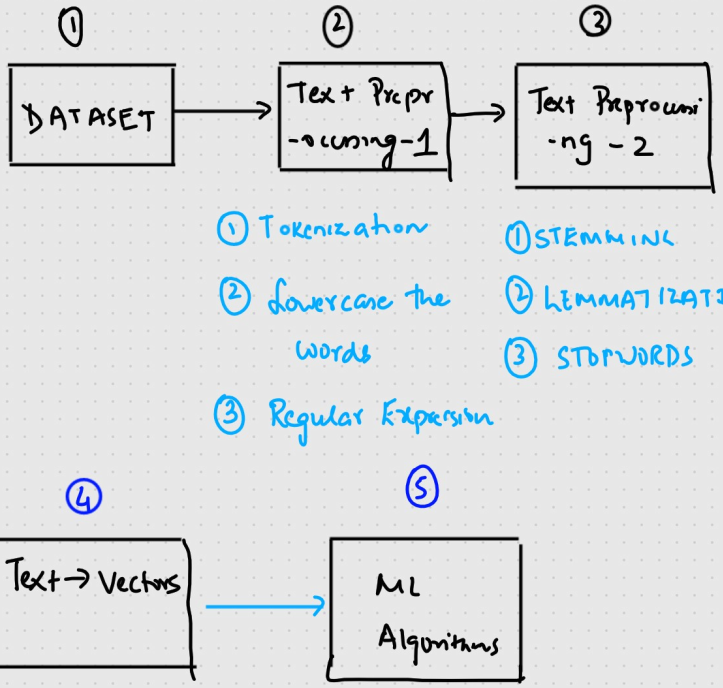
Text Preprocessing → What we have learnt?

Sentiment Analysis

	Text	O/P
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is Amazing	1
D4	Burger is bad	0



D1 → [1 0 1 1] 1

Gensim

① One Hot Encoded
② Bag of Words (BOW)
③ TF-IDF
④ Word2Vec
⑤ Avg Word2Vec

① One Hot Encoding

	Text	O/P
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is Amazing	1

Test [Burger is bad]

Vocabulary size: 7

Vocabulary {unique words}

The	food	is	good	bad	Pizza	Amazing
1	0	0	0	0	0	0
0	1	0	0	0	0	0

D1 4×7
[[1 0 0 0 0 0 0],
[0 1 0 0 0 0 0],
[0 0 1 0 0 0 0],

D2 4×7
[[1 0 0 0 0 0 0],
[0 1 0 0 0 0 0],
[0 0 1 0 0 0 0],

D3 3×7
[[0 0 0 0 0 1 0],
[0 0 1 0 0 0 0],
[0 0 0 0 0 0 1]]

[0 0 0 1 0 0 0] [0 0 0 0 1 0 0]

{50K vocabulary size}

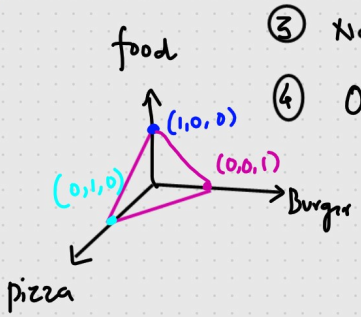
Advantages

① Easy to implement with python
[sklearn OneHotEncoder, pd.get_dummies()]

Disadvantages

① Sparse matrix → Overfitting
② ML Algorithm → Fixed size I/p
③ No semantic meaning is getting captured
④ Out of Vocabulary (OOV).

food	pizza	burger
1	0	0
0	1	0
0	0	1



② Bag of Words

Dataset-

Text	O/p
He is a good boy	1
She is a good girl	1
Boy and girl are good	1

lower all the words case

Stopwords

S1 → good boy
S2 → good girl good
S3 → Boy girl good school

Test

Vocabulary	frequency
good	3
boy	2
girl	2

	[good	boy	girl]	O/p
S1	[1	1	0]	1
S2	[1	0	1]	1
S3	[1	1	1]	1

Binary BOW and BOW

{1 and 0}

{Count will get updated based on frequency}

Advantages

① Simple and Intuitive

② Fixed Sized I/p → ML Algorithms

Disadvantages

① Sparse matrix or array → Overfitting

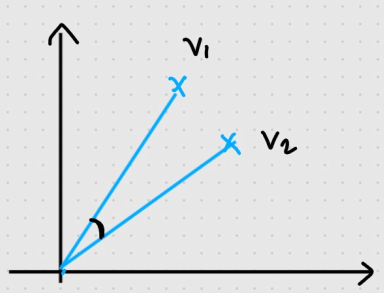
② Ordering of the word is getting changed

③ Out of Vocabulary (OOV).

④ Semantic meaning is still not captured.

The food is good → [1 1 1 0 1] → v_1

The food is not good → [1 1 1 1 1] → v_2



↓ Similar

⑤ N-grams

Eg: bigrams, trigrams

S1 → The food is good

S2 → The food is not good

	food	not	good
→	1	0	1
→	1	1	1

Bigram

	food	not	good	food good	food not	not good
S1	1	0	1	1	0	0
S2	1	1	1	0	1	1

Sklearn → n-grams = (1,1) → unigrams

= (1,2) → unigram, bigram

= (1,3) → unigram, bigram, trigram

= (2,3) → Bigram, trigram.

④ TF-IDF [Term Frequency – Inverse Document Frequency]

S1 → good boy

S2 → good girl

S3 → boy girl good

$$\text{Term Freq (TF)} = \frac{\text{No. of rep of words in sentence}}{\text{No. of words in sentence}}$$

$$\text{IDF} = \log_e\left(\frac{\text{No. of Sentences}}{\text{No. of sentences containing the word}}\right)$$

Term Frequency * IDF

	S1	S2	S3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

Words	IDF
good	$\log_e(3/3) = 0$
boy	$\log_e(3/2)$
girl	$\log_e(3/2)$

Final TF-IDF

	[good	boy	girl]
Sent 1	0	$\frac{1}{2} \times \log_e(3/2)$	0
Sent 2	0	0	$\frac{1}{2}\log_e(3/2)$
Sent 3	0	$\frac{1}{3}\log_e(3/2)$	$\frac{1}{3}\log_e(3/2)$

Ofs

Bow

1	1	0
1	0	1
1	1	1

Advantages

① Intuitive

② Fixed Size → Vocab size

③ Word Importance is getting captured

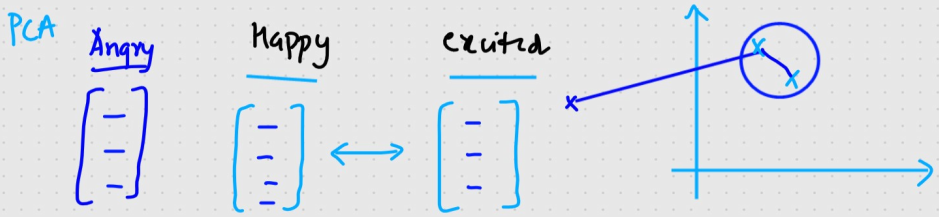
Disadvantages

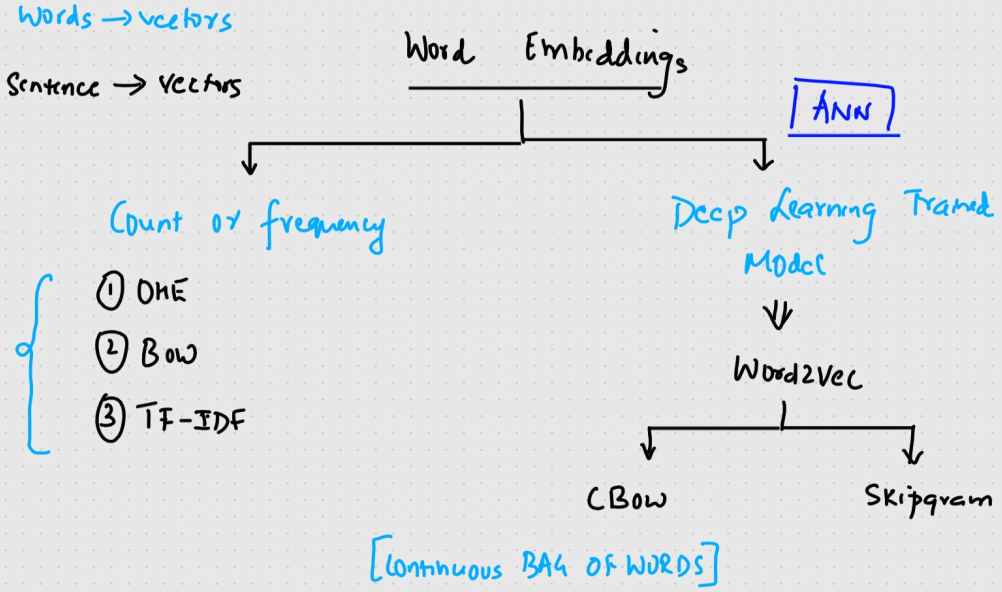
① Sparsity still exists

② OOV

Word Embeddings [Wikipedia]

In natural language processing (NLP), word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning.





Word2Vec → Feature Representation

Google

Word2vec is a technique for natural language processing published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector.

Vocabulary → Unique words → Corpus.

Feature Representation

	Boy	Girl	KING	QUEEN	Apple	Mango
Gender	-1	1	-0.92	+0.93	0.01	0.05
Royal	0.01	0.02	0.95	0.96	-0.02	0.02
Age	0.03	0.02	0.75	0.68	0.95	0.96
Food	-	-	-	-	0.91	0.92
				-	-	-

300 dimension ! nth

$$\begin{bmatrix} - \\ - \end{bmatrix} \quad - \quad - \quad - \quad - \quad - \quad - \quad - \quad - \quad -$$

[KING - BOY + QUEEN = GIRL]

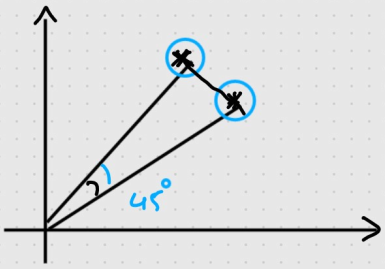
KING [0.95, 0.96] MAN [0.95 0.98]

QUEEN [-0.96, 0.95] WOMEN [-0.94, -0.96]

KING - MAN + QUEEN = WOMEN

Cosine Similarity

Distance = 1 − Cosine Similarity

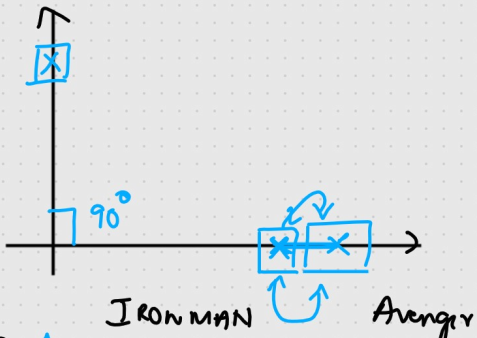


Cosine-Sim = $\cos 45^\circ = \frac{1}{\sqrt{2}} = 0.7071$

Distance = 1 − 0.7071

↳ = 0.29

1 − 1 = 0



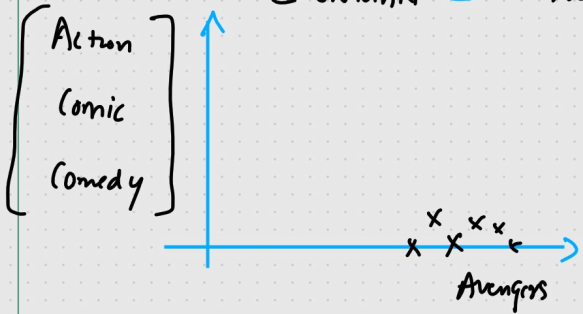
Distance = 1 − 0

= 1

Distance = 1 − cos0

= 1 − 1

= 0//.



ANN, Loss, Optimizers

(*) Word2Vec → CBOW [Continuous Bag of Words]

→ Skipgram.

Pretrained Model

Train A Model For Scratch

① CBOW [Continuous Bag of Words]

CORPUS ÷ DATASET

[iNeuron Company IS Related To DATA SCIENCE] → Vocabulary

300 dimension — 300 ← Window size = 5 ⇒ Feature Representation — OHE

[_ _ ___]

I/p

→ [iNeuron, Company, Related, To]

→ Company, IS, To, DATA

→ Is, Related, DATA, SCIENCE

O/p

IS

Related

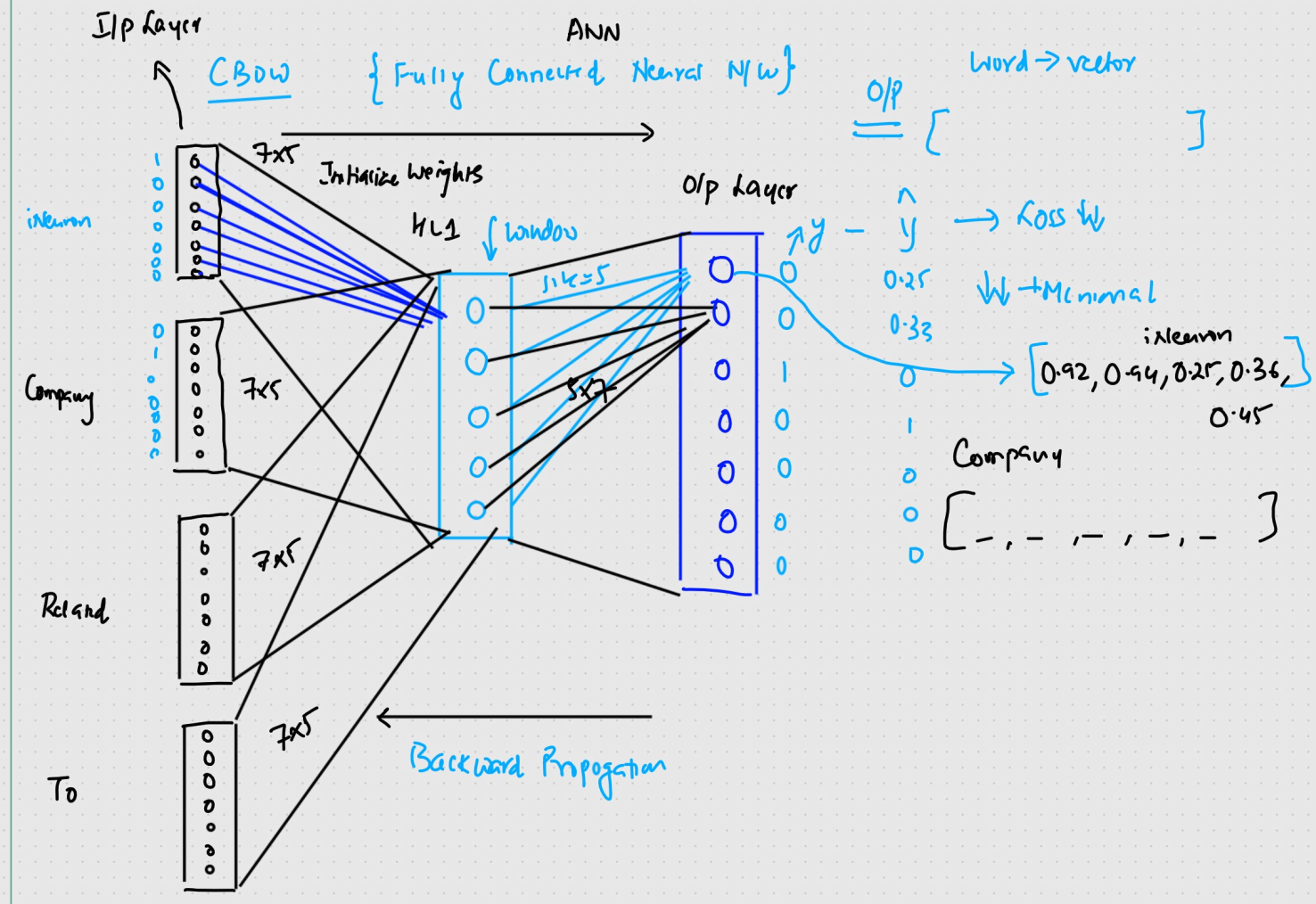
To

iNeuron [1 0 0 0 0 0 0]

Company [0 1 0 0 0 0 0]

Related [0 0 0 1 0 0 0]

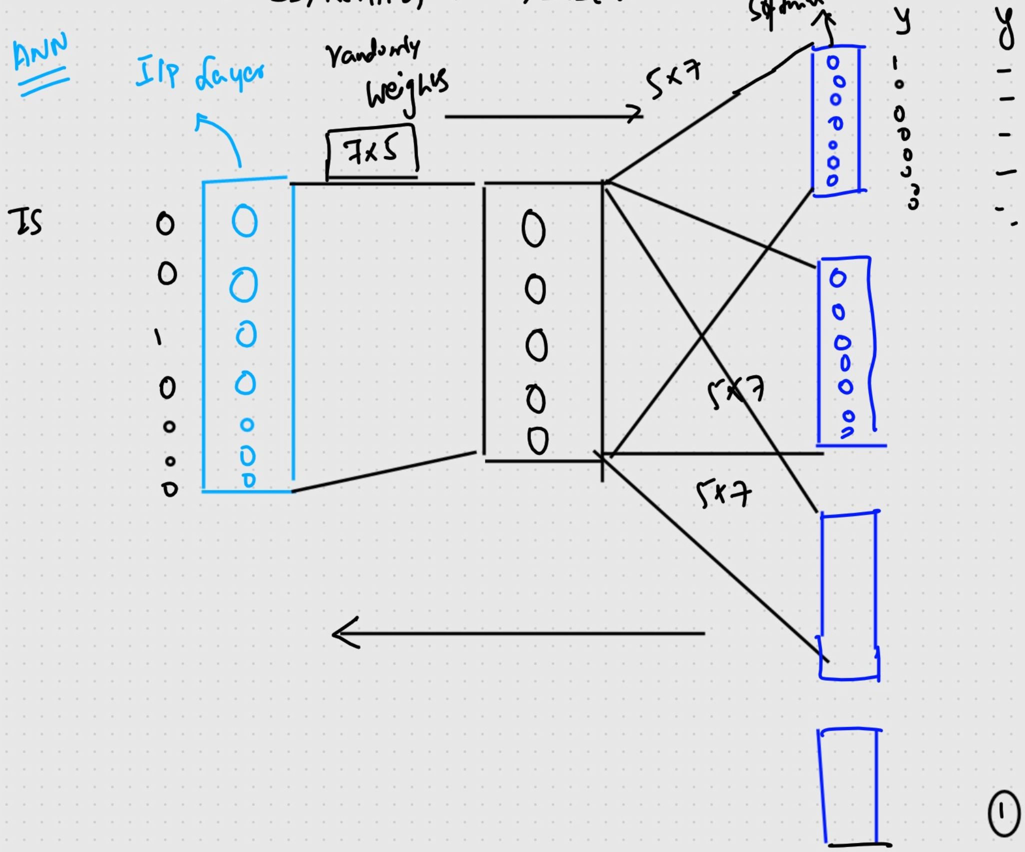
to 0 0 0 0 1 0 0



② Skipgram – Word2Vec

Window size=5

O/p	I/p
→ [iNeuron, Company, Related, To]	IS
→ Company, IS, To, DATA	Related
→ Is, Related, DATA, SCIENCE	To



Loss function ↓

When should we apply CBOW or SkipGram.

{ Small Dataset → CBOW
Huge Dataset → SkipGram }

Improve

CBOW or SkipGram

① Increasing the Training Data

② Increase the window size

– vector dimension is also increasing.

Google Word2vec

Gensim

3 billion words → Google News

feature representation of 300 dimension vectors]

Cricket → [- - - - - - - - - - - - -].

Advantages of Word2Vec

*) Sparse Matrix ⟶ Dense Matrix

*) Semantic Info is getting captured [Monet, good]

*) Vocabulary Size ⟶ Fixed set of dimension vectors
Google Word2Vec [300 dimension]

*) OOV is also solved

⊛ Avg Word2Vec

	Text	O/P
D1	The food is good	1
D2	The food is bad	0
D3	Pizza is Amazing	1

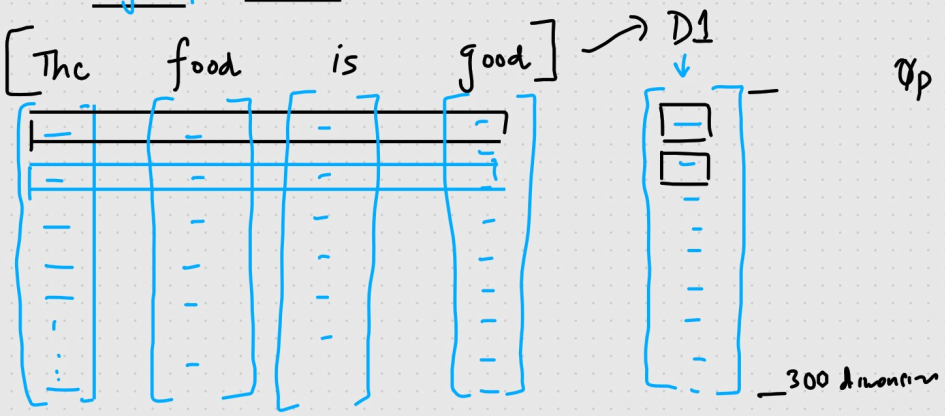
Avg word2vec

[- - - - - - - - -]
[ ]
[ ]

O/P
1

Google pretrained Word2Vec Model

Avg Word2Vec

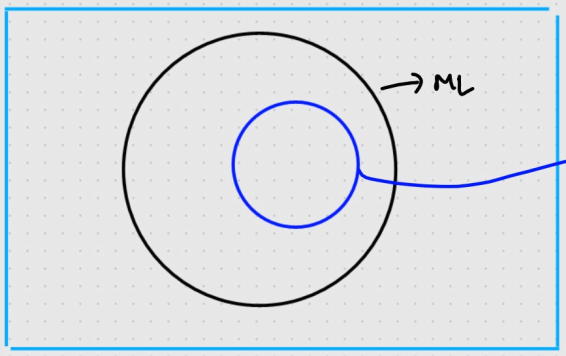


⊛
Gensim, Glove

↳ pretrained Google Word2Vec

↳ Train A Word2Vec From Scratch

Deep learning



Deep learning

1. ANN → Artificial Neural N/W → Classification, Regression
2. CNN → Convolutional Neural N/W → I/P : Images, Video frames → RCNN, MASKED RCNN, Detection, YOLO V5, V6, V7
3. RNN → Recurrent Neural N/W → NLP → NLP, Time Series

Computer Vision
Objection Detection

I/P: Text, Time Series

⇓

{ Word Embedding, LSTM RNN, GRU RNN, Bidirectional LSTM RNN, Encoder Decoder, Transformers, BERT }

FRAMEWORK

TENSORFLOW

End to End Project

② Why Deep Learning Is Becoming popular ?

2005 ⟶ Facebook, Youtube, Whatsapp, Linked, Twitter } ⇒ Social Media platform

⋮

2011-2012

⇓

DATA WAS GENERATED

Exponentially

Image, Text, Documents,

Videos

↳ Big DATA ⟶ HADOOP → unstructured DATA

⇓

2011 ⇒ Cloudera, Hortonworks

Netflix → movie streaming platform

⇓

Recommendation System

⇓

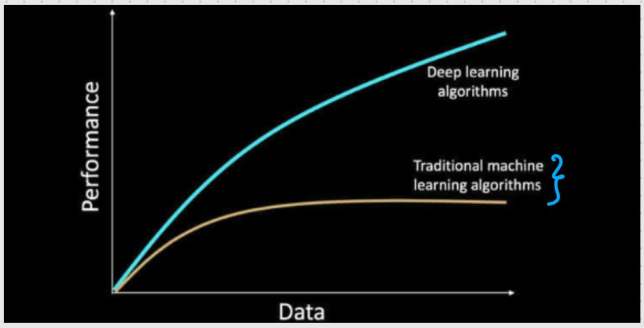
Increase Revenue

2011-2012 :

① Hardware Requirement ⟶ GPU's cost↓↓↓ → NVidia GPU Price↓↓↓

TRAIN MODELS

② Huge amount of data is getting generated → Deep Learning Models Perform well



③ Deep learning is been used in Many Domains

1. Medical
2. Ecommerce
3. Retails
4. Marketing . . . .

④ Frameworks Open source

Community size ↑↑

Tensorflow

⇓

Google

Pytorch

⇓

Facebook

⇓

More Research

③ Perceptron [Artificial Neuron or Neural Network Unit]

① Input layer ✓

② Hidden layer ✓

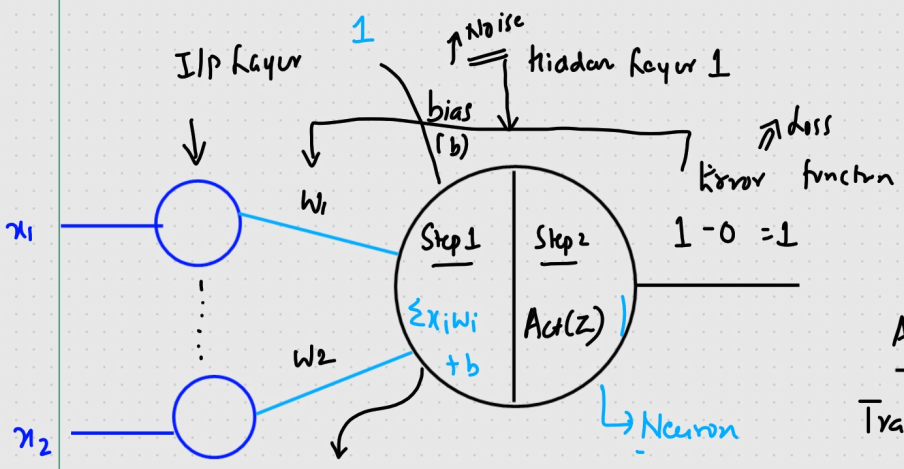
③ Weights ✓

④ Activation function ✓

[Single layered NN] ⇓ Binary classifier

DATASET

	x_1 Id	x_2 No. of study hours	o/p Pass/Fail
→	95	3	0
→	110	4	1
→	100	5	1



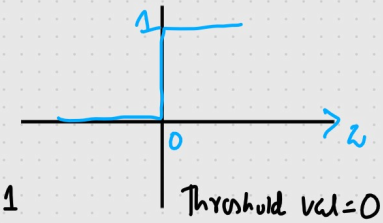
$Z = w_1x_1 + w_2x_2 + b$

$Z = \sum_{i=1}^{n} x_i w_i + b$

Activation function

Transform the output between 0 to 1 / -1 to 1

Step Function

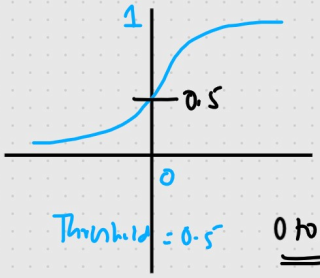


0 to 1

Threshold val = 0

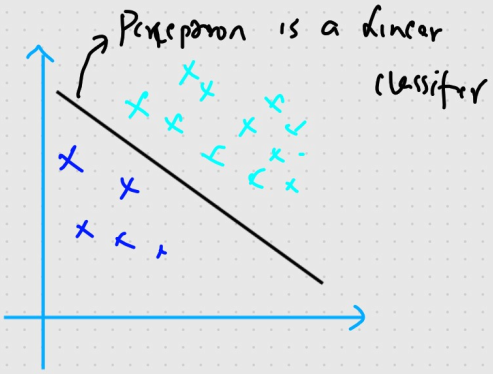
$\begin{cases} 0 & Z \le 0 \\ 1 & Z > 0 \end{cases}$

Sigmoid Function



Threshold = 0.5 — 0 to 1

$\begin{cases} 1 & Z > 0.5 \\ 0 & Z \le 0.5 \end{cases}$



Step 1

$$Z = \sum_{i=1}^{n} w_i x_i + b$$

$$Z = b + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

$$y = mx + c$$

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 \; - - - + \beta_n x_n$$

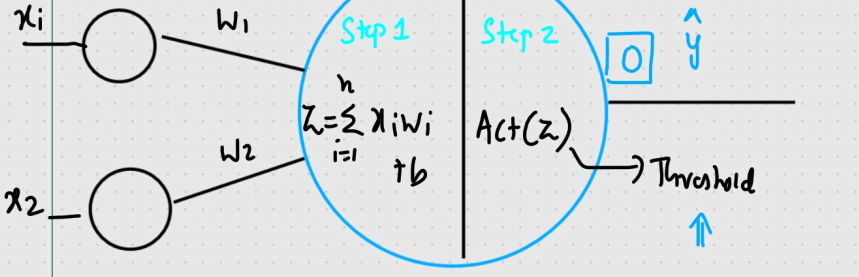
linear problem Statement

Perceptron Models

- Single Layer Perceptron Model
- Multi Layered Perceptron Model [ANN]

Single Layer Perceptron Model

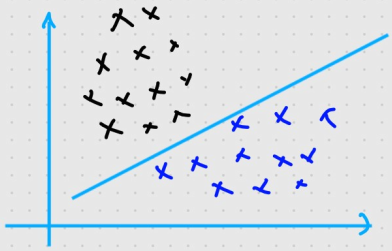
Feed Forward Neural NW



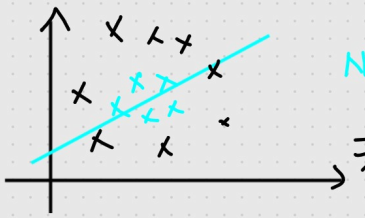
Multi Layered Perceptron Model

1. Forward Propogation
2. Backward "
3. Loss functions
4. Activation functions
5. Optimizers

Linear Separable problem



Binary classifier ⇓ linearly separable



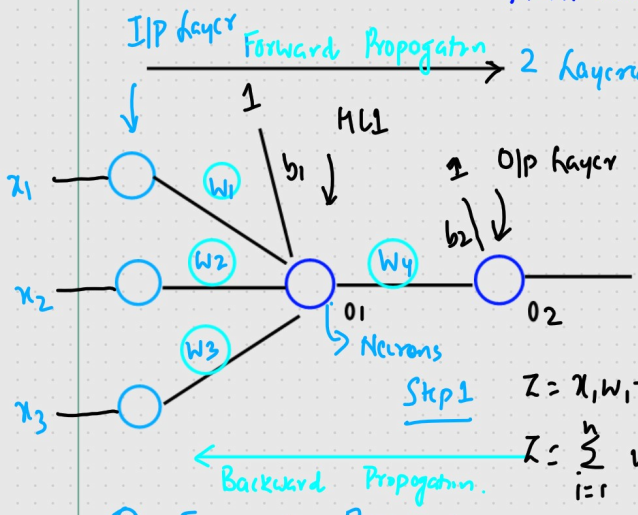
Non linear separable ⇒ Multi Layered NN

5 Multi layered Perceptron Model [Artificial Neural N/W]

① Forward Propogation
② Backward Propogation → Geoffrey Hinton →

③ Loss function
④ Optimizers ✓
⑤ Activation function.

