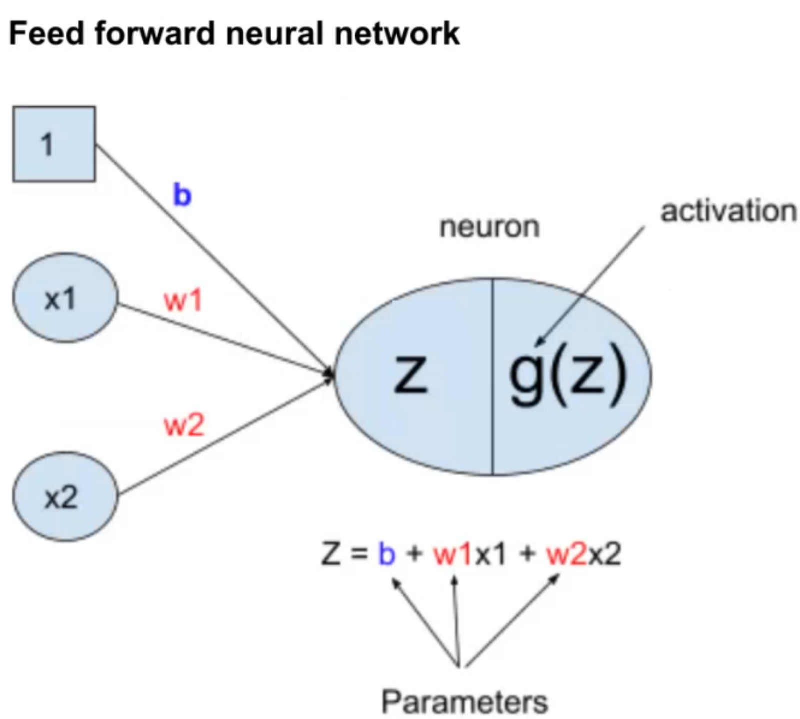
**CS 419M Introduction to Machine Learning** **Spring 2021-22**

