COMPREHENSIVE DEEP LEARNING NOTES

INDEX

ANN

Exploring neural networks and backpropagation.

RNN

Modeling sequential data with RNNs, LSTMs.

TRANSFORMER

Transforming NLP with attention mechanisms.

CNN

Revolutionizing image processing and recognition.

ENCODER DECODER

Sequence-to-sequence learning and translation.

Deep Learning

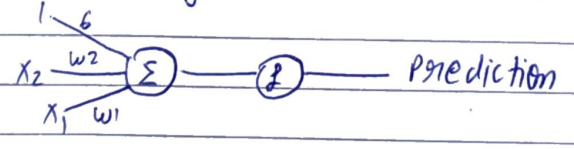
① Deep Learning is a subfield of AI and ML that is inspired by the structure of a human brain

② DL algorithm uses multiple layers to progressively extract high level feature from the raw input

③ DL vs ML
- (i) Data Dependency - DL is more Data Hungry
- (ii) Hardware Dependency - DL use GPU, ML use CPU
- (iii) Training Time - DL training time is high
- (iv) Feature selection - DL does not required pre build features
- (v) Interpretability - DL models are not interpretable

④ Type
- (i) MLP (Multi layer Perceptron)
- (ii) CNN (Convolutional Neural Network)
- (iii) RNN (Recurrent Neural Network)
- (iv) Auto encoders
- (v) GAN (Generative Adverserial Network)

⑤ Perceptron
- (i) It is a algorithm used for supervised machine learning problem
- (ii) we can also say it is a mathematical model / function
- (iii)



x_1, x_2 = Input

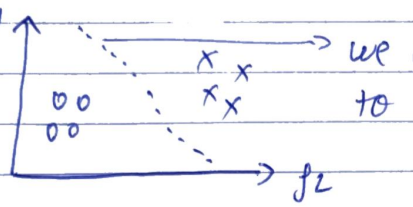
w_1, w_2, b = weigh and bias

$\Sigma = w_1x_1 + w_2x_2 + b$

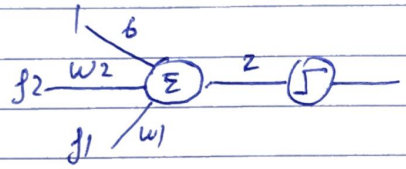
f = activation function = Scale down $w_1x_1 + w_2x_2 + b$ in a range of values

(iv) Perceptron Trick

1. A Algo used to train perceptron
2. Lets suppose we have a data with 2 input features $f1$, $f2$ and one output feature ($f3$) = categorical column, This is a Binary classification problem



→ we want to use perceptron trick to make this classification line



→ Here we are using Σ (step function) as activation function

$\Sigma = f1\,w1 + f2\,w2 + b$,
we need to Find the value of $w1$, $w2$, and b

Step function (Σ) = $\begin{cases} 1 & x \geq 0 \\ 0 & < 0 \end{cases}$

3. Algo

3.1 Guesses a random value of $w1$, $w2$, b

3.2 Question Random Point - (that the Point correctly positioned or not)

3.2.1 If (yes) correctly positioned → Do nothing to line, ~~Question another Random Point~~

3.2.2 If (No) - make changes in bias and weight so that so that now line is correctly positioned for that point

loop

3.3 Do this in a loop for 1000 or 10000 Epoch

Note :- Transformation maths

① Example — (cross OR ball)

P1 → wrongly classified Point (4, 5)
Line = $2x + 3y + 5 = 0$

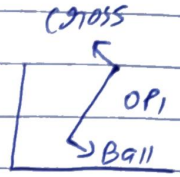
Step1 - Add 1 at the end of the Point = (4, 5, 1)
Step2 - Let Learning rate = 1, ⇒ (4 * 1), (5 * 1), (1 * 1)
Step3 - if (we want to move the Point in (-)ve region) line weight - Points
if (we want to move the Point in (+)ve region) line weight + Points

It can also be writen as = $w_n = w_0 + \eta(y_i - \hat{y}_i)x_i$

2	3	5
-4	5	1
-2	-2	4

w_n = new weights of line
w_0 = current weight + bais of line
x_i = coordinates of that Points +1

here is new line = $-2x - 2y + 4 = 0$



(v) Perceptron Loss Function

1. Problem with Perceptron trick

(a) It is possible the model never converge

(b) Perceptron trick does not use loss function, so we are not sure about our model accuracy.

2. Loss Function

$$L = \frac{1}{n}\sum_{i=1}^{n} \max(0, -y_i f(x_i))$$

n = number of rows

$f(x_i) = w_1x_1 + w_2x_2 + b$

3. We use gradient Decent

$$L = \frac{1}{n}\sum_{i=1}^{n} \max(0 - y_i(w_1 x_{i1} + w_2 x_{i2} + b)$$

$$w_1 = w_1 - \eta\frac{\partial L}{\partial w_1} \qquad w_2 = w_2 - \eta\frac{\partial L}{\partial w_2} \qquad b = b - \eta\frac{\partial L}{\partial b}$$

$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f(x_i)} \times \frac{\partial f(x_i)}{\partial w_1}$, Similarly for $\frac{\partial L}{\partial w_2}$, $\frac{\partial L}{\partial b}$

$$\frac{\partial L}{\partial f(x_i)} = \begin{cases} 0 & \text{if } y_i f(x_i) \geq 0 \\ -y_i & \text{if } y_i f(x_i) < 0 \end{cases}, \quad \frac{\partial f(x_i)}{\partial w_1} = x_{i1}$$

$$\frac{\partial L}{\partial w_1} = \begin{cases} 0 & \text{if } y_i f(x_i) \geq 0 \\ -y_i x_{i1} & \text{if } y_i f(x_i) < 0 \end{cases}$$

$$\frac{\partial L}{\partial w_2} = \begin{cases} 0 & \text{if } y_i f(x_i) \geq 0 \\ -y_i x_{i2} & \text{if } y_i f(x_i) < 0 \end{cases}$$

$$\frac{\partial L}{\partial b} = \begin{cases} 0 & \text{if } y_i f(x_i) \geq 0 \\ -y_i & \text{if } y_i f(x_i) < 0 \end{cases}$$

3. Perceptron is very flexible

Loss function	Activation function	output
① Hinge loss	step function	Perceptron / binary classifier
② log loss (Binary cross entropy)	sigmoid function	logistic regression
③ categorical cross entropy (Sparse)	softmax	multi class classification ~~softmax regression~~
④ Mean Sauare error	linear	linear regression

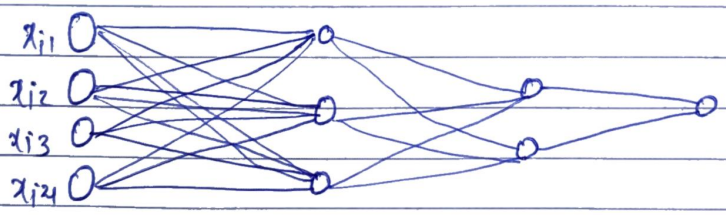
4. Why we need multi layer Perceptron and Issue with Perceptron

4.1 Perceptron does not work for Non-linear Data, thats why we use multi layer perceptron

5) Multi layer Perceptron

6.1 Finding the number of Trainble Parameter in a Nueral network

(a) Trainable Parameter = weights and bias



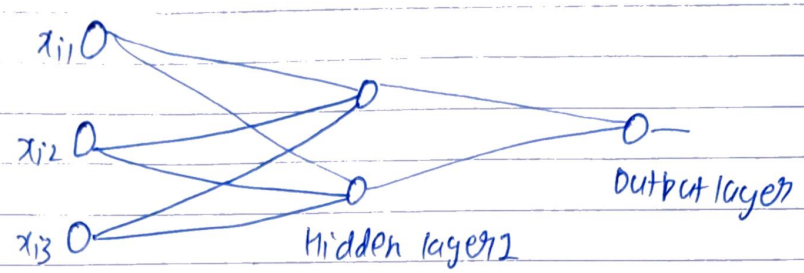
Blw Input layer and Hidden layer1 = number of weights = $4 \times 3 = 12$
number of bias = 3 (3 nodes in layer 1)

Blw Hidden layer 1 and 2 = number of weights = $3 \times 2 = 6$
number of Bias = 2

Blw hidden layer 2 and olp layer = number of weights = $2 \times 1 = 2$
number of Bias = 1

Total Trainble Parameter = $12 + 3 + 6 + 2 + 2 + 1 = 26$

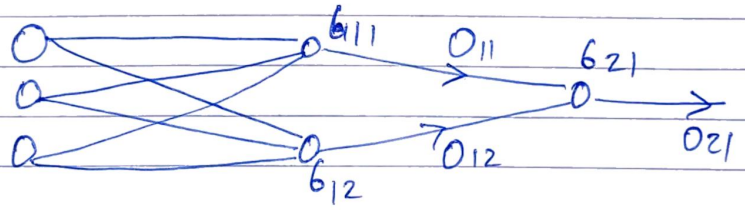
6.2 Notation of weights and Bias and output

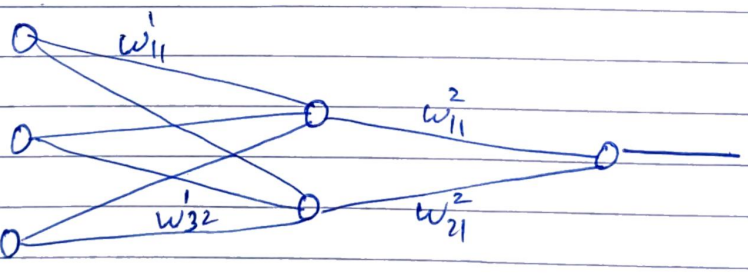


Bias = b_{ij} = i = layer number
j = node number

output = O_{ij} = i = layer number
j = node number

weight = w_{ij}^{k} = i = input layer, node number
k = receiving layer number
j = receiving layer, node number





6.3 Geometric Intuition of Perceptron

(a) without weight = P1 ∟ , P2 ∟ + = ⊥ = ∟ → we are able to capture non linearity

(b) with weight

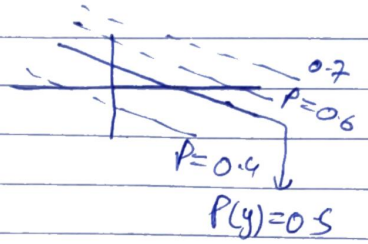
P1 ∟ × w = 10

P2 ∟ × w = 5 + = ∟ = Here P1 shape more Dominating.

