Artificial Neural Networks

Lecture 13 – HCCDA-AI

Imran Nawar

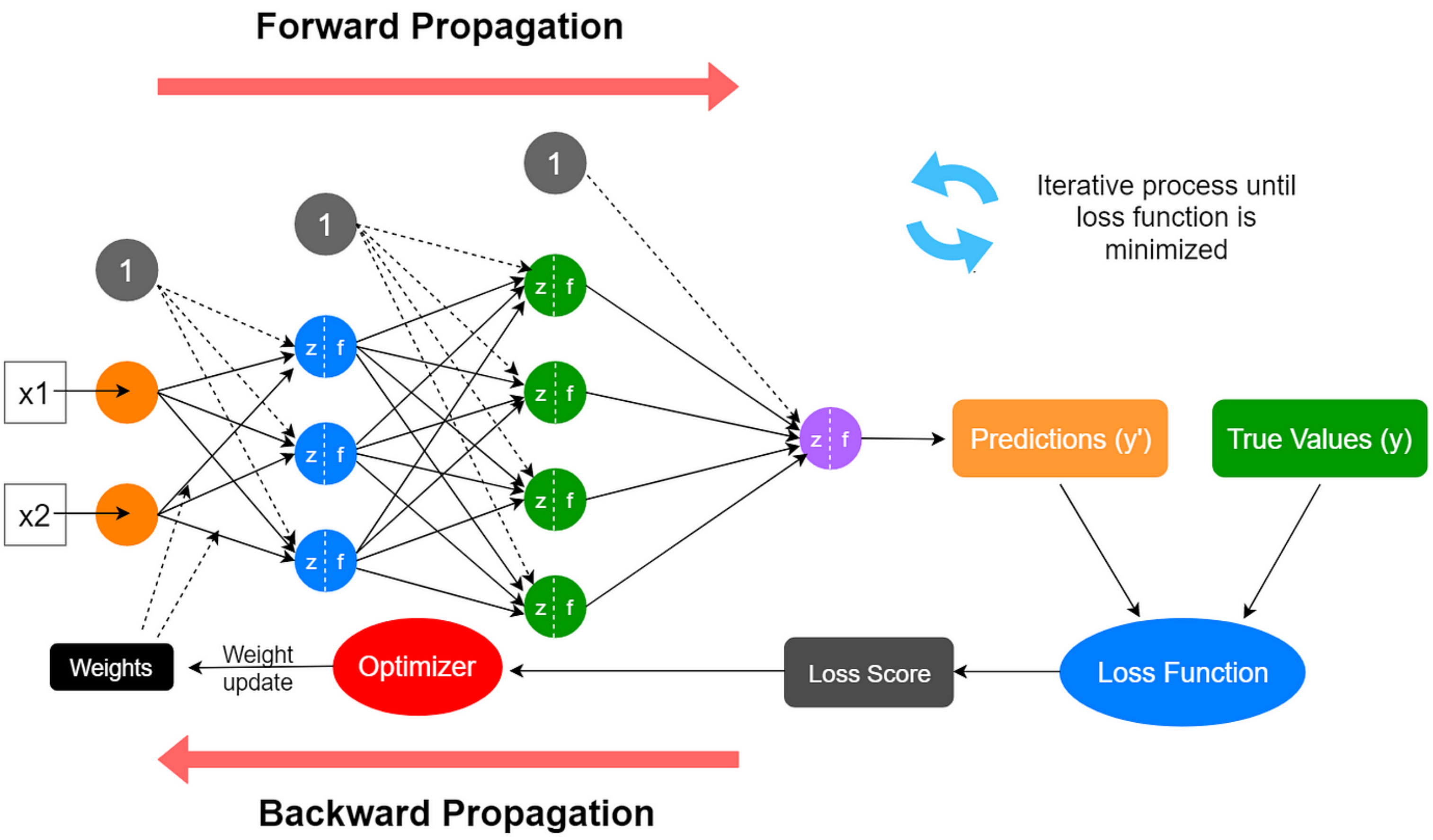
Neural Networks Training Process – Overview

- To train a neural network, we optimize its parameters using backpropagation and gradient descent.