# Lecture 17: Feed Forward Neural networks, CNN and their Training

*Lecturer: Abir De* *Scribe: Group 1*

## 17.1 Feed Forward Neural Network

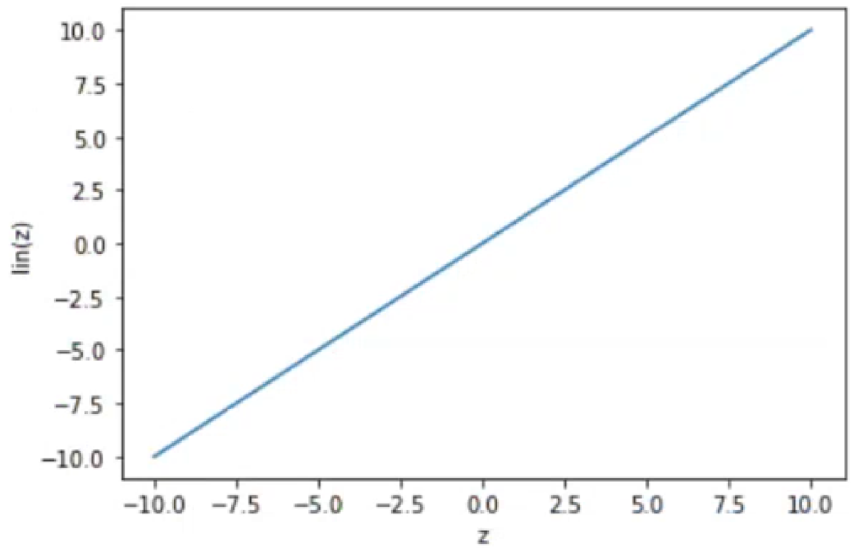
### 17.1.1 Neuron



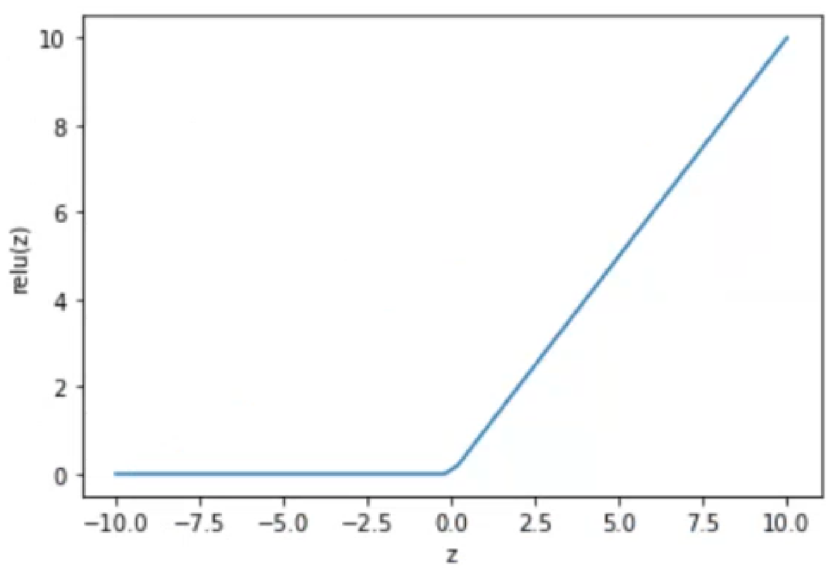

- Neuron is the building block of the neural network, Logically it performs two steps, linear combination of the inputs and then non-linear activation (Note it can be linear as well, eg: last layer for linear regression).
- In the above figure, $x_1$ and $x_2$ are inputs, b is the bias, $w_1$ is the weight corresponding input $x_1$ and $w_2$ is the weight corresponding to input $x_2$.
- First we perform the linear combination to get z $= w_1x_1 + w_2x_2 + b$ next we pass this z through typically non linear activation to get the output of the neuron g(z).
- Here $w_1$, $w_2$ and b are parameters for each neuron that can be trained.

### 17.1.2 Types of activation functions used

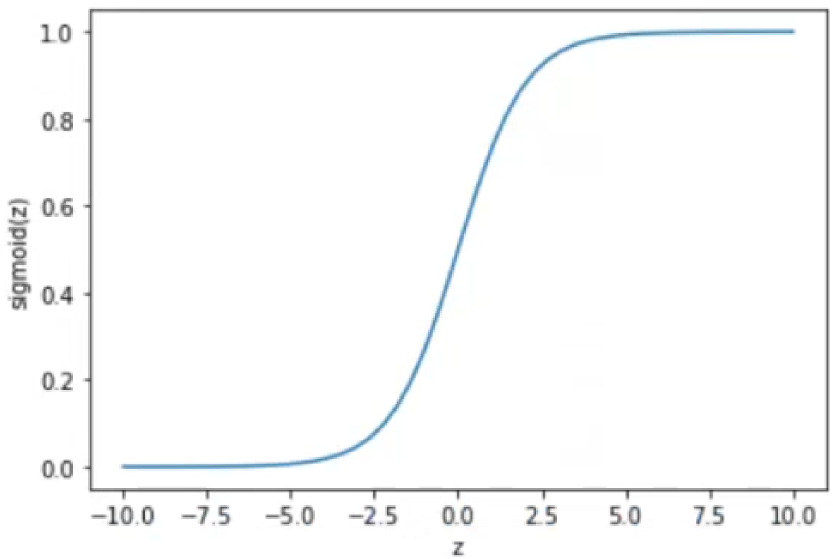
- Linear activation: g(z) = z, typically used in output layer of regression