Multi layered NN



x_1	x_2	x_3	Pass/Fail
IQ	Study hours	Play hours	O/p
95	4	4	1
100	5	2	1 ←
95	2	7	0

Error → Loss function = (1 − 0.51129) ≈ 0.49

Step 1 $Z = x_1w_1 + x_2w_2 + x_3w + b_1$

$Z = \sum_{i=1}^{n} w_i^T x_i + b$

② Activation(Z)

W_1 W_2 W_3
[0.01, 0.02, 0.03] [0.001]

① Forward Propogation

Hidden Layer 1

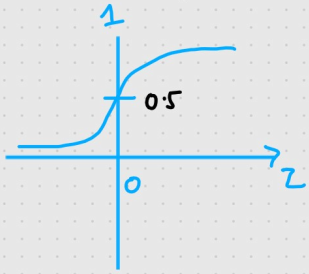
Step 1: $Z = 95 \times 0.01 + 4 \times 0.02 + 4 \times 0.03 + 1 \times 0.01$

$= 1.151$

Step 2: Activation (Z)

Sigmoid.

$= \frac{1}{1+e^{-Z}}$



$f(Z) = \frac{1}{1+e^{-1.151}} = 0.759$

$O_1 = 0.759$

Hidden Layer 2

$W_4 = 0.02$

$b_2 = 0.03$

Step 1: $Z = O_1 * W_4 + b_2$

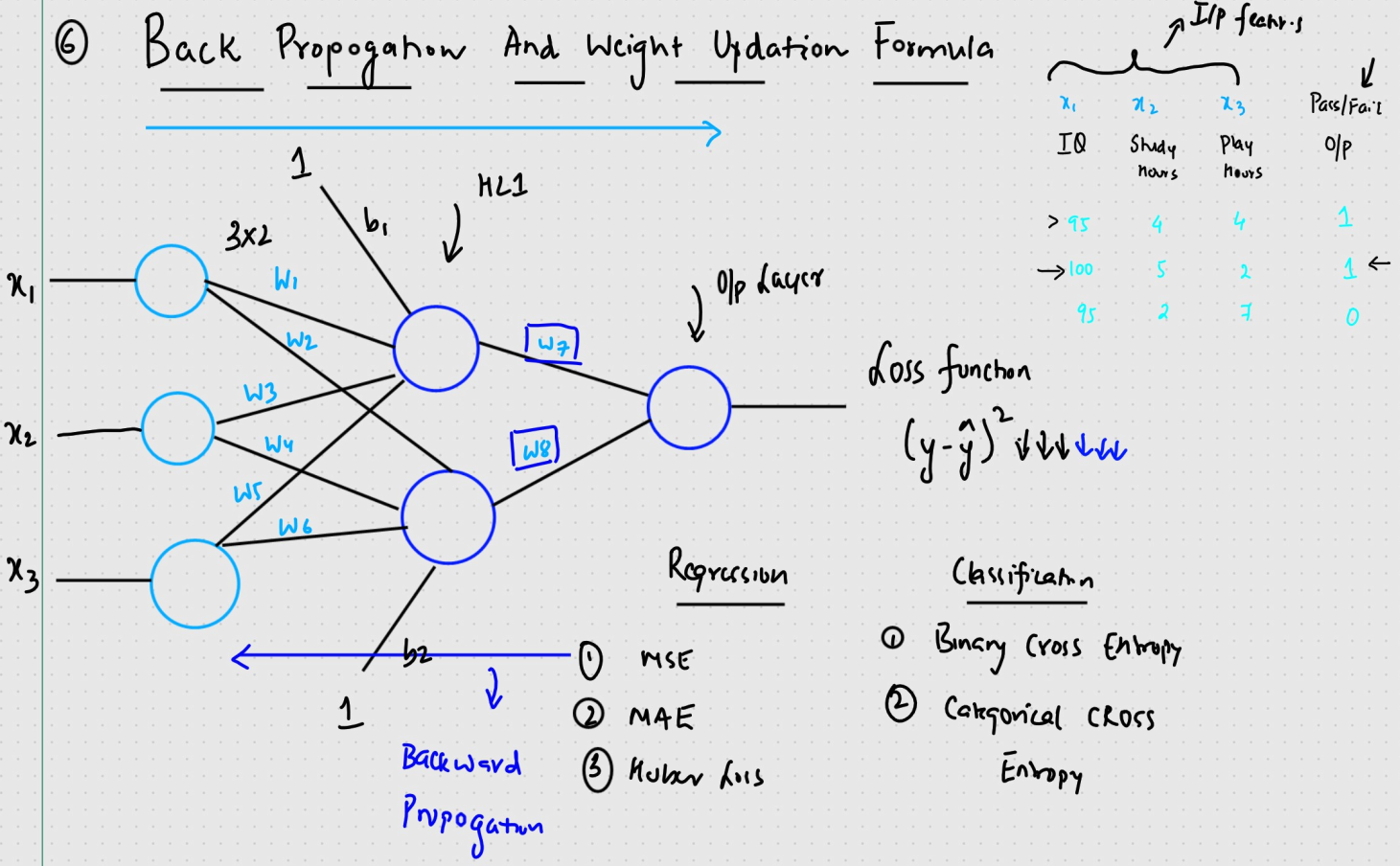
$= 0.759 * 0.02 + 0.03$

$= 0.04518$ //

Step 2 : Activation(z) $\frac{1}{1+e^{-(0.04518)}} = 0.511129$

$O_2 = 0.511129 \Rightarrow \hat{y}$

⑥ Back Propogation And Weight Updation Formula



I/p features

x_1	x_2	x_3	Pass/Fail
IQ	Study hours	Play Hours	O/p
> 95	4	4	1
→ 100	5	2	1 ←
95	2	7	0

Loss function $(y-\hat{y})^2$

Regression

① MSE
② MAE
③ Huber Loss

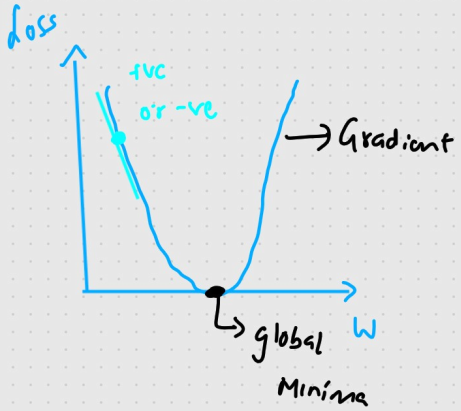
Classification

① Binary Cross Entropy
② Categorical CROSS Entropy

Weight updation Formula

$$W_{7_{new}} = W_{7_{old}} - \eta \left[\frac{\partial L}{\partial W_{7old}} \right]$$ ← slope

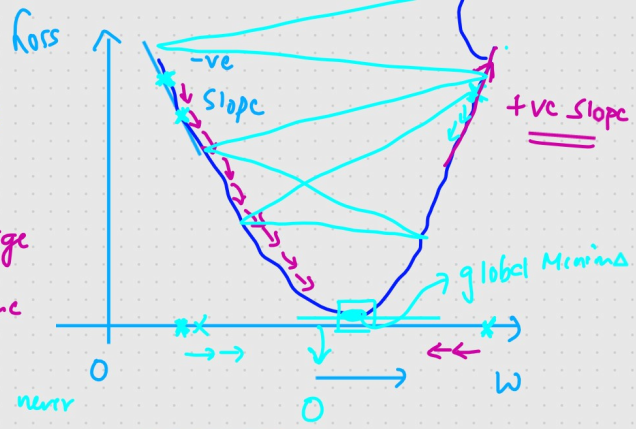
$$W_{8new} = W_{8old} - \eta \frac{\partial L}{\partial W_{8old}}$$ ← Learning Rate



$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$ ⇒ Weight Updation Formula.

Gradient Descent ↗ Optimizers

Optimizers : To reduce the loss value



η = large value ⇓ It may never converge

$W_{new} = W_{old} - \eta\,(-ve)$

$= W_{old} + \eta\,(+ve)$

η → Learning Rate

$W_{new} >> W_{old}$

0.001 → Small value.

$W_{new} = W_{old} - \eta\,(+ve)$

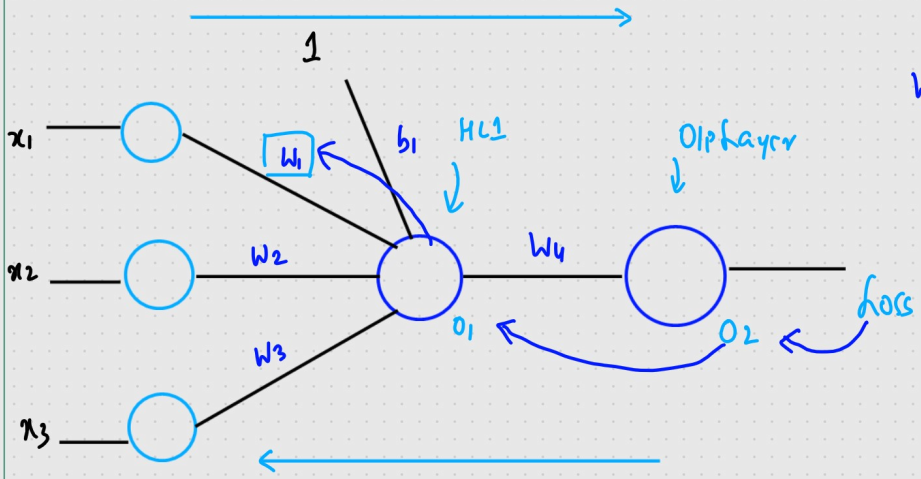
$= W_{old} - \eta\,(+ve)$

$W_{new} << W_{old}$

When W reaches global Minima

$W_{new} = W_{old}$

⑦ Chain Rule of Derivative



$$W_{new} = W_{old} - \eta \frac{\partial h}{\partial W_{old}}$$

$$W_{4_{new}} = W_{4_{old}} - \eta \left[\frac{\partial h}{\partial W_{4_{old}}}\right]$$

$$\frac{\partial h}{\partial W_{4old}} = \frac{\partial h}{\partial O_2} * \frac{\partial O_2}{\partial W_{4old}} \Rightarrow \text{Chain Rule of Derivation}$$

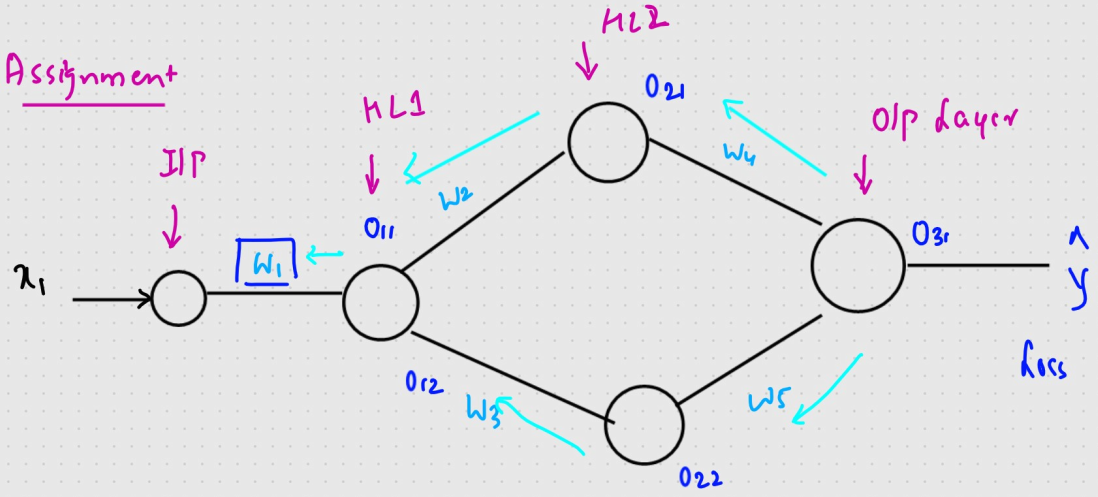
$$W_{1new} = W_{1old} - \eta \left[\frac{\partial h}{\partial W_{1old}}\right]$$

$$\frac{\partial h}{\partial W_{1old}} = \frac{\partial h}{\partial O_2} * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}$$

W_{2new}

W_{3new}.

Assignment



$$W_{1new} = W_{1old} - \eta \left[\frac{\partial h}{\partial W_{1old}}\right]$$

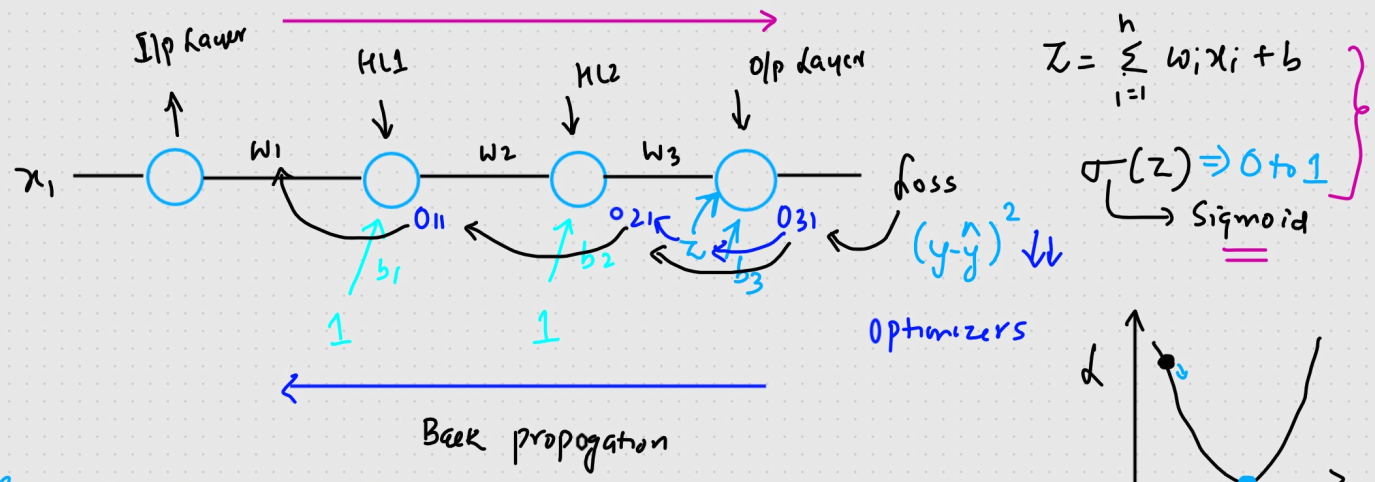
$$\Downarrow$$

$$\frac{\partial h}{\partial W_{1old}} = \left[\frac{\partial h}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} * \frac{\partial O_{11}}{\partial W_{1old}}\right]$$

$$+$$

$$\left[\frac{\partial h}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial O_{12}} * \frac{\partial O_{12}}{\partial W_{1old}}\right]$$

⑧ Vanishing Gradient Problem And Activation functions $\phi(z)$

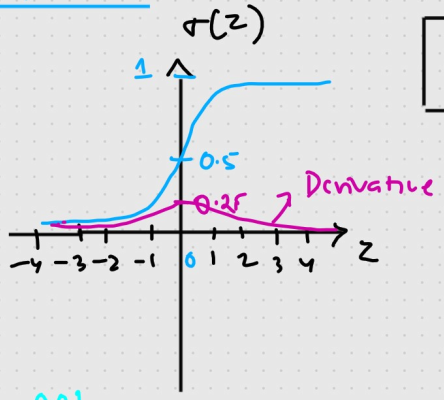


$$z = \sum_{i=1}^{n} w_i x_i + b$$

$\sigma(z) \Rightarrow 0$ to 1 → Sigmoid

⑧ Sigmoid Activation function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$0 \le \sigma(z) \le 1$$

Vanishing Gradient Problem.

Derivative $(\sigma(z))$

$$0 \le \sigma(z) \le 0.25$$

$$W_{1\,new} = W_{1\,old} - \eta \left[\frac{\partial L}{\partial W_{1\,old}}\right]$$

η = 0.01

$\frac{\partial L}{\partial W_{1\,old}}$ ⇒ small value

$\Rightarrow W_{1\,new} \approx W_{1\,old}$ (Equal)

$\frac{\partial L}{\partial W_{1\,old}}$ ⇒ small value

$[\ ] \quad [0-0.25] * [0-0.25] * [0-0.25]$ ⇒ 0.000001 ⇒ Smaller value

$$\frac{\partial L}{\partial W_{1\,old}} = \frac{\partial L}{\partial O_{31}} * \left[\frac{\partial O_{31}}{\partial O_{21}}\right] * \frac{\partial O_{21}}{\partial O_{11}} * \frac{\partial O_{11}}{\partial w_{11}}$$

$$O_{31} = \sigma\left(\overbrace{w_3 * O_{21} + b_3}^{z}\right) \qquad z = w_3 * O_{21} + b_3$$

$$O_{31} = \sigma(z)$$

$$\frac{\partial O_{31}}{\partial O_{21}} = \frac{\partial(\sigma(z))}{\partial(z)} * \frac{\partial z}{\partial O_{21}}$$ (Sigmoid) { Chain Rule }.

$0 \leq \sigma(z) \leq 0.25$ * $\frac{\partial((w_3 * o_{21}) + b_3)}{\partial(o_{21})}$

⇓

Derivative of Sigmoid

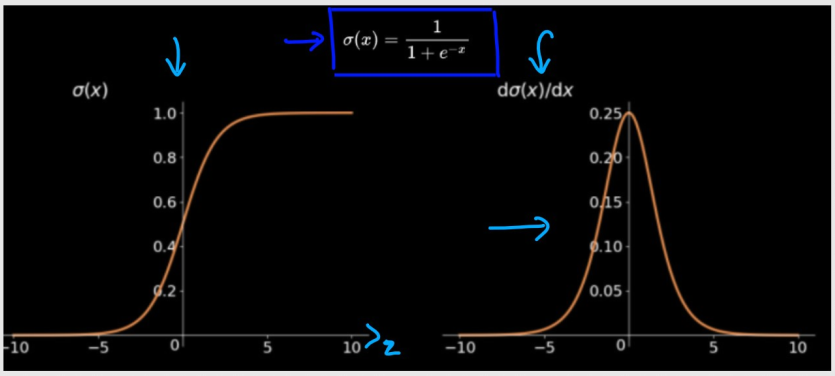
$$\frac{\partial o_{31}}{\partial o_{21}} = 0 \leq \sigma(z) \leq 0.25 * w_{3_{old}}$$

(*) To fix this problem Researchers started exploring other Activation function

(*) Tanh (*) Relu (*) Prelu (*) Swiss

Activation Functions

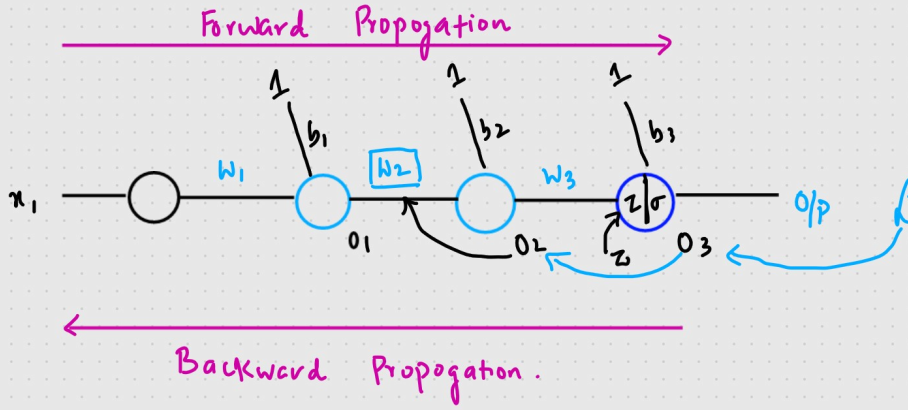
① Sigmoid Activation function [0 to 1] $z = \sum_{i=1}^{n} w^T x + b$

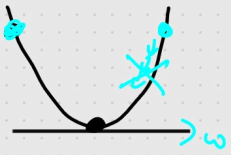


$\sigma(z) \Rightarrow$ 0 to 1

$\phi(z) \Rightarrow$

$\frac{\partial \sigma(z)}{\partial z} = 0$ to 0.25





$$w_{2new} = w_{2old} - \eta \left[\frac{\partial h}{\partial w_{2old}} \right]$$

η: 0.001

$\frac{\partial h}{\partial w_{2old}}$: 0.00001

$\Rightarrow w_{2new} \approx w_{2old} \Rightarrow$

$$\frac{\partial h}{\partial w_{2old}} = \frac{\partial h}{\partial o_3} * \left[\frac{\partial o_3}{\partial o_2} \right] * \frac{\partial o_2}{\partial w_2}$$

$0.20 * 0.01 \times$

Let $z = (o_2 * w_3) + b_3$

$$\frac{\partial o_3}{\partial o_2} = \frac{\partial(\sigma(z))}{\partial z} * \frac{\partial z}{\partial o_2}$$

[0 to 1]

$$= [0 - 0.25] * \frac{\partial[(o_2 * w_3) + b_3]}{\partial o_2}$$

$$= [0 - 0.25] * w_3 \Rightarrow \text{Small value} \Rightarrow$$

Advantages

① Binary classification Suitable.

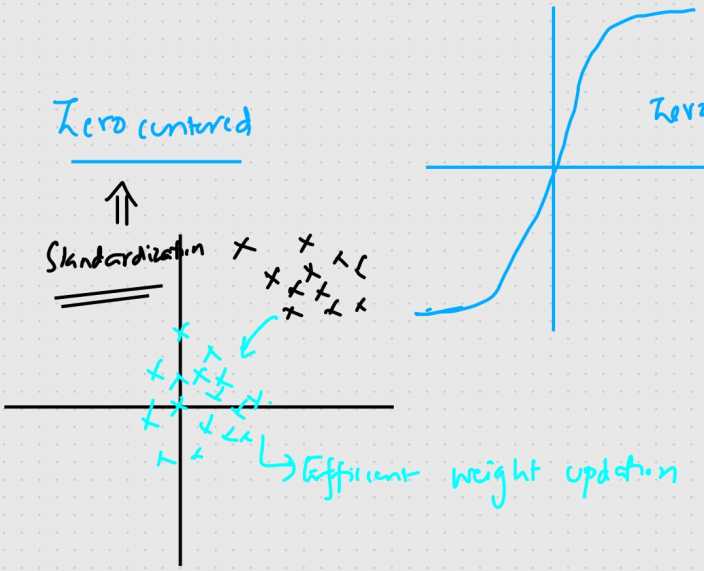
② Clear prediction i.e very close 1 or 0

Disadvantages

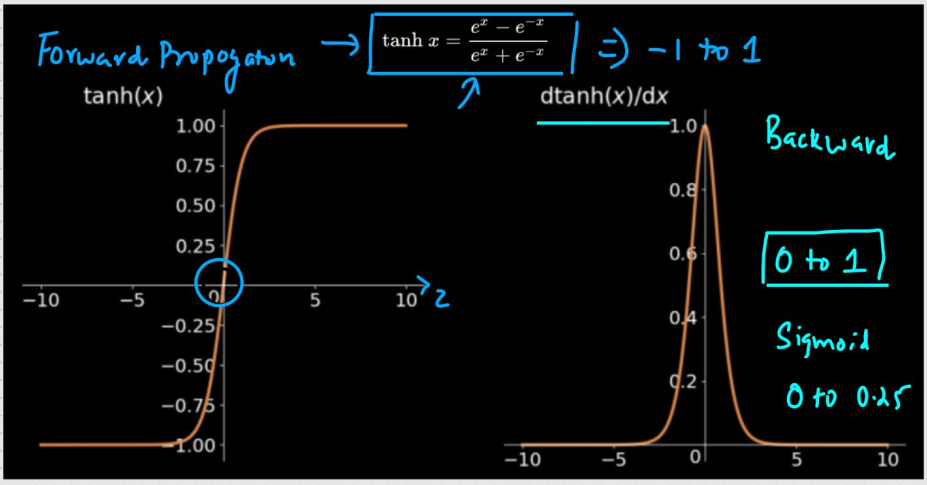
① Prone to vanishing Gradient Problem.

