Fourth Industrial Summer School

Advanced Machine Learning

Neural Networks and Deep learning-part2

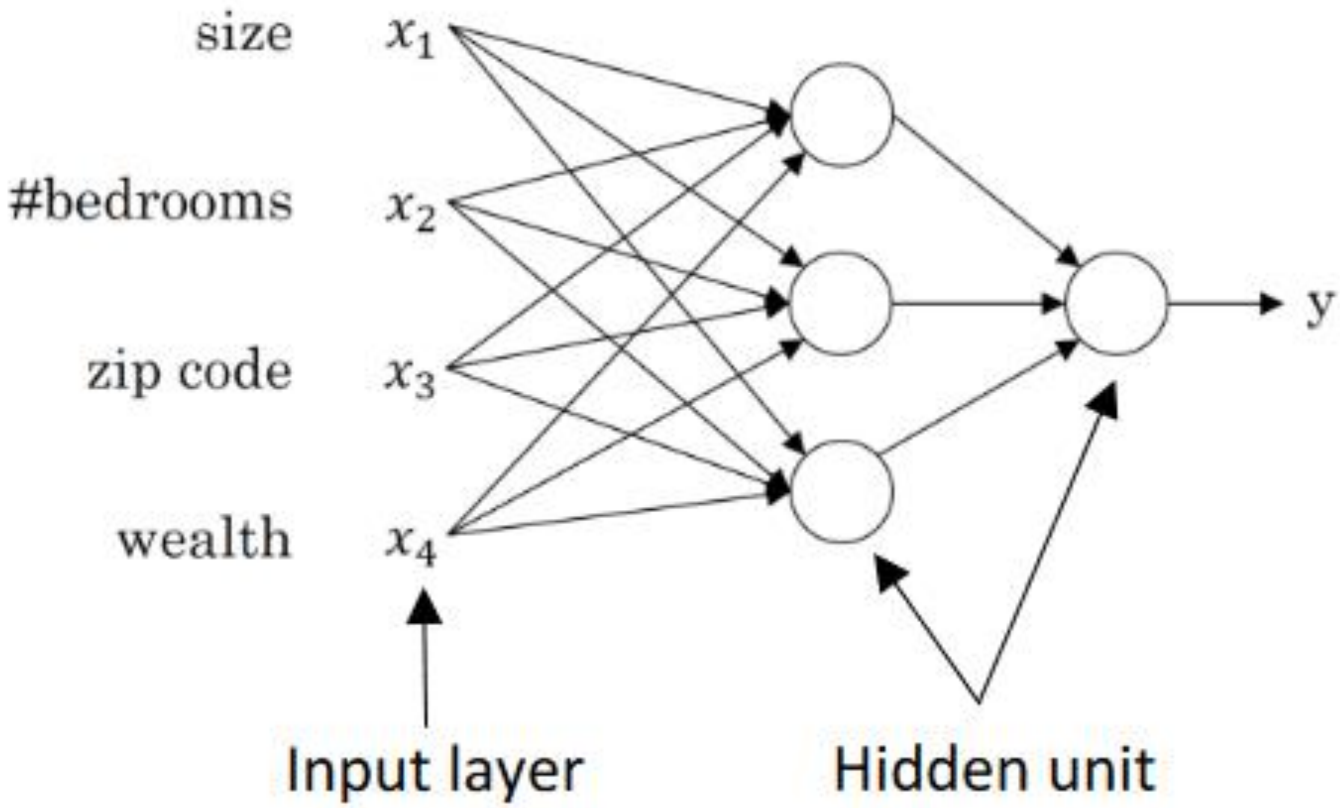
Session Objectives

- ✓ Introduction
- ✓ Fundamentals
- ✓ Neural Network Intuitions
- ✓ 2-Layer Neural Network