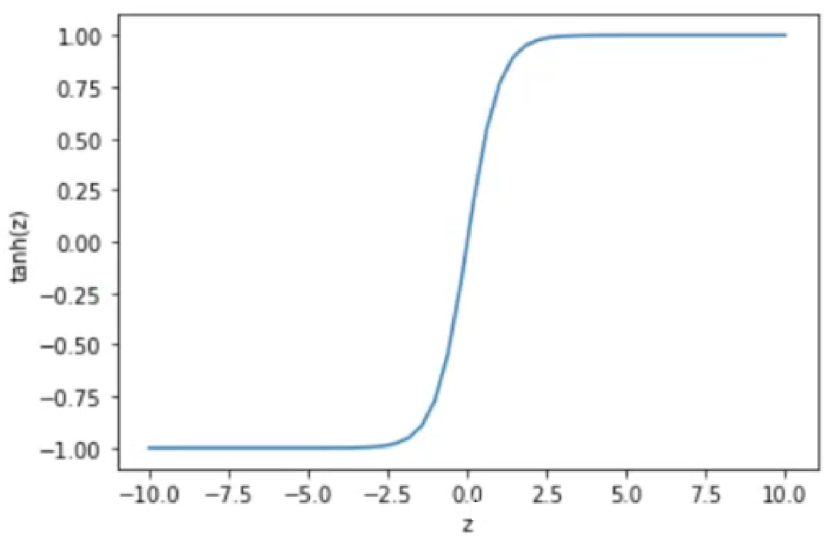

- Non-linear activation functions:
    1. relu(z) = max(0, z)



2. $\text{sigmoid(z)} = \dfrac{1}{(1 + exp(z))}$

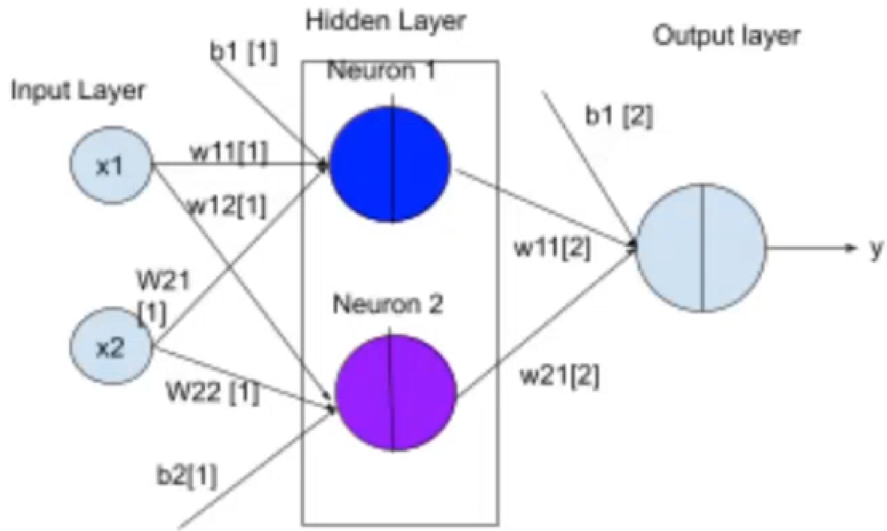



3. $\tanh(\text{z}) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$




- These are typically used activation functions, however there are other activation functions like Leaky ReLU, Softmax, Maxout, ELU etc.

### 17.1.3 Joining the neurons to form Layers




- In this network, we got two neurons in the hidden layer (any layer in between input and output layer is called hidden layer) and one neuron in the output layer.
- The number of units in the input layer is equal to the number of input features.
- In a feed forward neural network, every unit in the current layer is connected to every unit in the next layer.
- [1] stands for layer 1 and [2] stands for layer 2 and so on. Square brackets : layer number.

### 17.1.4 Activation function for each layer

1. **Input Layer**
   - Input layer doesn't contain neurons, so no activation.
2. **Hidden Layer**
   - ReLU (typically)
3. **Output Layer**
   (a) **Single neuron**
   - Regression: Linear activation
   - Binary Classification: Sigmoid Activation: Predicts Pr[y=1/x]

   (b) **K neurons**

- Regression: Linear activation
- Multi-class classification: Softmax activation

**Question:** Why can't we use Sigmoid for hidden layer?
Ans: When we do back propagation for bigger neural networks, the gradient becomes smaller and smaller from each layer when we use sigmoid activation. This is called Vanishing gradient problem.