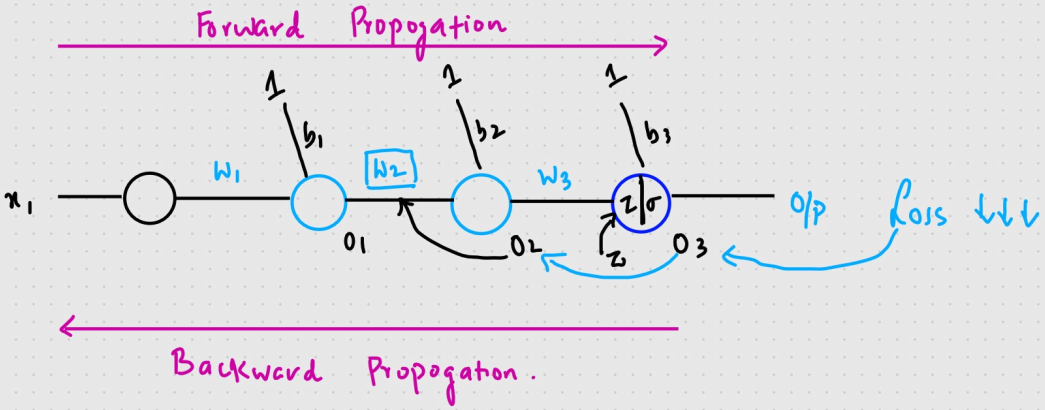
② Function output is not Zero centred ⇒ Efficient weight updation

③ Mathematical operation are relatively time consuming



② Tanh Activation Function





$$\frac{\partial h}{\partial w_{2old}} = \frac{\partial h}{\partial o_3} * \frac{\partial o_3}{\partial o_2} * \frac{\partial o_2}{\partial w_2}$$

$0.20_{11} * 0.01 \times$

Let $z = (o_2 * w_3) + b_3$

$$\frac{\partial o_3}{\partial o_2} = \frac{\partial(\tanh(z))}{\partial z} * \frac{\partial z}{\partial o_2}$$

[0 to 1]

$$= [0-1] * \frac{\partial[(o_2 * w_3) + b_3]}{\partial o_2}$$

$$= [0-1] * w_3 \Rightarrow \text{Small value} \Rightarrow$$

Advantages

① Zero Centric ⇒ Weight updation is Efficient

Disadvantages

① Prone to Vanishing Gradient Problem

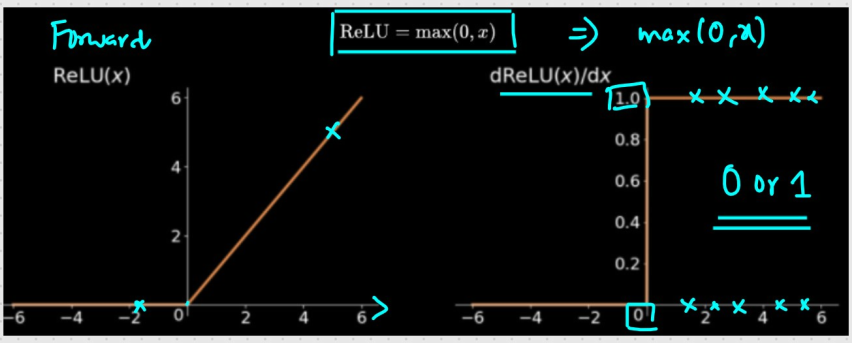
② Time complexity

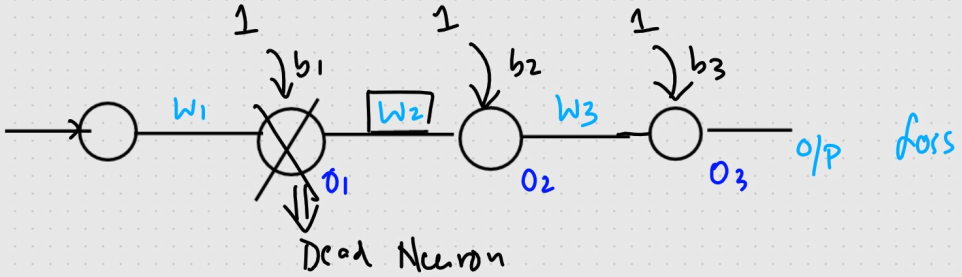
③ Relu Activation Function

[Rectified Linear Unit].

Tanh ⇒ 0 to 1

Sigmoid ⇒ 0 to 0.25





$$\frac{\partial h}{\partial w_{2old}} = \frac{\partial h}{\partial o_3} * \frac{\partial o_3}{\partial o_2} * \frac{\partial o_2}{\partial w_2}$$

0.20_{11} * 0.01 ×

Let $z = (o_2 * w_3) + b_3$

$$\frac{\partial o_3}{\partial o_2} = \frac{\partial(\text{relu}(z))}{\partial z} * \frac{\partial z}{\partial o_2} \qquad [0 \text{ to } 1]$$

$$= [0 \text{ or } 1] * \frac{\partial[(o_2 * w_3) + b_3]}{\partial o_2}$$

$$= [0 \text{ or } 1] * w_3 \quad \Rightarrow \text{Small value} \Rightarrow$$

(-ve, +ve)

Derivative of

If Relu output is $\boxed{1}$ $\Rightarrow$ Weight updation will happen

If Relu output is $\boxed{0}$ $\Rightarrow$ Dead Neuron

$$W_{2new} = W_{2old} - \eta \left[\frac{\partial h}{\partial w_{2old}}\right] \Rightarrow 0$$

If Derivative of Relu(z) is 0

$$\boxed{W_{2new} \approx W_{2old}} \Rightarrow \text{Dead Neuron}$$

If $z = +ve$ $\quad \frac{\partial\, \text{Relu}(z)}{\partial z} = 1$

If $z = -ve$ $\quad \frac{\partial\, \text{Relu}(z)}{\partial z} = 0$

Advantages

① Solving Vanishing Gradient Problem

② Max(0, x) → Calculation is superfast. The Relu function has a linear Relationship

③ It is much faster than Sigmoid or Tanh.

Disadvantages

① Dead Neuron

② Relu function O/P

(0, x) ⇒ 0 or +ve number

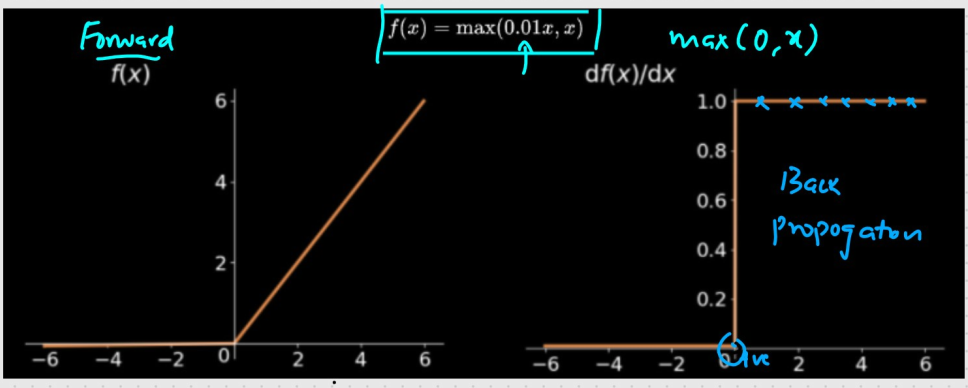
⇓

It is not Zero centric

④ Leaky Relu And Parametric Relu

⇓

max(αx, x) → hyperparameter α = alpha = 0.01, 0.02, 0.03



Relu ⟶ Dead Neuron ⟶ Dead Relu Problem

Advantages

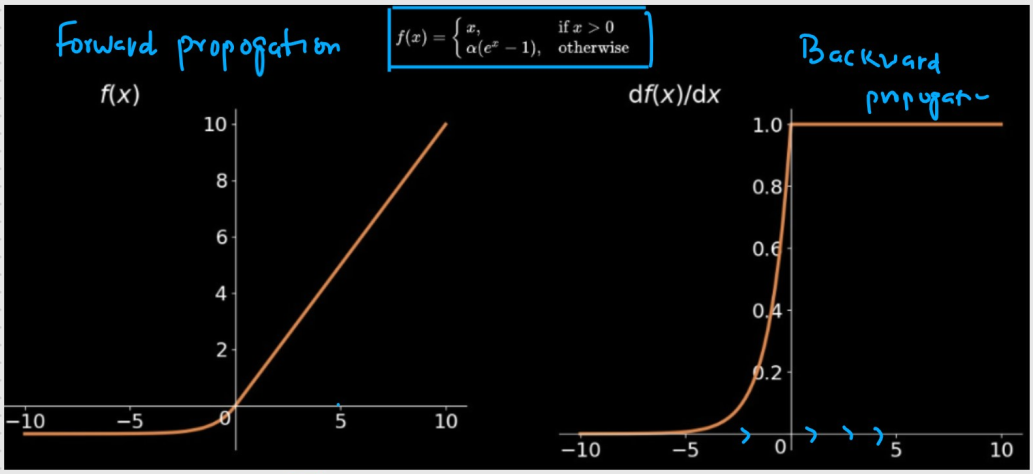
① Leaky Relu has all the advantage of Relu

② It removes the Dead Relu Problem

Disadvantage

① It is not Zero centric

⑤ ELU (Exponential Linear Units)



[It is used to solve Relu problems]

Advantages

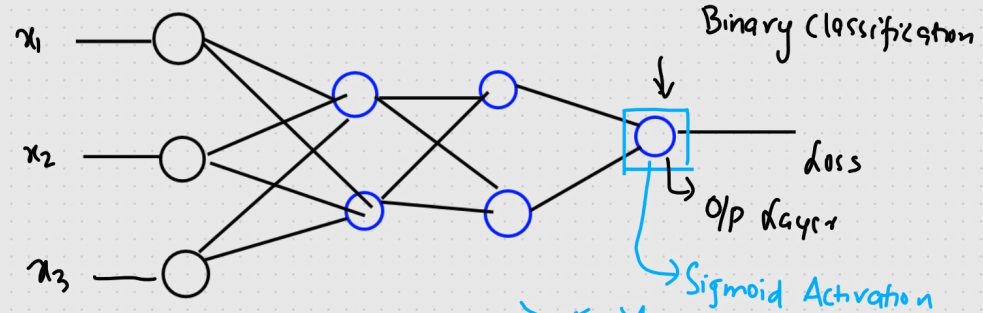
① No Dead Relu Issues

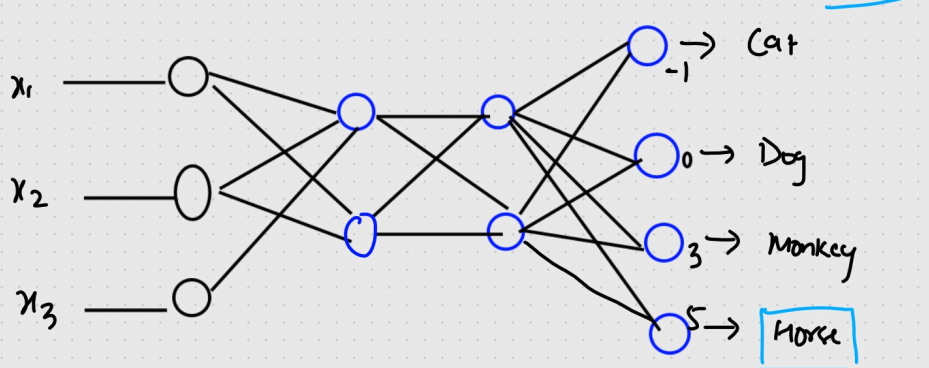
② Zero centered

Disadvantage

1) Slightly more computationally Intensive.

⑥ Softmax Activation function [Multiclass classification Problem]





Softmax Activation

$$\text{Softmax} = \frac{e^{y_i}}{\sum_{k=0}^{n} e^{y_k}}$$

$y_i = 0 * w + b$

Softmax $\Rightarrow$ Cat $= \frac{e^{-1}}{e^{-1+0+3+5}} = 0.00033$

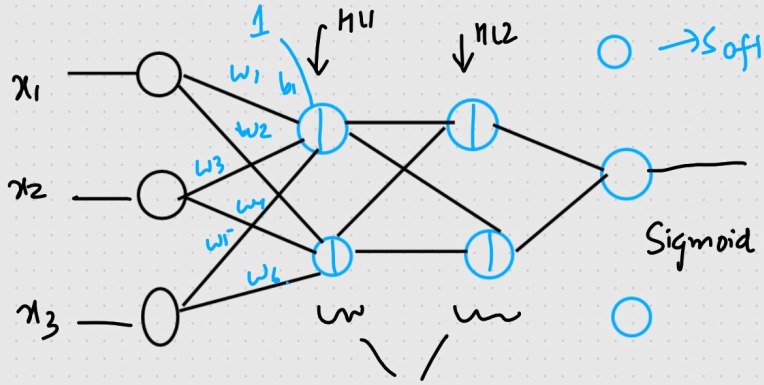
Dog $= \frac{e^{0}}{e^{-1+0+3+5}} = 0.0024$

Monkey $= \frac{e^{3}}{e^{-1+0+3+5}} = 0.0183$

Horse $= \frac{e^{5}}{e^{-1+0+3+5}} = 0.1353$

Pr(Horse) $= \frac{0.1353}{0.00033 + 0.0024 + 0.0183 + 0.1353} \approx 86\%$

(7) Which Activation Function To Use When ?



↓ O/P layer

→ Softmax ⇒ Multiclass Classification

Sigmoid → Binary classification

Hidden Layers → Sigmoid or Tanh

⇓

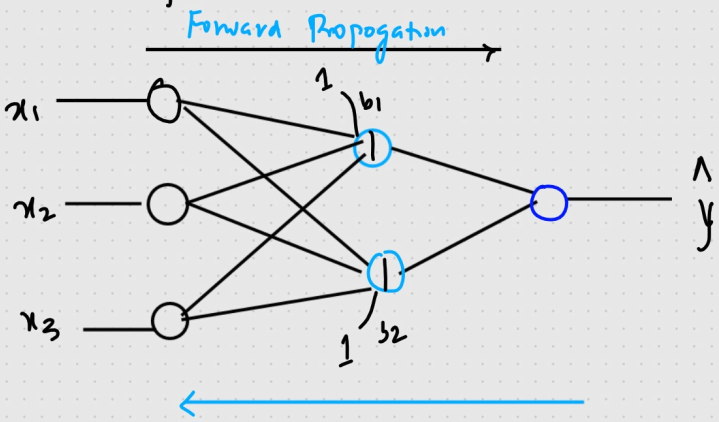
Vanishing Gradient Problem

Relu or its variant

Prelu, Leaky Relu,

Elu

Loss function And Cost Function



$$\text{Cost fn} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad \text{(Error)}$$

$$\text{Loss} = (y - \hat{y})^2$$

Optimizers

Gradient Descent

x_1	x_2	x_3	O/P ($\hat{y}$)
–	–	–	0
–	–	–	1
–	–	–	0
–	–	–	1

Loss function

$$MSE = (y - \hat{y})^2$$

Costfunction

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

ANN

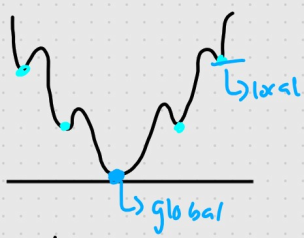
- Classification
- Regression
  1. Mean Squared Error (MSE)
  2. Mean Absolute Error (MAE)
  3. Huber Loss
  4. RMSE

① Mean Squared Error (MSE)

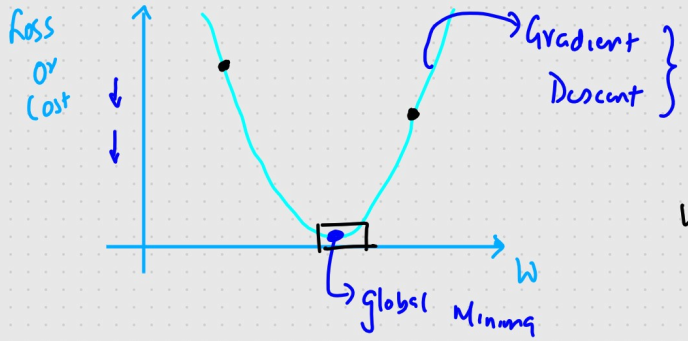
$$\text{loss function} = (y - \hat{y})^2$$

$$\text{Cost fn} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

↳ Quadratic Equations



Weight updation



$$W_{new} = W_{old} - \eta \left[\frac{\partial h}{\partial W_{old}} \right]$$

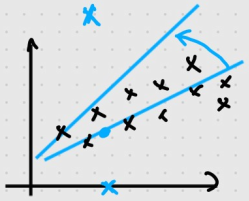
Slope

Advantages

1. MSE is Differentiable
2. It has 1 local or global Minima
3. It converges faster

Disadvantages

1. Not Robust to outliers



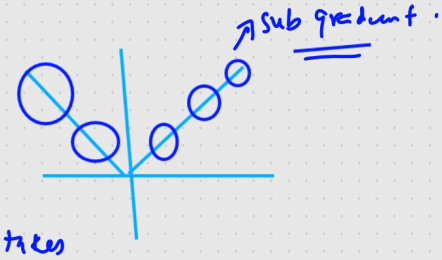
penalizing the Error

Cost function = $\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

2. Mean Absolute Error (MAE)

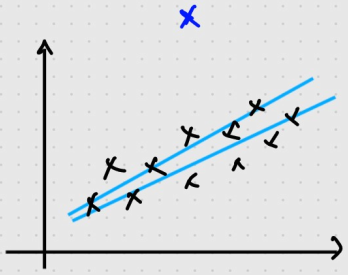
Loss fn : $|y - \hat{y}|$

Cost fn = $\frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$



Advantages

1. Robust to outliers



Disadvantages

1. Convergence usually takes time in MAE

3. Huber loss

1. MSE
2. MAE

MSE

No outlier

Hyperparameter

Cost fn = $\begin{cases} \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2 & \text{if } |y - \hat{y}| \leq \boxed{\delta} \end{cases}$

$$\begin{cases} \delta|y-\hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

⇓

MAE

④ RMSE (Root Mean Squared Error)

$$= \text{Cost function} = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}}$$

Advantages

Disadvantages

② Loss or Cost function For Classification Problems

Classification ⟶ CROSS Entropy
- ⟶ Binary CROSS Entropy {Binary}
- ⟶ Categorical CROSS Entropy {Multiclass}
- ⟶ Sparse Categorical CROSS Entropy ↳{Multiclass}

① Binary CROSS Entropy

Log Loss

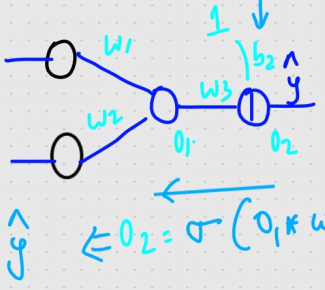
⇓

$$\text{loss} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$

y = Actual value

$\hat{y}$ = Predicted value

$$\text{loss} = \begin{cases} -\log(1-\hat{y}) & \text{if } y = 0 \\ -\log(\hat{y}) & \text{if } y = 1 \end{cases}$$



Binary classification

Sigmoid

$$\hat{y} = \frac{1}{1+e^{-z}}$$ ⇒ Sigmoid Activation function

② Categorical CROSS Entropy (Multiclass classification)

OHE → One Hot Encoding

	f_1	f_2	f_3	O/P	Good (j=1)	Bad (j=2)	Neutral (j=3)
→	2	3	4	Good	1	0	0
→	5	6	7	Bad	0	1	0
→	8	9	10	Neutral	0	0	1

C = No of categories

i : 1 to n

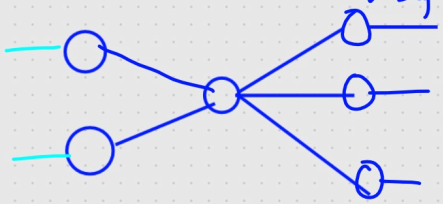
$$L(x_i, y_i) = -\sum_{j=1}^{C} y_{ij} * \ln(\hat{y}_{ij})$$

Actual value ⇐ $y_{ij} = \begin{bmatrix} y_{11} & y_{12} & y_{13} & \cdots & y_{1C} \\ y_{21} & y_{22} & y_{23} & \cdots & y_{2C} \end{bmatrix}$

$$y_{ij} = \begin{cases} 1 & \text{if the element is in the class} \\ 0 & \text{Otherwise} \end{cases}$$

Prediction ⇐ $\hat{y}_{ij}$ ⇒ Softmax Activation $= Sof(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

↓ softmax Activation



O/p $\hat{y}_{ij}$ = Probabilities

[0.1, 0.2, 0.3, 0.2, 0.2] ← 1

Categorical Cross Entropy ⇒ [0.2, 0.3, 0.5]

⇓

[This also gives the probability of other categories]

③ Sparse Categorical Cross Entropy

[0 (0.2), 1st (0.3), 2nd (0.5)] → Categories

⇓

2nd Index ⇒ O/p

[0 (0.2), 1 (0.3), 2nd (0.1), 3rd (0.2), 4th (0.2)] ⇒ 1th Index

⇓

Category ⇒ O/p

Disadvantage

① Loosing info about the probability of other category.

④ Right Combination

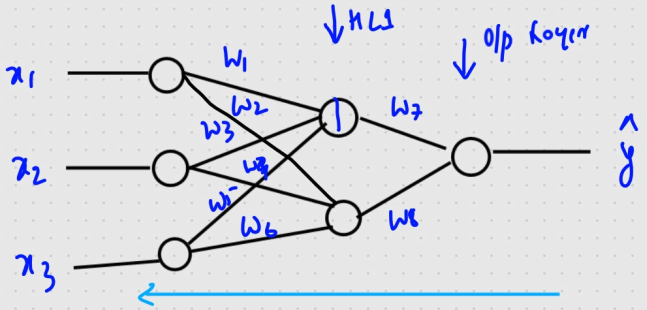
Activation applied ⇓

	Hidden Layers	O/p Layer	Problem Statement	Loss function
①	Relu or its Variants	Sigmoid	Binary classification	Binary Cross Entropy
②	Relu or its Variants	Softmax	Multi class	Categorical or Sparse CE
③	Relu or its Variants	linear	Regression	MSE, MAE, Huber loss, RMSE

Optimizers

1. Gradient Descent
2. Stochastic Gradient Descent (SGD)
3. Mini batch SGD
4. SGD with Momentum
5. Adagrad and RMSPROP
6. Adam Optimizers

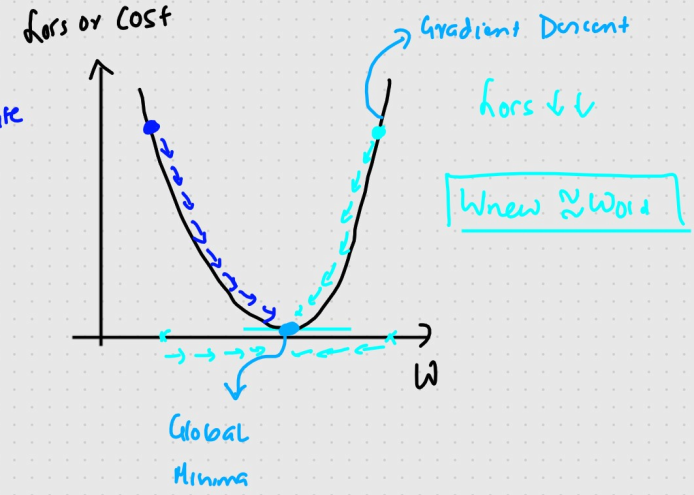
1. Gradient Descent Optimizer



Backward propogation

$$loss = [\quad] \downarrow\downarrow\downarrow$$

Optimizers



Learning Rate

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

$W_{new} \approx W_{old}$

MSE

$$\text{loss fn} = (y - \hat{y})^2 \qquad \text{Cost fn} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Dataset = 1000 Data points

Epochs, Iteration

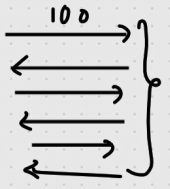
1 Epoch = 1 Iteration

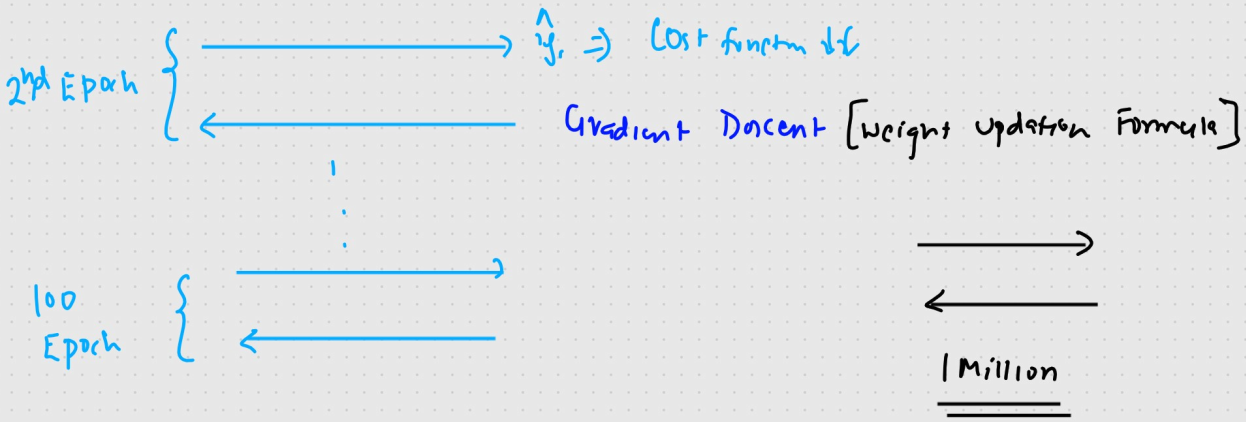
10 Iteration

$100/10 = 100$

1 Epoch { 1000 datapoint → $\hat{y}_i$ ⇒ Cost function ↓↓ ← Weights will get updated

Gradient Descent optimizer [Weight updation]





Advantages

① Convergence will happen.

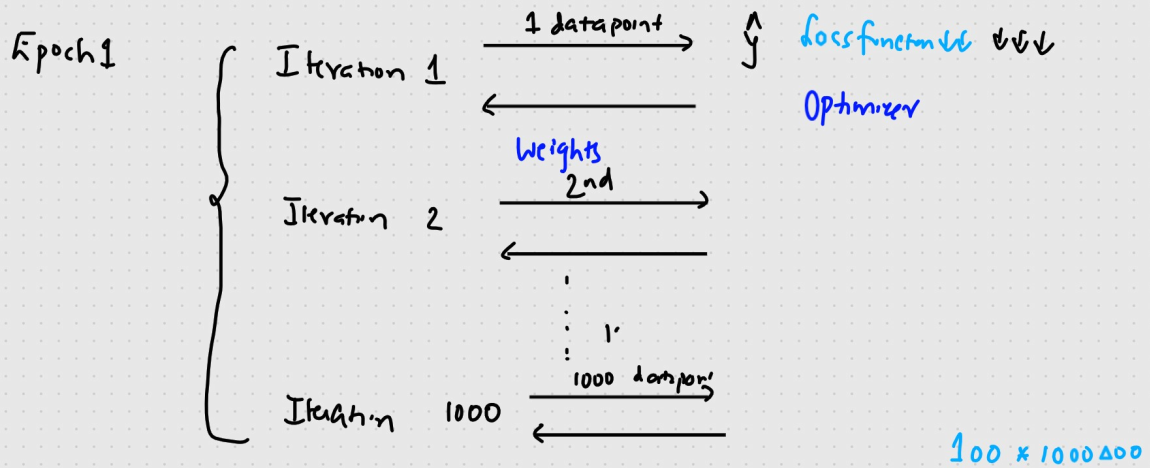
Disadvantage

① Huge Resource RAM, GPU ⇓ Resource Intensive.