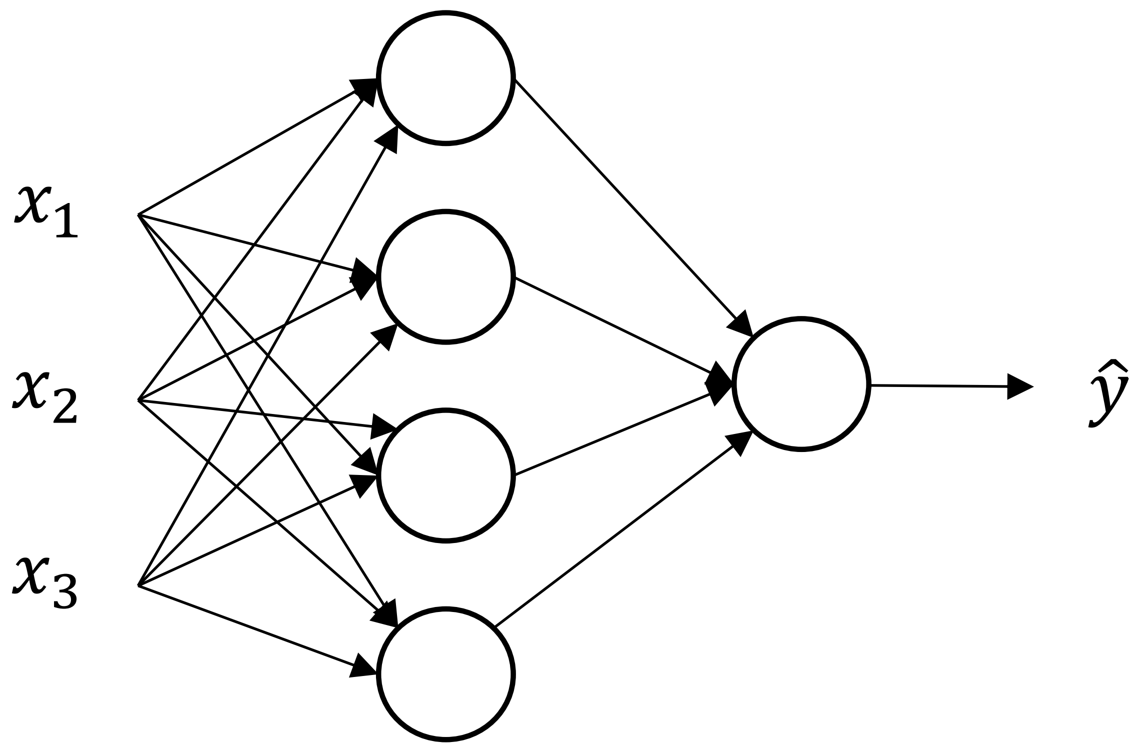
Neural Networks

Basic Architecture

Architecture of a standard neural network

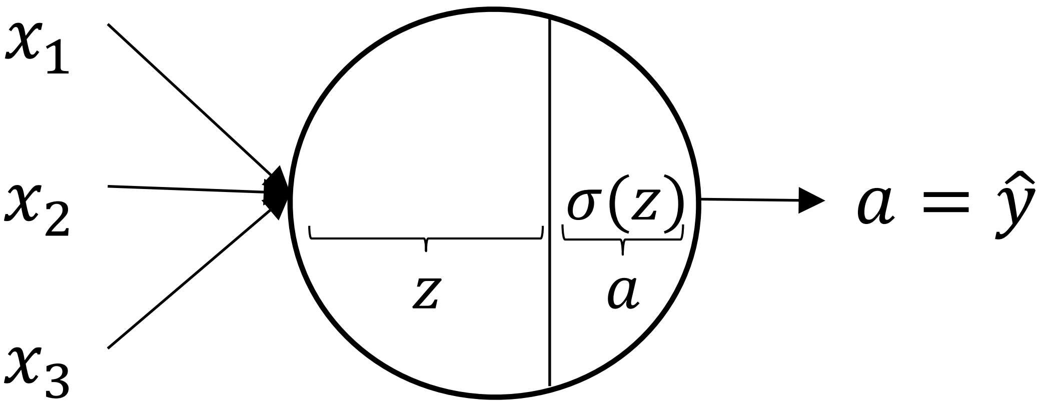


Neural Network Representation



- Normally termed as 2-layer neural network. (input layer is not counted).