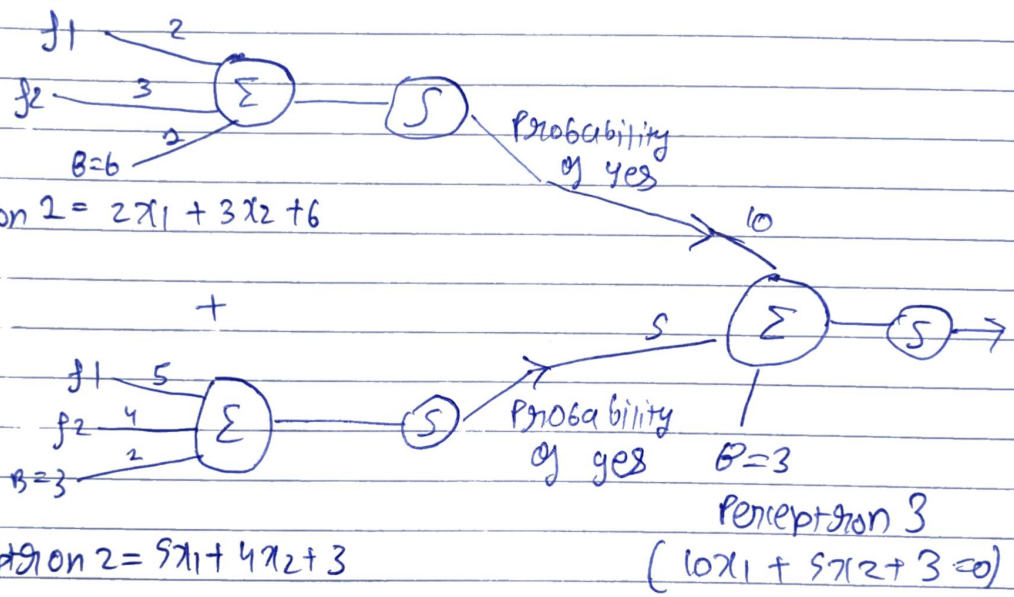
6.4 Math Intuition

(a) Pre Requisite

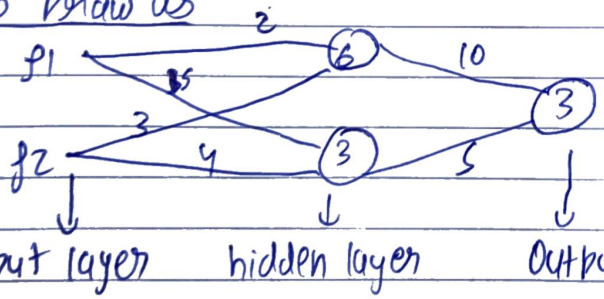
- Here we are using Sigmoid as Activation Function
- Log Loss as Loss function
- Sigmoid → return Probability = $\frac{1}{1+e^{-z}}$
- Probability at the line = 0.5
  P(Probability of yes) = 0.5



(b) working



Also Draw as

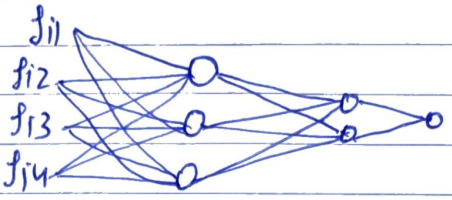


6.5 How we can change architecture of nueral network

1. Increase number of nodes in Hidden layer
2. Increase hidden layers
3. Increase input layer nodes
4. Increase output layer nodes (used for multi class classification problem)

6.6 Forword Propagation

(a) Let suppose we have a data with 4 input features and 1 output feature
(b) Let suppose we have a nueral network like this



Total Tainble feature = $4\times3+3+3\times2+2+2\times1+1 = 26$

(c) Prediction = $\sigma(W^T X + B)$

Here σ = Activation Function
W = weight metrics
W^T = Transpose of weight metrics
X = value of features
B = value of Bias

(d) Example - For layer 1

$$\sigma\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}^T \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ [illegible] \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix} \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ x_{i4} \end{bmatrix} + \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ [illegible] \end{bmatrix}\right)$$

$$\Rightarrow \sigma\begin{bmatrix} w_{11}x_{i1} + w_{21}x_{i2} + w_{31}x_{i3} + w_{41}x_{i4} + b_{11} \\ w_{12}x_{i1} + w_{22}x_{i2} + w_{32}x_{i3} + w_{42}x_{i4} + b_{12} \\ w_{13}x_{i1} + w_{23}x_{i2} + w_{33}x_{i3} + w_{43}x_{i4} + b_{13} \end{bmatrix} = \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix}$$

For layer 2

$$\sigma\left(\begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \\ w^2_{31} & w^2_{32} \end{bmatrix}^T \begin{bmatrix} O_{11} \\ O_{12} \\ O_{13} \end{bmatrix} + \begin{bmatrix} b_{21} \\ b_{22} \end{bmatrix}\right) = \sigma\begin{bmatrix} w^2_{11}O_{11} + w^2_{21}O_{12} + w^2_{31}O_{13} + b_{21} \\ w^2_{12}O_{11} + w^2_{22}O_{12} + w^2_{32}O_{13} + b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix}$$

For layer 3

$$\sigma\left(\begin{bmatrix} w^3_{11} \\ w^3_{21} \end{bmatrix}\begin{bmatrix} O_{21} \\ O_{22} \end{bmatrix} + [b_{31}]\right) = \sigma\left(\left[w^3_{11}O_{21} + w^3_{21}O_{22} + b_{31}\right]\right) = O_{31} = \hat{y}$$

6.7 Loss Function in Deep learning

(a) Regression = MSE, MAE, huber loss

~~(b) categorical~~

(B) classification = Binary cross entropy, hinge loss, categorical cross entropy

(c) GAN = Discriminatry loss, Min max gan loss

(d) Autoencoder = KL divergence

(e) Object Detection = Focal loss

(f) Embedding = triplet loss

(i) MSE (mean Saucare error)

(a) Loss Function = $(y_i - \hat{y}_i)^2$

(b) cost Function = $1/n \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

(c) Output layer Activation Function = linear

(d) Advantages → ① easy to understant ② 1 minima ③ Differentiable

(e) Disadvantages → ① Not robust to outliers

(ii) MAE (mean Absolute Error)

(a) Loss Function = $|y - \hat{y}|$

(b) cost Function = $\frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$

(c) Advantages → ① Robust to outliers ② Error unit is same as output

(d) Disadvantages → Not differentiable

(iii) Haber loss

(a) Loss Function = $\begin{cases} 1/2 (y - \hat{y})^2 & \text{for } |y - \hat{y}| \le \delta \\ \delta |y - \hat{y}| - 1/2 \delta^2 & \text{otherwise} \end{cases}$

δ = Hyper Parameter

(b) works Good if data has to much outliers

(iv) Binary Cross Entropy (log loss)

(a) used for binary class classification problem

(b) $L = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$

(c) Activation function for output layer = Sigmoid

(v) Categorical Cross Entropy

(a) used for multi class classification problem

(b) $L = -\sum_{j=1}^{K} y_j \log(\hat{y}_j)$

K = number of classes in data

(c) Cost Function = $\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log(\hat{y}_{ij})$

(d) Activation Function for Output layer = Softmax Regressor

(vi) Sparse categorical Cross Entropy

(a) used for multi class classification problem

(b) Faster then categorical cross entropy, Almost similar to categorical cross entropy

6.8 Back Propagation

(a) Algo use to train nueral network using gradient decent

Note → Back Propagation algo use memorization

(b) Algo :-

```
Epochs = 10
for i in range(Epochs):
    for j in range(X.shape[0]):        → shuffle the data
        → select 1 row (random)
        → Predict (using forward Propagation)
        → calculate loss
        → update weight and baias using GD
```

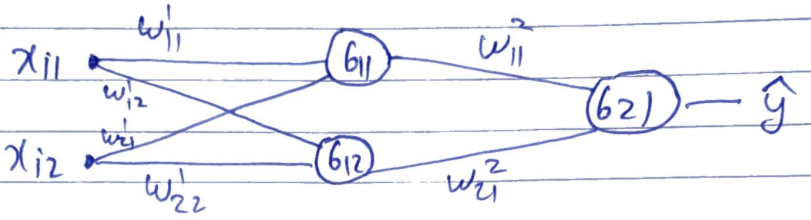
$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$, $B_{new} = B_{old} - \eta \frac{\partial L}{\partial b}$

(c) Example

(i) initalize all weight ($w=1$) and all bias ($b=0$)

(ii) Here we are using Activation Function = linear and Loss Function = MSE (mean squared error)