② Stochastic Gradient Descent (SGD)

1000 datapoint

Epochs & Iteration



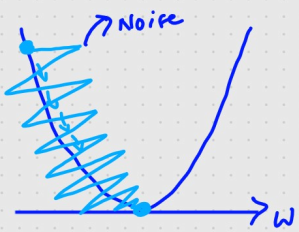
100 × 1000 = 100000

Epochs

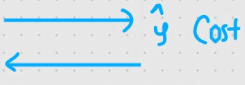
1000000 datapoint

Advantage

Ⓐ Solve Resource Issue



Disadvantage



Ⓑ Time Complexity ↑↑

Ⓒ Convergence will also take more time.

Ⓓ Noise gets Introduced

③ Mini Batch SGD

No.of iterations = $\frac{100000}{1000}$ = 100 iterations

Epoch, Iteration, Batch-size

Datapoints = 100000

batch_size = 1000

MSE

Cost fn = $\sum_{i=1}^{1000} (y_i - \hat{y}_i)^2$ ↓↓

Epoch 1

Iteration 1 → 1000 →

← change the weights

Optimizer ⇒ Mini Batch SGD

Iteration 2 → 1000 →

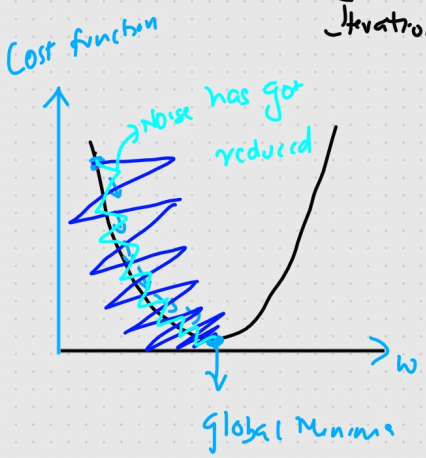
Iteration 3 →

⋮

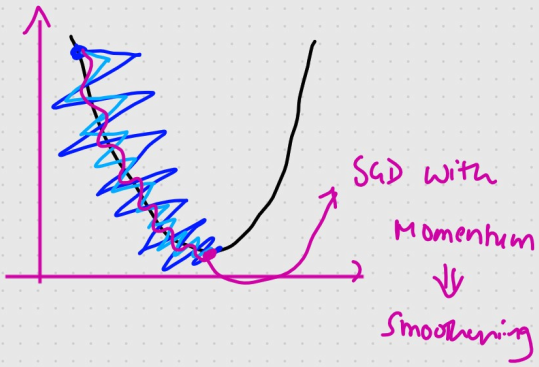
Iteration 100 →

8gb 1

16gb 500



Batch Size



Advantages

① Convergence speed will increase

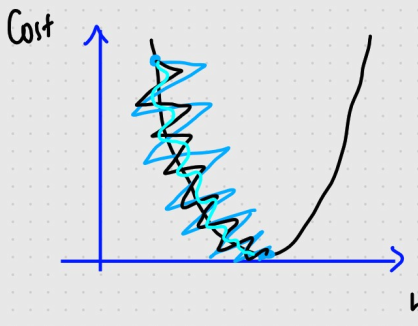
② Noise will be less when compared to SGD

③ Efficient Resource usage (RAM)

Disadvantage

① Noise still exists

④ SGD With Momentum



Weight Updation formula

learning rate

$$W_{new} = W_{old} - \eta \frac{\partial h}{\partial W_{old}}$$

$$b_{new} = b_{old} - \eta \frac{\partial h}{\partial b_{old}}$$

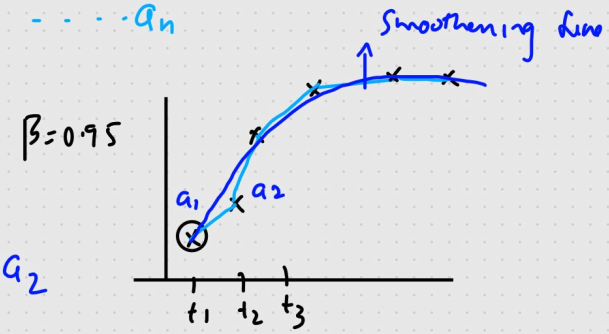
$$W_t = W_{t-1} - \eta \frac{\partial h}{\partial W_{t-1}}$$

↓↓↓

Exponential Weight Average {Smoothening} ⇒ ARIMA, SARIMAX

Time = t_1 t_2 t_3 t_4 - - - t_n Time Series

Values = a_1 a_2 a_3 a_4 - - - - a_n



$$V_{t_1} = a_1$$

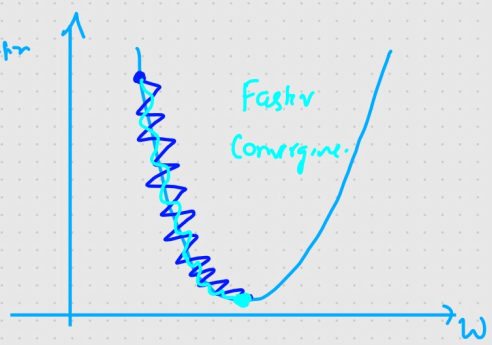
$$V_{t_2} = \boxed{\beta} * V_{t_1} + (1-\beta) * a_2$$

$\beta = 0$ to 1

$$= 0.95 * a_1 + (0.05)\, a_2$$

$$V_{t_3} = \beta * V_{t_2} + (1-\beta) * a_3$$

$$= 0.95 \left[0.95 * a_1 + (0.05)\, a_2 \right] + (0.05) * a_3$$



Advantage

① Reduces the noise

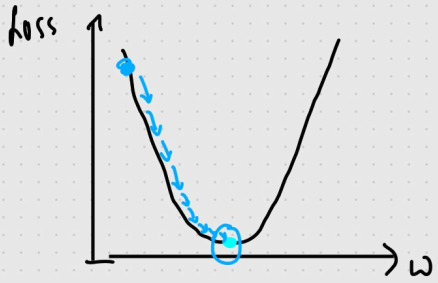
② Quick Convergence

⑤ Adagrad ÷ Adaptive Gradient Descent

η = fixed ⇒ Dynamic learning Rate

$$W_t = W_{t-1} - \eta \frac{\partial h}{\partial w_{t-1}}$$

→ Learning Rate = 0.001



As the convergence happen the learning rate should change

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

→ Epsilon ≈ small value

$\eta' \downarrow\downarrow$ $\alpha_t \uparrow\uparrow$

$\alpha_t \uparrow\uparrow$

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial h}{\partial w_t}\right)^2$$

0.00001 ≈ 0

$$W_t = W_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

⇓

$W_t \approx W_{t-1}$

$t=1 \rightarrow t=2 \rightarrow t=3$

$\eta = 0.01$ $\eta = 0.005$ $\eta = 0.003$

Disadvantage

① η' → Possibility to become a very small value ≈ 0

⑥ Adadelta And RMSPROP

Exponential Weight Average

$\beta = 0.95$ $\quad S_{dw_t} = 0$

$$S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial h}{\partial w_{t-1}}\right)^2$$

$$\eta' = \frac{\eta}{\sqrt{S_{dw_t} + \epsilon}}$$

[Dynamic LR + Smoothening EWA]

$$w_t = w_{t-1} - \eta' \frac{\partial h}{\partial w_{t-1}}$$

⑦ Adam Optimizer

SGD with Momentum + RMSPROP [Dynamic LR + Smoothening]

$$w_t = w_{t-1} - \eta' V_{dw}$$

Weight Updation

$$b_t = b_{t-1} - \eta' V_{db}$$

Bias Updation

$$\eta' = \frac{\eta}{\sqrt{S_{dw_t} + \epsilon}}$$

$S_{dw_t} = 0$

EWA

$$S_{dw_t} = \beta * S_{dw_{t-1}} + (1-\beta)\left(\frac{\partial h}{\partial w_{t-1}}\right)^2$$

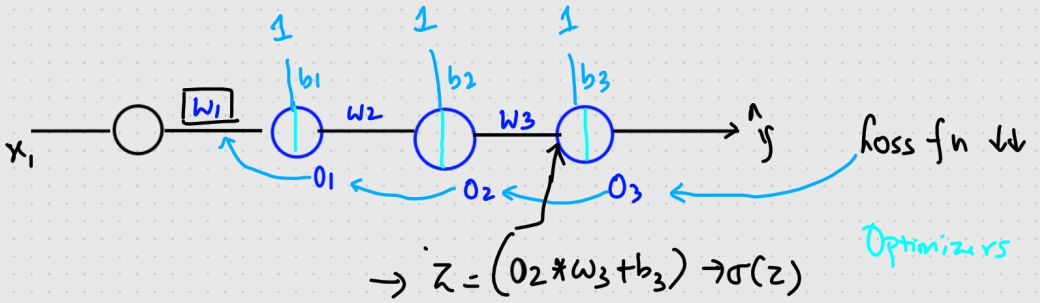
[EWA]

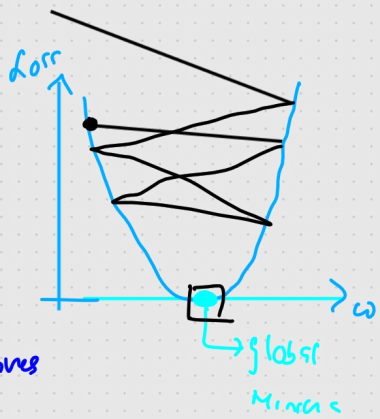
$$V_{dw_t} = \beta * V_{dw_{t-1}} + (1-\beta)\frac{\partial h}{\partial w_{t-1}}$$

$$V_{db_t} = \beta * V_{db_{t-1}} + (1-\beta)\frac{\partial h}{\partial b_{t-1}}$$

⇒ Momentum
↳ Smoothening

Exploding Gradient Problem ⇒ Weight Initialization Technique





$$W_{1new} = W_{1old} - \left[\eta \left[\frac{\partial h}{\partial W_{1old}}\right]\right]$$

{ $W_{inew} >>> W_{1old}$; $W_{inew} <<< W_{1old}$ }

Chain Rule of Derivatives

$$\frac{\partial h}{\partial W_{1old}} = \frac{\partial h}{\partial O_3} * \left[\frac{\partial O_3}{\partial O_2}\right] * \frac{\partial O_2}{\partial O_1} * \frac{\partial O_1}{\partial W_{1old}}$$

big * big * big * big ⇒ big value

Weight initialization ⇓ Very high value

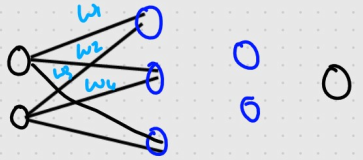
$$\frac{\partial O_3}{\partial O_2} = \left[\frac{\partial \sigma(z)}{\partial z}\right] * \frac{\partial z}{\partial O_2}$$

$$= [0 - 0.25] * \frac{\partial (O_2 * W_3 + b_3)}{\partial O_2}$$

$$= [0 - 0.25] * W_3 \Rightarrow 500 - 1000$$

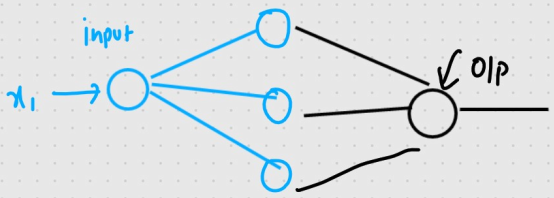
Weight Initializing Techniques

1. Uniform Distribution ✓
2. Xavier / Glorot Initialization ✓
3. Kaiming He Initialization ✓



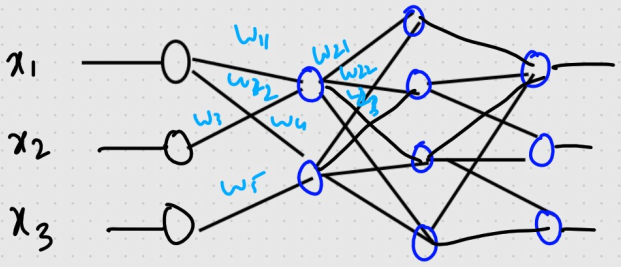
Key Points

1. Weights should be small ✓
2. Weights should not be same ✓
3. Weights should have good variance



input = 1

Output = 1



input = 3

O/p = 3

① Uniform Distribution

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-1}{\sqrt{input}}, \frac{1}{\sqrt{input}}\right]$

$\left[\frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right]$

② Xavier/Glorot Initialization

Researcher $\longrightarrow$ Xavier Glorot

① Xavier Normal Init

$W_{ij} \sim N(0, \sigma)$

$$\sigma = \sqrt{\frac{2}{(input + output)}}$$

② Xavier Uniform

$W_{ij} \sim$ Uniform Distribution $\left[\frac{-\sqrt{6}}{\sqrt{input + Output}}, \frac{\sqrt{6}}{\sqrt{i/p + o/p}}\right]$

③ Kaiming He Initialization

① He Normal

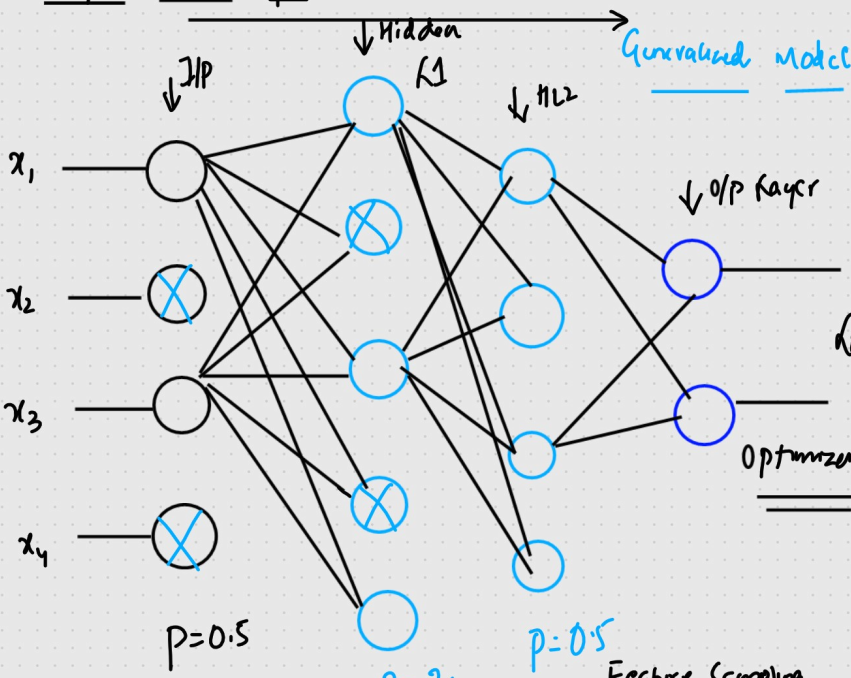
$$W_{ij} \sim N(0, \sigma)$$

$$\sigma = \sqrt{\frac{2}{input}}$$

② He uniform

$$W_{ij} \sim \text{Uniform Dimbution} \left[-\sqrt{\frac{6}{input}}, \sqrt{\frac{6}{input}} \right]$$

Drop Out Layer



Generalized Model ⇐ ⇒ Overfitting

Model → Training Acc ⇒ 90%
Test Acc ⇒ 60%

Focus ↓↓ Machine Learning

Optimizers

DT

⇒ Overfitting

⇓

Generalized Model

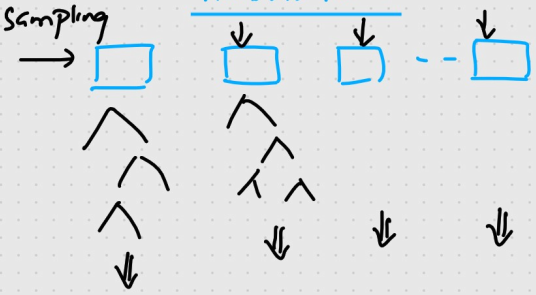
$0 \leq p \leq 1$

⇓

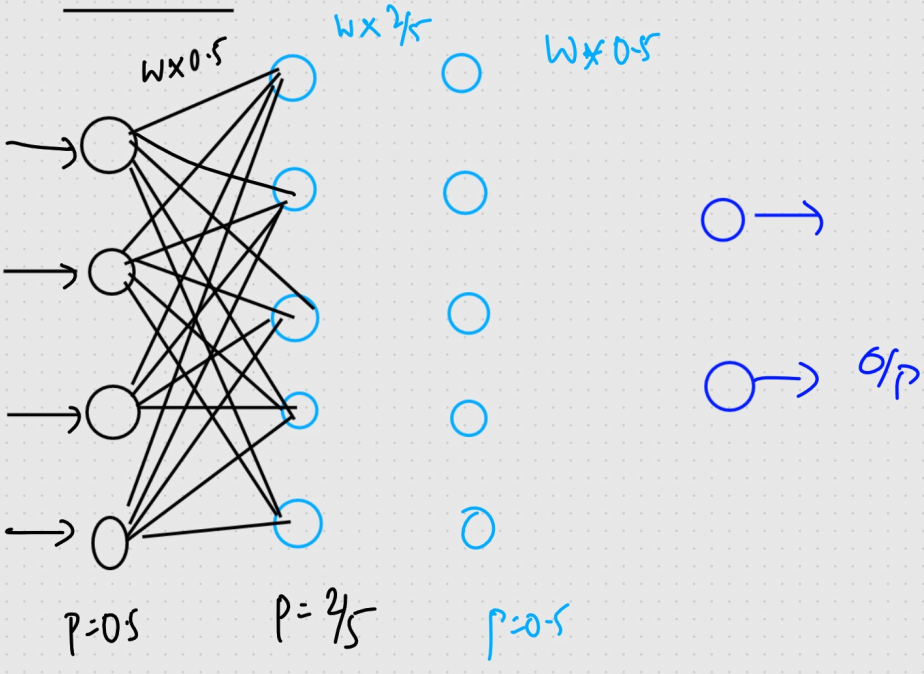
Hyperparameter

Feature Sampling
Row Sampling

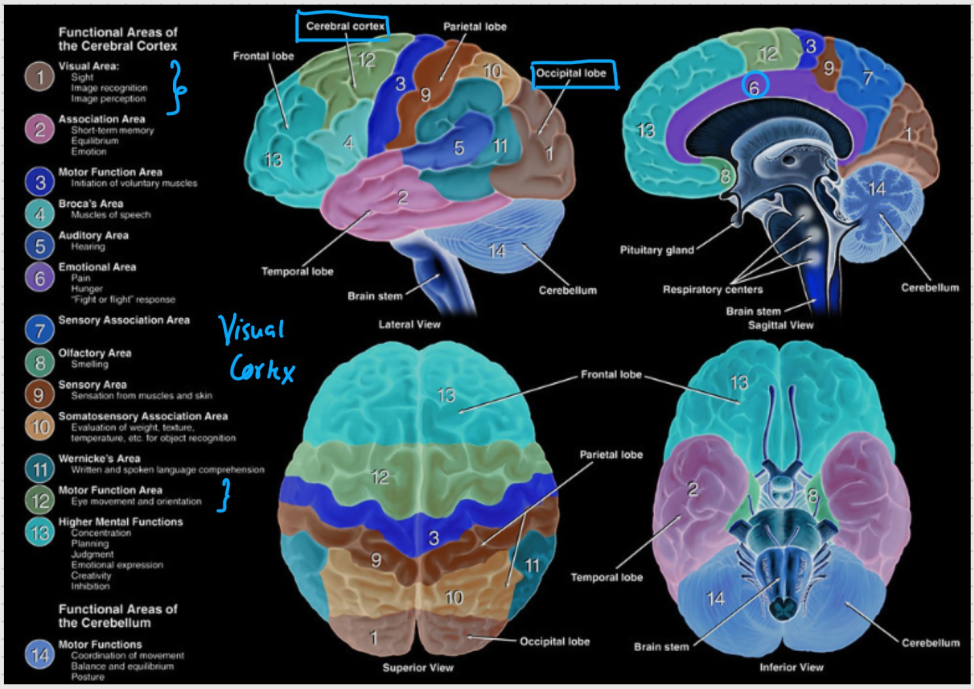
Random Forest



Test data



Convolutional Neural N/w



https://www.dana.org

① ANN → Supervised Learning → [Classification, Regression]

Dataset : I/p features O/p

② CNN : I/p => Images Eg: Image classification, Object Detection, Segmentation

② Cerebral Cortex And Visual Cortex

Visual Cortex (V1-V5) [Region of the brain that receives, integrates and processes visual information relayed from the retinas].

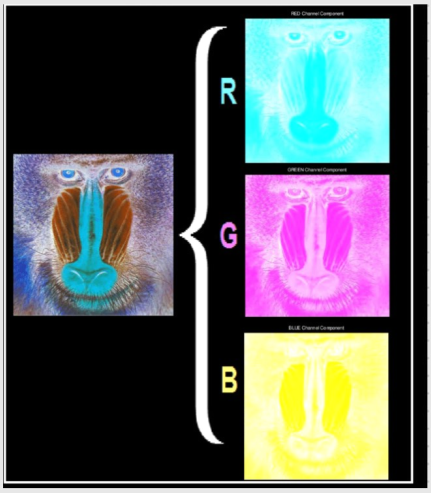
→ V1 → Primary Visual Cortex [Orientation of edges And lines]

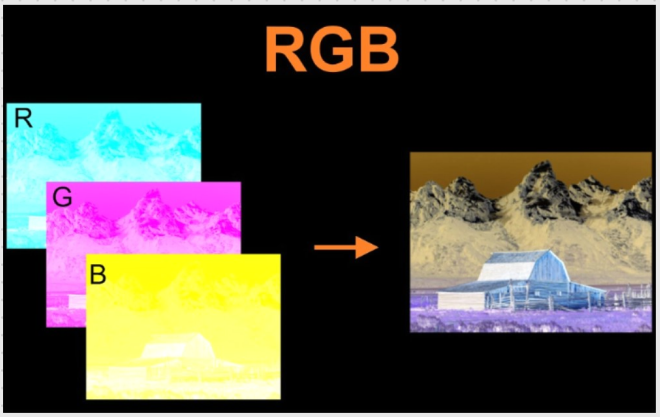
V2 [Differences in color, complex patterns, Object orientation]

V3 V4 V5 [Object Recognition]

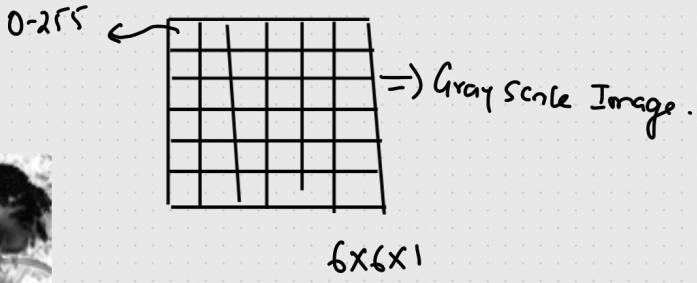
Visualize the Image

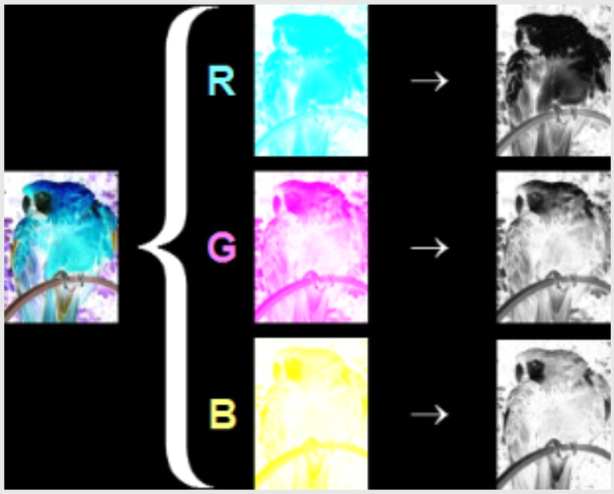
③ RGB Images And Gray Scale Images



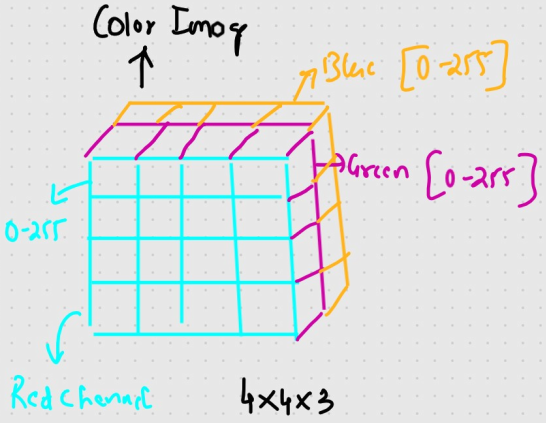


https://www.researchgate.net/





https://commons.wikimedia.org/



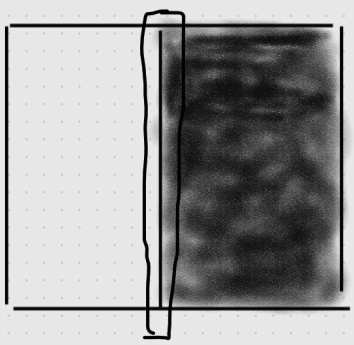
④ Convolution Operation In CNN

→ (0,1)

0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255
0	0	0	255	255	255

6x6x1

⇒



Step 1

① Normalize

Divide by 255

+1	+2	+1
0	0	0
-1	-2	-1

Convolution operation

Stride=1

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

6 x 6 x 1

n = 6

*

f = 3

+1	0	-1
+2	0	-2
+1	0	-1

3 x 3

filters

Vertical edge filters

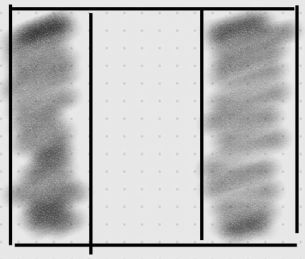
=

o/p = 4

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

4 x 4

$n - f + 1 = 6 - 3 + 1 = 4$



⇐

255	0	0	255
255	0	0	255
255	0	0	255
255	0	0	255

⑤ Padding In CNN

① Zero padding

② Neighbour padding

0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	
0	0	0	0	1	1	1	
0	0	0	0	1	1	1	
0							

6x6 n=6 8x8

*

f = 3

+1	0	-1
+2	0	-2
+1	0	-1

=

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

6x6

$n - f + 2p + 1$

$6 - 3 + 2p + 1 = 6$

$3 + 2p + 1 = 6$

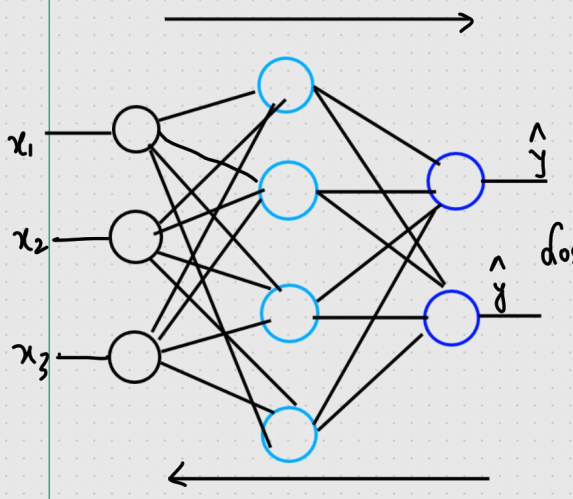
$2p = 6 - 4$

$p = 2/2 = 1$

7x7 3x3 7x7

How much padding you need to apply?