### 17.1.5 Evaluation of the above NN

1. **Classification**
   - Output y = sigmoid($b_1[2] + w_{11}[2]g_{z1} + w_{21}[2]g_{z2}$).
   - Here $g_z 1$ is the output of Neuron 1 and $g_z 2$ is the output of the Neuron 2 of the hidden layer.
   - $g_{z1} = relu(b_1[1] + w_{11}[2]x_1 + w_{21}[1]x_2)$
   - $g_{z2} = relu(b_2[1] + w_{12}[2]x_1 + w_{22}[1]x_2)$
2. **Regression**
   - Output y = linear($b_1[2] + w_{11}[2]g_{z1} + w_{21}[2]g_{z2}$).
   - $g_{z1} = relu(b_1[1] + w_{11}[2]x_1 + w_{21}[1]x_2)$
   - $g_{z2} = relu(b_2[1] + w_{12}[2]x_1 + w_{22}[1]x_2)$

**Question:** RELU is not differentiable at x=0 right?
Ans: Correct, we usually take left derivative or right derivative at that point.

### 17.1.6 Matrix - Vector Form

1. **Neuron Level Vectorization**
   - x1 and x2 are not vectors, they are scalars
   - $\mathbf{x}$ = [1 x1 x2], $\mathbf{w}$ = [b w1 w2]
   - $g_{z1} = relu(\mathbf{w}_1[1]^T\mathbf{x})$, $g_{z2} = relu(\mathbf{w}_2[1]^T\mathbf{x})$
   - Dimension of $\mathbf{w}_1$ & $\mathbf{w}_2$ is 1x3 and dimension of $\mathbf{x}$ is 3x1
   - $y = sigmoid(\mathbf{w}_1[2]^T[1\ g_{z1}\ g_{z2}])$
2. **Layer Level Vectorization**
   - ${\mathbf{w}_1}^T = \begin{bmatrix} b_1 & w_{11} & w_{21} \\ b_2 & w_{12} & w_{22} \end{bmatrix}$

- Input matrix $\mathbf{x}^T = \begin{bmatrix} 1 & x_1^1 & x_2^1 \\ 1 & x_1^2 & x_2^2 \\ 1 & x_1^3 & x_2^3 \end{bmatrix}$
- x1 = [1 $x_1{}^1$ $x_2{}^1$] , here x1 denotes training example 1 and similarly x2 = [1 $x_1{}^2$ $x_2{}^2$] denotes training example 2

3. **Linear Combination of the First Layer**
   - $\mathbf{z}_1$=$\mathbf{w}_1{}^T\mathbf{x}$
   - Output is $\mathbf{g_{z1}}$=relu(**z1**)

### 17.1.7 Example

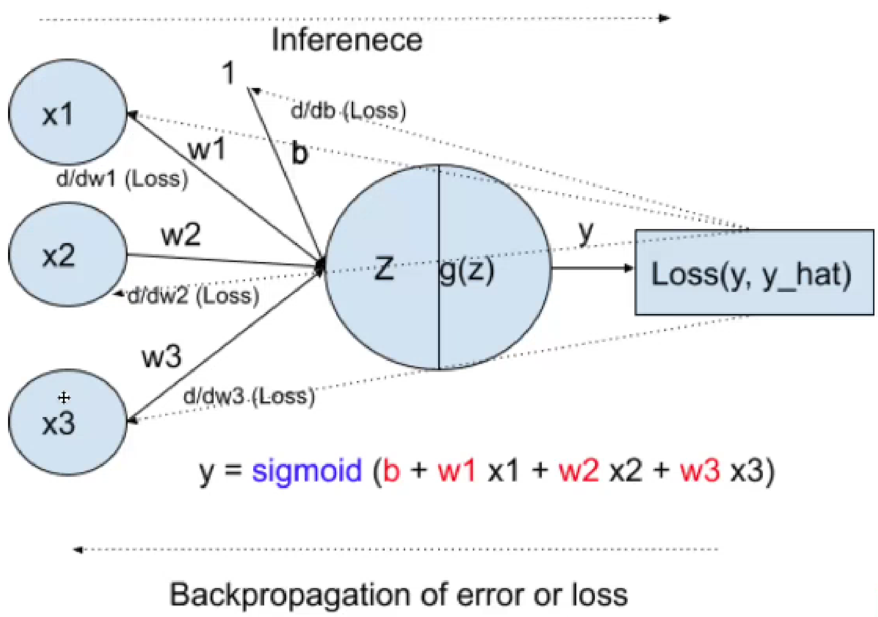
- $\mathbf{x}^T$ = [1 2 3]
- $\mathbf{w}^T = \begin{bmatrix} 0.5 & -1 & 2 \\ 0.2 & 2 & -1 \end{bmatrix}$