(iii) Network



For gradient decent we need to calculate these 9 derivative

$$\frac{\partial L}{\partial w^2_{11}}, \frac{\partial L}{\partial w^2_{21}}, \frac{\partial L}{\partial b_{21}}, \frac{\partial L}{\partial w^1_{11}}, \frac{\partial L}{\partial w^1_{21}}, \frac{\partial L}{\partial b_{11}}, \frac{\partial L}{\partial w^1_{12}}, \frac{\partial L}{\partial w^1_{22}}, \frac{\partial L}{\partial b_{12}}$$

$$\frac{\partial L}{\partial w^2_{11}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w^2_{11}} = -2(y_i - \hat{y}_i) \times \frac{\partial}{\partial w^2_{11}}(o_{11} w^2_{11} + o_{12} w^2_{21} + b_{21}) = -2(y_i - \hat{y}_i) o_{11}$$

$$\frac{\partial L}{\partial w^2_{21}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w^2_{21}} = -2(y_i - \hat{y}_i) o_{12} \qquad \frac{\partial L}{\partial b_{21}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial b_{21}} = -2(y_i - \hat{y}_i)$$

$$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial w^1_{11}} = -2(y_i - \hat{y}_i) \times w^2_{11} \times \frac{\partial}{\partial w^1_{11}}(x_{i1} w^1_{11} + x_{i2} w^1_{21} + b_{11}) = -2(y_i - \hat{y}_i) w^2_{11} x_{i1}$$

$$\frac{\partial L}{\partial w^1_{21}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial o_{12}} \times \frac{\partial o_{12}}{\partial w^1_{21}} = -2(y_i - \hat{y}_i) w^2_{21} x_{i2} \qquad \frac{\partial L}{\partial b_{11}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial o_{11}} \times \frac{\partial o_{11}}{\partial b_{11}} = -2(y_i - \hat{y}_i) w^2_{11}$$

$$\frac{\partial L}{\partial w^1_{12}} = -2(y_i - \hat{y}_i) w^2_{21} x_{i1}$$

$$\frac{\partial L}{\partial w^1_{22}} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial o_{12}} \times \frac{\partial o_{12}}{\partial w^1_{22}} = -2(y_i - \hat{y}_i) w^2_{21} x_{i2}$$

$$\frac{\partial L}{\partial b_{12}} = -2(y_i - \hat{y}_i) w^2_{21}$$

Note

$\hat{y}_i = o_{11} w^2_{11} + o_{12} w^2_{21} + b_{21}$

$o_{11} = x_{i1} w^1_{11} + x_{i2} w^1_{21} + b_{11}$

$o_{12} = x_{i1} w^1_{12} + x_{i2} w^1_{22} + b_{12}$

Note :- A Pro tip about gradient Decent

if $\frac{\partial L}{\partial w}$ = (+) = mean increase in weight lead to increase in L(Loss)

if $\frac{\partial L}{\partial w}$ = (-) = mean increase in weight lead to decrease in L(loss)

if $\partial L / \partial w$ = (+)

↳ we need to decrease the value of w so that L also decrease

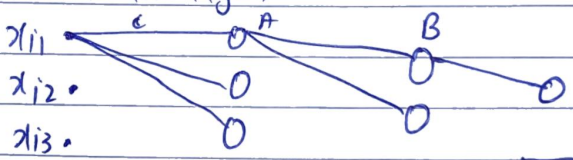
↳ $w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$

if $\partial L / \partial w$ = (-)

↳ we need to increase the value of w so that L decrease

↳ $w_{new} = w_{old} - \eta\left(-\frac{\partial L}{\partial w}\right) = w_{old} + \eta \frac{\partial L}{\partial w}$

(a) Example (Multiple Hidden layer)



Calculate derivative for A B and C

$\hat{y} = O_{21} w_{11}^3 + O_{22} w_{21}^3 + b_{23}$

$O_{21} = O_{11} w_{11}^2 + O_{12} w_{21}^2 + O_{13} w_{31}^2 + b_{21}$

$O_{22} = O_{11} w_{12}^2 + O_{12} w_{22}^2 + O_{13} w_{32}^2 + b_{22}$

$O_{11} = x_{i1} w_{11}^1 + x_{i2} w_{21}^1 + x_{i3} w_{31}^1 + b_{11}$

$O_{12} = x_{i1} w_{12}^1 + x_{i2} w_{22}^1 + x_{i3} w_{32}^1 + b_{12}$

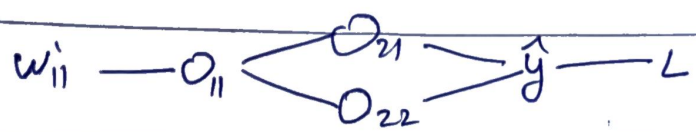
$O_{13} = x_{i1} w_{13}^1 + x_{i2} w_{23} + x_{i3} w_{33}^1 + b_{13}$

we want to calculate $\frac{\partial L}{\partial w_{11}^3}$, $\frac{\partial L}{\partial w_{11}^2}$, $\frac{\partial L}{\partial w_{11}^1}$

we can calculate $\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_{11}^3}$, $\frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial w_{11}^2}$

But we can't for $\frac{\partial L}{\partial w_{11}^1}$ = Because = $\frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial O_{21}/O_{22}} \times \frac{\partial O_{21}/O_{22}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}^1}$

Because → Both O_{21} and O_{22} depend on O_{11}



NOTE

In maths if:-

$x \rightarrow f(x), g(x) \rightarrow h(f(x), g(x))$

And then,

$\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f(x)} \times \frac{\partial f(x)}{\partial x} + \frac{\partial h}{\partial g(x)} \times \frac{\partial g(x)}{\partial x}$

$$\frac{\partial L}{\partial w_{11}^{1}} = \frac{\partial L}{\partial \hat{y}} \times \left[\frac{\partial \hat{y}}{\partial O_{21}} \times \frac{\partial O_{21}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}^{1}} + \frac{\partial \hat{y}}{\partial O_{22}} \times \frac{\partial O_{22}}{\partial O_{11}} \times \frac{\partial O_{11}}{\partial w_{11}^{1}} \right]$$

6.8 Types of Gradient Decent in Neural Network

(a) Types – Batch Gradient Decent
- Stochastics Gradient Decent
- Mini Batch Stochastics Gradient Decent

(b) Batch Gradient Decent
(i) For each epoch, we first do prediction on complete dataset, then calculate the loss and then update the weights and bias
(ii) Epoch = Number of time we calculate weights and bias

(c) Stochastics Gradient Decent
(i) The Gradient Decent we have studied in Back propagation

(d) Mini Batch Stochastics Gradient Decent
(i) Divide training data into small batch
(ii) For each epoch, shuffle data of each batch, then calculate the loss and update the weight and bias

(e) Speed = BGD > MBGD > SGD
Converge = BGD < MBGD < SGD

Note
(i) why Batch size is provided in multiple of 2? Ans = To use Ram effectively (Ram is desined to Handle binary value)
(ii) If row = 400, batch size = 150
then batches will be = 150, 150, 100

(7) Vanishing Gradient Decent Problem
(i) when the gradient of the loss function with respect to the Parameters of the earlier layer in the network become exceedingly small. This makes it difficult for the network to learn effectively

(ii) causes
(a) Very Deep Networks
(b) Certain Activation Functions (Sigmoid, tanh)
(c) Poor weight initialization

(iii) Is it Exclusive to Backpropagation
NO, It can affect any Gradient-based optimization method.

(iv) Symptoms - If loss is not decreasing, it may be because of VGP

(v) Solutions - (i) Reduce hidden layers
(ii) Use ReLU Activation Function
(iii) Do proper weight initialization
(iv) Use Batch normalization
(v) Residual network

⑧ Exploding Gradient Problem

(i) when the gradient grow exponentially large the updates to the network weights become excessively large, causing the model to become unstable and the loss function to diverge

(ii) cause - (a) Very deep Networks
(b) Poor weight initialization
(c) Long sequence in RNNs

(iii) Symptoms - Loss increase rapidly instead of decreasing

(iv) Solution - (a) Gradient clipping
(b) Weight initialization
(c) Use small learning rates
(d) Use Batch normalization
(e) Residual Network

(9) How to improve the performance of a Neural Networks

9.1 Problems we face

(i) Vanishing and exploding gradient Decent

(ii) NOT enough data

Solution - Transfer learning
Unsupervised pre Training

(iii) Slow training

Solution - optimizer
learning rate scheduler

(iv) over fitting

Symtoms - (i) High Training Accuracy But low validation Accuracy
(ii) High validation loss But low Training loss

Cause - (i) Deep Networks (ii) Insufficient training Data
(iii) High epoch

Solution - (i) Dropouts (ii) Regularization (iii) Add more data
(iv) Reduce layers (v) Early stopping

9.2 How to Decide value of these Parameters

(i) Number of Hidden layers - stop adding layer when model start overfitting

(ii) Number of neurons per layer - sufficient/more then what is required

(iii) Batch size → Large Batches - Fast But not good result
→ Small Batches - Slow But good result

(iv) Epochs - very high epoch + Early Stopping

9.3 Early Stopping

(i) Early Stopping is used to stop training when performance stops improving

(ii) Issues with wrong Epoch - large Epoch = lead to overfitting
less Epoch = ~~model will not converge~~ model will not converge

9.4 Data Scaling

(i) For unnormalize data → training will be slow, reason = normalization improves the gradient decent convergence.

9.5 Dropout layers

(i) used to reduce overfitting
(ii) Dropping out a fraction of the neurons during training
(iii) During Prediction, dropout layer does not applied, All the neurons are used for prediction, The weight of each neurons calculated like this = $W(1-P)$
(iv) For Example - A layer have dropout = 0.2, then weight of the layer = $W(1-0.25) = 0.75W$

(v) Tip

(a) Effect of P - very low value of P = lead to overfitting
- very high value of P = lead to underfitting
- it must be Blw 0.2 to 0.5

(b) Start applying dropout from last layer and if you see improvement then move to other layers

(c) For CNN - Range of P = 0.4 to 0.5
For ANN - Range of P = 0.1 to 0.5
For RNN - Range of P = 0.2 to 0.5

9.6 Regularization

(i) L1 → Penalty term = $\frac{\lambda}{2n} \sum_{i=1}^{k} ||w_i||$ λ = Hyper parameter, n = number of row

(ii) L2 → Penalty term = $\frac{\lambda}{2n} \sum_{i=1}^{k} ||w_i||^2$

(iii) used to reduce overfitting.