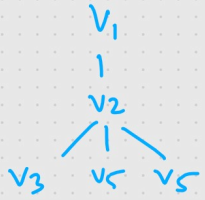
⑥ Operation of CNN Vs ANN



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

$$relu(z)$$

Visual Cortex



→ Relu operation $max(0,x)$

Image

0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1

$*$

+1	0	-1
+2	0	-2
+1	0	-1

f_1

f_2

f_3

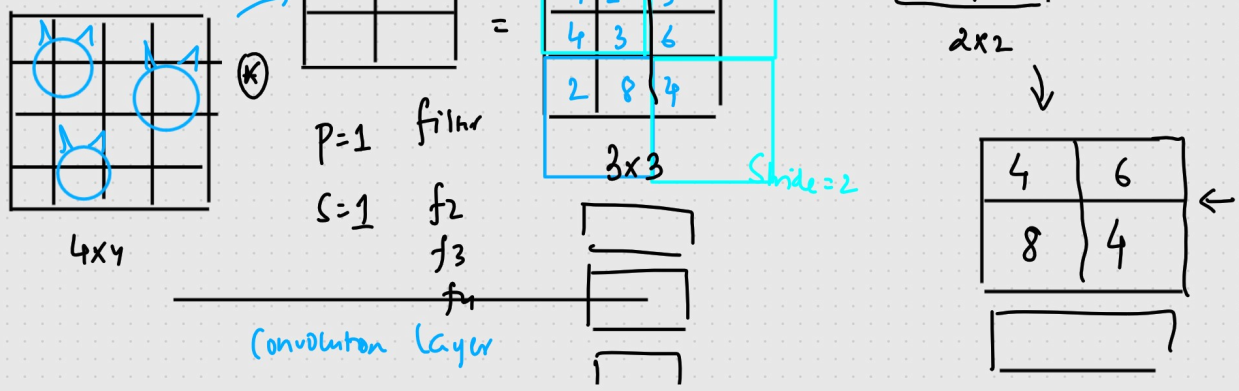
f_n

$=$

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

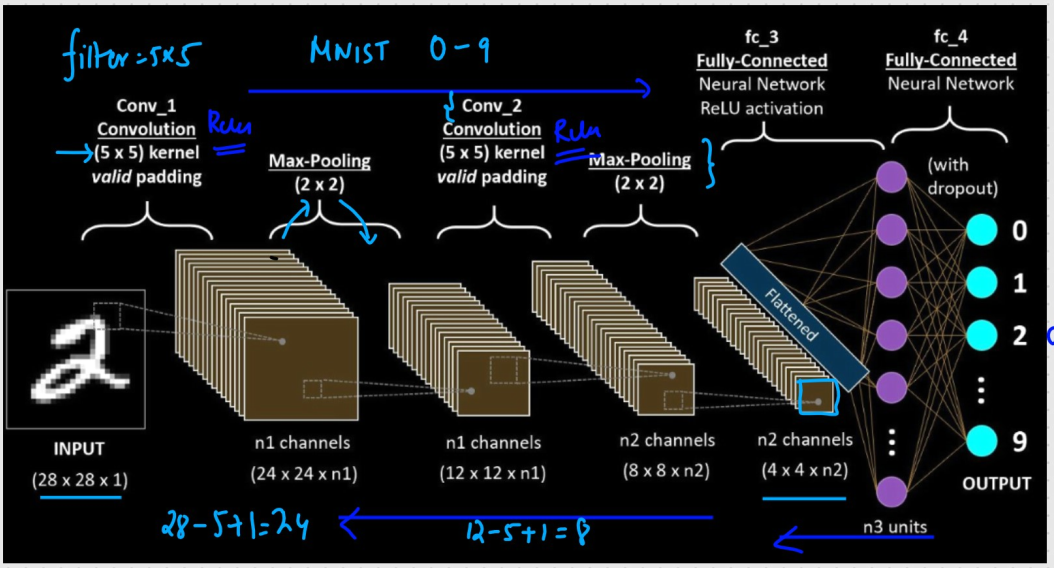
Convolution Layer

⑦ Max Pooling, Min Pooling, Mean Pooling



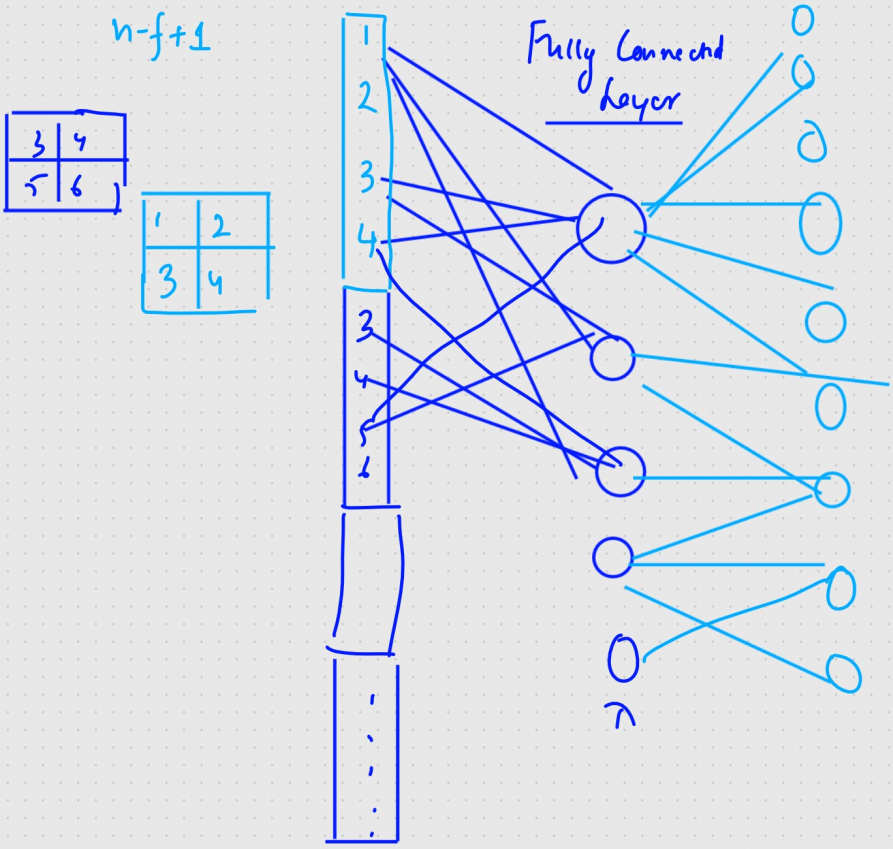
⑦ Location Invariant

⑧ Fully Connected Layer In CNN [Flattened Layer]

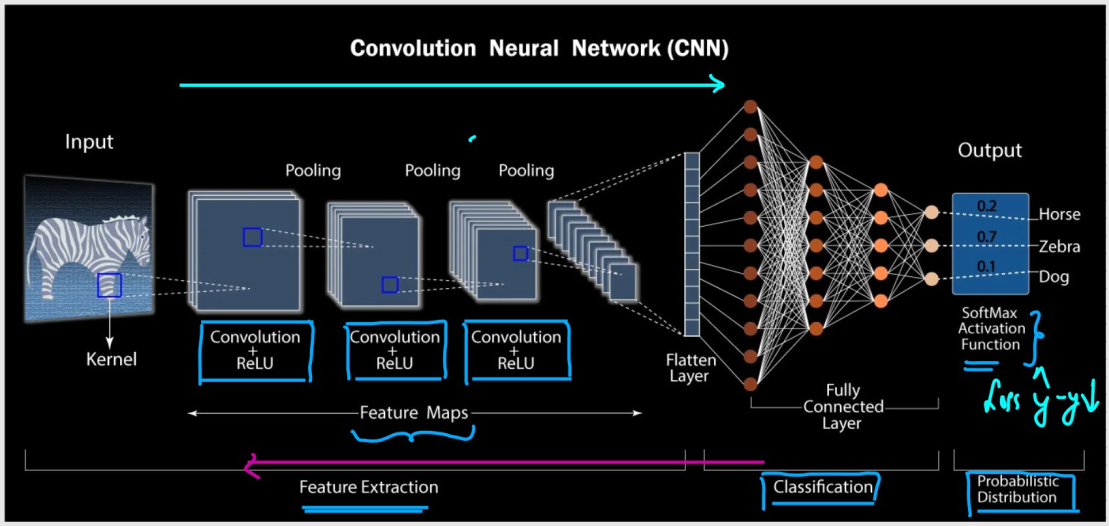


Loss function ↓↓

Optimizer



⑨ CNN Complete Example



https://developersbreach.com/convolution-neural-network-deep-learning/

0	0	0			
0	0	0			
0	0	0			
0	0	0			
0	0	0			
0	0	0			

*

+1	0	-1
+2	0	-2
+1	0	-1

0	-4	-4	0
0	-4	-4	0
0	-4	-4	0
0	-4	-4	0

Day 6 - NLP {Recurrent Neural N/W}

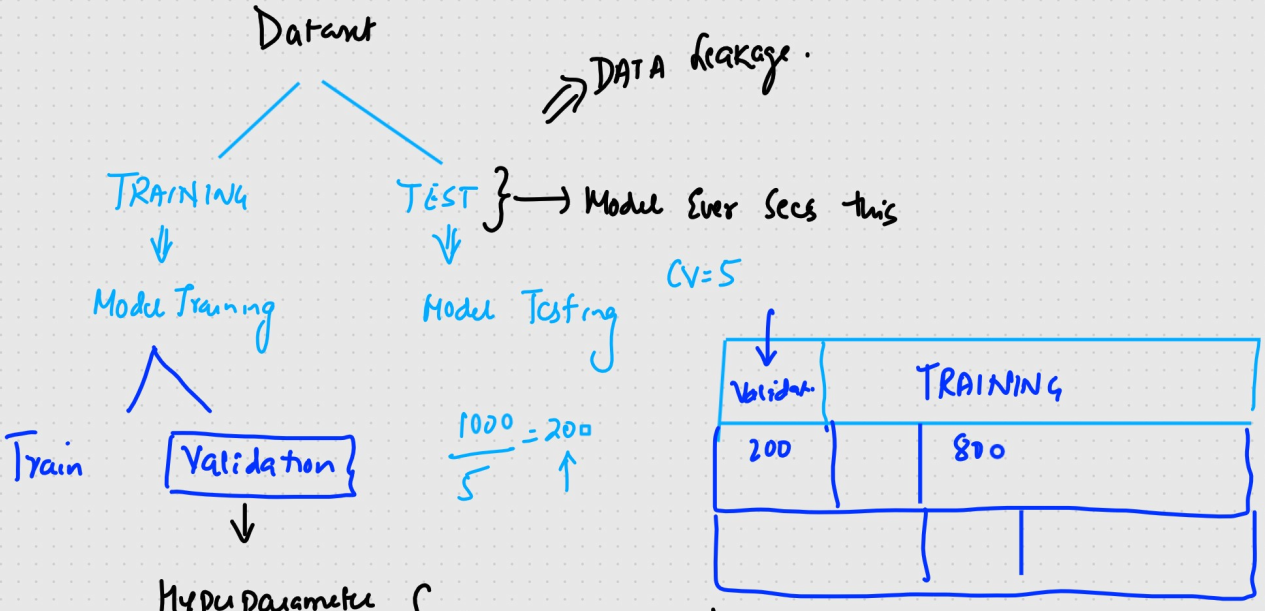
① BOW, TFIDF, Word2vec, Avg Word2Vec {Python Practical Implementation}

CBOW Skip Gram

② Deep learning → ① RNN ② LSTM RNN ③ GRU RNN ④ Bidirectional LSTM RNN
⑤ Encoders - Decoders ⑥ TRANSFORMERS ⑦ BERT

Interview Question

① Train vs Test Vs Validation →



② Why RandomForest instead of Decision Tree {Answer}.

DT → {Low Bias High Variance}

{ Low Bias Low Variance } → Random Forest }.

① Recurrent Neural N/W ⇒ Text → Vectors

Machine Learning

Word Embedding ← Word2vec, AvgWord2vec

Chatbot : Question and Answer → Sequence of words

Language Translation → Hindi → English

Text generation → A sentence → Suggestion → Completion of Sentences

Grammatically

GMAIL AutoSuggestion

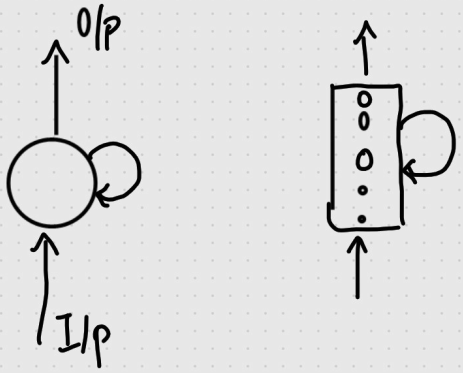
Deep learning

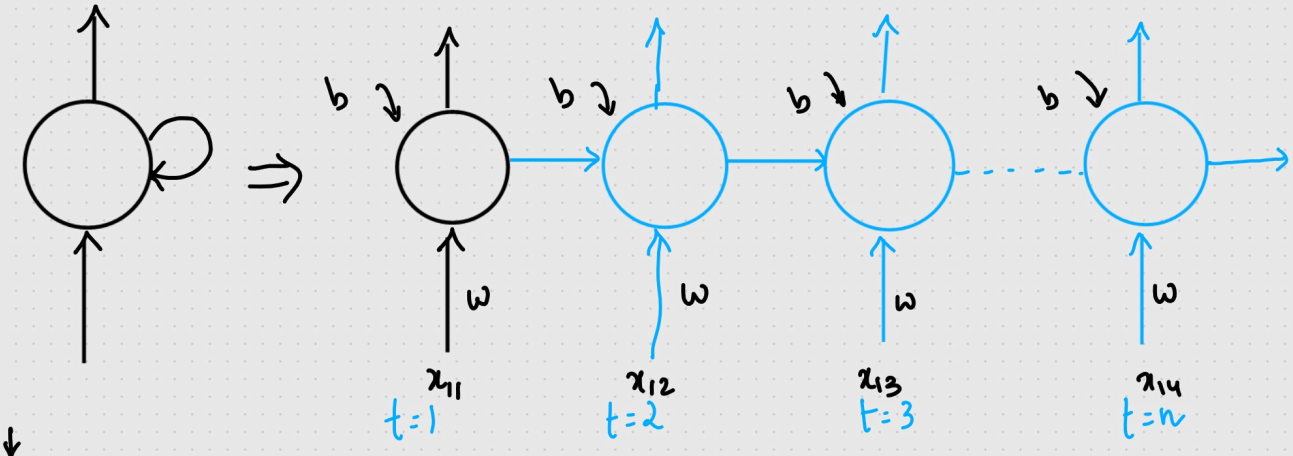
① RNN ② LSTM RNN ③ Transformers ④ BERT

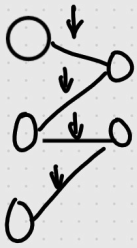
Word2vec ⇒ Deep learning

Words ⇒ Vec

① Recurrent Neural N/W







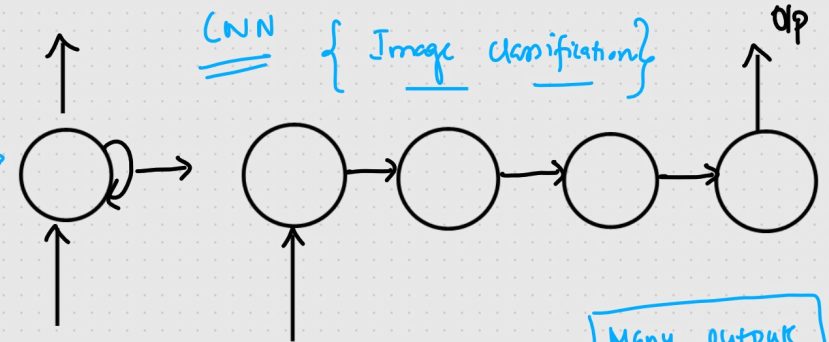
Sentiment Analysis → The food is good → Positive (O/p)

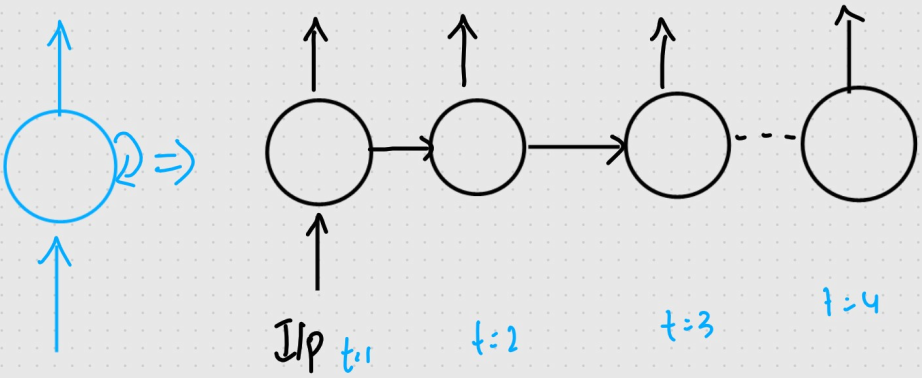
$\langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$

x_{11} → Word2vec → Vectors → $d = 300$

Types of RNN

1. One to One RNN
2. One to Many RNN
3. Many to One RNN
4. Many to Many RNN



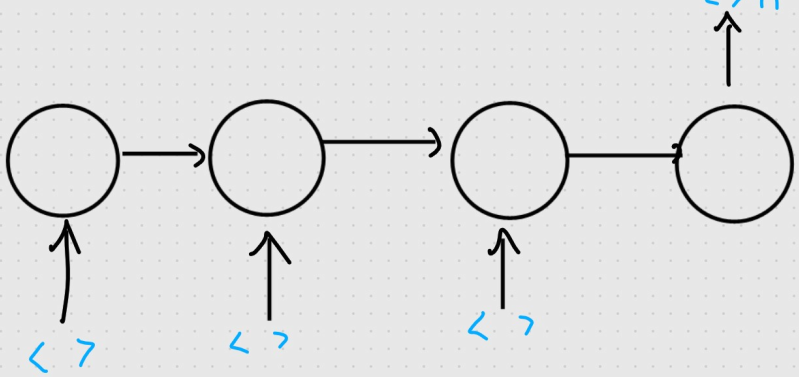


Eg: Music Generation, Text Generation

Google Search Suggestion, Music Recommendation

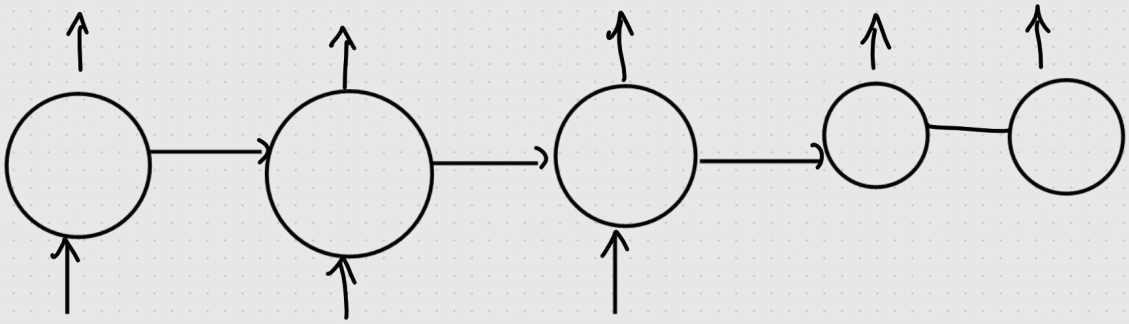
③ Many to One

Sentiment Analysis



{ Predict Next Day Sales }

④ Many to Many



① Language Translation
② Question answers
③ Chatbots.

{ ① Forward Propogation
② Backward Propogation }

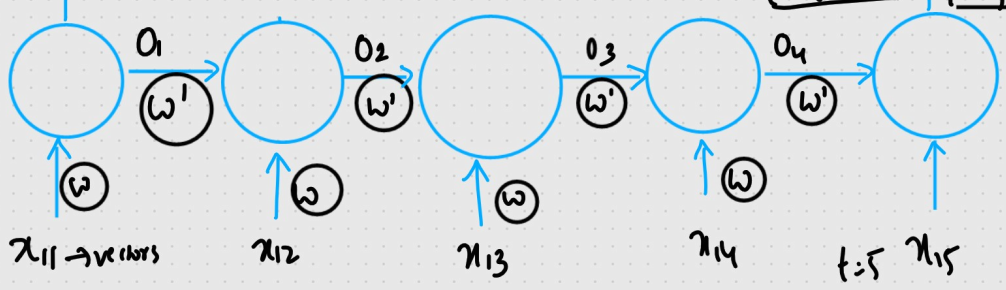
Time Series Data

① Forward Propogation In RNN

Sentiment Analysis → Many to One

Activation

Binary: Sigmoid + Multiclass: Softmax



loss = $(y-\hat{y})$

t=1 t=2 t=3 t=4 O/P

The food is very good Positive.

x_{11} x_{12} x_{13} x_{14} x_{15}

$O_1 = f(x_{11} * w)$

$O_2 = f(x_{12} * w + O_1 * w_1')$

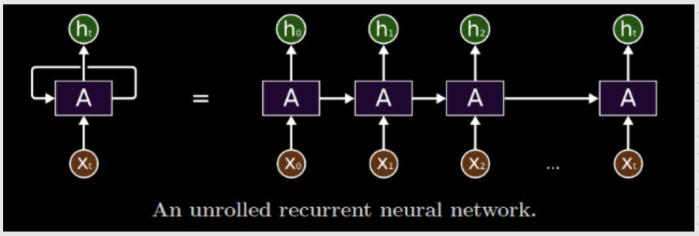
$O_3 = f(x_{13} * w + O_2 * w')$

LSTM RNN [Long Short Term Memory RNN]

RNN ⟶ Long Term Dependencies ⟶ Vanishing Gradient Problem

1. RNN ⟶ Problem ? ✓
2. Why LSTM RNN ? ✓ Basic Representation.
3. How LSTM RNN works → Long Term Memory ✓ / Short Term Memory ✓
4. LSTM Architecture
5. Working of LSTM RNN

Problems With RNN → Long Term Dependency



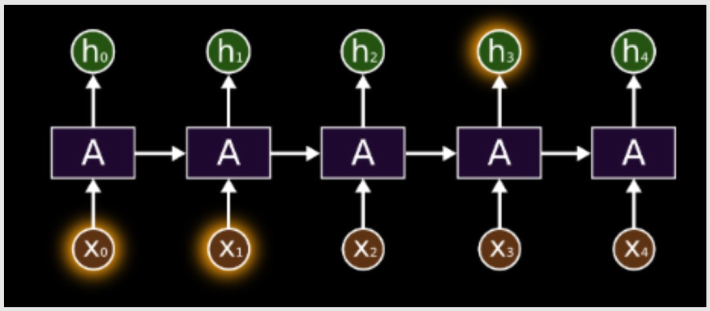
Vanishing Gradient Problem

Task:

Next Word In a Sentence

The color of the sky is blue

further context



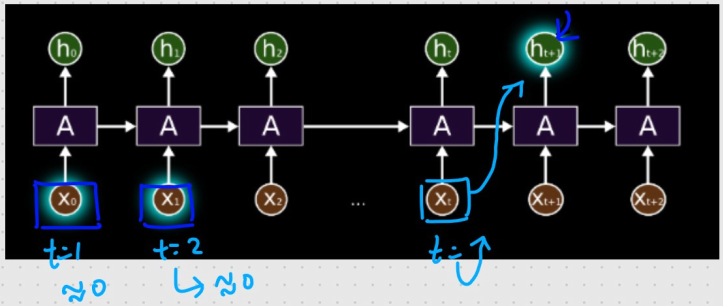
Gap is less

Huge gap O/p ← Context

I grew up in India .... I speak fluent ____ ← Context

Name of language

further Context



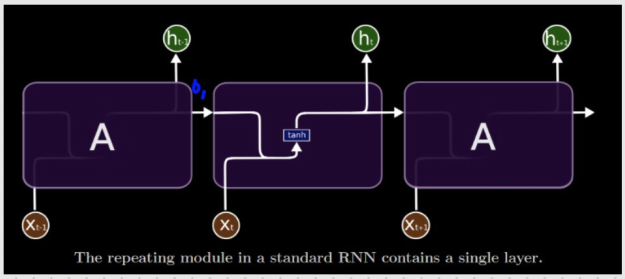
[0-0.25] [0-1] Huge Gap → Long Term Dependency

RNN → Long Term Dependency → Vanishing Gradient Problem

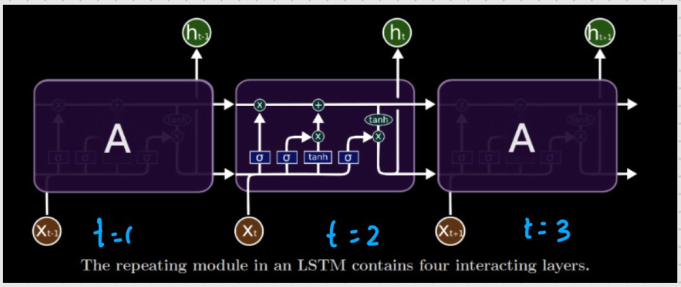
↳ Chain Rule → ≈0.

Basic Representation of RNN And LSTM RNN

LSTM RNN

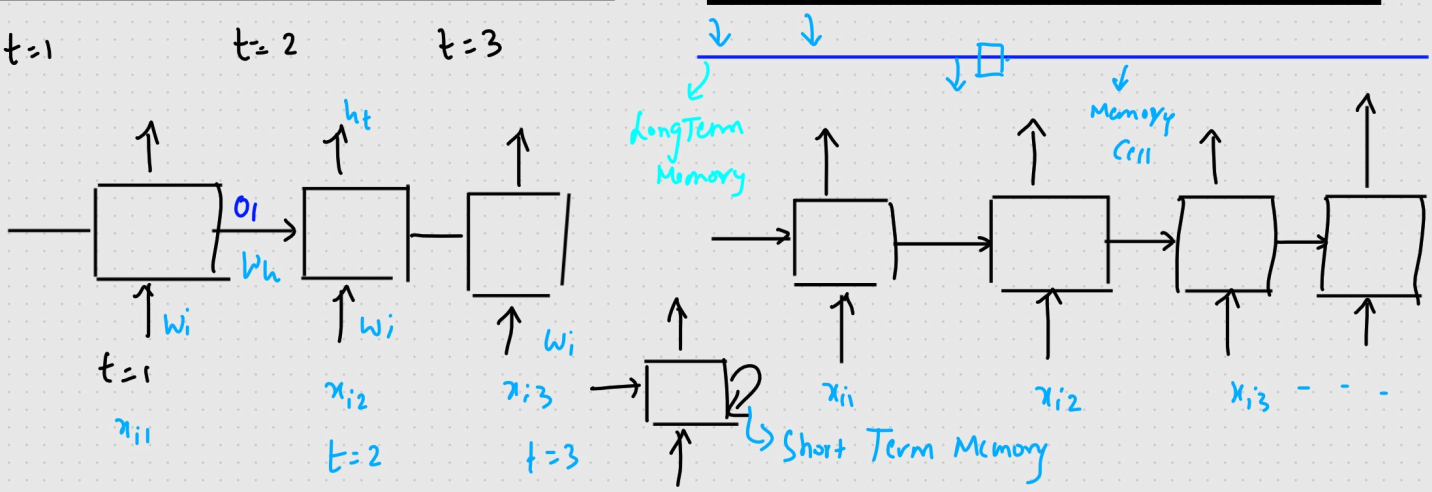


The repeating module in a standard RNN contains a single layer.