Steps:

1. **Initialize weights & biases**
2. **Forward pass** (compute predictions)
3. **Compute loss** – Measure how far predictions are from actual values.
4. **Backpropagation** – Calculate gradients to adjust parameters.
5. **Update weights** using gradient descent
6. **Repeat** until convergence
7. **Evaluate** and adjust hyperparameters

Neural Networks Training – Visual Summary

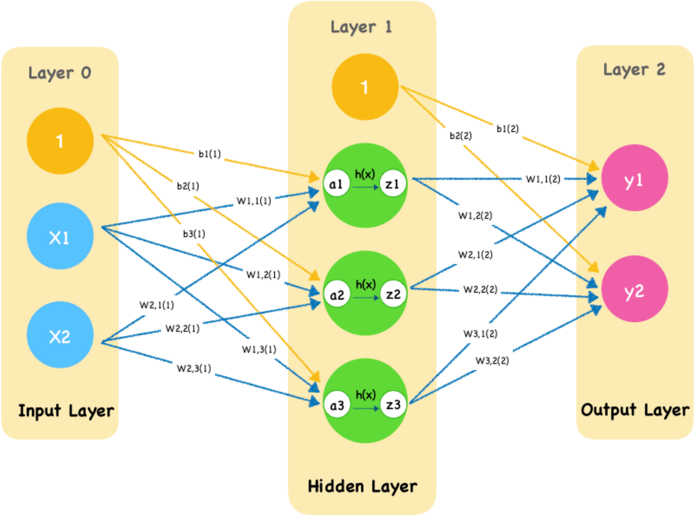


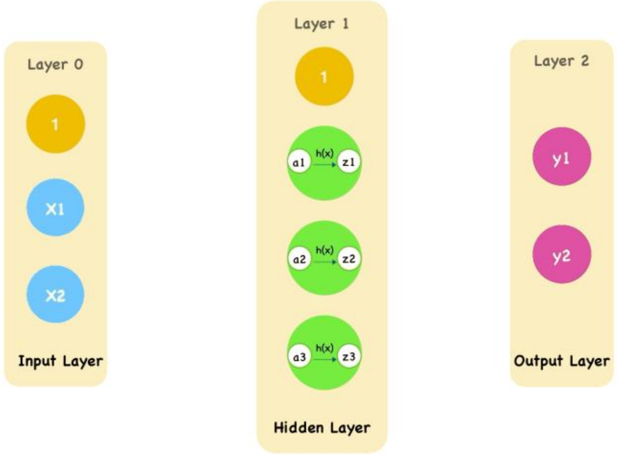
Forward Propagation

- Forward propagation refers to the process of feeding input through the network to generate predictions.

Steps:

1) Input data is provided to the input layer.
2) Weighted sum of inputs is calculated at each neuron.
3) Activation function is applied.
4) The output of each layer serves as input to the next layer.
5) The final layer produces predictions.



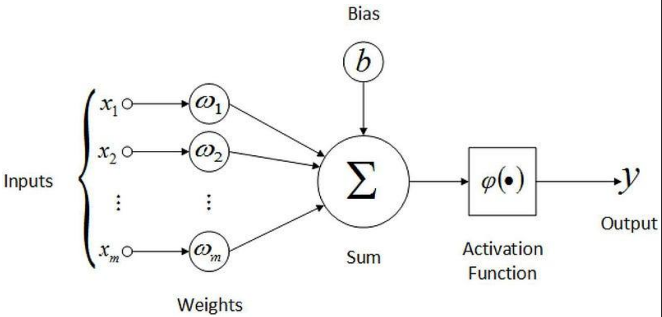


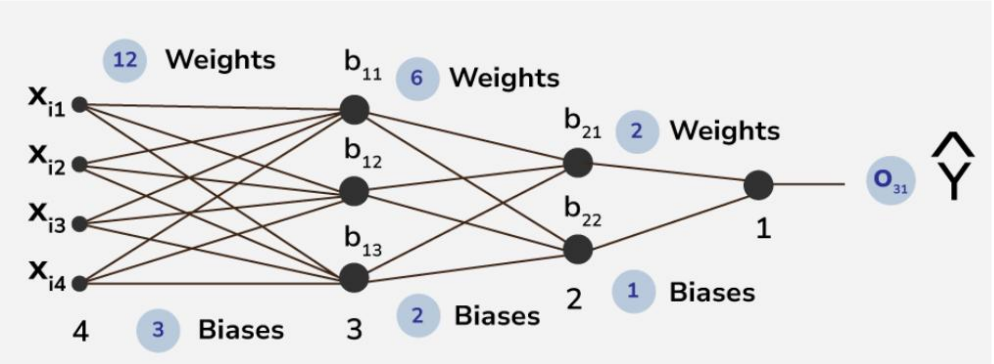
Key Steps for Forward Propagation

1) **Initialize weights and biases** (small random values)
2) **Compute weighted sum** (multiply inputs with weights and add biases.)