9.7 Activation Function

(i) Need - Activation function introduce non-linearity into the model

(ii) ideal activation function characteristics

(a) Activation function should be Non-Linear

(b) Activation function should be differentiable (to allow for gradient-based optimization methods)

(c) Activation function should be non-saturating (saturating activation function effectively cause the gradient vanishing)

(d) Activation function should be zero centered (mean=0, normalized)

(e) computational efficiency

(iii) Sigmoid

(a) Formula = $\frac{1}{1+e^{-x}}$ (b) Shape 

(c) output range = 0, 1 (d) Saturating Function thats why never used in hidden layers

(iv) tanh

(a) Formula = $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ (b) output range = -1, 1

(v) Softmax

(a) Formula = $\frac{e^{x_i}}{\sum_j e^{x_j}}$ (b) Range = 0, 1

(vi) Relu (Rectified linear Unit)

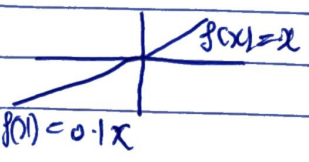
(a) Formula = max (0, x) (b) Shape = 

(c) variants

linear → Leaky relu, Parametric relu

non linear → Selu, elu

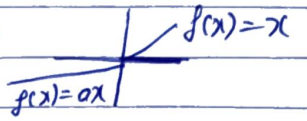
(vii) Leaky relu

(a) Formula = $f(z) = \begin{cases} 0.01 \times z & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$ (b) Shape 

(viii) Parametric Relu

(a) Formula = $f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$

(b) shape



a = trainable parameter

(ix) Exponential linear Unit (Elu)

(a) Formula = $f(x) = \begin{cases} x & \text{if } x \geq 0 \\ a(e^x - 1) & \text{if } x < 0 \end{cases}$

(b) shape

(x) Scaled Exponential Linear unit (Selu)

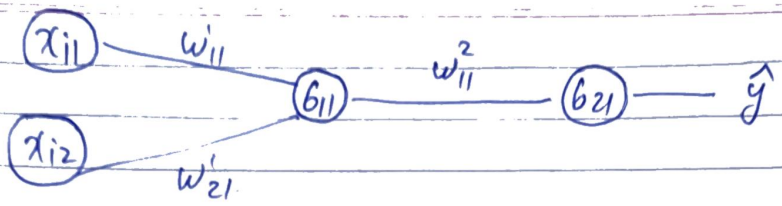
(a) Formula = $f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ ae^x - a & \text{if } x < 0 \end{cases}$ $a = 1.6732$, $6 = 1.0507$

Activation Function	Differentiable	Non-linear	Non-Saturating Activation	zero centered	computational Efficiency	Dying relu problem
sigmoid	✓	✓	✗	✗	✗	
tanh	✓	✓	✗	✓	✗	
softmax	✓	✓	✗	✗	✗	
relu	✗	✓	✓	✗	✓	✓
Leaky relu	✓	✓	✓	close	✓	✗
Parametric	✓	✓	✓	close	✓	✗
elu	✓	✓	✓	close	✗	✗
selu	✓	✓	✓	✓	✗	✗

9.8 Dying Relu Problem

(i) Relu Neurons that output zero for any input, leading to stop contributing to the learning process

(ii) cause - if $z < 0$ of a relu neuron



$z_{11} = w^1_{11} x_{i1} + w^1_{21} x_{i2} + b_{11}$

$z_{21} = w^2_{11} o_{11} + b_{21}$

$o_{11} = \text{relu}(z_{11})$

$o_{21} = \text{linear}(z_{21}) = \hat{y}$

Let assume $z_{11} < 0$, that make $o_{11} = 0$

Gradient decent

$$w^1_{11} = w^1_{11} - \eta \, \partial L / \partial w^1_{11}$$
$$w^1_{21} = w^1_{21} - \eta \, \partial L / \partial w^1_{21}$$

$$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{21}} \frac{\partial z_{21}}{\partial o_{11}} \frac{\partial o_{11}}{\partial z_{11}} \frac{\partial z_{11}}{\partial w^1_{11}} = 0 \quad \left\{ o_{11} = 0 \rightarrow \frac{\partial o_{11}}{\partial z_{11}} = 0 \right\}$$

Similarly $\frac{\partial L}{\partial w^1_{21}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z_{21}} \frac{\partial z_{21}}{\partial o_{11}} \frac{\partial o_{11}}{z_{11}} \frac{\partial z_{11}}{\partial w^1_{21}} = 0 \quad \left\{ 0 = 0 \rightarrow \frac{\partial o_{11}}{\partial z_{11}} = 0 \right\}$

$w^1_{11} = w^1_{11} - 0$ {this lead to no learning}

$w^1_{21} = w^1_{21} - 0$

(iii) causes (a) poor weight initialization
(b) high negative Bias
(c) large learning Rate

(iv) Once a relu neuron z become < 0, it is considered dead and not recoverable

(v) Solution — (a) set low learning rate
(b) use relu variant
(c) set bias 0.01

9.9 Weight initialization technique

(i) choosing wrong weight initialization technique lead to these problem
(a) vanishing Gradient problem (b) Exploding Gradient problem
(c) slow convergence

(ii) wrong weight initialization technique
(a) zero initialization (b) Non-zero costant initialization
(c) Random initialization with small weight
(d) Random initialization with large weight

(iii) what to do :- Random initialization But with correct variance
(iv) Techniques (a) xavier/Glorot weight initialization (Default in Keras)
(b) He weight initialization

For tanh and sigmoid activation function → xavier/Glorot
For relu activation function → He weight initialization work better.

9.10 Batch Normalization

(i) Batch normalization consists of normalizing activation vector from hidden layer using mean and variance of the current batch.

(ii) This normalization step is applied right before or right after the non-linear Function.

(iii) Adv - (a) speed up training (b) Regularizes the model
(c) Decrease the importance of initial weights.

9.11 Optimizer

(i) challenges in using gradient decent as optimizer

(a) Deciding the value of learning rate

slow value → lead to slow convergence
large value → lead to miss the minima point

(b) Same learning rate for all the dimensions

(c) converge at local minima

(d) Saddle Point

It is a point where slop of 1 direction is going up and slop of another direction is going downwards, this make a big region where slop remain same

which mean $\frac{\partial L}{\partial w} = 0$, $w_n = w_0 - \eta \frac{\partial L}{\partial w} \Rightarrow \boxed{w_n = w_0}$

(ii) Gradient decent alternative

(a) momentum (b) adagrad (c) NAG (d) RSM prop (e) Adam

(iii) ~~momentum~~ EWMA

(a) Exponentially weighted moving Average or Exponential weighted Average

(b) a technique used to find trend in time series based data.

(c) Formula

$$V_t = \beta V_{t-1} + (1-\beta) x_t$$

β = Hyper Parameter (0-1)
x_t = observation / data

(d) EWMA assign exponentially decreasing weight to older observation and give more importance to recent observation

$$V_t = \beta V_{t-1} + (1-\beta) x_t$$
$$V_0 = 0$$
$$V_1 = \beta V_0 + (1-\beta) x_1 = (1-\beta) x_1$$
$$V_2 = \beta V_1 + (1-\beta) x_2 = \beta(1-\beta) x_1 + (1-\beta) x_2$$
$$V_3 = \beta^2 (1-\beta) x_1 + \beta(1-\beta) x_2 + (1-\beta) x_3$$

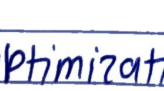
here for $V_3 \rightarrow x_1 \times \beta^2$, $x_2 \times \beta$, $x_3 \times \beta$

(e) Impact of β

High β (close to 1): The EWMA place more weight on past values and less weight on the most recent observation. The EWMA react slowly to recent changes in the data.

Low β (close to 0): The EWMA place more weight on the most recent observation and less weight on past values. The EWMA react quickly to recent changes in the data.

(ii) Convex and non-convex optimization

convex optimization =  , Non-convex opti. = 

Issue with non-convex graph = local minima, saddle point, high curvature.

(v) Momentum

(a) Problem solve - local minima, Saddle Point, high curvature.

(b) The main idea momentum optimization is to acceleration the gradient vector in the right direction, leading to faster converging towards the minima and reducing oscillations

(c) Formula

$$W_{t+1} = W_t - V_t \qquad \left(V_t = \underbrace{\beta * V_{t-1}}_{\text{momentum term}} + \eta \underbrace{\nabla w_t}_{\text{gradient of } w_t} \right)$$

Hyperparameter - η, β

if $\beta = 0$, then momentum = = gradient decent

(d) Disadvantages

momentum, Because of momentum the gradient decent make oscillation on global minima, wasting some time.

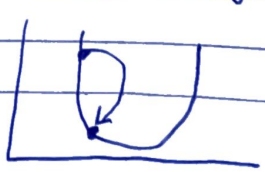
(vi) Nesterov Accelerated Gradient

(a) mostly perform better then momentum. Builds on the concept

(b) The main idea behind NAG is to anticipate the future position of the parameters and compute the gradient not at the current position but at this anticipated future position

(c) Formula

→ First we apply momentum term and make the jump to new point (look ahead term)



→ Now we calcalate the gradient at the new point

→ Now on the basis of this new gradient we decide where to move.

$w_{ea} = w_t - \beta V_{t-1}$ (step 1)
↳ look ahead
$V_t = \beta V_{t-1} + \eta \nabla w_{ea}$ [calculating slop at new point]
$w_{t+1} = w_t - V_t$ (update rule)

(d) Disadvantages : convergence at local minima

(vii) Adaptive Gradient Algorithm

(a) Adagrad adapts the learning rate for each parameter individually making larger updates for infrequent parameter and smaller updates for frequent parameter.

