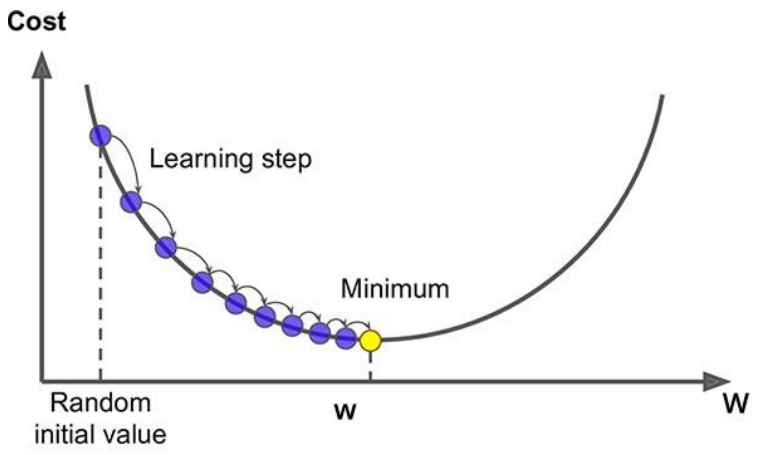
Gradient Descent for Neural Networks

Spring 1400 by Matin Zivdar

Gradient Descent

An optimization problem trying to find a local minimum for a **differentiable** function capable of solving a wide range of problems. Here are the steps of this algorithm:

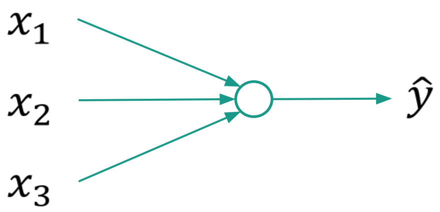
1. Calculate the local gradient at x.
2. x = x - α * gradient

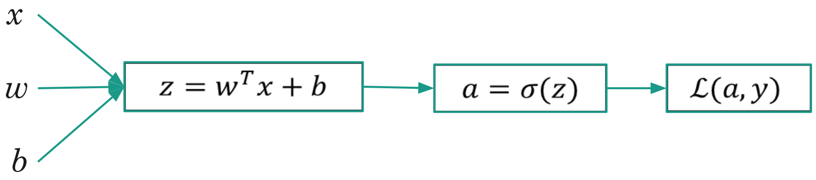


What is a
Neural Network?

Threshold Logic Unit

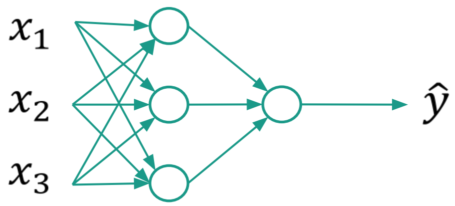
Neuron's job:
- Multiply each input and its weight.
- Sum over them.
- Apply a "activation function".

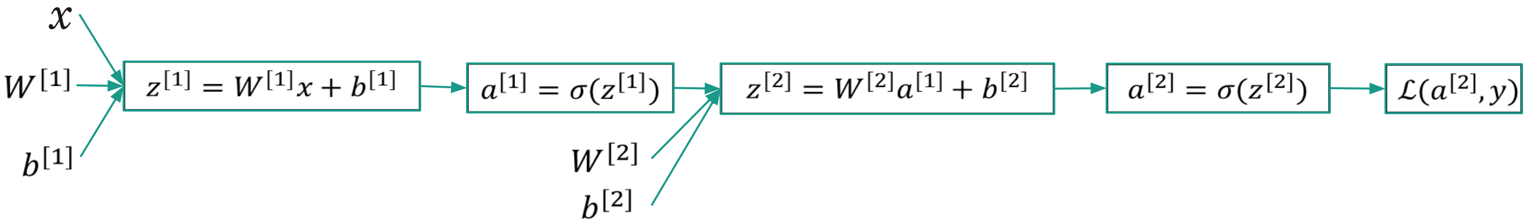




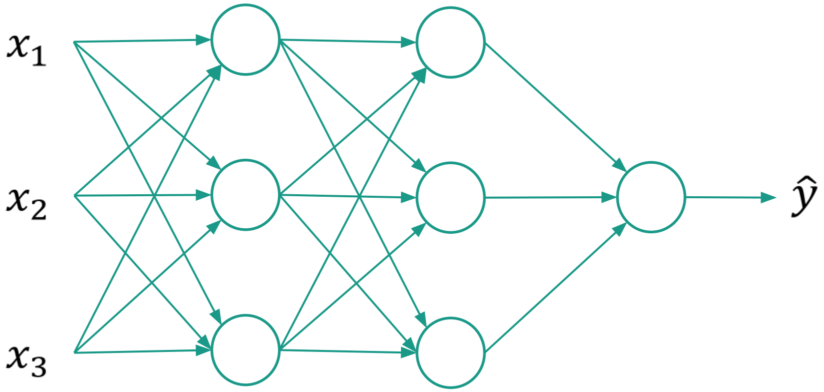
Neural Network

You can form a neural network by stacking together a lot of TLUs. Whereas previously, these nodes corresponds to two steps to calculations.