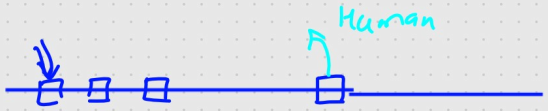
The repeating module in an LSTM contains four interacting layers.

t=1 t=2 t=3

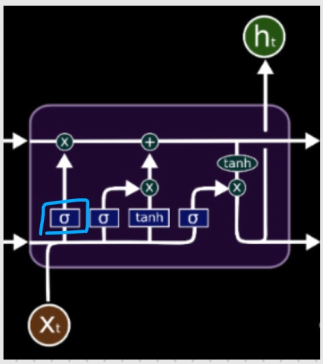


LSTM RNN → Long Term Memory
→ Short Term Memory

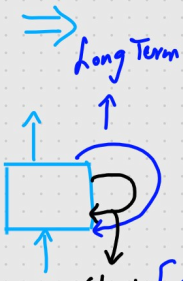
Conveyance But : Luggages



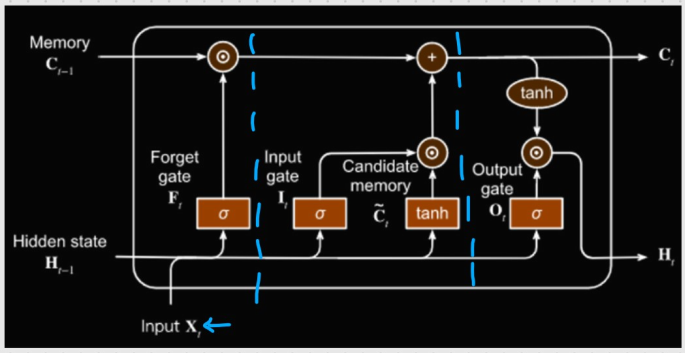
LSTM Architecture



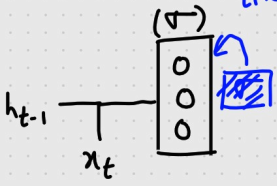
Basic Architecture

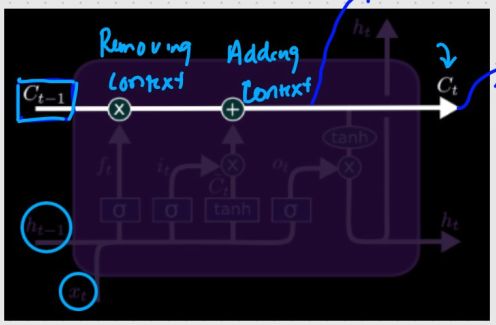


LSTM RNN

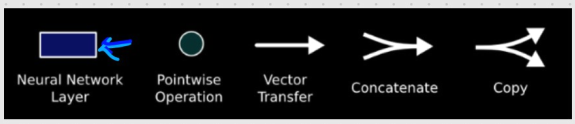


{Neural Nw Layer}.





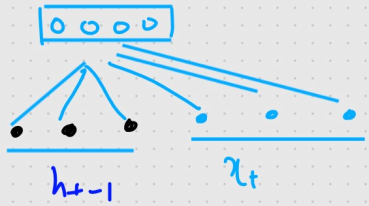
Long Term Memory



Combining 2 vectors

$h_{t-1} = [1 \quad 2 \quad 3]$

$x_t = [2 \quad 3 \quad 4]$



Duplicate copy

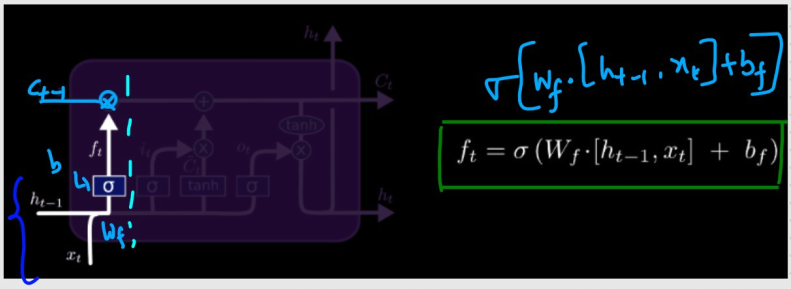
V1 $[1 \quad 2 \quad 3]$ ← tanh

V2 $[4 \quad 5 \quad 6]$

⊗ ⇒ $[4 \quad 10 \quad 18]$

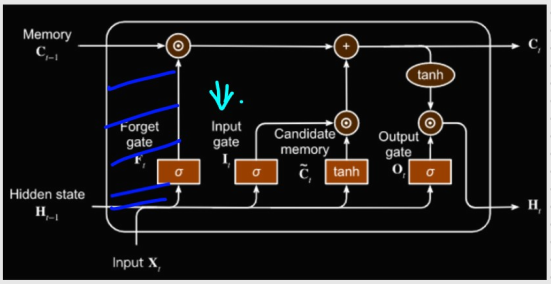
⊕ ⇒ $[5 \quad 7 \quad 9]$

tanh ⇒ $[\tanh(1) \quad \tanh(2) \quad \tanh(3)]$



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$\sigma\left[W_f \cdot [h_{t-1}, x_t] + b_f\right]$$



Forget Gate

h_{t-1} = Hidden state of previous time stamp

x_t = Word passed as i/p in the current time stamp

Text: x_{11} x_{12} x_{13} x_{14}

Next Word: y_{15}

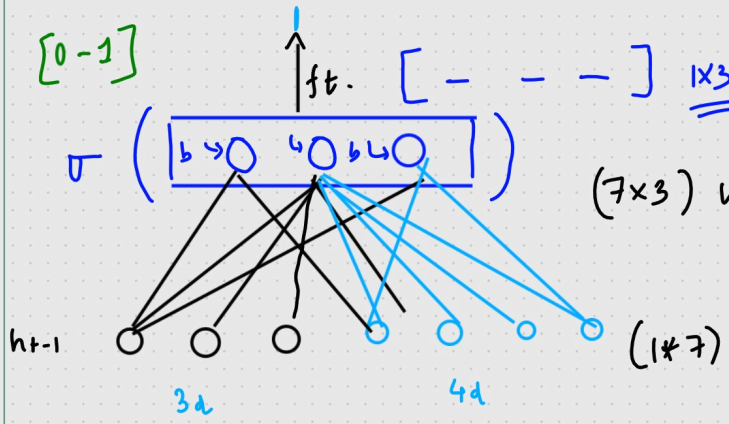
$[0 \quad 2 \quad 4 \quad 1]$ $[4 \quad 5 \quad 12]$ - - - - -

x_t x_{t+1}

h_{t-1} $[1 \quad 2 \quad 4]$ 3d

C_{t-1} $[4 \quad 2 \quad 1]$ 3d ⇐

Memory Cell



t=2

$ft = [0 \;\; 0 \;\; 0]$ ← Complete Sentence

Context changed

① $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[0 \;\; 0 \;\; 0]$

$= [0 \;\; 0 \;\; 0]$ ← Removing all the previous Context

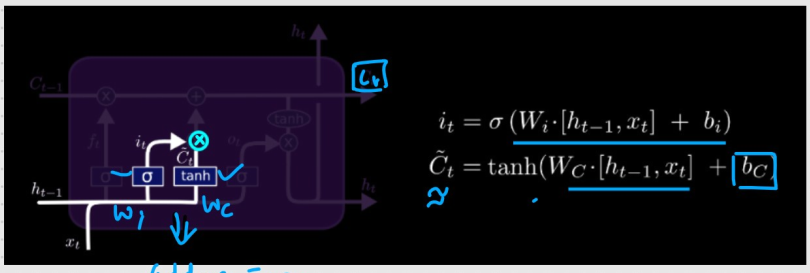
$ft = [1 \;\; 1 \;\; 1]$

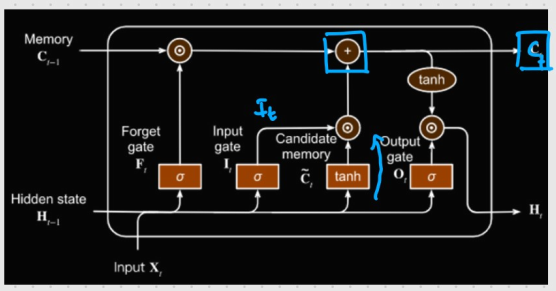
② $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[1 \;\; 1 \;\; 1] = [6 \;\; 8 \;\; 9]$

③ $C_{t-1} = [6 \;\; 8 \;\; 9]$ ⊗ $[0.5, 1, 0.5] = [3 \;\; 8 \;\; 4.5]$

Conclusion : Based on the context → Forget gate will let go some information or will not let go some Info {Forgetting}.

② Input Gate And Candidate Memory

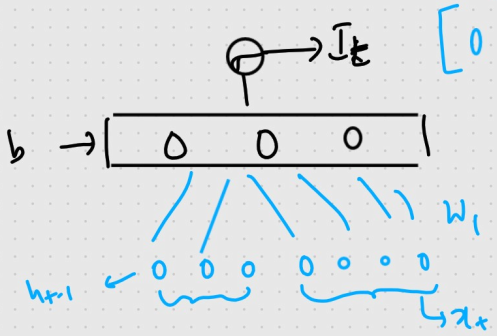


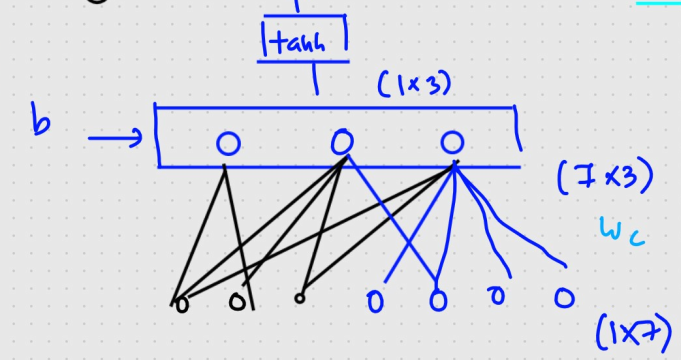


Adding Info

$I_t = [\ 2\ \ 4\ \ 6\ ]$ —— ⊛ —— C_t $[\ 0\ \ 2\ 0\ ]$ ⇒ I/p Gate

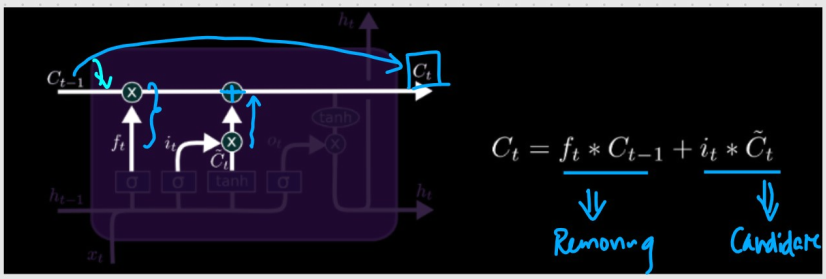
$[\ 0\ \ 8\ \ 0\ ]$





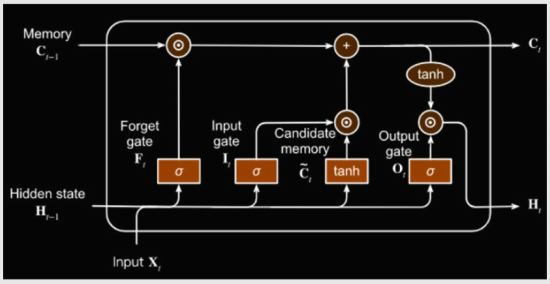
Context = If any information needed to be added it the memory

C_{t-1} ⟶ The information will be added



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

⇓ Removing ⇓ Candidate



I stay in India - - - - - -

and I speak → English → Hindi

or Forgetting Some info

Memory.

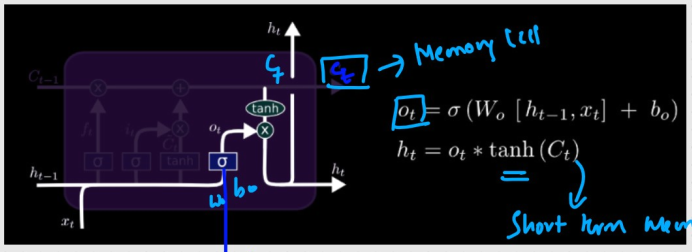
Forget Gate

I/p Gate ⊛ Candidate memory

+

Ct-1 ⇒ Ct.

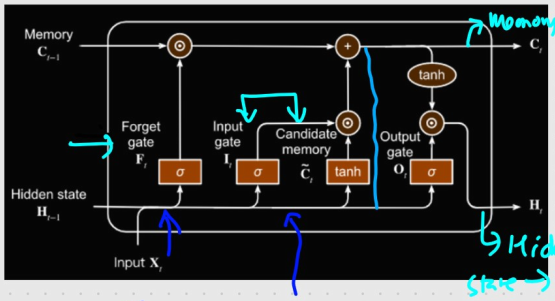
Output gate LSTM RNN



→ Memory Cell

→ Short term Memory.

→ Neural Network

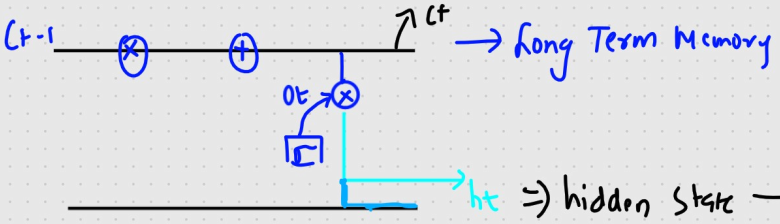


Memory

→ Hidden state → ST Memory.

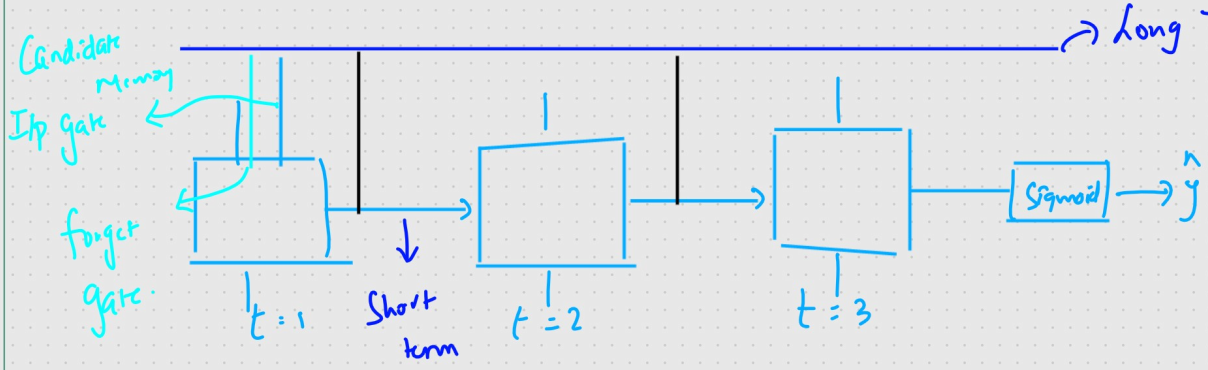
Forget — Add Info → Context

Some Info → context → Memory Cell



→ Long Term Memory

⇒ hidden state → Short term Memory



→ Long Term

loss ↓↓.

[Wi, Wc, Wo] → Updating ← Back Propogation

GRU RNN ⇒ LSTM Variant

Training Data With LSTM RNN

{Training} Text Paragraph

I Went to Restaurant and order burger

The burger looked tasty and crispy

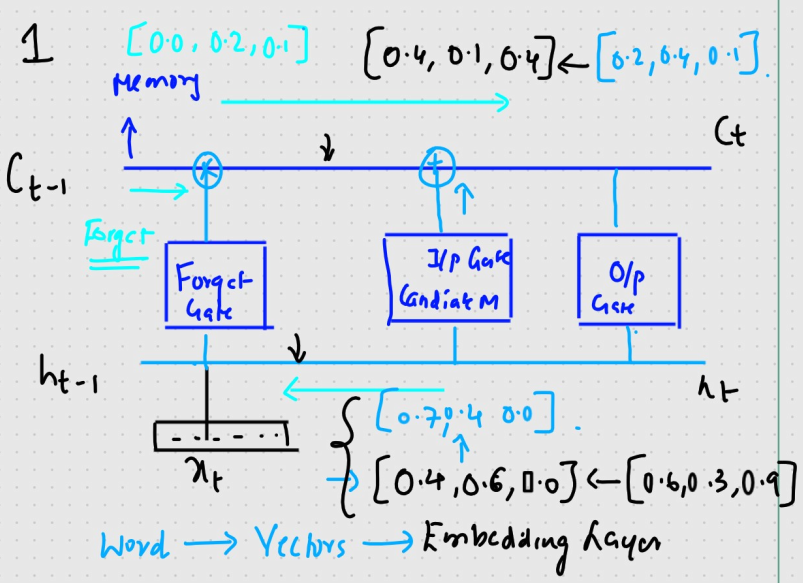
But burger is not good for health

It has lot of fats, cholestrol

But this burger was made with Whey protein and only vegetable were used, so it was good.

O/p [good/bad] gate

Forget gate | I/p gate | O/p gate



Word → Vectors → Embedding Layer

Word2Vec [3 dimension-vector]

→ [[Good] [Bad] [Healthy]] ← Black Box

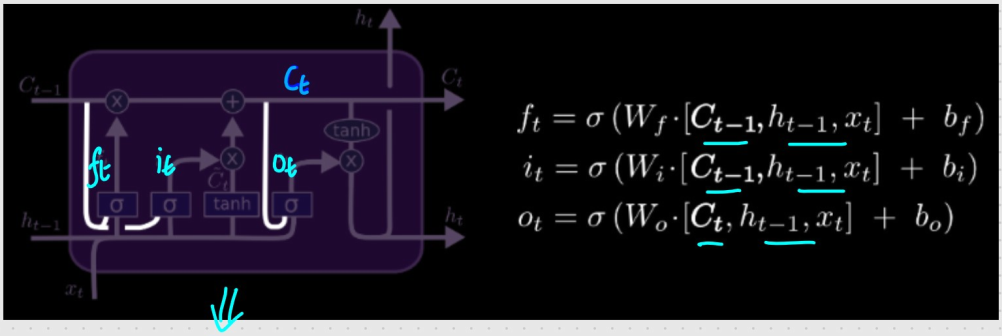
Tasty [0.9 0.0 0.1] ← 3 d.

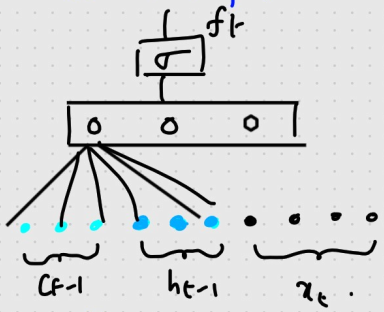
Variants of LSTM RNN

LSTM RNN [1970-80]

LSTM Variants Introduced By Gers & Schmidhuber [2000]

↳ Research paper





focus

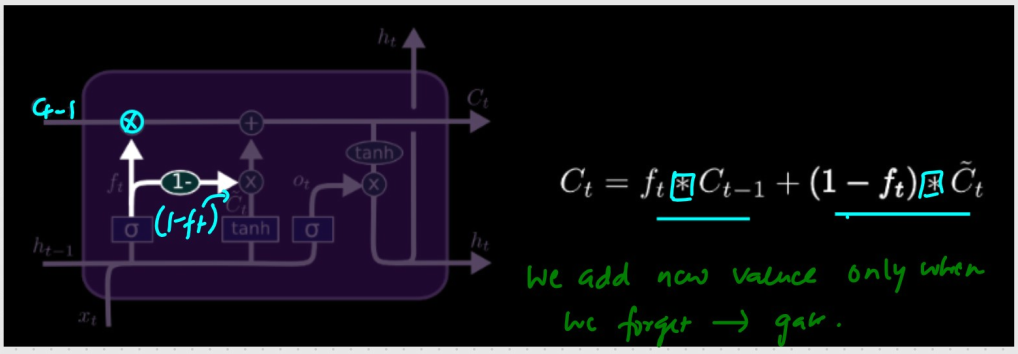
Connections → From Memory cell to forget gate

i/p gate ⇒ Peephole Connections

O/p gate

Peephole Connections : We let the gate layers look at the cell state

Another variation → Coupling Forget And I/p Gates



$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

We add new value only when we forget → gate.

Goal : We only forget when we're going to i/p something in its place.

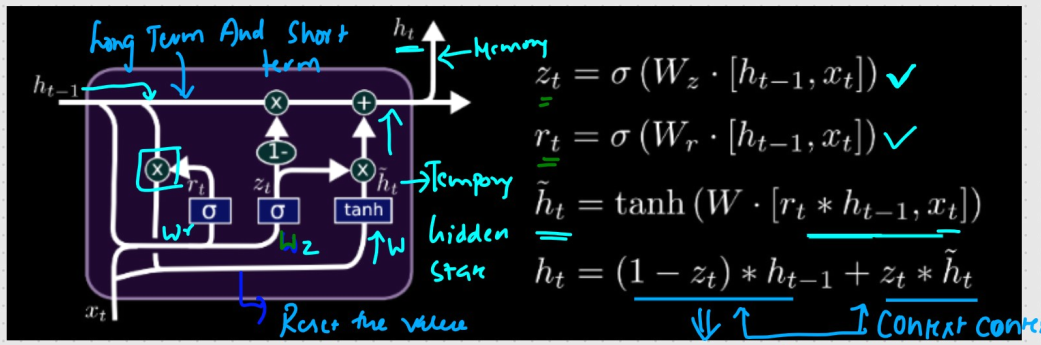
⇓

We only i/p new values to the state when we forget something older.

Instead of seperately deciding what to forget and what we should add new Info, we make this decision together.

GRU → Gated Recurrent Unit [Cho, et al [2014]]

1980 → LSTM
2000 - Variants
2014 → GRU



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$
$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Context content

⇓ Coupled.

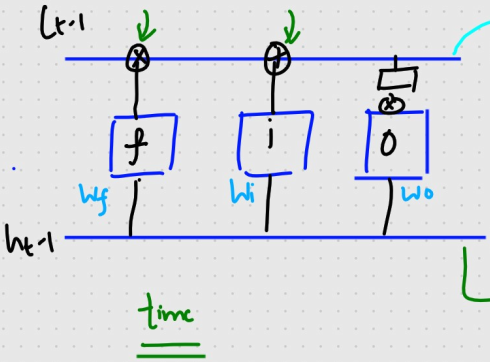
z_t ⇒ Update Gate ←

r_t ⇒ Reset Gate ←

$\tilde{h}_t$ ⇒ Temporary hidden state.

Trainable parameters

bias → $[W_f, W_i, W_o]$

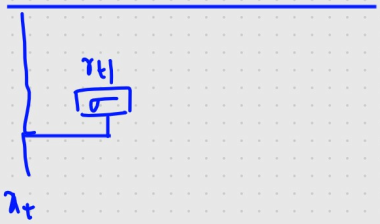


LSTM : Complex Architecture → forget, i/p + Candidate memory, o/p

Training Time ↑↑

z_t ,

Reset gate = r_t =



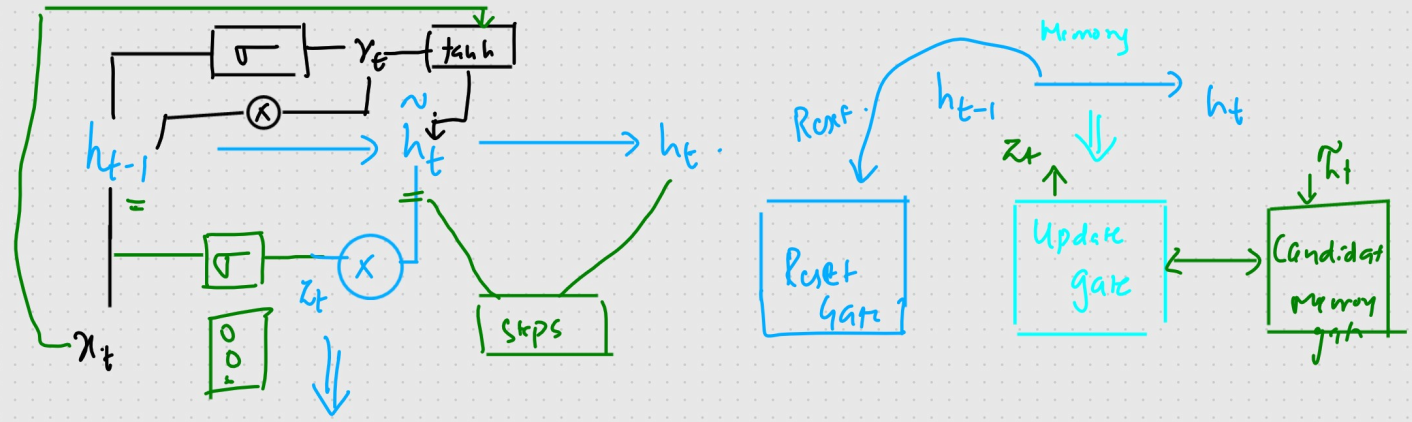
→ Resetting some info from h_{t-1} ⇒ Memory → LTM + STM

h_{t-1} [0.6 0.5 0.3 0.9]

$\otimes \leftarrow r_t$ [0.2 0.4 0.8 0.2]

$\downarrow$ $\downarrow$ $\downarrow$ $\downarrow$

$x_t \rightarrow$ [0.12 0.20 0.24 0.18] ← Resetting → Context



What context Info needs to be Added

$\downarrow$

Candidate hidden state. [current Context]

$\downarrow$

Imp → Add Info

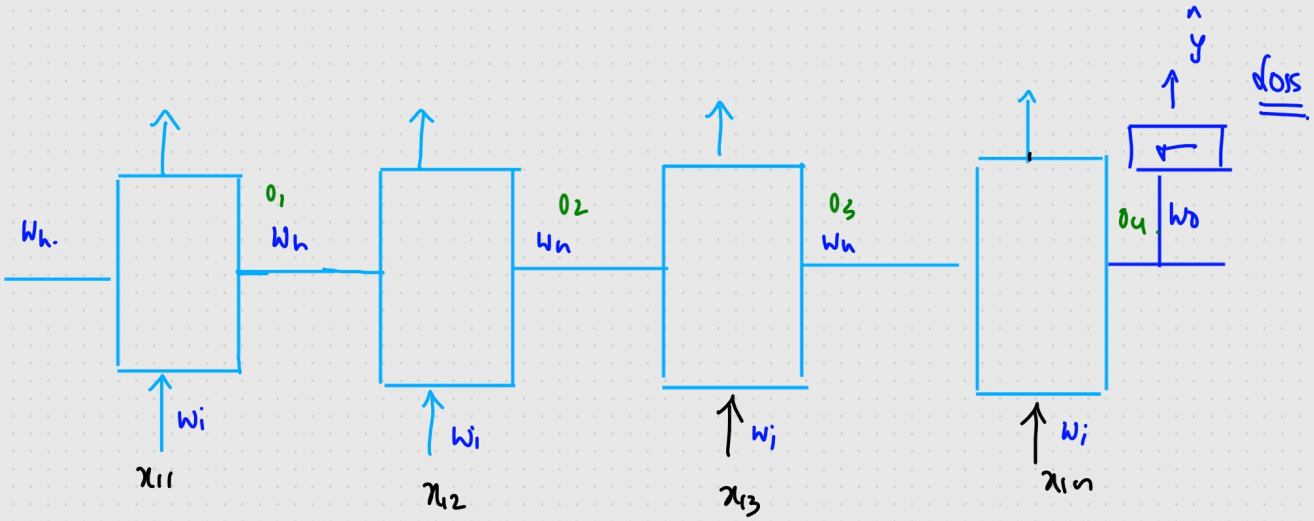
$z_t \Rightarrow$ Len Info

Bidirectional RNN

① Simple RNN ⟶ practical Implement ⟶ Embedding Layers

② LSTM, GRU Variants RNN ⟶ Practical Implement

③ Bidirectional RNN / LSTM RNN



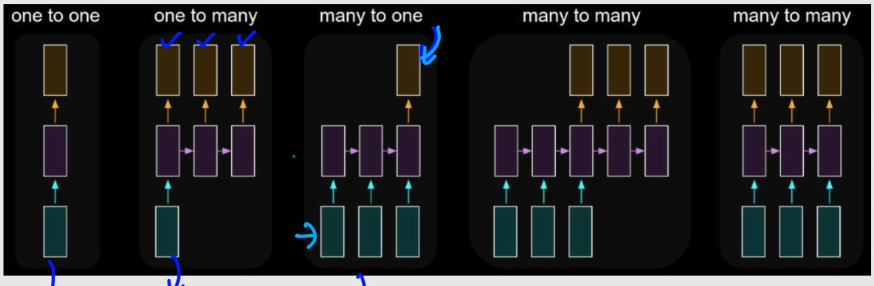
Types of RNN

① One to Many RNN

② Many to One RNN

③ Many to Many RNN.