$$z = w.x + b$$

3) **Apply activation function** (pass the result through an activation function (Sigmoid, ReLU, etc.,).)
4) Pass output to next layer
5) Output layer generates final prediction.





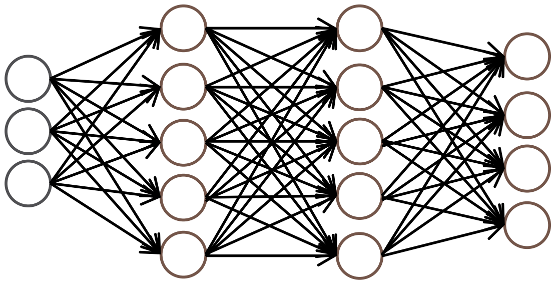
Python Code: Forward Propagation (Simple Example)

```
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Input values
x = np.array([0.5, 0.8])

# Initialize weights and bias
w = np.array([0.2, 0.4])
b = 0.1

# Compute weighted sum
z = np.dot(x, w) + b

# Apply activation function
output = sigmoid(z)
print("Final Output:", output)
```

Neural Networks: Learning

Cost Function

Neural Network (Classification)



$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

Binary classification

$y = 0$ or 1

1 output unit

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Neural Networks: Learning

Backpropagation algorithm

Backpropagation – Intuition

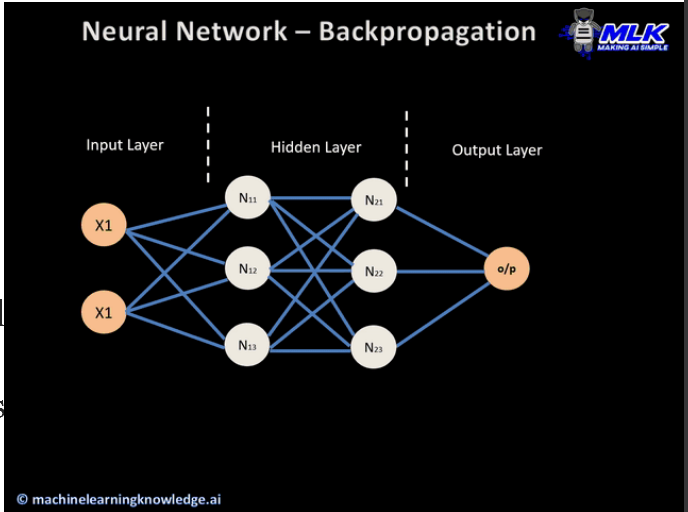
Backpropagation is an algorithm used to minimize the cost function by updating weights based on the error obtained at the output layer.

Key Steps:

- Perform forward pass
- Compute error/loss
- Compute gradients of loss with respect to weights using chain rule.
- Propagate gradients backward layer by layer
- Update weights using gradient descent
- Repeat

Important Notes:

- Backpropagation adjusts weights using partial derivatives.
- It enables efficient training by distributing errors backward.



Backpropagation Algorithm

An algorithm for trying to minimize the cost function.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

$$\min_{\Theta} J(\Theta)$$

Need code to compute:

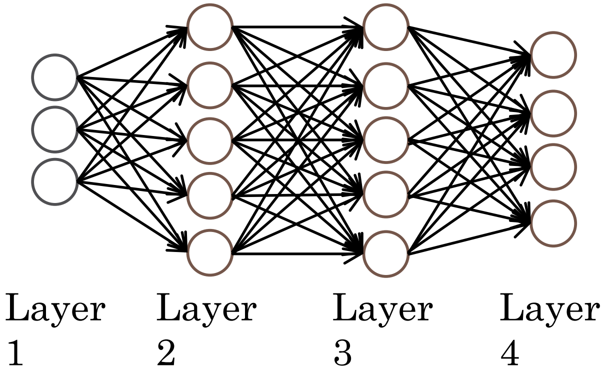
- $J(\Theta)$
- $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$