(b) works better for :- if input feature values have different range
if input feature values are sparses

(c) Formula

$$w_{t+1} = w_t - \frac{\eta \nabla w_t}{\sqrt{V_t + \varepsilon}}$$ (ε → non-zero very small number)

$$V_t = V_{t-1} + (\nabla w_t)^2$$

(d) Div - learning rate Decay (that why not used in neural network)

(viii) Root mean Squared Propagation (RMS prop)

(a) RMSprop modifies AdaGrad by introducing a decaying avg of past squared gradients which help stabilize the learning rate over time.

(6) It is one of the optimizing techniques for neural networks

(c) Formula

$$w_{t+1} = w_t - \frac{\eta \nabla w_t}{\sqrt{v_t + \varepsilon}}$$

ε = very small non zero number.

$$v_t = \beta v_{t-1} + (1-\beta)(\nabla w_t)^2$$

(ix) Adam

(a) Adam (Adaptive Moment Estimation) is an optimization algorithm that combines the advantages of 2 other popular optimization method: Adagrad, RMSProp

(b) Best optimizing technique.

(c) Formula

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \nabla w_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla w_t)^2$$

Bias correction

$$\hat{m}_t = (m_t) / (1 - \beta_1^t)$$
$$\hat{v}_t = (v_t) / (1 - \beta_2^t)$$

→ this t mean epochs number.

β_1 and β_2 = Trainable Parameter ($\beta_1 = 0.9$, $\beta_2 = 0.99$ generally)

$$w_{t+1} = w_t - \frac{h \hat{m}_t}{\sqrt{v_t + \varepsilon}}$$

(10) Convolutional Neural Networks

10.1 Special kind of neural network for processing data that has a grid like topology like time series data (1D) or image (2D)

10.2 CNN Architecture

(a) Input Layer

(b) Convolutional Layer
- contain multiple [filter (kernels) + relu]
- Output = feature map
- we can also do batch normalization after convolution layer before the pooling layer

(c) Pooling layer
- output = tensor

(d) Flatten layer
- convert high dimensional data into 1D

(e) Fully connected layer
- like ann

10.3 Why we not use ann on image data

- overfitting
- High computational cost
- loss of arrangement of pixels

10.4 Basics of images

(a) Types of images → Grey Scaled images
→ colored images

(b) Grey Scaled images
→ Store image like 2d array in 1 channel
→ value varies from 0 (black) to 255 (white)

(c) colored image
↳ Store image like 2d array in 3 channel
↳ Shape (228, 228, 3)

10.5 convolutional layer

(a) Core Building block of a CNN

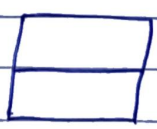
(b) It applies a convolution operation to the input, passing the result to the next layer.

(c) Intuition
1 convolutional layer = Detect primitive features
2 convolutional layer = Detect slightly more complex feature
3 convolutional layer = detect advance features

(d) Steps
- Initialize the number filter and size of filter (generally 3x3)
- Slide the filter over the input image
- Perform element wise multiplication and summation

Note - The value of filter in cnn are not manually decided. Instead they are learned automatically during the training process through backpropagation. Initially filters are initialized with small random value.

(e) Example

Input image =  Edge detection convolution operation → 

Output

- Image = $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix}$ Size (6×6)

$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ → Kernel filter

$*$ → convolution Operation

$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = 0\times1 + 0\times-1 + 0\times-1 + 0\times1 + 0\times1 + 0\times1 = 0$

$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} = 0$

→ repeat this = feature map $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

= 

(3) Formula If image size (n×n), if filter size (m×m)

⇒ Feature map size = (n-m+1), (n-m+1)

like image size = (6,6), filter Size (3×3)

feature map Size = (6-3+1), (6-3+1) = (4,4)

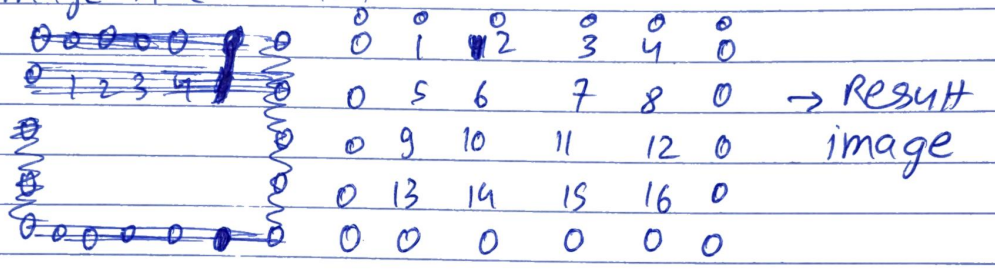
10.6 Padding

(a) Need

- Loss of data because of convolution operation
- The border pixels will be part of less convolution as compare to centre pixels

(b) Padding refers to adding extra pixels around the border of the input image

(c) In Keras are 2 types of Padding → valid (no padding)
→ same (Apply padding)

(d) Intuition
→ Add zero row and column at the border of image
→ image size = 4×4
→ 

(e) Formula if image size (n×m), if filter size (m×m)
p = padding

Feature map size = $(n+2p-m+1), (n+2p-m+1)$

10.7 Strides

(a) Strides determine the step size by which the filter moves across the input image. The stride affects the spatial dimensions of the output feature map.

(b) Why to use → To reduce computing power
→ To capture high level features

(c) Not so much important now days

(d) Formula image size (n×n), filter size (m×m), p = padding
s = Strides

Feature map size = $\left(\frac{n+2p-m}{s}+1, \frac{n+2p-m}{s}+1\right)$

(e) Effect of strides

Stride (1,1) = The output feature map is larger and closer in size to the input

Stride (3,3) = The output feature map is smaller, reducing the computational load, information loss.

10.8 Pooling layer

(a) Problem with convolution operation
- Memory issue = convolution operation consume a lot of memory
- Translation variance = the convolution operation will become location dependent.

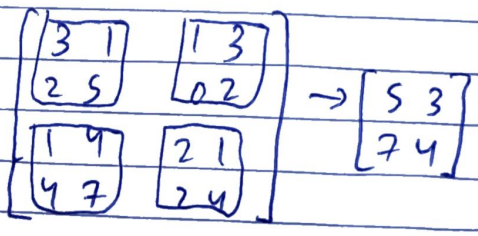
(b) Types - max pooling, min pooling, avg pooling, L2 pooling, global pooling

(c) Parameters - size (2,2), stride, type

(d) Adv - memory reduced, translation invariance, max pooling → enhance feature

(e) div - loss of information
- image segmentation required translation variance.

(f) Implementation - feature map



Pooling = size = 2x2
stride = 0
type = max pooling

10.9 Data Augmentation

(a) Data augmentation is a technique used to increase the diversity and quantity of training data without actually collecting new data

(b) Why it is used
- increase data size
- Reduce overfitting
- Enhance generalization

(c) we do not apply data augmentation on test data set.

(d) common data augmentation techniques

- Geometric Transformations
  Rotation, translation, scaling, Flipping
- Color Transformations
  Brightness adjustment, contrast adjustment, saturation adjustment
- Occlusion
  Random cropping, cutout (randomly using masking out square regions of the input during training)
- Elastic Distortions
  Applying random elastic distortions to the image.
- Noise injection
  Introducing random noise to the image.

10.10 Pretrained Models

(a) Pretrained models are neural network model that have been previously trained on a large dataset and can be reused by new, related tasks.

(b) Benefits - Reduced Training Time
- Improved performance
- Resource efficiency

(c) Div - If pretrained model does not trained on the our current target task (solved by transfer Learning)

(d) Commonly used Pretrained Models

- Image Model :- ResNet, VGG16, VGG19, Inception, MobileNet EfficientNet
- NLP Model :- GPT, BERT, TS, ROBERTa, Transformer.

10.11 Transfer learning

(a) A technique where a pretrained model is used as a starting point for a new task, leveraging the knowledge gained from the original task.

(b) Adv - Reduced Training time
- Improved Performance
- Resource Efficiency

(c) Type
- Feature extraction
- Fine - Tuning

(d) Feature Extraction

- Use the pretrained model as a fixed feature extractor
- In this approach we freeze the weights of the pretrained model and only train the final layers specific to the new task
- In case of cnn we re-train fully connected layer and freeze the convolutional layer
- used when your model are already trained on kind a similar problem - using dog vs cat classifier on lion vs tiger classifier.

(e) Fine Tuning

- Unfreeze some or all of the layers of the Pretrained model and jointly train them with the new layers on the new task. This allows the model to adapt the pretrained weight to the specifics of the new task

- In case of cnn we retrain fully connected layer and some last convolution layer also

- used when your model are not already trained on kind a similar problems

II Functional API

(a) Assumptions of sequential models

(i) 1 input
(ii) 1 output
(iii) All layers are linearly attached

(b) The Function API allow us to build more complex neural network architecture as compared to Sequential API

(c) The Function API allow us to build model with multiple input, multiple output, shared layers or non-linear topology.

12 RNN

12.1 RNN Basics

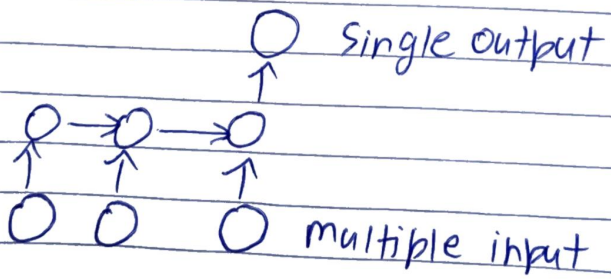
(a) Designed to process sequential data

(b) RNN have connections that form directed cycles, enabling them to maintain a memory of previous inputs in the sequence

(c) Good For - Time series analysis, language modeling.