④ One to One RNN

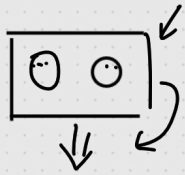


One i/p
One O/p

One i/p
O/p Many

Image scanch
I/p Many
O/p One.

Language Translation
Many I/p
Many O/p.



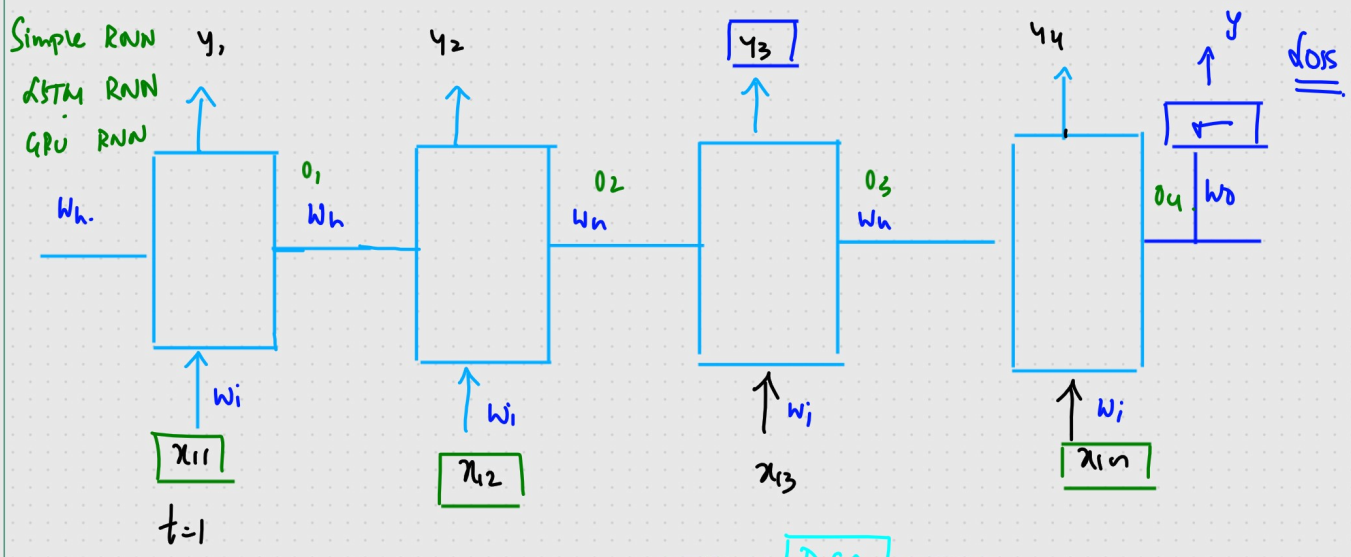
There is a dog and cat

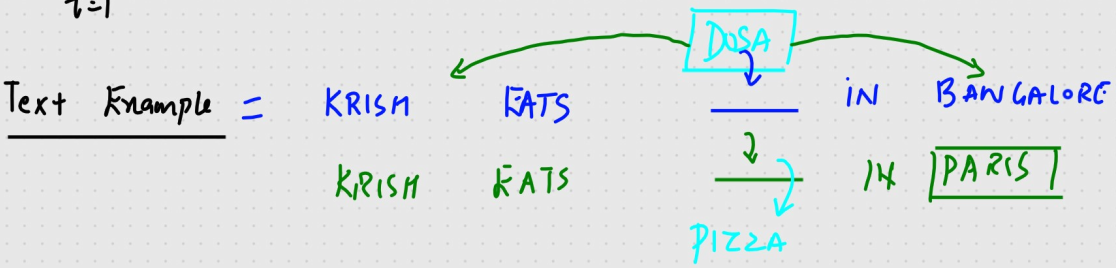
Eg:

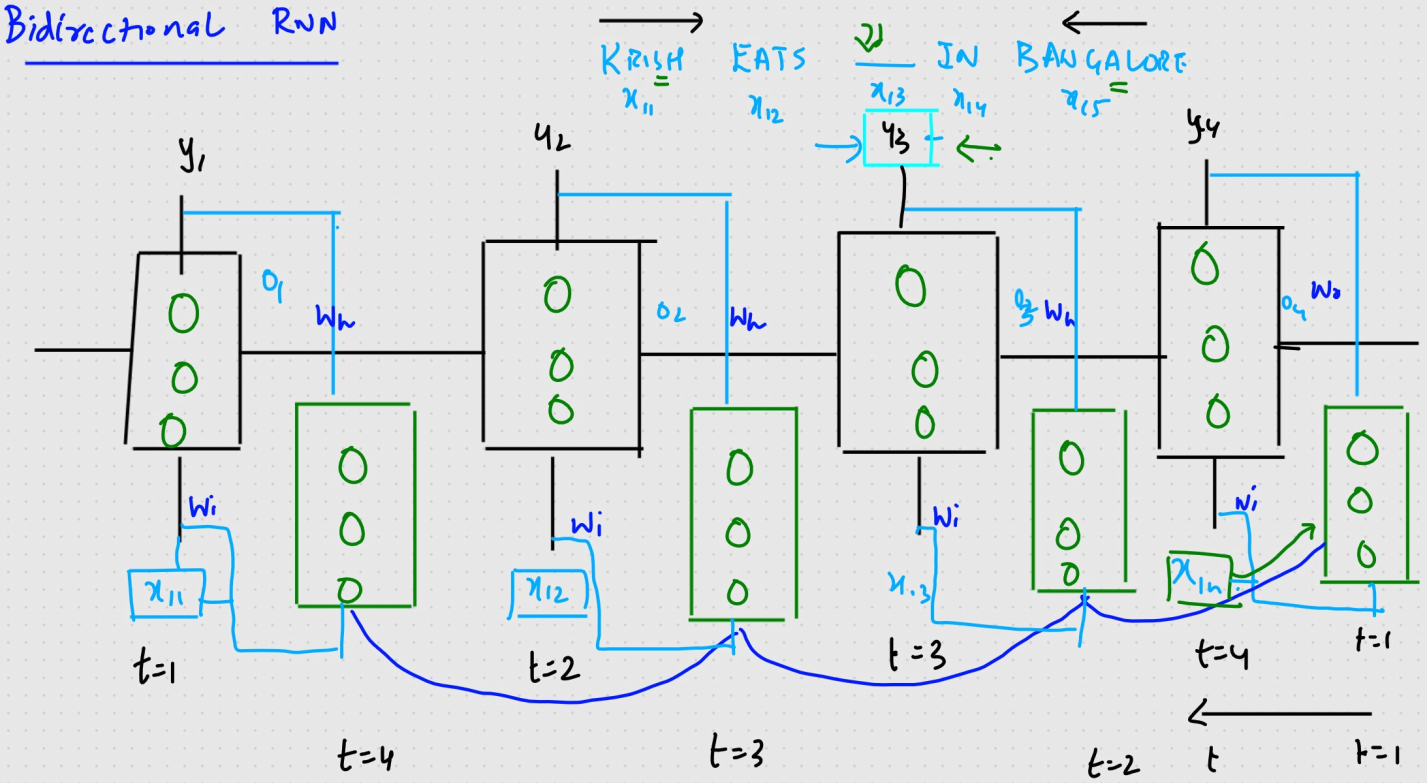
Eg: Image Captioning

A cat eating food

o eating







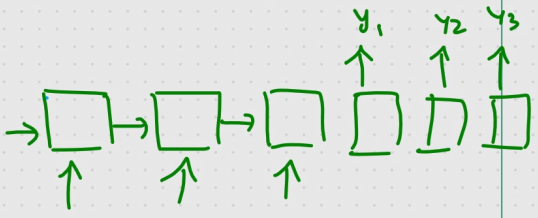
① Forward propogation → Equation ←

Encoder And Decoder

1. Simple RNN → Vanishing Gradient Problem
2. LSTM RNN → } Long Short Term Memory.
3. GRU RNN → }
4. Bidirectional RNN ←

Type of RNN

1. Many to many RNN



Encoder And Decoder

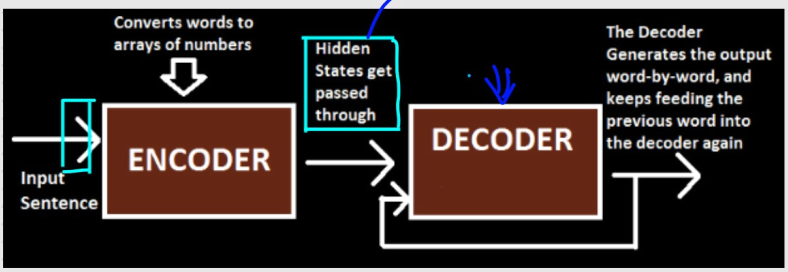
Sequences I/p — O/p Sequence of Words

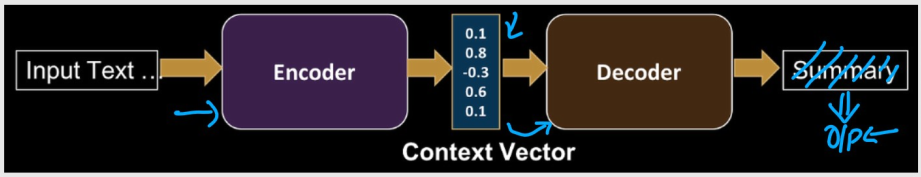
Eg: One language To other

English → French

Eg: Linked chat → Hi, How are you?

Simple Working



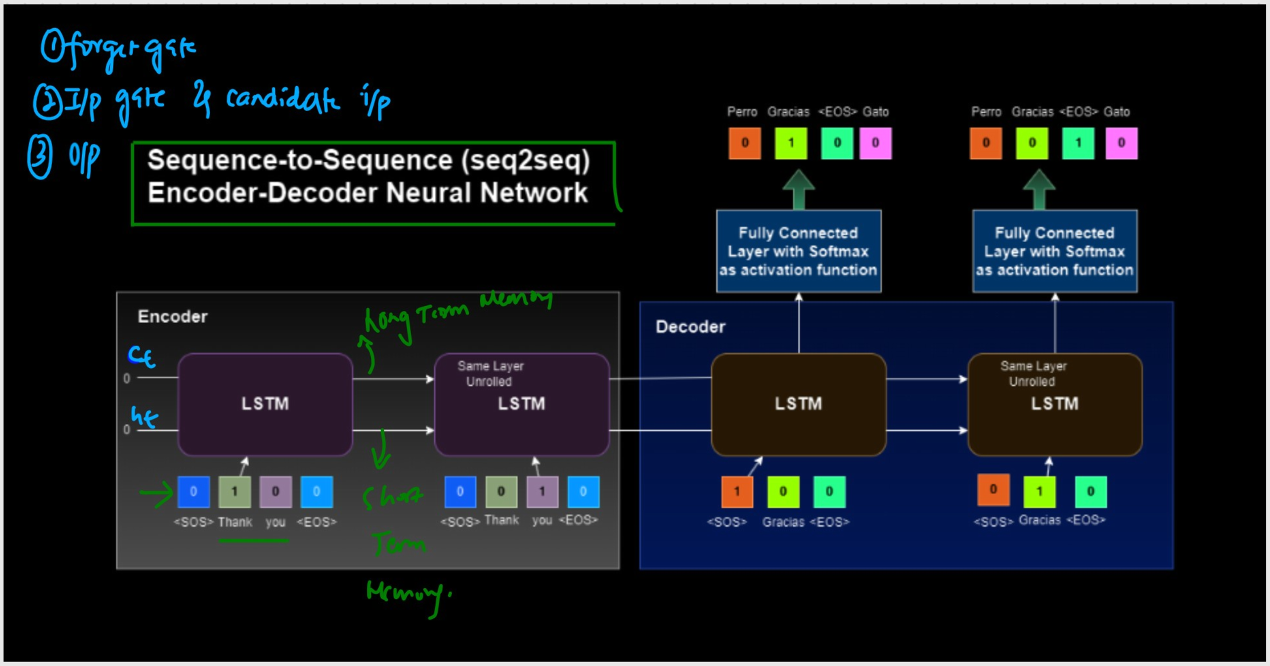


1. Encoder ⇒ I/p ⇒ Context Vector ⇐ Vectors
2. Decoder ⇒ ⇒ O/P

Usecase

1. Language Translation
2. Text Generation
3. Text Suggestion.

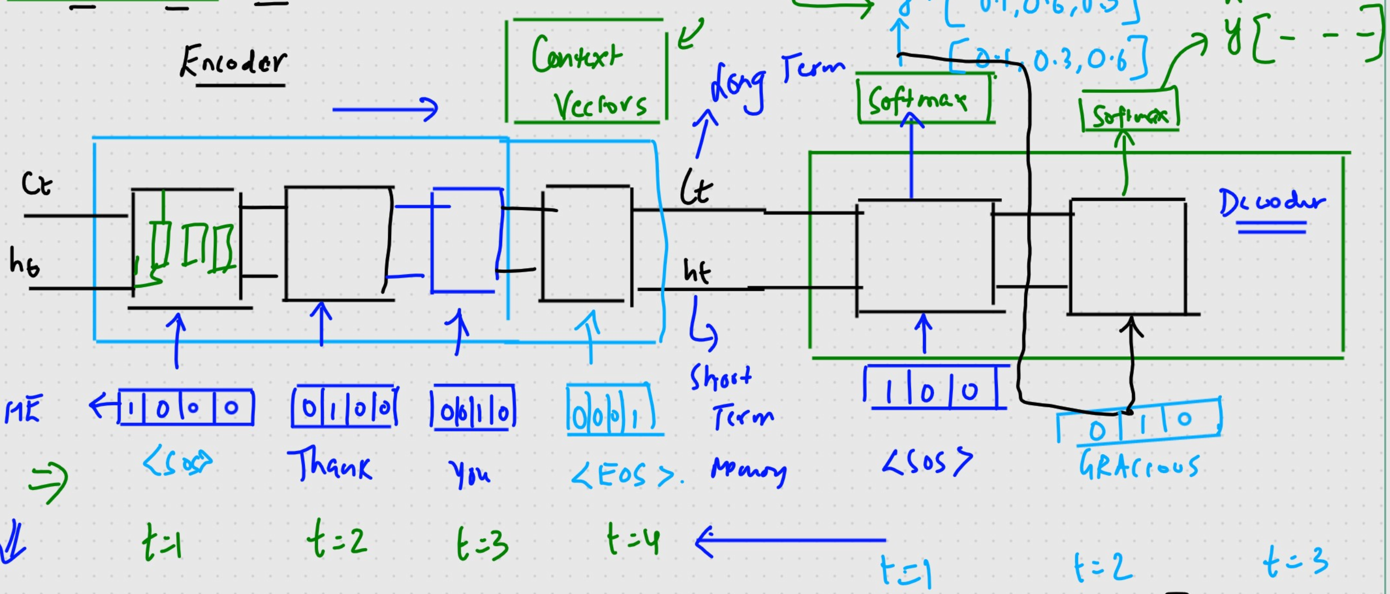
RNN → Vanishing Gradient Problem



Dataset — True o/p — y truth

English — French — {Training}

<SOS> Thank you <EOS>. Gracias → <SOS> GRACIAS <EOS>. GRACIOUS



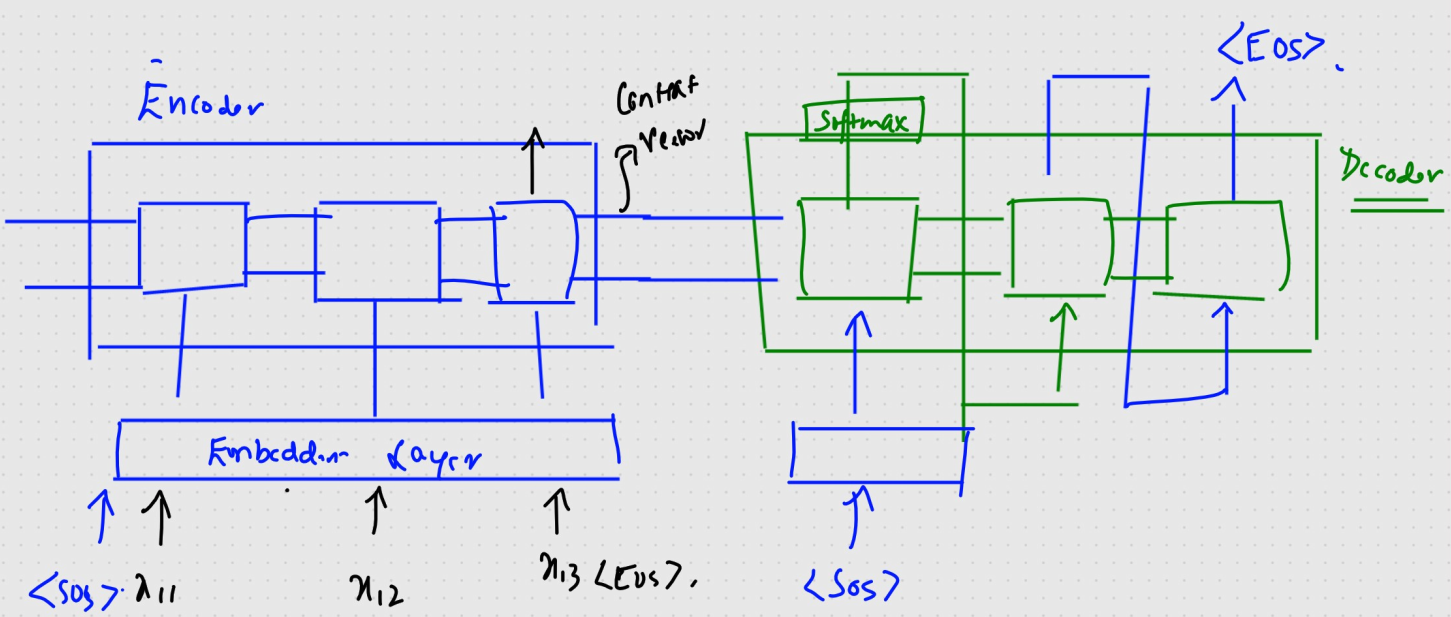
⇒ Context of this Sentence.

Optimizer

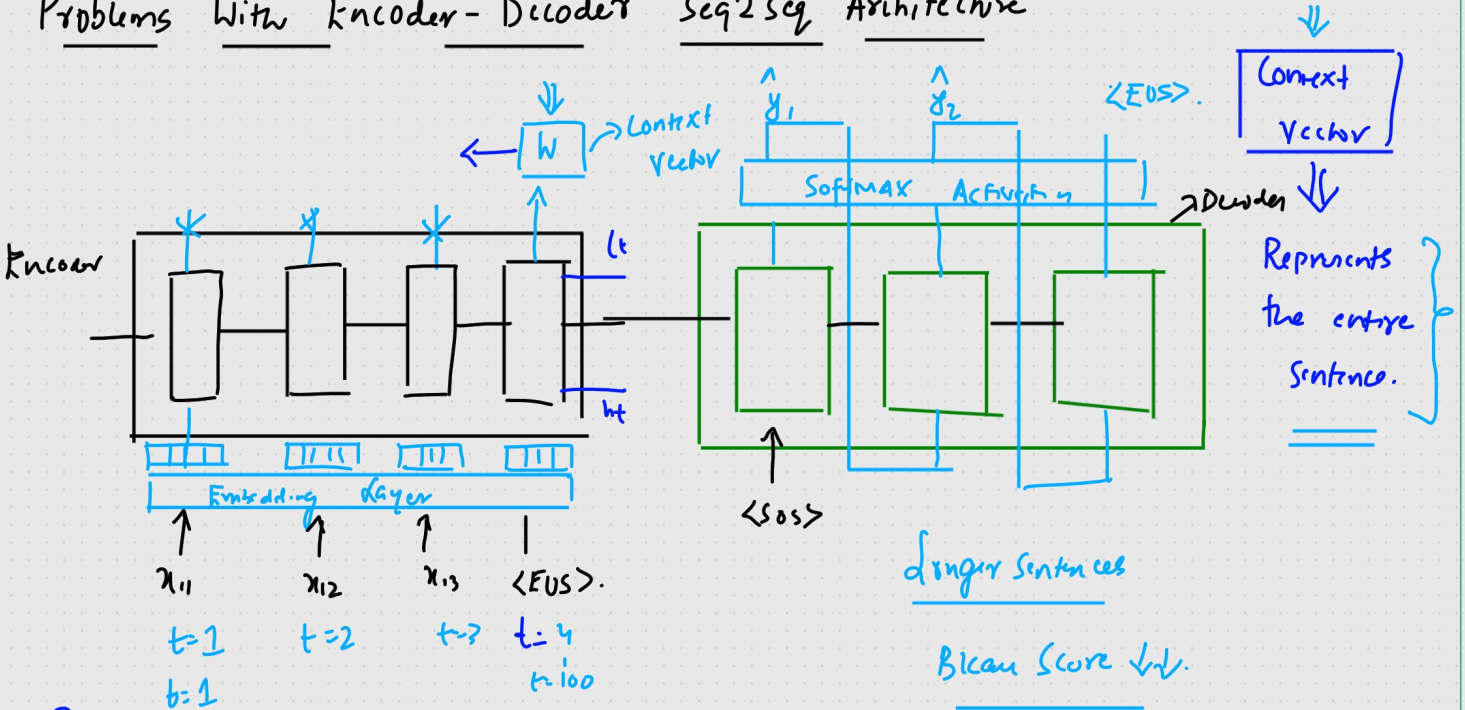
Loss $(y-\hat{y})^2$

$\hat{y}$ [- - -] [- - -] [- - -]

y [. . .] [. . .] [. . .]



Problems With Encoder-Decoder Seq2seq Architecture



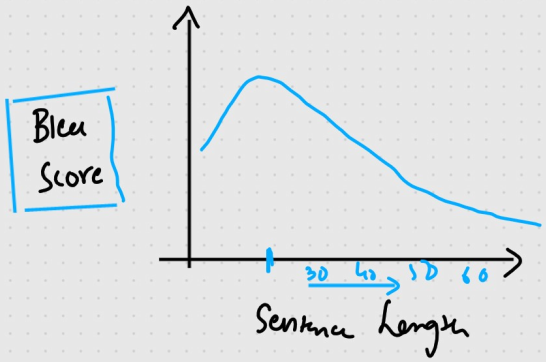
Context Vector

Represents the entire sentence.

Longer Sentences

Bleau Score ↓↓.

Researchers : Sentences of varying length



⇒ Seq to Seq Data.

(*) Attention Mechanism → Seq2Seq Network

longer paragraph → {context vector}.

+

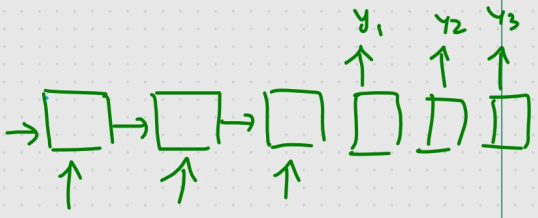
{context}

Encoder And Decoder

1. Simple RNN → Vanishing Gradient Problem
2. LSTM RNN → } Long Short Term Memory.
3. GRU RNN → }
4. Bidirectional RNN ←

Type of RNN

1. Many to many RNN



Encoder And Decoder

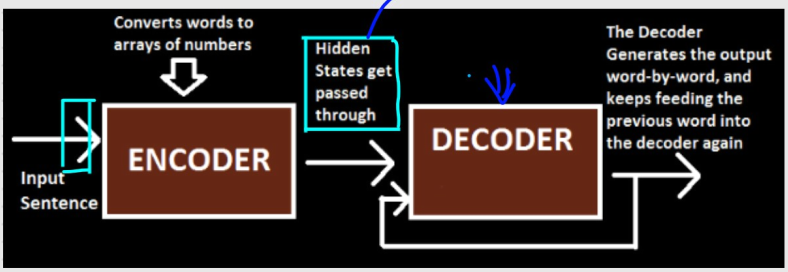
Sequences I/p — O/p Sequence of Words

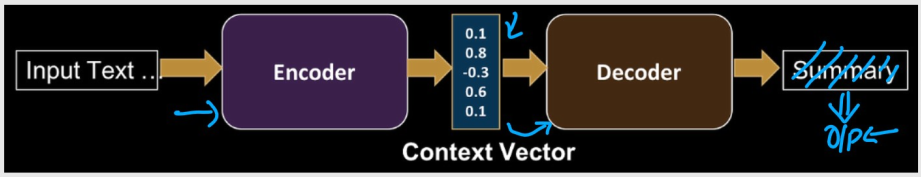
Eg: One language To other

English → French

Eg: Linked chat → Hi, How are you?

Simple Working



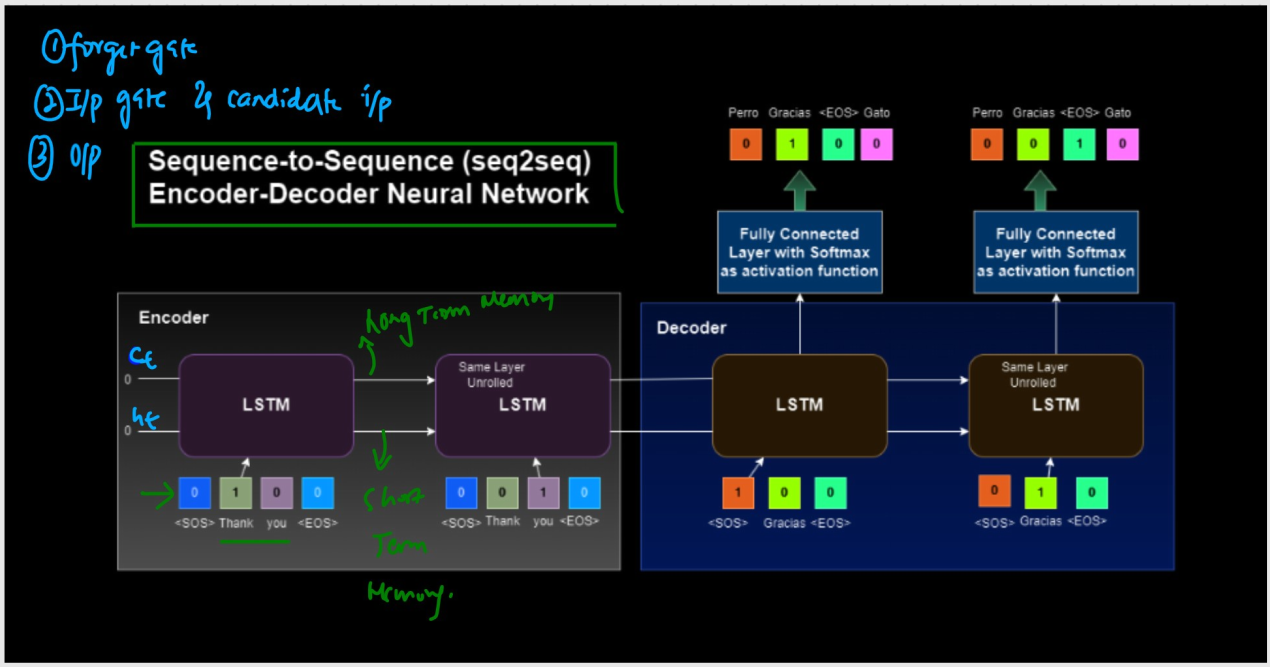


1. Encoder ⇒ I/p ⇒ Context Vector ← Vectors
2. Decoder ⇒ ⇒ O/P

Usecase

1. Language Translation
2. Text Generation
3. Text Suggestion.

RNN → Vanishing Gradient Problem



① forget gate
② I/p gate & candidate i/p
③ O/p

Dataset — True O/p — y_{truth} {Training}

English — French

<SOS> Thank you <EOS>. Gracias → <SOS> GRACIAS <EOS>. GRACIOUS <EOS>.