Gradient Computation

Given one training example (x, y):

Forward propagation:

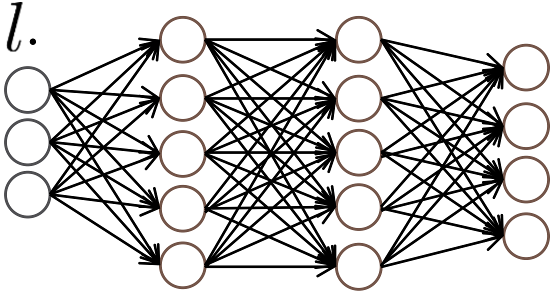
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



Gradient computation: Backpropagation algorithm

In order to compute the derivative we are going to use an algorithm called back propagation.

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Backpropagation: The process of Error Correction

- The term "backpropagation" comes from the method of computing the error at the output layer and propagating it backward.

- It follows a layer-wise approach, computing delta (δ) values for each layer:
  - Compute $\delta^{(4)}$ for the output layer.
  - Compute $\delta^{(3)}$ for the third hidden layer.
  - Compute $\delta^{(2)}$ for the second layer.

Mathematical Representation:

$$\frac{\partial}{\partial \theta_{ij}{}^{(l)}} J(\theta) = a_j^{(l)} \delta^{(l+1)} \qquad \textit{Ignoring } \lambda \textit{ if } \lambda = 0$$

- Using backpropagation and computing δ terms allows quick calculation of partial derivatives for all parameters.
- The backpropagation algorithm effectively trains a neural network using the chain rule.
- Each forward pass is followed by a backward pass to update weights and biases.
- This fine-tuning helps in reducing the error rate after each training iteration.

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

We can also vectorize this

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)}$ if $j = 0$

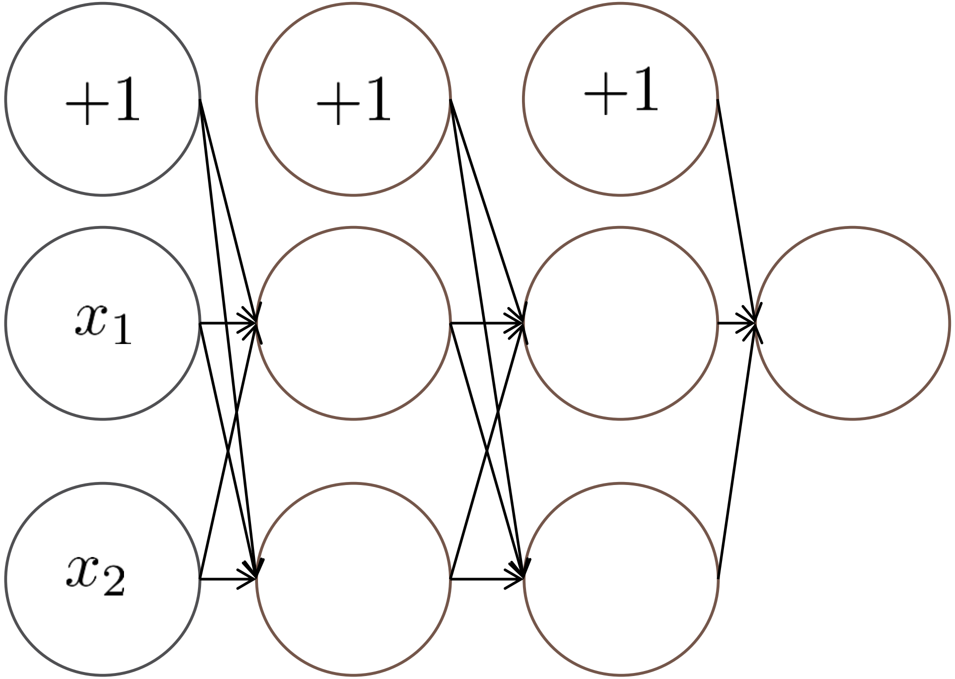
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Andrew N