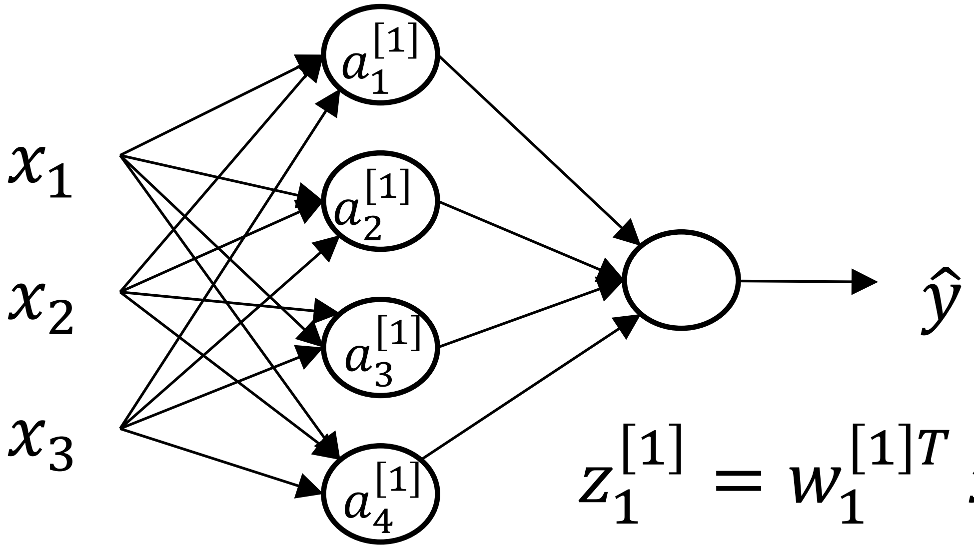
Neural Network Representation



$$z = w^T x + b$$

$$a = \sigma(z)$$

Computing a Neural Network's Output



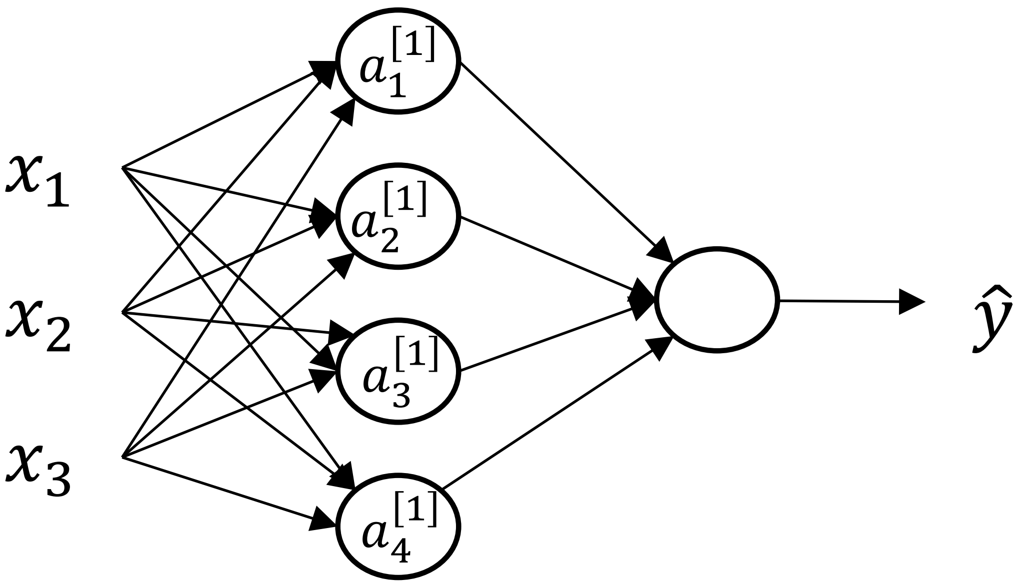
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \; a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \; a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \; a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \; a_4^{[1]} = \sigma(z_4^{[1]})$$

Computing a Neural Network's Output



Given input x:

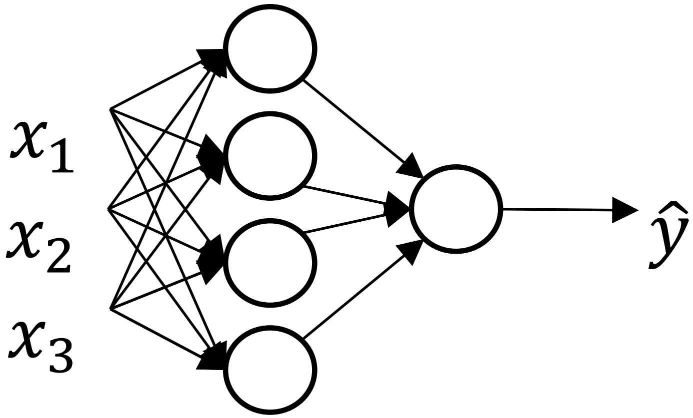
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Vectorising across multiple examples