1. **Non Vectorised**
   - $g_z$1[1] = relu(b1[1]+w11[1]*x1+w21[1]*x2) = relu(0.5-1*2+2*3) = relu(4.5) = 4.5
   - $g_z$2[1] = relu(b2[1]+w12[1]*x1+w22[1]*x2) = relu(0.2+2*2-1*3) = relu(1.2) = 1.2
2. **Vectorised**
   - $\mathbf{z1} = \mathbf{w}^T\mathbf{x} = \begin{bmatrix} 0.5 & -1 & 2 \\ 0.2 & 2 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5*1 & -1*2 & 2*3 \\ 0.2*1 & 2*2 & -1*3 \end{bmatrix} = \begin{bmatrix} 4.5 \\ 1.2 \end{bmatrix}$
   - Output is $\mathbf{g_{z1}} = \text{relu}(\mathbf{z1}) = \begin{bmatrix} 4.5 \\ 1.2 \end{bmatrix}$

### 17.1.8 Training of feed-forward Neural Network

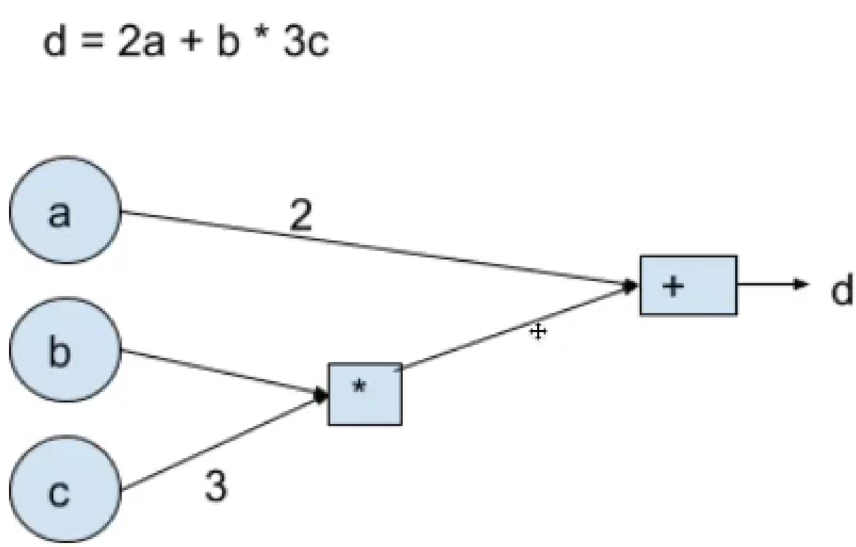
**Question:** What is the training problem here?
Ans: Estimate weights(w) such that the loss is minimised.

- Loss for Regression: **Mean-Squared Error**(prediction, actual)
- Loss for Binary Classification: **Binary Cross Entropy**(prediction, label)
- Loss for Multi-class Classification: **Categorical Cross Entropy**(prediction, label)




- Loss is not directly related to weights($w_1$,$w_2$,$w_3$) but connected through intermediary levels.
- So, $\frac{\partial Loss}{\partial w} = 0$ is not valid here. Hence, we need to do Backpropagation.