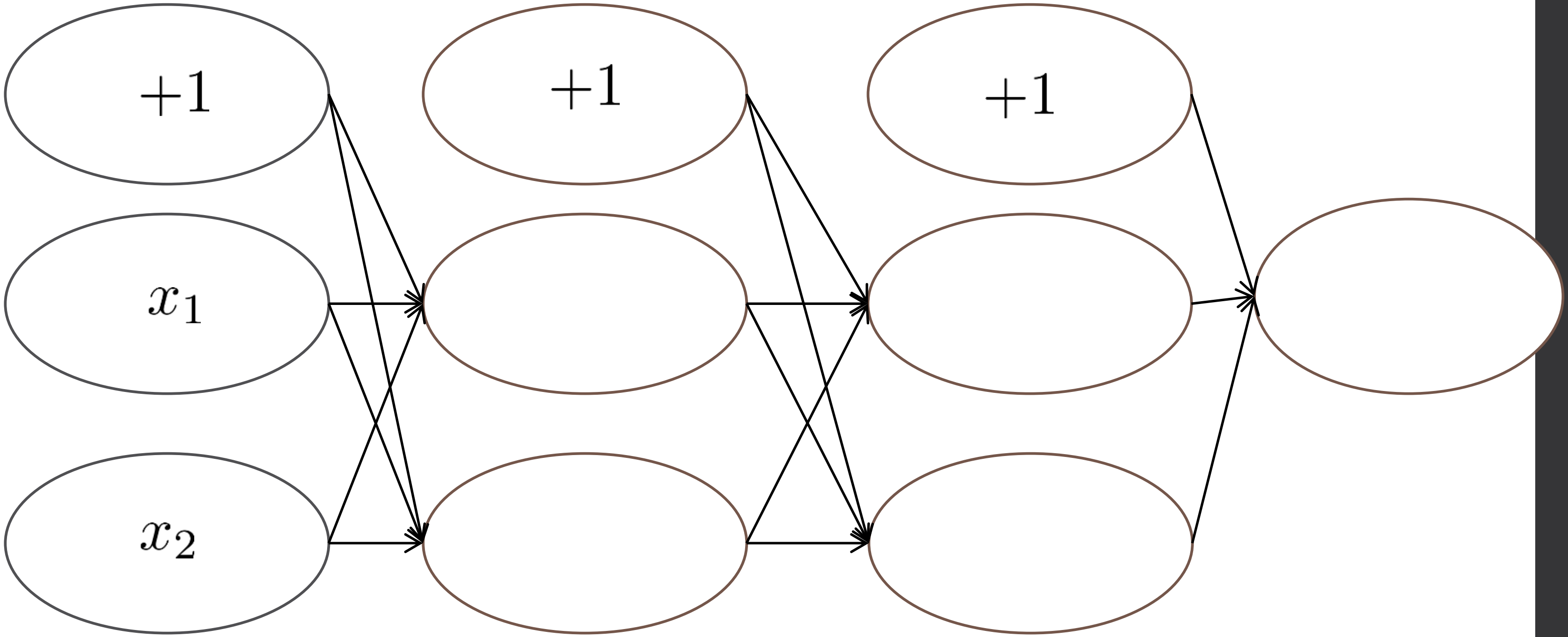
Neural Networks: Learning

Backpropagation intuition

Forward Propagation



Forward Propagation



$$z_1^{(3)} = \theta_{10}^{(2)} + \theta_{11}^{(2)}.a_1^{(2)} + \theta_{12}^{(2)}.a_1^{(2)}$$

What is backpropagation doing?

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1-y^{(i)}) \log(1-(h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1-y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

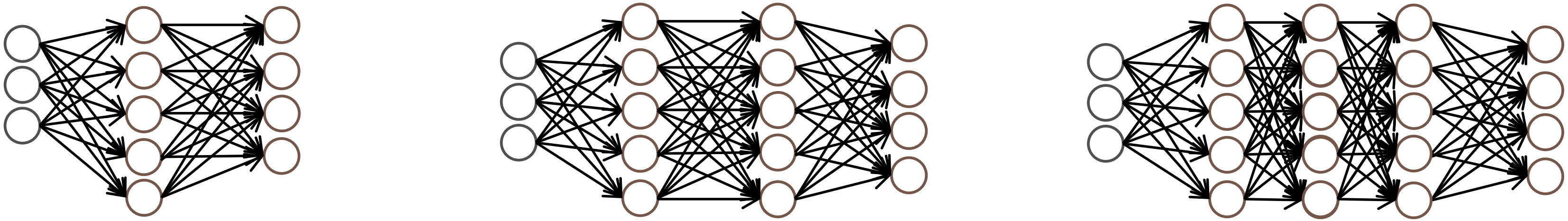
i.e. how well is the network doing on example i?