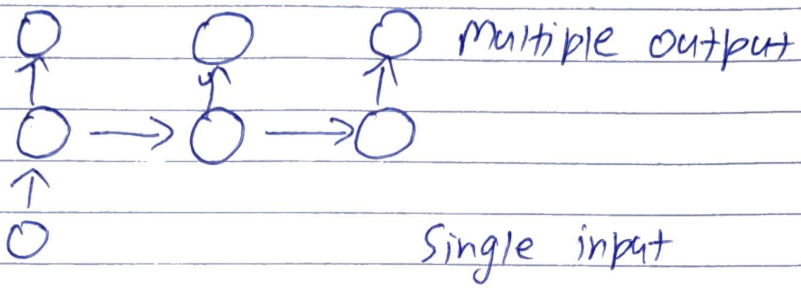
(d) Types
- (i) Many to one RNN
- (ii) One to many RNN
- (iii) Many to many RNN
- (iv) One to One RNN

(i) Many to one RNN



Application - Sentiment analysis, Rating prediction

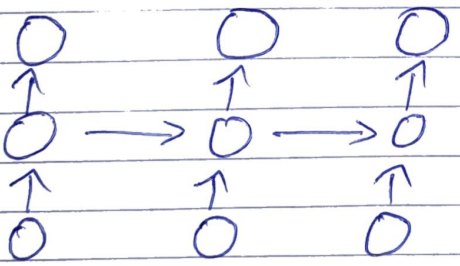
(ii) One to Many RNN



Application - Image captioning

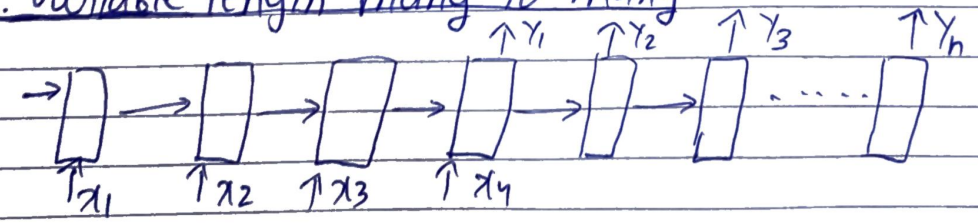
(iii) Many to Many RNN

1. Also know as Seq2Seq models
2. Types
   - Same length many to many
   - Variable length many to many
3. Same length many to many



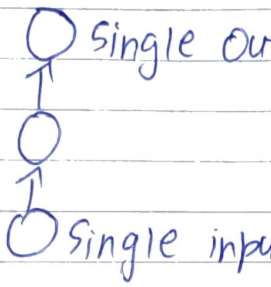
Application - Part of speech tagging
- Name Entity recognition

4. variable length many to many



Application - Machine translation

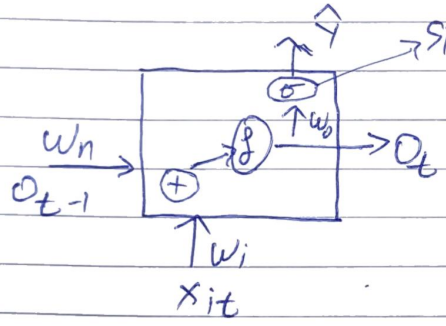
(iv) Single to Single RNN / One-to-One RNN



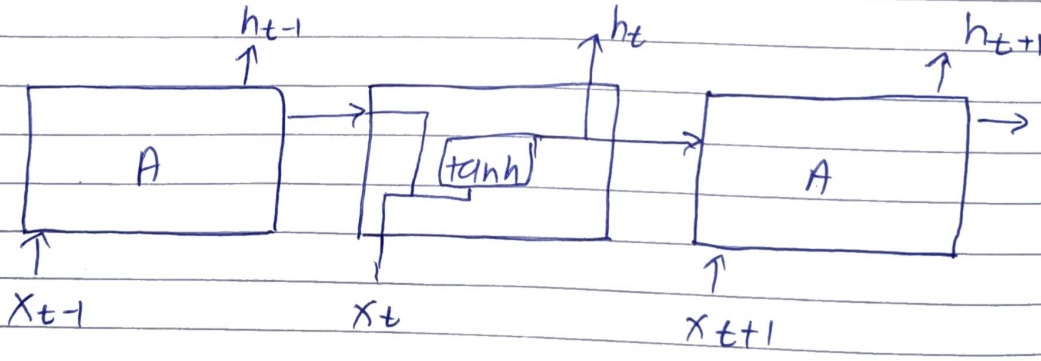
Application - image classification

☆ Technically this is not rnn

(e) RNN architecture



i = row number
t = time step
$f(w_i x_{it} + O_{t-1} w_n) = O_t$
$g = \text{Sigmoid}(O_t w_o)$



(f) variation of RNN

(i) Long Short Term Memory (LSTM)
(ii) Gated Recurrent Unit (GRU)
(iii) Bidirectional RNN (Bi RNN)
(iv) Echo State Networks (ESN)

(g) Why ann is not good for text data

(i) The order of words in a sentence is important for meaning. ANN do not inherently account for the order of input features, treating each word as independent

(ii) ANN lack the ability to maintain and utilize context over long sequences

(iii) High dimensionality of text data

(iv) Variable length of text sequences.

(h) Difference between RNN and ANN

RNN	ANN
(i) RNN process input sequence based on time steps.	(i) ANN process input simultaneously in a feed forward manner.
(ii) RNN are trained using Backpropagation through time (BPTT)	(ii) ANN are trained using standard backpropagation algorithm
(iii) RNN maintain a hidden state that captures info about the sequence seen so far. which makes it capable of capturing dependencies over time.	(iii) Each input is processed independently without considering previous input in the same training instance.

(i) Bidirectional RNN

(a) Capture information from both past and future states in a sequence

(b) Application - Name entity recognition
- Part of speech tagging
- Sentiment analysis
- Machine translation

(j) Problem with simple RNN

(i) Vanishing gradient problem
(ii) Exploding gradient problem

(iii) Different in learning Long-Term-Dependencies

(a) Simple rnn struggle to capture dependencies that span long sequences due to the vanishing gradient

(b) Solution
→ Use relu activation functions
→ Better weight initialization
→ Use LSTM / GRU

(iv) Unstable training because of exploding gradient decent.

12.2 LSTM

(a) A type of rnn designed to better capture long-term dependencies in sequential data.

(6) They are particularly effective at addressing the vanishing gradient problem that simple RNN face, enabling them to learn from and remember important information over long sequences.

(c) Components of LSTM

(i) Input
- Previous hidden state
- Previous cell state
- Input for the current time

(ii) Output
- current cell state
- current hidden state
-

(iii) Processing
- calculate hidden state
- Add or remove information from cell state

(iv) Cell State (C_t)
- It carrying relevant information through the Entire sequence.
- Help in maintaining long-term dependencies.

(v) Hidden State (h_t)
- Output of the LSTM at each time step.

(v) Gates
- → Forget Gate
- → Input Gate
- → Output Gate

(vi) Forget Gate
- Decides what to forget from the cell state based on current input and short term context

(vii) Input Gate
- Decides what new information to add to cell state

(viii) Output Gate
- Produces the new hidden state based on the updated cell state

(d) Why LSTM are effective

(i) Mitigating vanishing gradients - The state allows gradients to follow unchanged over long sequences.

(ii) Selective Memory - The gates (input, output, forget) enable selective memory, allowing the network to retain or discard information as needed.

12.3 Gated Recurrent Unit (GRU)

(a) GRU is a type of rnn architecture designed to impact the performance of traditional rnn

(b) Architecture

(i) Rest Gate
- Decides how much of the past info should be forgotten

(ii) Update Gate
- Decide how much of the previous hidden state should be passed to the current hidden state

(iii) Input
- Previous hidden state
- Input for the current time

(iv) State
- combines the previous hidden state and the candidate activation based on the update gate.

(c) Advantages of GRU
- Simple architecture as compared to LSTM
- Less computationally Intensive as compared to LSTM
- Efficient memory use

(d) GRU vs LSTM

Feature	GRU	LSTM
Gates	2(Update, Reset Gates)	3(Input, Output, Forget Gate)
memory cell	No memory cell	have separate memory cell
Hidden State	Combine Hidden and cell stat	Separate hidden and cell State
update process	Fewer Parameter (simple)	complex
Parameters	Fewer Parameter	More Parameter
Time	Less training time	More Training Time
Performance	Comparable to LSTM	Robest
Complexity	Simple architecture	Complex architecture
Computation	Less intensive	more intensive

13 Encoder Decoder

13.1 Encoder

(i) The encoder processes the input sequence and compresses the information into a context vector

(ii) This typically involves serval layers of LSTM / GRU

13.2 Decoder

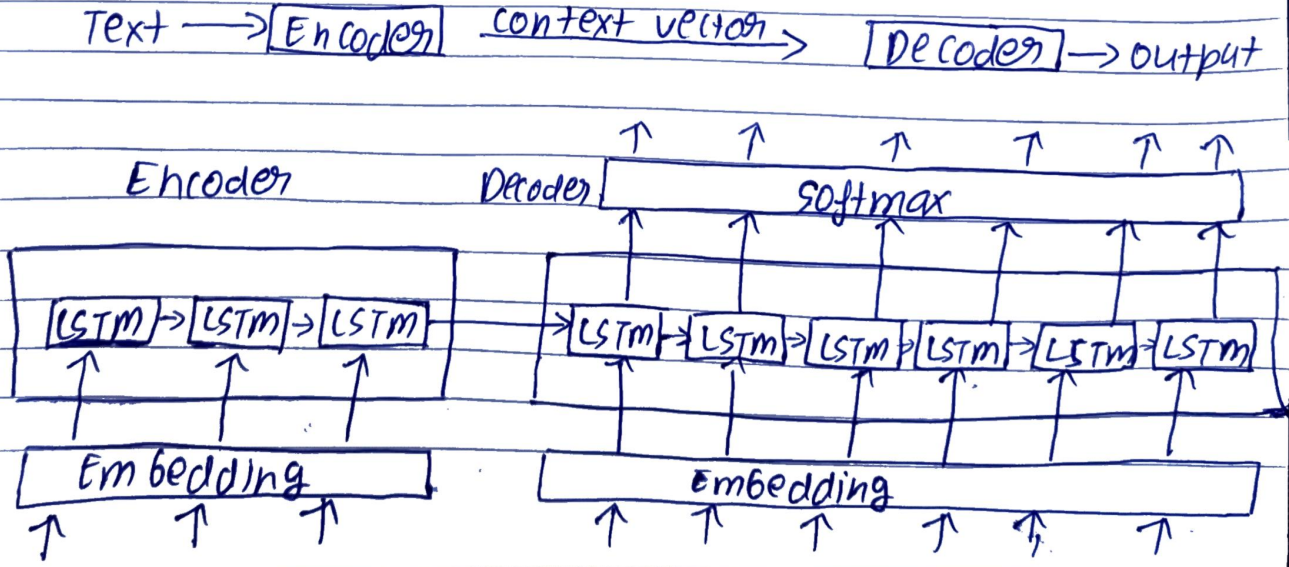
(i) The decoder takes the context vector from the encoder and generates the output sequence