### 17.1.9 Example to understand this concept



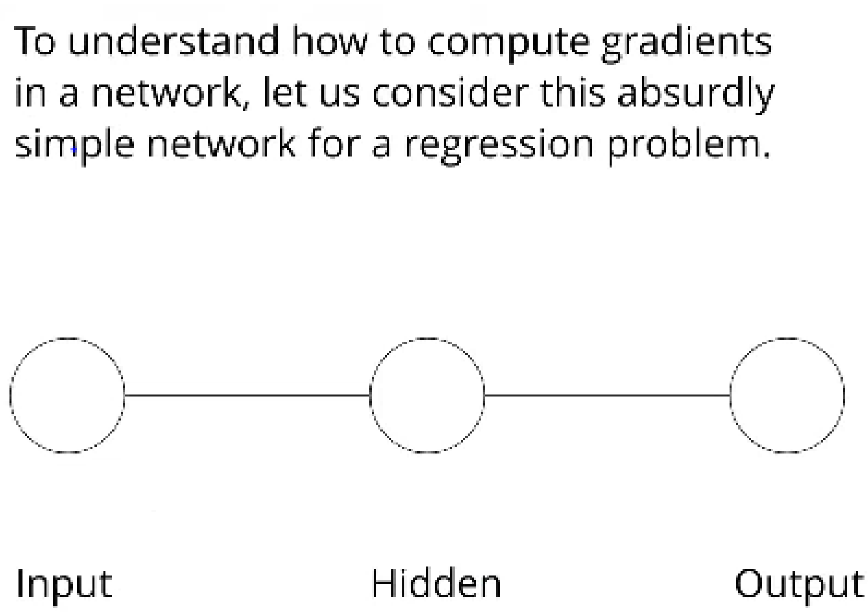

**d** is analogous to the loss and **a,b,c** are analogous to the weights of our model, which are not directly connected to the output(loss).

d = $z_1$ + $z_2$ = 2a + b*$z_3$ = 2a + b*3c

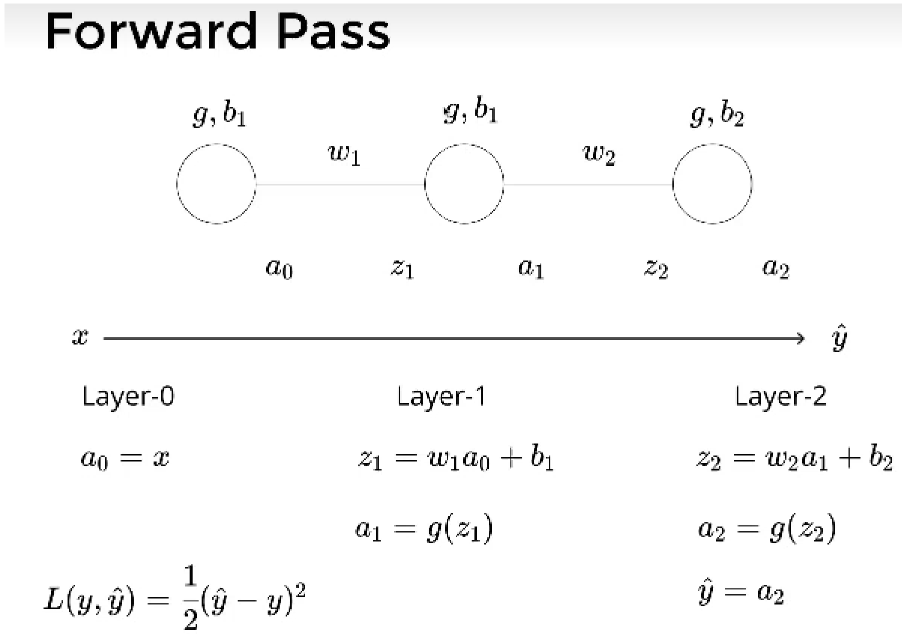
We know that **d** is connected to **a** through $z_1$.
So we apply **Chain-Rule**

$\frac{\partial d}{\partial a} = \frac{\partial d}{\partial z1} * \frac{\partial z1}{\partial a} = 2$
Similarly we can do for **b and c**.

### 17.1.10 Backpropagation




In order to compute the gradients, we first need to compute loss on a training example.