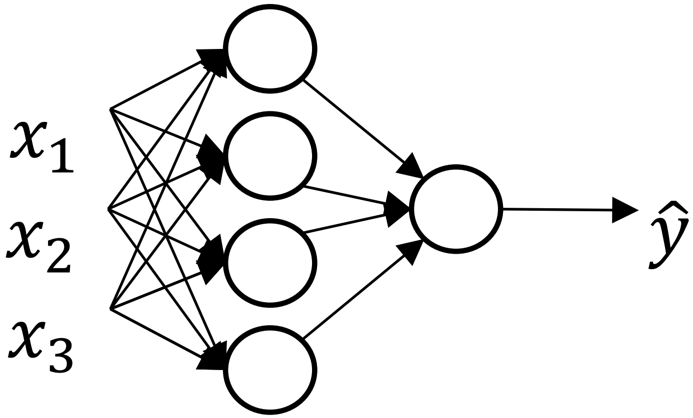
for i = 1 to m

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

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$\mathrm{A}^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \ldots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Vectorising across multiple examples

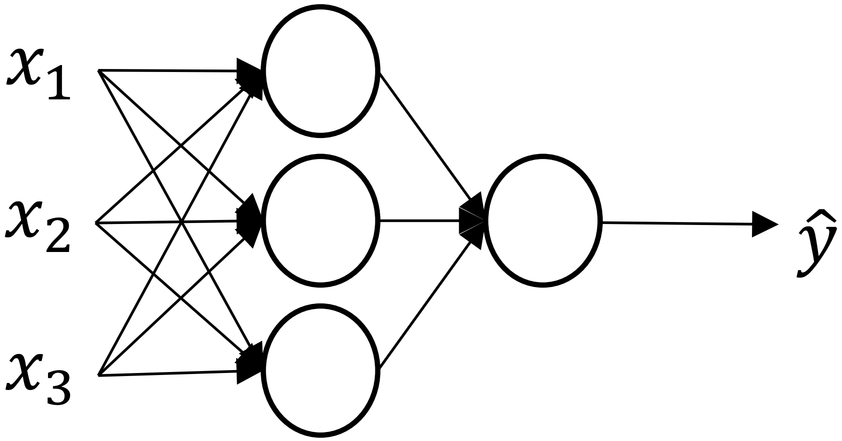



```
for i = 1 to m
```

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

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

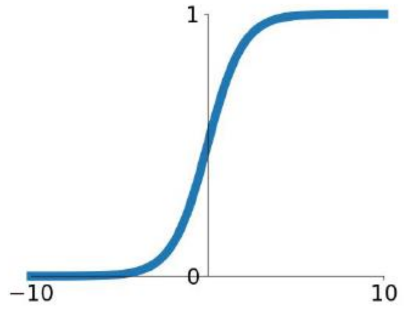
Activations



Given x:

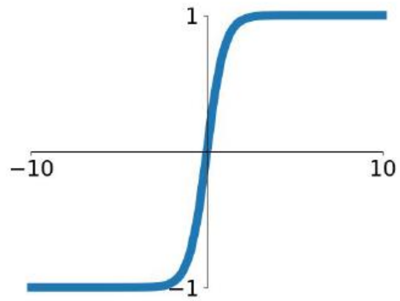
$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Different activation functions



Sigmoid

- Saturated neurons "kill" the gradients
- Sigmoid outputs are not zero-centered
- exp() is a bit compute expensive



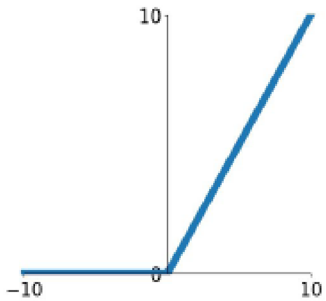
tanh(x)

- Still kills gradients when saturated

Different activation functions

ReLU

$\max(0, x)$

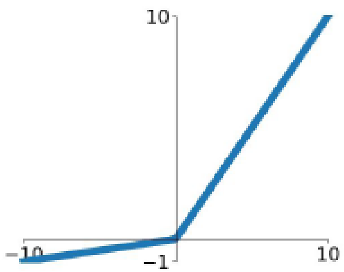


- Does not saturate
- Computationally efficient
- Converges much faster
- Not zero-centered output
- Loosing half the spectrum

- will not "die".