Neural Networks: Learning

Putting it together

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



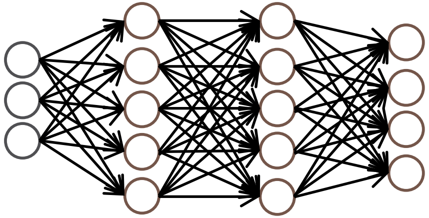
No. of input units: Dimension of features $x^{(i)}$
No. output units: Number of classes
Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

```
for i = 1:m
```

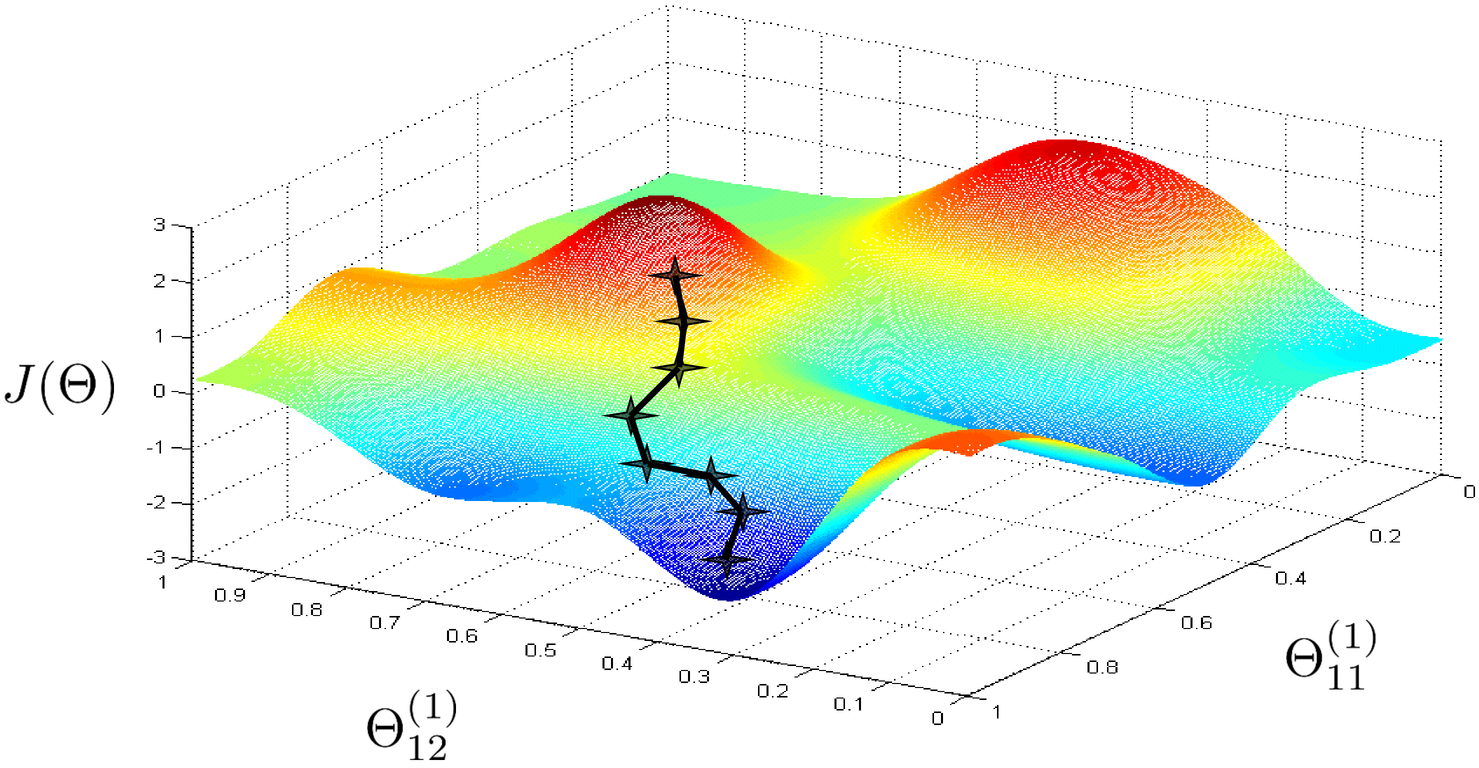
Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).