**g** is non-linear activation. **b** is the bias.**a** is the activation from previous layer. **w** is the weight. **z** is linear combination. **L** is the loss

**For Layer-2**
$\frac{\partial L}{\partial w2} = \frac{\partial L}{\partial a2} * \frac{\partial a2}{\partial z2} * \frac{\partial z2}{\partial w2}$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1} \cdot g'(z_1) \cdot a_0 \qquad \frac{\partial L}{\partial w_2} = (\hat{y} - y) \cdot g'(z_2) \cdot a_1$$

$$\frac{\partial L}{\partial a_1} = (\hat{y} - y) \cdot g'(z_2) \cdot w_2$$

**Batch Gradient Descent**

1. Randomly initialize W. [In case of neural networks, do not initialize parameters to 0 or to the same number. There are some specialized initialization of parameters for neural networks: (i) He's initialization or (ii) Xavier initialization.]

2. Repeat until convergence stop if loss is not changing
   a. for i in range(0, len(W)):
   i. w[i] (new) := w[i] (old) - alpha d/dw[i] J(W) Gradient of loss function w.r.t. W (**Comes**

**from Backpropagation**)
b. Update w simultaneously

alpha is the learning rate.

### 17.1.11 Gradient$_{(hiddenlayers)}$

## Gradients (hidden layers)

Activations -> Pre-activations-> Weights

$\boldsymbol{A}_l^{(g)}$ gradient of the loss w.r.t the activations

$\boldsymbol{Z}_l^{(g)}$ gradient of the loss w.r.t the pre-activations

$\boldsymbol{W}_l^{(g)}$ gradient of the loss w.r.t the weights

If $\boldsymbol{A}_l^{(g)}$ is already known, we can compute the rest of the gradients:

$$\boldsymbol{Z}_l^{(g)} = \boldsymbol{A}_l^{(g)} \odot g'(\boldsymbol{Z}_l)$$

$$\boldsymbol{W}_l^{(g)} = \boldsymbol{A}_{l-1}^T \boldsymbol{Z}_l^{(g)}$$

$$\boldsymbol{A}_{l-1}^{(g)} = \boldsymbol{Z}_l^{(g)} \boldsymbol{W}_l^T$$

The activation function in the final layer for regression is just the identity function. There is a slight abuse of notation: $\boldsymbol{A}_{\boldsymbol{L}}^{(g)}$ and $\boldsymbol{Z}_{\boldsymbol{L}}^{(g)}$ are vectors.

$$\boldsymbol{A}_{\boldsymbol{L}}^{(g)} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

$$\boldsymbol{Z}_{\boldsymbol{L}}^{(g)} = g'(\boldsymbol{Z}_l) \odot \boldsymbol{A}_{\boldsymbol{L}}^{(g)}$$

$$= \boldsymbol{A}_{\boldsymbol{L}}^{(g)}$$

These equations will make more sense if we compare them with their forward-pass counterparts:

$$\boldsymbol{Z}_l = \boldsymbol{A}_{l-1}\boldsymbol{W}_l + \boldsymbol{b}_l \qquad\qquad \boldsymbol{Z}_l^{(g)} = \boldsymbol{A}_l^{(g)} \odot g'(\boldsymbol{Z}_l)$$

$$\boldsymbol{A}_l = g(\boldsymbol{Z}_l) \qquad\qquad \boldsymbol{W}_l^{(g)} = \boldsymbol{A}_{l-1}^T \boldsymbol{Z}_l^{(g)}$$

$$\boldsymbol{A}_{l-1}^{(g)} = \boldsymbol{Z}_l^{(g)} \boldsymbol{W}_l{}^T$$

For a moment think of all of them as scalars. We can show that these equations are the matrix equivalents of the chain-rule.

### 17.1.12 Gradient Descent

## Gradient Descent (Neural Networks)

We now have all the ingredients to completely specify the learning algorithm.