(ii) Similar to encoder, it contain serval layers of LSTM/GRU

13.3 context vector

The context vector is a fixed size vector that encapsulates the information from the entire input sequence

13.4 Architecture



13.5 challenges of vanilla Encoder-Decoder

(i) Fixed length context vector lead to loss of information

(ii) Gradient Vanishing / Exploding Problem

(iii) Inability to capture long Range Dependency

13.6 Attention mechanism

(i) It allows the decoder to focus on different parts of the input sequence at each step of the output generation

(ii) The decoder get a weighted sum of all encoder hidden states where the weights are dynamically computed based on the decoder's current state.

(iii)

Encoder → Attention mechanism → decoder

(iv) we can't use trans fer learning for Encoder-Decoder with attention mechanism

Because Encoder-Decoder with attention mechanism train sequentially (which lead to slow training) that's lead to High cost and time for training, Henre we can't train it on very huge data set.

14. Transformer

14.1 Transformers are a type of neural network architecture designed to handle sequential data

14.2 Architecture

(i) Encoder - The encoder is responsible for processing the input sequence and generating a set of encoder representation

(ii) Decoder - The decoder generates the output sequence, using the encoder representions from the encoder

(iii) Output layer - The final linear layer transforms the decoder's output into logits which are converted into probability using a softmax function to produce the final output tokens.

14.3 Advantages

(i) Scalability (ii) Transfer learning (iii) Multi mode input output (iv) Flexible architecture (v) Transfer learning

14.4 Disadvantages

(i) High computational cost (ii) Large amount of data requires (iii) Prone to overfitting (iv) High energy consumption

14.5 Self Attention mechanism

(i) mechanism that take static embedding and generate contextual embedding.

(ii) Issue with static embedding

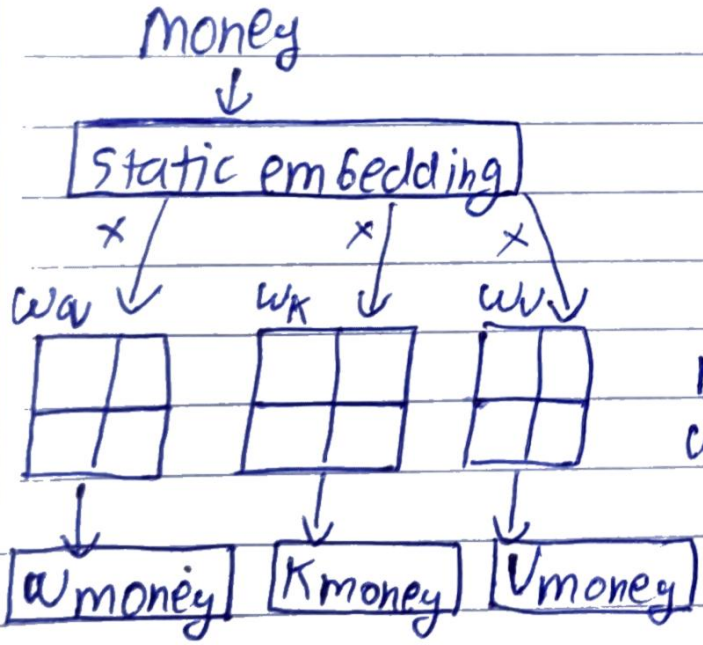
A word can have more then 1 meaning, But static embedding capture only one meaning.

(iii) working

Creating contextual embedding for line →

"money bank grows"

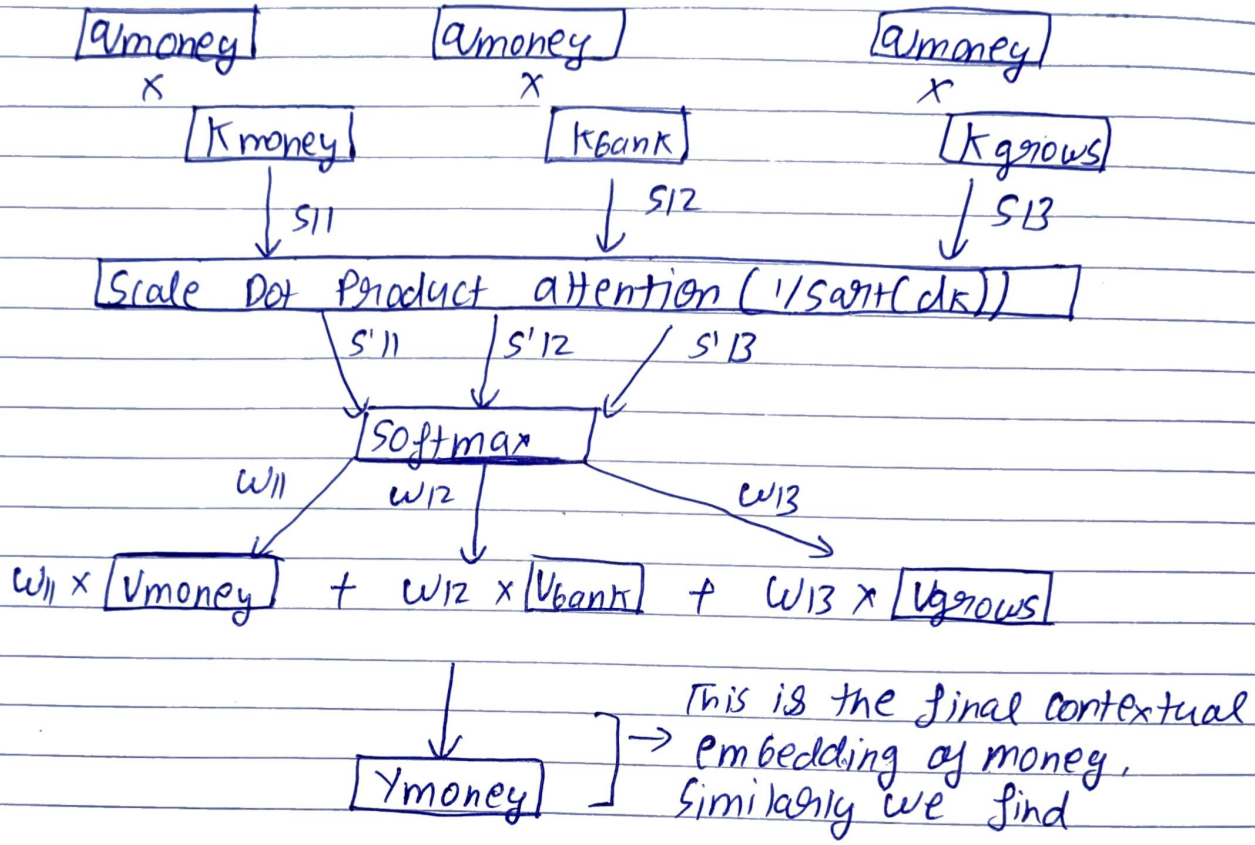
step 1:- making 3 embedding for each world (query, key, value)



these (q, k, v matrics) are obtained by multiplying the word static embedding with 3 learned weighted matrix, initially they were initialized with random number and obtain correct value through training

Similarly we find q_{Bank} k_{Bank} v_{Bank}, q_{grows}

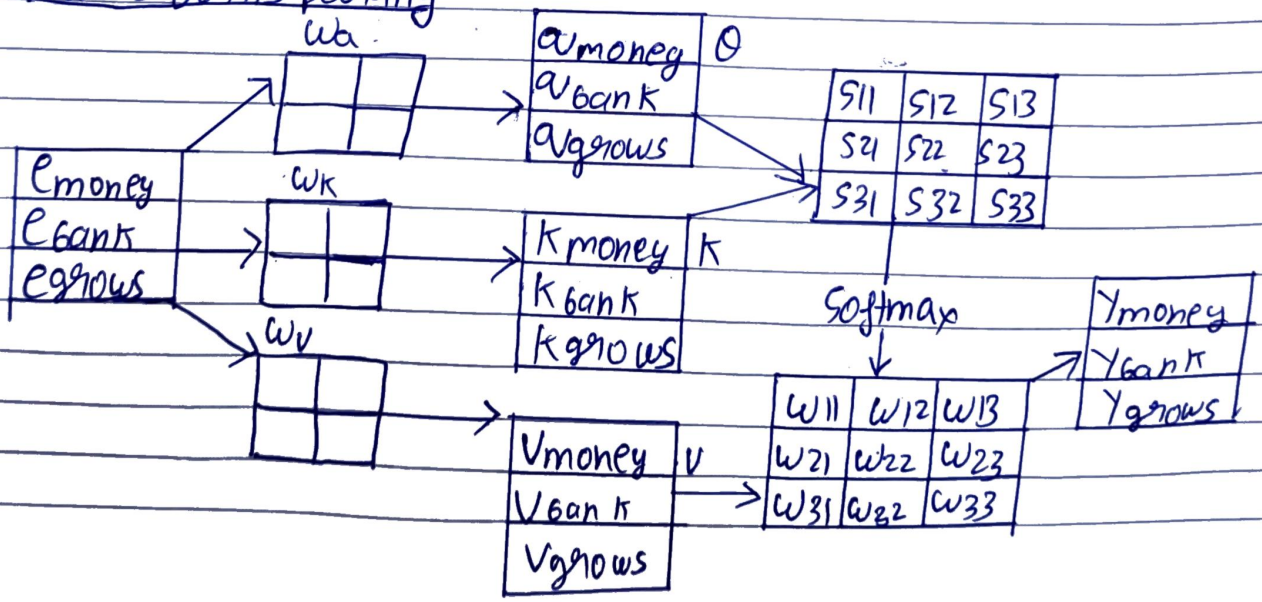
Step2 : Finding Contextual Embedding



This is the final contextual embedding of money. Similarly we find

Ybank, Ygrows

How we do this parally



(iv) Scaled Dot Product Attention

To stabilize the gradient during training, the dot product are scaled by the square root of the dimension of the key vector (d_k)