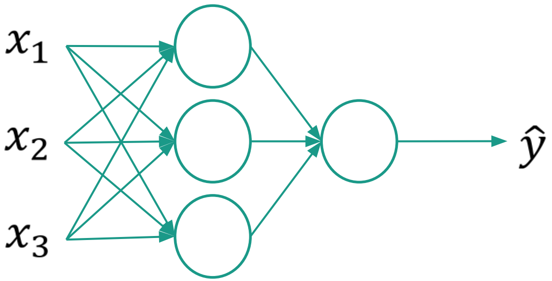




Neural Network Representation

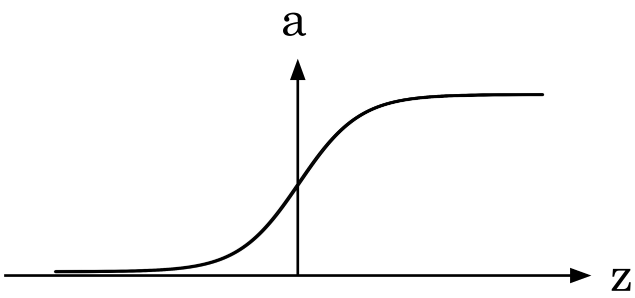


Activation Functions

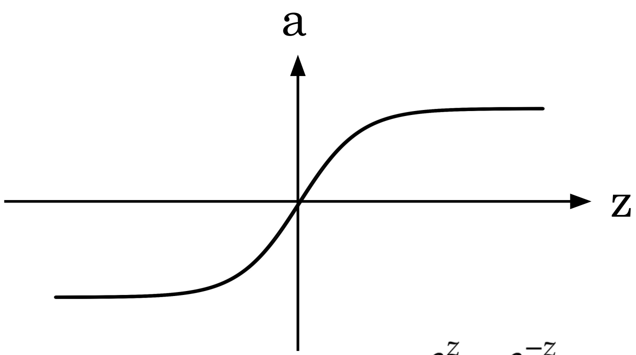


Given x:

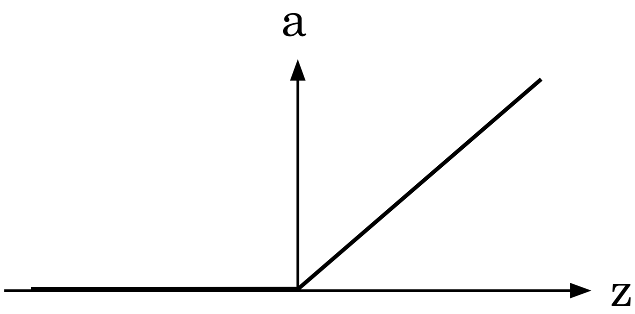
$$
\begin{aligned}
\mathrm{z}^{[1]} &= W^{[1]}x + b^{[1]} \\
\mathrm{a}^{[1]} &= \sigma\left(\mathrm{z}^{[1]}\right) \\
\mathrm{z}^{[2]} &= W^{[2]}x + b^{[2]} \\
\mathrm{a}^{[2]} &= \sigma\left(\mathrm{z}^{[2]}\right)
\end{aligned}
$$



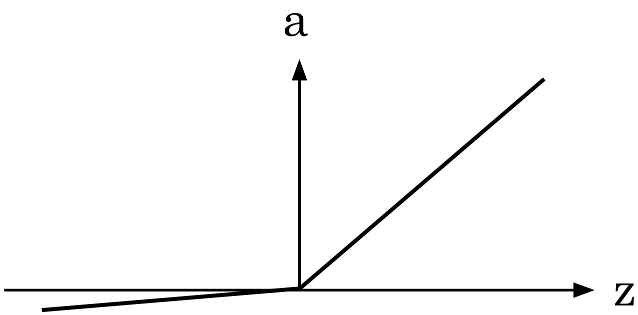
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$



Relu: $a = max(0,\ z)$



Leaky Relu: $a = max(0.01z,\ z)$

Gradient Descent for Neural Networks

Parameters: $w^{[1]}, b^{[1]},\ w^{[1]},\ b^{[2]}$

Cost Function: $J\left(w^{[1]}, b^{[1]},\ w^{[1]},\ b^{[2]}\right) = \frac{1}{m}\sum \mathscr{L}(\hat{y} - y)$

Gradient Descent:

Repeat {

Compute Predicts $(\hat{y},\ i = 1,\ldots,m)$

$$dw^{[1]} = \frac{dJ}{dw^{[1]}},\quad db^{[1]} = \frac{dJ}{db^{[1]}},\quad \ldots$$

$$w^{[1]} := w^{[1]} - \alpha \,.\, dw^{[1]}$$
$$b^{[1]} := b^{[1]} - \alpha \,.\, db^{[1]}$$
$$. \; . \; .$$

}

Homework ^-^

Formulas for computing derivatives

Forward Propagation:

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1](i)}$$
$$a^{[1](i)} = tanh\left(z^{[1](i)}\right)$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2](i)}$$
$$\hat{y}^{(i)} = a^{[2](i)} = \sigma\left(z^{[2](i)}\right)$$

Vectorized Forward Propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}\left(Z^{[1]}\right) = tanh\left(Z^{[1]}\right)$$
$$Z^{[2]} = W^{[2]}X + b^{[2]}$$
$$A^{[2]} = g^{[2]}\left(Z^{[2]}\right) = \sigma\left(Z^{[2]}\right)$$

Formulas for computing derivatives (contd.)

Backward Propagation:

$$dz^{[2]} = a^{[2]} - y$$
$$dW^{[2]} = dz^{[2]} a^{[1]}$$
$$db^{[2]} = dz^{[2]}$$
$$dz^{[1]} = W^{[2]^T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$
$$dW^{[1]} = dz^{[1]} a^{[1]}$$
$$db^{[1]} = dz^{[1]}$$

Vectorized Backward Propagation:

$$dZ^{[2]} = A^{[2]} - Y$$
$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$
$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Thanks for your attention.