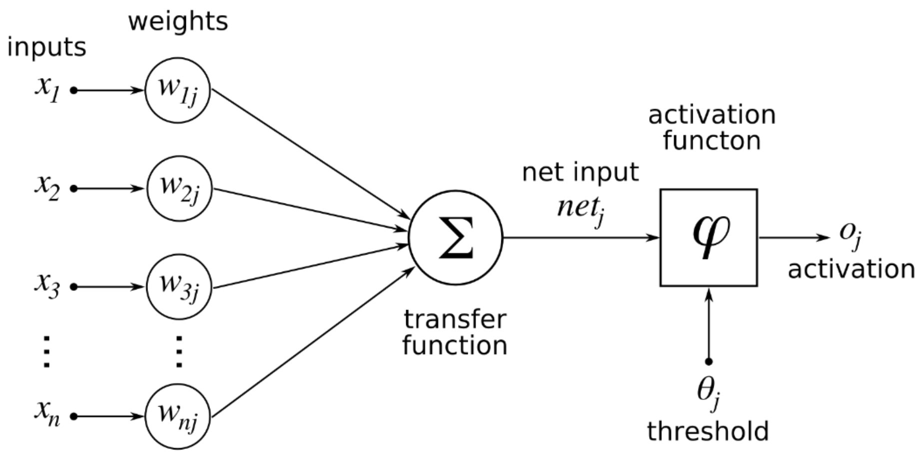
SL03: Neural Networks

Perceptron:

- A perceptron is a simple computational imitation to a brain cell. It allows computations to fire under certain stimuli, allowing for learning.



- The Perceptron calculates the sum of products of the inputs *X* and their corresponding weights *W*, and then compare the result with an activation threshold. If the sum is greater than or equal the threshold θ, the Perceptron fires (outputs one).
- The Perceptron can be used to model Boolean functions:
  - AND: $w_1 = ½$, $w_2 = ½$, $\theta = ¾$
  - OR: $w_1 = 1$, $w_2 = 1$, $\theta = 1$
  - XOR: Requires 2 Perceptrons.
- Perceptron training:
  There're two ways to learn the weights of a Perceptron:
  - Perceptron Rule:
    → The Perceptron Learning Rule defines how to update the weights in an iterative way.
    → It works by "rewarding" the correct weights and "punishing" the incorrect ones.

$$w_i = w_i + \Delta w_i$$
$$\Delta w_i = \eta(y - y')x_i$$
$$y' = (\sum_i w_i x_i \geq 0)$$

    → The Perceptron Learning Rule works by adjusting the weights based on the neuron's performance.
    → In the best case, the difference between the predicted value *y'* and the training example *y* is zero, which means that the perceptron was able to correctly predict the value of *y*, so the weight will not be updated. It keeps repeating this process till the error saturates at 0.
    → Otherwise, the weights will be adjusted to correct the error in prediction.
    → η: This is called the "learning rate". It controls the learning process, so that the weights get adjusted by a small value each time to avoid overshooting.

- → The Perceptron Learning Rule works only on linearly separable datasets, where we can separate the positive and negative examples using a straight line (or a half plane).
- → If the data is linearly separable, the Perceptron will eventually find the suitable line to separate the examples.
- → The problem is, there's no obvious way to determine if the data is linearly separable or not, especially with higher dimensions.
- → One workaround it to run the algorithm on the data and see if it ever stops.

- Gradient Descent:
  - → Gradient Descent is an algorithm that can work with data that isn't linearly separable.
  - → Let *D* be the training set with ordered pairs (*x*, *y*), where *y* is the target value. Gradient Descent works by taking the dot product *a* of *w* and *x* (which is essentially a simple linear regression), and then minimize the error of this activation using calculus.

$$a = \sum_i x_i w_i$$
$$y' = \{a \geq 0\}$$
$$E(w) = \frac{1}{2} \sum_{(x,y)\in D} (y - a)^2$$

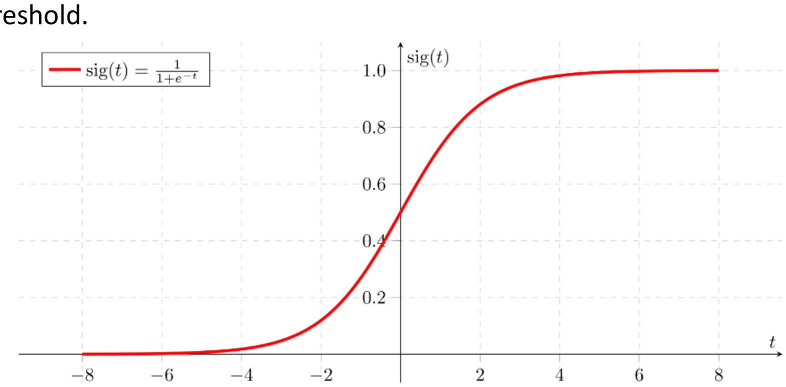
  - → To minimize error $E(w)$, we calculate the partial derivative of $E(w)$ with respect to each of the individual weights:

$$\frac{dE}{dw_i} = \frac{d}{dw_i} \frac{1}{2} \sum_{(x,y)\in D} (y-a)^2$$
$$= \frac{1}{2} \times 2 \sum_{(x,y)\in D} (y-a) \frac{d}{dw_i} - \sum_i x_i w_i$$
$$= \sum_{(x,y)\in D} (y-a)(-x_i)$$

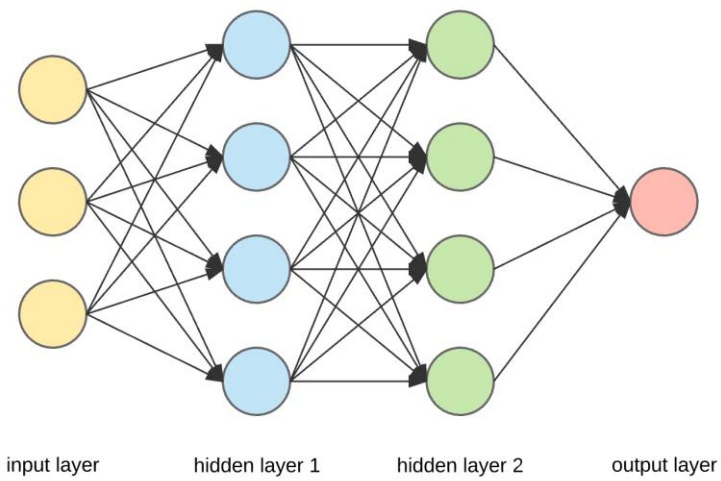
  - → The error decreases fastest in the direction opposite to the gradient.
  - → Robust to data that is not linearly separable.
- Perceptron Rule vs Gradient Descent:
  - → Perceptron Learning $\Delta w_i = \eta(y - y')x_i$
  - → Gradient Descent $\Delta w_i = \eta(y - a)x_i$
  - → The reason we can't use y' in Gradient Descent is that it will make the formula non-differentiable.

- Sigmoid Function:
  - The Sigmoid Function facilitates a differentiable threshold.

$$\sigma(a) = \frac{1}{1 + e^{-a}}$$
$$a \to -\infty \quad \sigma(a) \to 0$$
$$a \to +\infty \quad \sigma(a) \to 1$$
$$D\sigma(a) = \sigma(a)(1 - \sigma(a))$$



Neural Networks:



- A Neural Network is a chain of relations between the input and the output, with hidden layers in between.
- Each hidden layer computes the weighted sum (sigmoid) of the layer before it.
- If sigmoid is used in the nodes of the hidden layers, then the mapping from input to output will be differentiable.
  - This means that we can determine how each weight in the network affects the mapping from input to output.
  - This facilitates Backpropagation, which means performing a backward distribution of the error values (computed at the output) back to the network layers to optimize the weights.
- Note that using sigmoid in the Neural Network doesn't always guarantee convergence. Sigmoid is not a hard threshold, like the Perceptron. In a multilayer setting, we might end up in a local minima.

Optimizing Weights:

- There're many optimization methods to find a global minima:
  - Momentum: An extra factor to push past the local minima.
  - Higher order derivatives: Use a combination of weights.
  - Randomized optimization.
  - Penalizing complexity: Penalizing more nodes, more layers or large weights.

Restriction Bias:

- The Restriction Bias determines what is the set of hypotheses we will consider, which in turn highlights the representation power of the learning algorithm.
- As we discussed, Perceptions can only work with linear data (half spaces). While sigmoid allow for representing more complex data, and using hidden layer increase the representation power even more.
- All these components of Neural Networks allow for modeling many types of functions:
  - Boolean: Network of threshold-like units.
  - Continuous: A single hidden layer with enough hidden nodes.
  - Arbitrary: Multiple hidden layers.
- We can conclude that Neural Networks have a low Restriction Bias because you they have the ability to model a wide variety of functions. This, however, increase the possibility of overfitting.
- A Neural Network will not only overfit because of excessive complexity but can also overfit because of excessive training.

Preference Bias:

- The Preference Bias determines which representation is preferred.
- How to initialize the weights?
  One preference in Neural Networks is to choose the initial weights to be small random values:
  - Random values provide variability that helps in avoiding local minima.
  - Small values have low complexity, which is preferred because larger weights can lead to overfitting.
- In summary, Neural Networks prefer simpler and generalizable representations