Training a neural network

5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
   Then disable gradient checking code.
6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

For neural network the cost function $J(\Theta)$ is non-convex.


$J(\Theta)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
$\Theta_{12}^{(1)}$
0
0.2
0.4
0.6
0.8
1
$\Theta_{11}^{(1)}$

Complete Learning Algorithm (Step-by-Step)

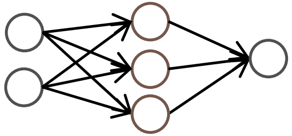
1) **Initialize Parameters** (Assign small random values to weights and biases)
2) **Feed Input Data** (*X*)
3) Perform forward propagation.
4) **Compute the loss** (Compare Predicted vs Actual Output)
5) **Apply Backpropagation**
6) **Update Parameters** (weights and biases) using gradient descent.
7) **Repeat for all epochs**
8) **Evaluate Model** – Test on unseen data.
9) Tune hyperparameters (e.g., learning rate, batch size, layers)

Neural Networks: Learning

Neural Networks and Overfitting

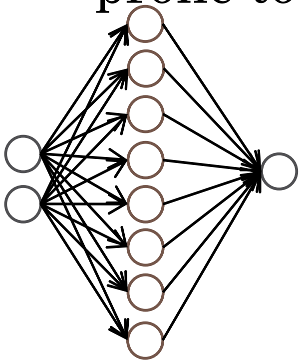
Neural networks and overfitting

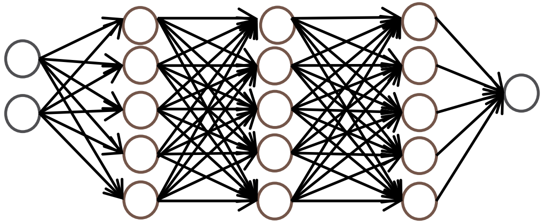
"Small" neural network
(fewer parameters; more prone to underfitting)



Computationally cheaper

"Large" neural network
(more parameters; more prone to overfitting)



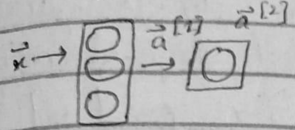


Computationally more expensive.

Use regularization (λ) to address overfitting.

TensorFlow

Building a neural network architecture




```
layer_1 = Dense(units=3, activation='sigmoid')
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])
x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

4x2

		y
200	17	1
120	5	0
425	20	0
212	18	1

```
y = np.array([1, 0, 0, 1])
model.compile(...)
model.fit(x, y)
```

targets

→ Sequential : Create a neural network ~~fo~~ by sequentially shrink together layer_1 and layer_2.

→ model.fit(x, y): tells tensorflow to take this neural network and train it on the data x and y.

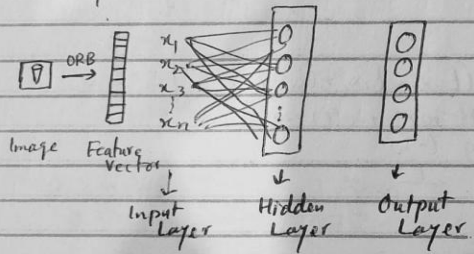
Task

Assignment :

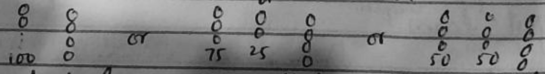
→ Take a dataset consisting of four classes (for practice you can also take more classes)

↳ Extract features using feature descriptor.

↳ Feed the extracted feature vectors to Artificial Neural Network (desinging based on your intuition).



→ For middle Layer take 50 or 100 nodes and then increase number of nodes and make different variations to see different results.



↳ For output layer there'll be four nodes (no. of classes), and use softmax.

↳ Make Dense Layer, compile and fit.

↳ Use Optimization functions...

↳ Find accuracy, fine tune....

Thank You