- Let $\boldsymbol{\theta}$ refer to all the parameters in the model.
- Let us define the following functions:

$$\hat{\boldsymbol{Y}} = \text{forward-pass}(\boldsymbol{X})$$

$$L = \text{loss}(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$$

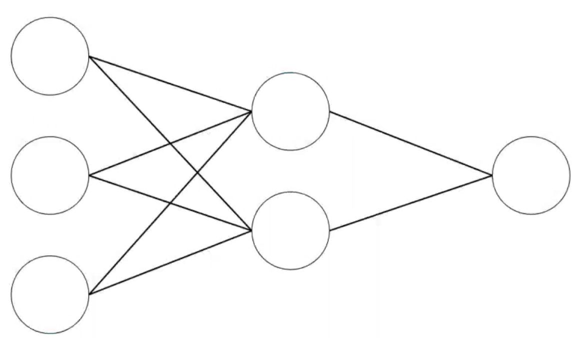
$$\boldsymbol{\theta}^{(g)} = \text{backward-pass}(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$$

Only the most important arguments are displayed here

### 17.1.13 Initialization

## Initialization

What happens if we initialize all the parameters to some constant value?



- Since the incoming weights are the same for all neurons in a layer, there is nothing to differentiate between two neurons.
- This symmetry means that they will evolve identically and will not learn different things.

Why did we not initialize all parameters to zero?

```
Initialize: θ ~ N(0,1)

for e = 1 to e = E:
```

$$\hat{\boldsymbol{Y}} = \text{forward-pass}(\boldsymbol{X})$$

$$L = \text{loss}(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$$

$$\boldsymbol{\theta}^{(g)} = \text{backward-pass}(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha\boldsymbol{\theta}^{(g)}$$

- Since the incoming weights are the same for all neurons in a layer, there is nothing to differentiate between two neurons.
- This symmetry means that they will evolve identically and will not learn different things.

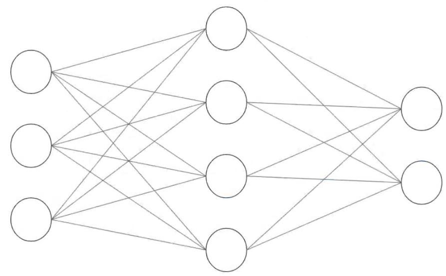
### 17.1.14 Regularization

There are two methods to regularize the model.

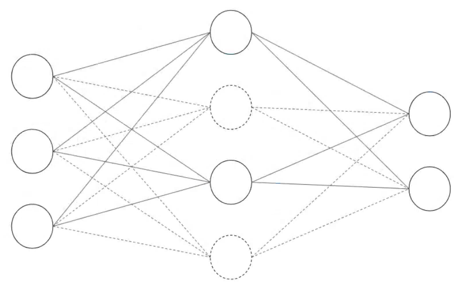
- L1/L2 Regularization
- Dropout

We are familiar with method(1)
Method(2) is particular to neural network (2)

Consider the following network with one hidden layer:



In each iteration of GD, randomly choose half the neurons in the hidden layer and "drop" them "out" of the network.



### 17.1.15 Dropout

At interference, use the full network, but halve the outgoing weights from the hidden layer. This is because only half of the neural were active during training.

Why does dropout work ?

- During the training phase, dropout can be seen as producing multiple networks, each having a different number of neurons in the hidden layers.
- At test time, we can think of the output of a network as averaging over the results of all these networks.

### 17.1.16 Transfer Learning

Transfer learning (TL) is a research problem in machine learning (ML) that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks.

## 17.2 Group Details and Individual Contribution

| Name | Roll Number | Sections |
|---|---|---|
| Garaga V V S Krishna Vamsi | 180070020 | 17.1.1, 17.1.2, 17.1.3, 17.1.4, 17.1.5 |
| Shrey Ganatra | 20D070074 | 17.1.8,17.1.9,17.1.10 |
| Abhinav Singh | 19D180002 | 17.1.6, 17.1.7 |
| Sristy Kushwaha | 180110088 | |
| Tejas Pravin Amritkar | 20D070081 | 17.1.11, 17.1.12, 17.1.13, 17.1.14, 17.1.15, 17.1.16 |