Leaky ReLU

$\max(0.1x, x)$



Derivatives of activation functions

Gradient descent for Neural Networks

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]^T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(\mathrm{z}^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

Update of parameters

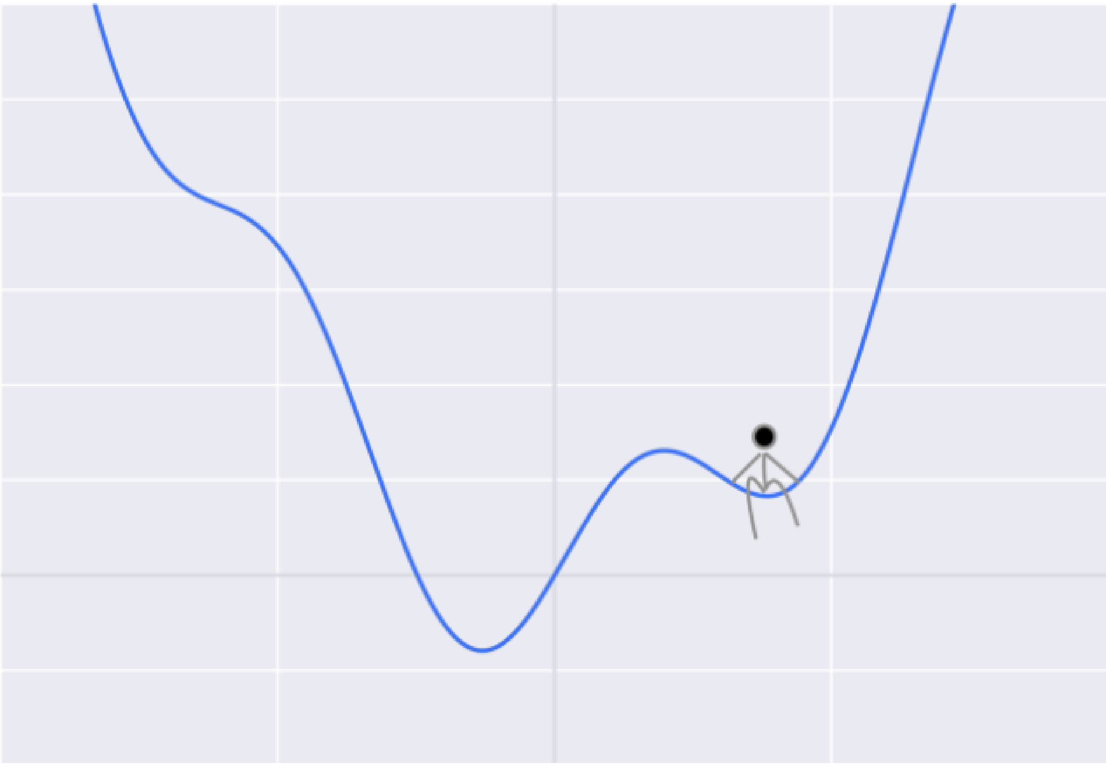
- $para_{t+1} = para_t - \alpha\partial(para)$

Learning

- Initialize parameters
- Loop n times:
  - Forward pass
  - Compute Cost (Cross entropy loss)
  - Back propagation
  - Update parameters

Loss function is not convex!

- There is a possibility of getting trapped in local minima
- Mainly on low-dimension feature space



Bob chillin at a local optima

https://hackernoon.com/life-is-gradient-descent-880c60ac1be8

Multi-class classification

- One-hot encoding:

Python:
- OneHotEncoder(categories='auto', sparse=False)
- onehotencoder.fit_transform(Y)

- Changing the dimension of the output layer

- Softmax:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

Exercise

References

- Introduction to Deep Learning, National Research University Higher School of Economics
- Andrew Ng, Neural Networks and Deep Learning, Stanford University
- Sanjoy Dasgupta, Machine Learning Fundamentals, UC San Diego
- https://playground.tensorflow.org
- Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar, Foundations of Machine Learning, second edition, The MIT Press
- Andrew Ng, Machine Learning Yearning, deeplearning.ai