$$\text{Scaled Score}(x_i, x_j) = \frac{Q_i \cdot K_j^T}{\sqrt{d_k}}$$

(v) $\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V$

(vi) Self Attention is called "self" because it operates on a single sequence of elements and relates each element at all other elements in the same sequence.

(vii) Problem with the self attention mechanism without multi head attention

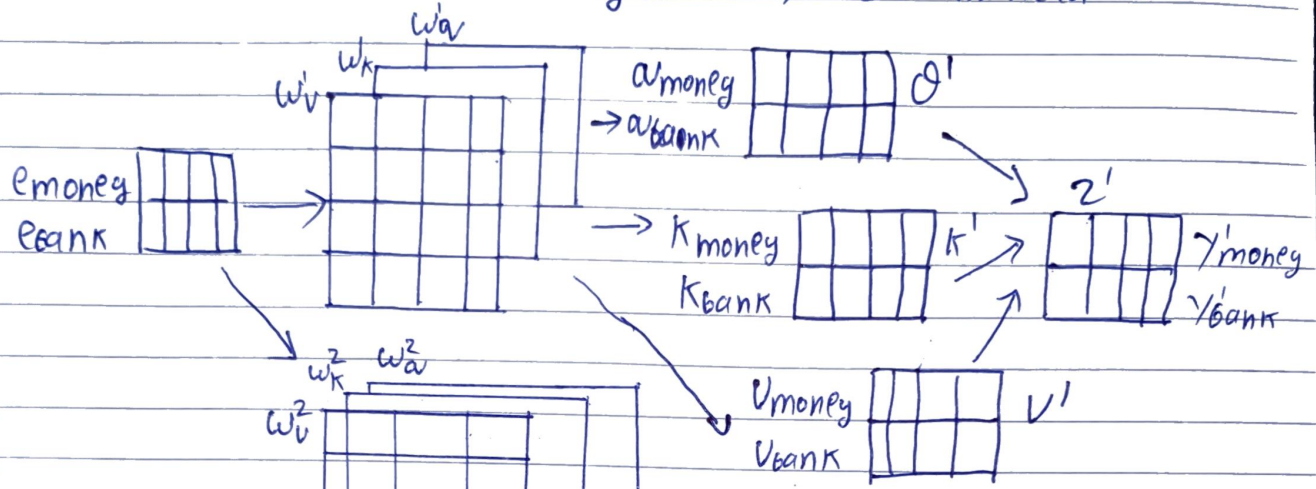
If a text has more then one perspective it will only gone capture the most prominent connection leading to miss out other important but obvious relation (Single Perspective)

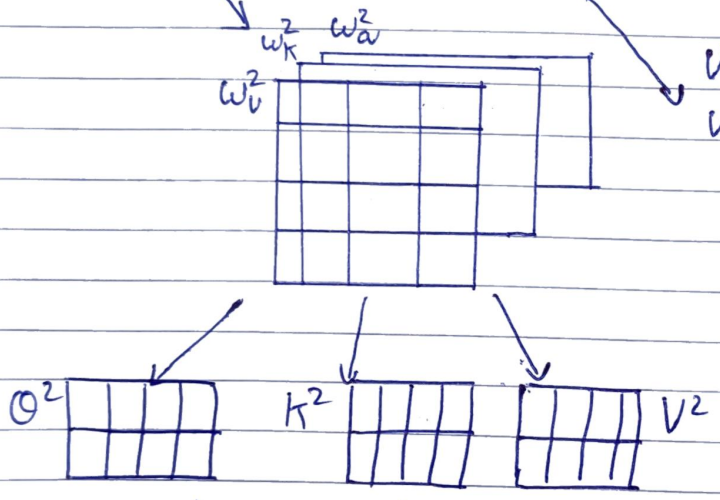
4.6 Multi Head Attention

(i) Multi-head attention is a mechanism used in transformers to enhance the model's ability to focus on different parts of the input sequence simultaneously and capture various types of relationship and dependencies.

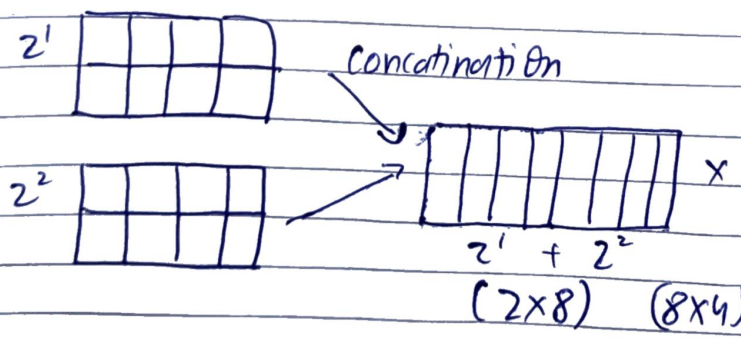
(ii) working

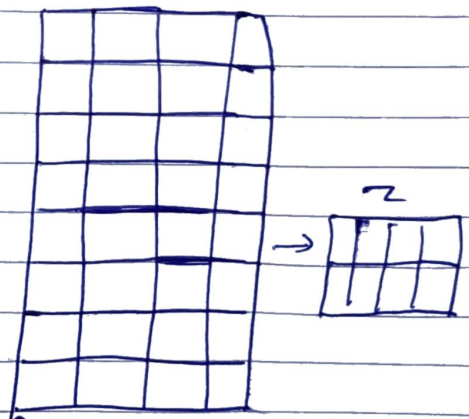
sentence = "money bank", let multi head = 2





we perform this linear Transformation so that we get the resultant embedding same size of input embedding





The value of this grid is random initially.

4.7 Positional Encoding

(i) Transformers process all tokens in parallel. So without positional encodings the model would have no information about the order of the tokens in the sequence.

(ii) Formula

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

PE are positional encoding for the even and odd dimensions of the position pos

pos is the position in the sequence

i represent the index of position encoding vector

dmodel is the dimension of the model (the size of embedding vectors)

(iii) Example sentence - "Tarun Singh"
embedding vector size = 6 (1×6)

Pos of Tarun = 1
Pos of Singh = 2
dmodel = 6
Tarun embedding = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
Singh embedding = [0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

now we need to calculate

Tarun positional encoding = [?, ?, ?, ?, ?, ?]
Singh Positional encoding = [?, ?, ?, ?, ?, ?]

For Tarun

$PE(0,0) = \sin\left(\frac{0}{10000^{0/6}}\right) = \sin(0) = 0$

$PE(0,1) = \cos\left(\frac{0}{10000^{2/6}}\right) = \cos(0) = 1$

$PE(0,2) = \sin\left(\frac{0}{10000^{2/6}}\right) = \sin(0) = 0$

$PE(0,3) = \cos\left(\frac{0}{10000^{2/6}}\right) = \cos(0) = 1$

$PE(0,4) = \sin\left(\frac{0}{10000^{4/6}}\right) = \sin(0) = 0$

$PE(0,5) = \cos\left(\frac{0}{10000^{4/6}}\right) = \cos(0) = 1 \rightarrow [0,1,0,1,0,1]$

Similarly we Find = [0.84, 0.54, 0.0001, 1.0, 0.0001, 1.0]
Sing positional encoding.

Once we find the Positional encoding

Final embedding = Embedding + Positional Encoding

Final embedding$_{Tarun}$ = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6] + [0, 1, 0, 1, 0, 1] = Result

Final embedding$_{singh}$ = Similarly we Find this

(iii) Without Positional encoding these 2 sentenie will have same meaning → Tarjan kill lion
lion kill Tarjan

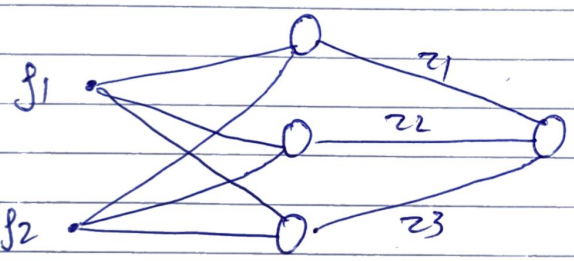
(iv) why we are using Sin and cosine

(a) Distinctive Pattern (b) Smooth variation

14.8 Layer Normalization

(i) Unlike batch normalization which normalize accross the batch dimension, layer normalization normalize accross the features of the every individual sample.

(ii)



f1	f2	z1	z2	z3
a	b	c	d	e

→ we normalize layer By layer.

(iii) why we dont use Batch normalization

- Dependency on Batch size
- Sequential Data Handling
- Does not support Parallel Processing.