$\hat{y}$ [y $\hat{y}$] ∀w

Encoder → Context Vectors → Decoder

Softmax [0.1, 0.6, 0.3] [0.1, 0.3, 0.6] $\hat{y}$ [- - -]

Long Term Memory — Short Term Memory

C_t, h_t

OHE ⇒ <SOS> Thank you <EOS>. — t=1 t=2 t=3 t=4

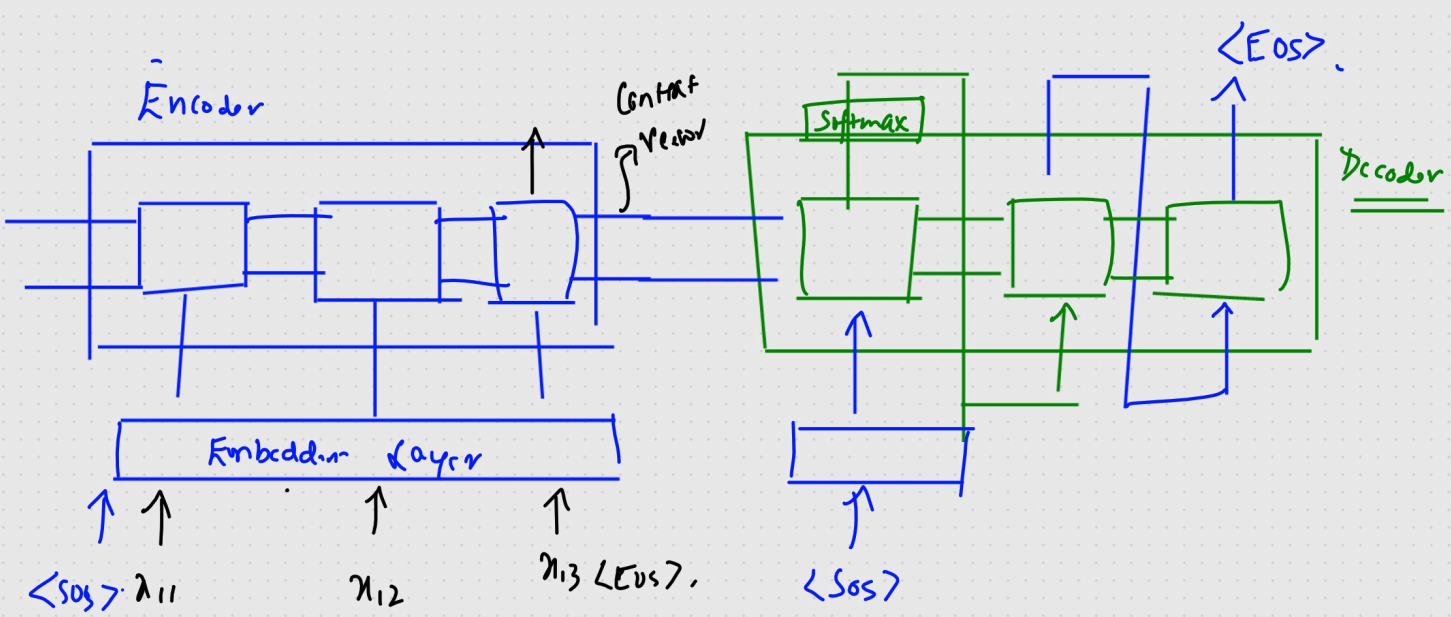
Context of this Sentence.

Optimizer

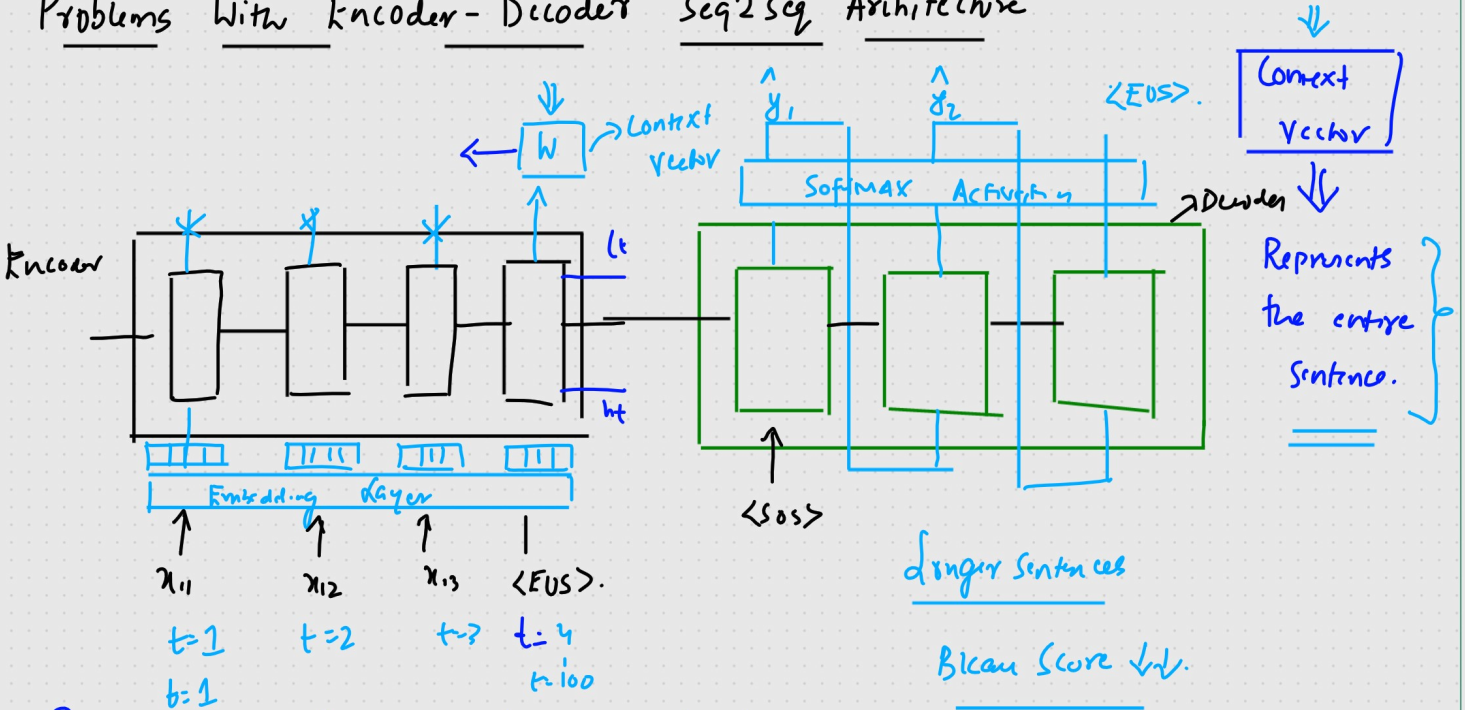
Loss $(y - \hat{y})^2$ ∀w

$\hat{y}$ [- - -] [- - -] [- - -]

y [1 1 1] [1 1 1] [1 1 1]



Problems With Encoder-Decoder Seq2seq Architecture



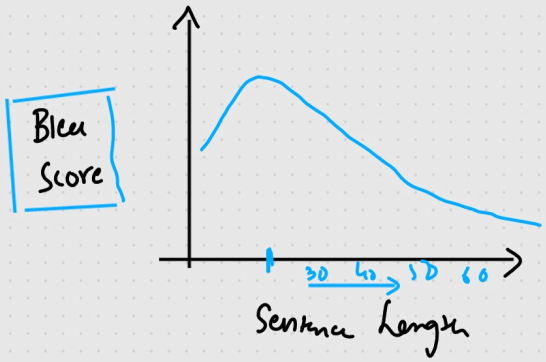
Context Vector

Represents the entire sentence.

Longer Sentences

Bleau Score ↓↓.

Researchers : Sentences of varying length



⇒ Seq to Seq Data.

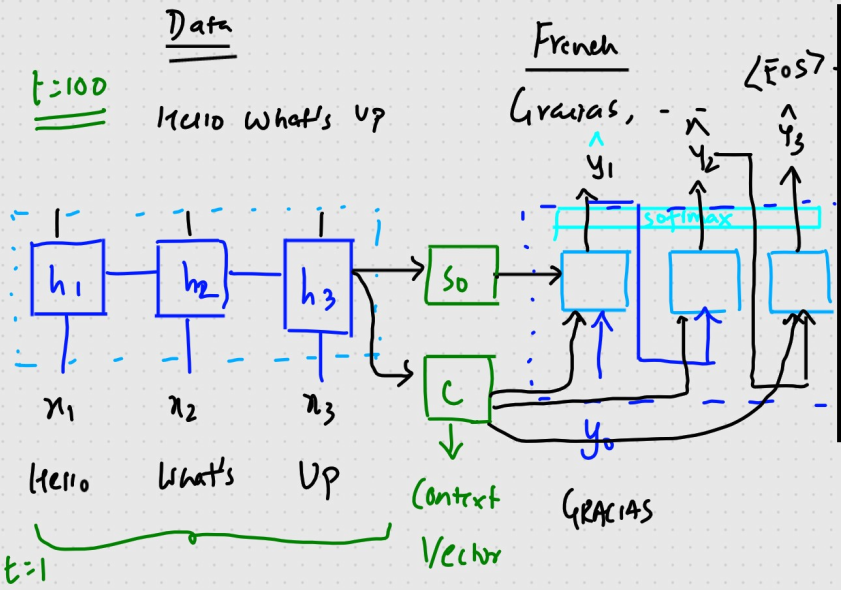
(*) Attention Mechanism → Seq2Seq Network

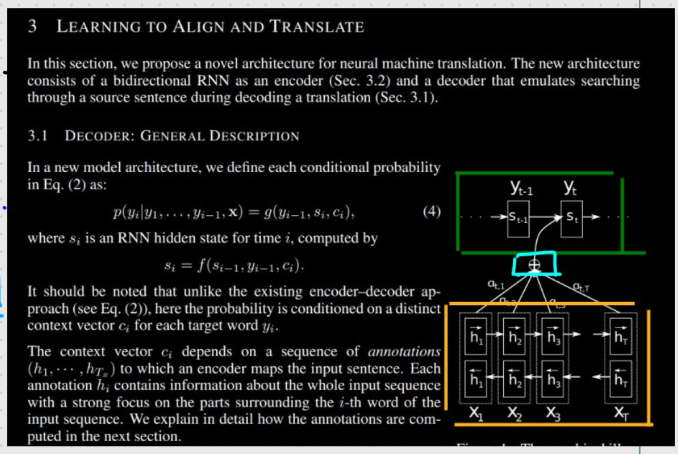
Longer paragraph → {context vector}.

+

{context}

Attention Mechanism | Seq2Seq Networks





3 LEARNING TO ALIGN AND TRANSLATE

In this section, we propose a novel architecture for neural machine translation. The new architecture consists of a bidirectional RNN as an encoder (Sec. 3.2) and a decoder that emulates searching through a source sentence during decoding a translation (Sec. 3.1).

3.1 DECODER: GENERAL DESCRIPTION

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\dots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i), \quad (4)$$

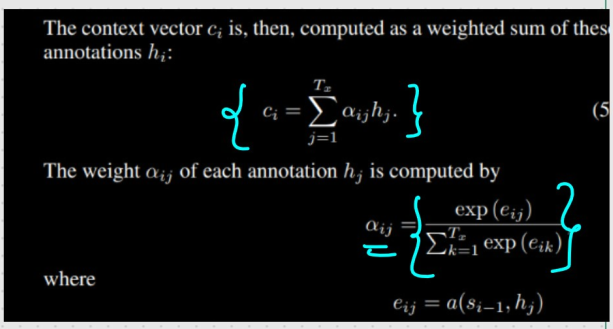
where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The context vector c_i depends on a sequence of *annotations* $(h_1,\cdots,h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

Encoder Decoder Architecture



The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5$$

The weight α_{ij} of each annotation h_j is computed by

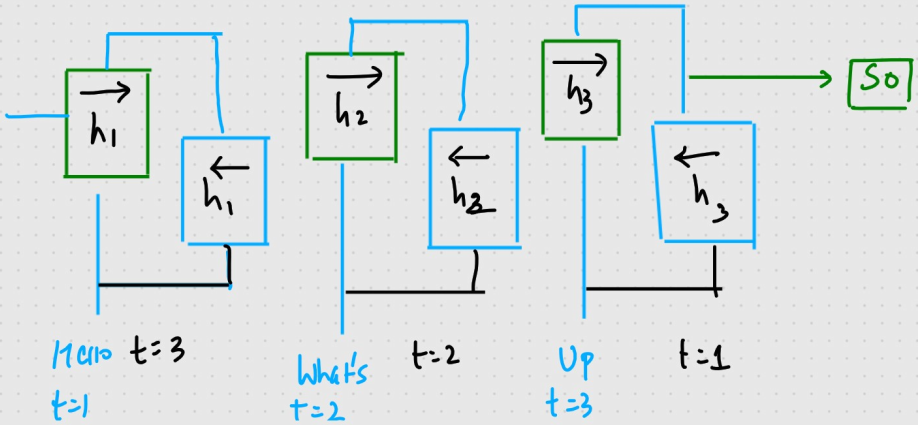
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

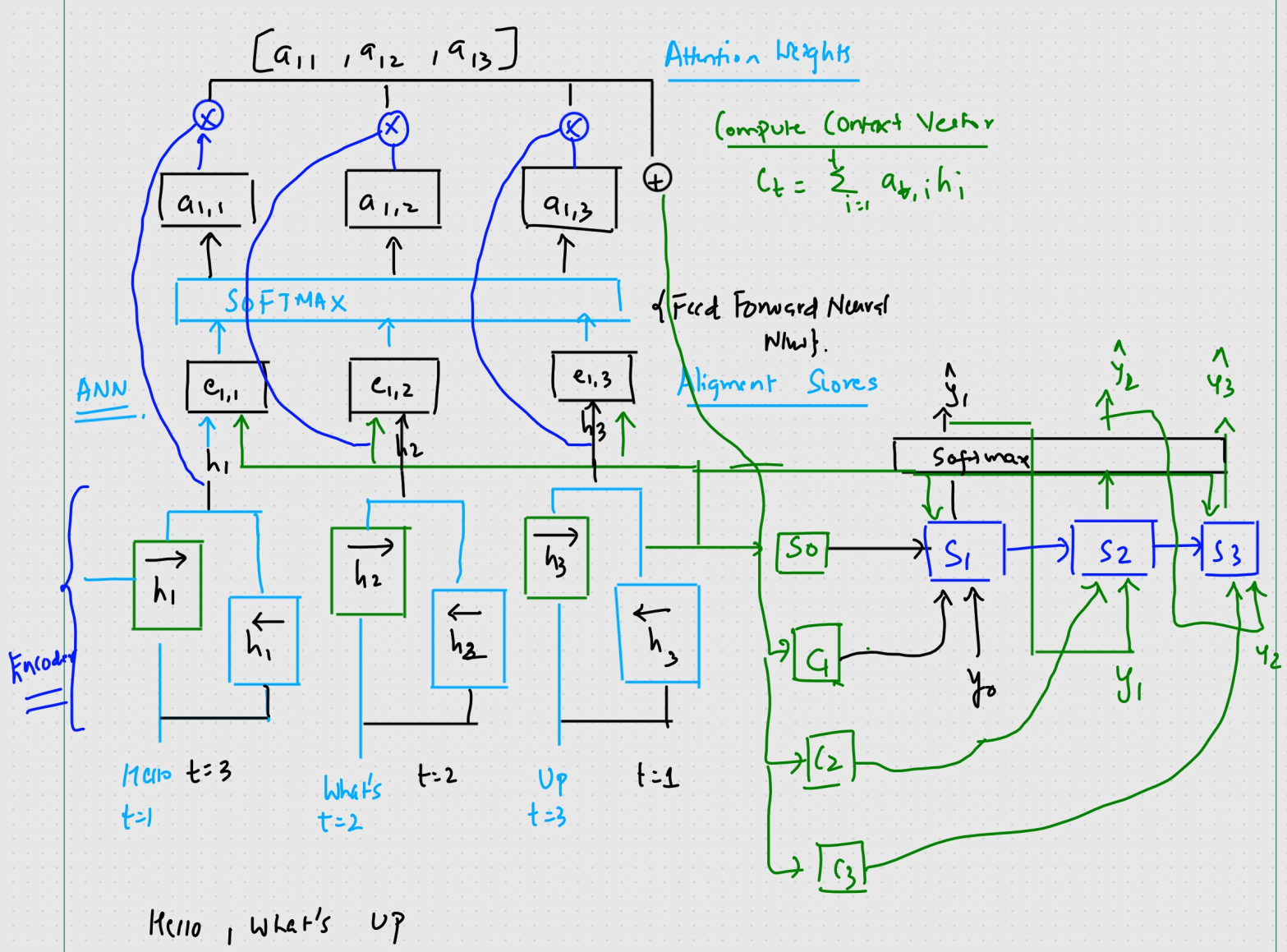
where

$$e_{ij} = a(s_{i-1}, h_j)$$

Attention Mechanism

https://erdem.pl/2021/05/introduction-to-attention-mechanism

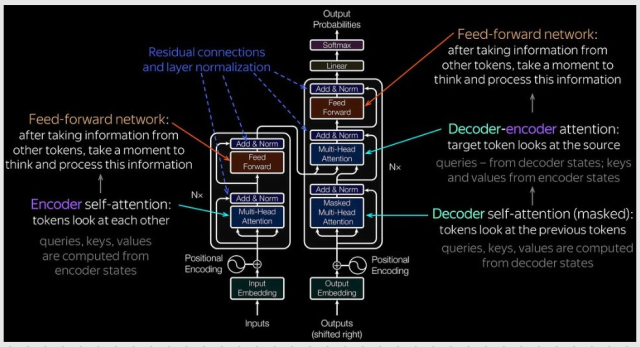



[a11 , a12 , a13]
Attention Weights
Compute Context Vector
$c_t = \sum_{i=1} a_{t,i} h_i$
a1,1
a1,2
a1,3
SOFTMAX
{Feed Forward Neural Nwk}.
Alignment Scores
ANN
e1,1
e1,2
e1,3
h1
h2
h3
Softmax
y1
y2
y3
Encoder
h1
h2
h3
S0
S1
S2
S3
G
y0
C2
C3
Hello t=3
t=1
What's t=2
t=2
Up t=3
t=1
Hello , What's Up

Introduction To Transformers

1) RNN / LSTM / GRU RNN
2) Encoder Decoder Architecture
3) ATTENTION MECHANISM
4) TRANSFORMERS
   1. Why Transformers?
   2. Architecture of Transformers?
   3. SELF ATTENTION → Q, K, V
   4. Positional Encoding
   5. Multi Head ATTENTION
   6. Combining the working of Transformers

Architecture



Generative AI → LLM, Multimodel

BERT, GPT ←

↓

Open AI → Chat GPT

↓

GPT-4o

1) What And Why → Transformers

Transformers in natural language processing (NLP) are a type of deep learning model that use self-attention mechanisms to analyze and process natural language data. They are encoder-decoder models that can be used for many applications, including machine translation. ⇒ Seq2Seq Task

Eg: Language Translation → Google Translation

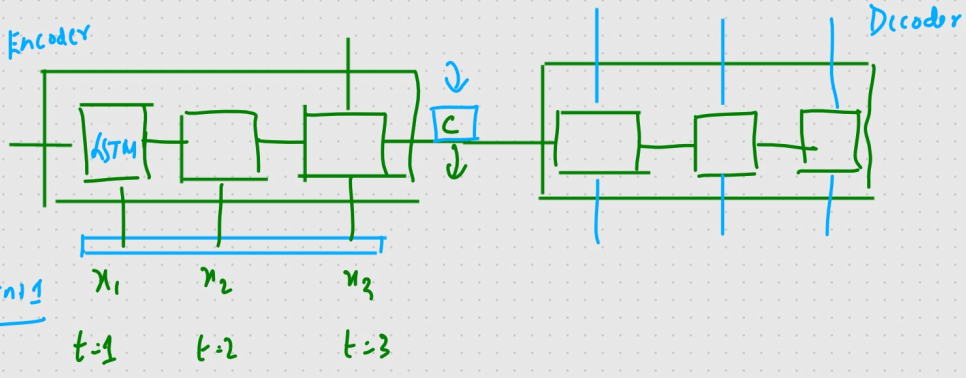
English → French

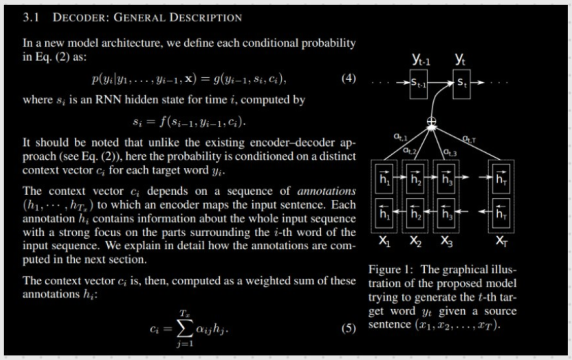
i/p ⇒ Many → o/p: Many. {Length of the sentence}.

Encoder – Decoder

Sentence Length ↑↑ → Bleau Score ↓↓

Length Sentence ↑↑.





3.1 Decoder: General Description

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\dots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i), \quad (4)$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

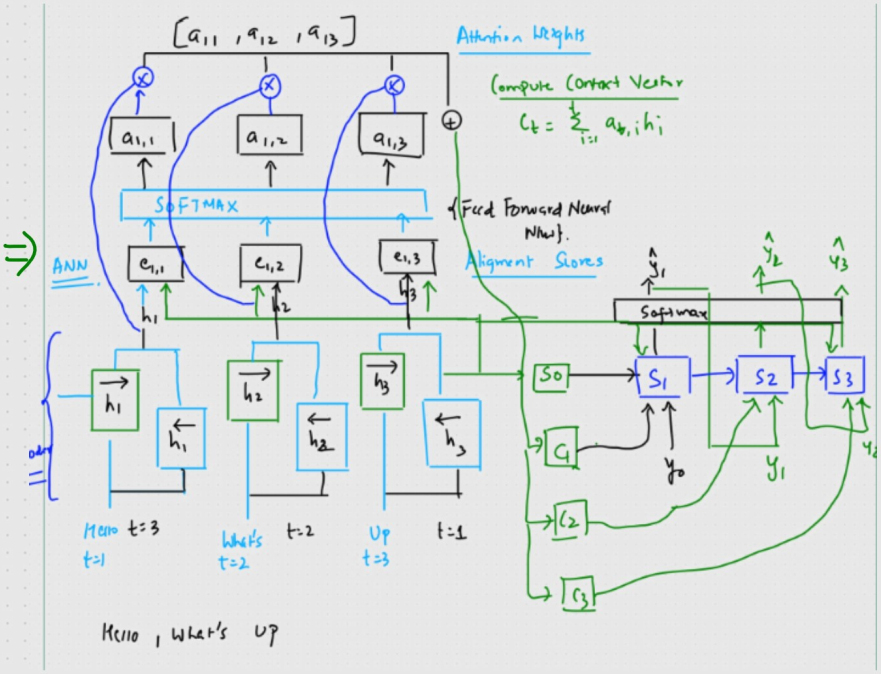
It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The context vector c_i depends on a sequence of *annotations* $(h_1,\cdots,h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.



Additional Context ⟶ Decoder

Long Sentence. Accuracy ↑↑.

Attention Mechanism ↓

① Parallely we cannot send all the words in a sentence ⟶ **Scalable**

DATASET ⟶ Huge ⟶ Scalable With Respect to Training.

TRANSFORMERS ✱ LSTM RNN

↳ Self Attention Module ← All the words will be parallely sent to encoder.

↓

Positional Encoding

Transformer ↑↑ DATASET ⟶ Amazing SOTA ← NLP

Transfer Learning ⟶ MultiModal Task ⟶ NLP + Image ←

Transformers ÷ AI Space ⟶ SOTA Model ⟶

BERT GPT ⟶ Transfer Learning ⟶ SOTA Models ⟶ DALL·E } LLM's Generative AI

↑↑

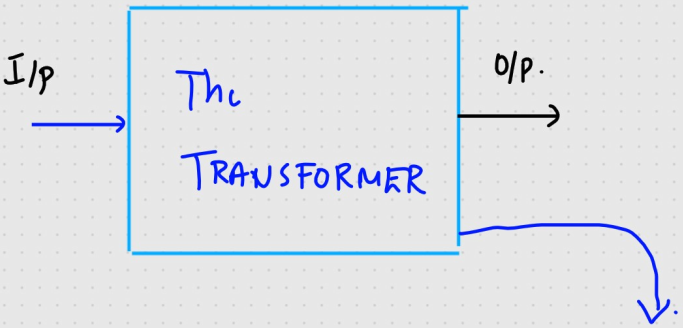
Train Huge Data

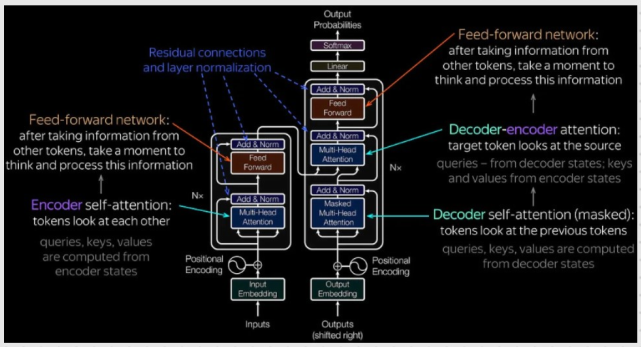
② Contextual Embedding → Self Attention — Contextual Vectors

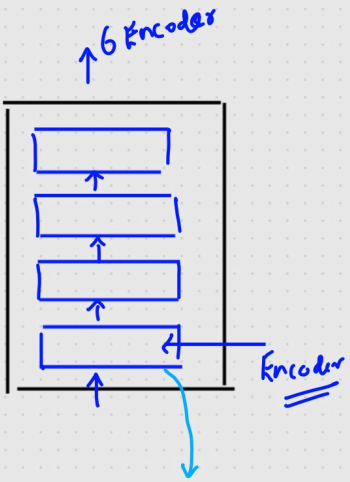
Eg: My name is KRISH And I play CRICKET

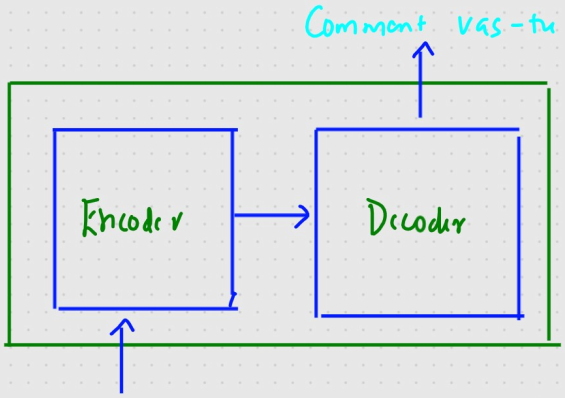
Word2Vec → Embedding Layer → Contextual vector Embedding

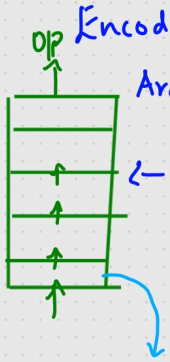
② Basic Transformer Architecture — {Seq2Seq Task} → Language Translation {Eng → French}

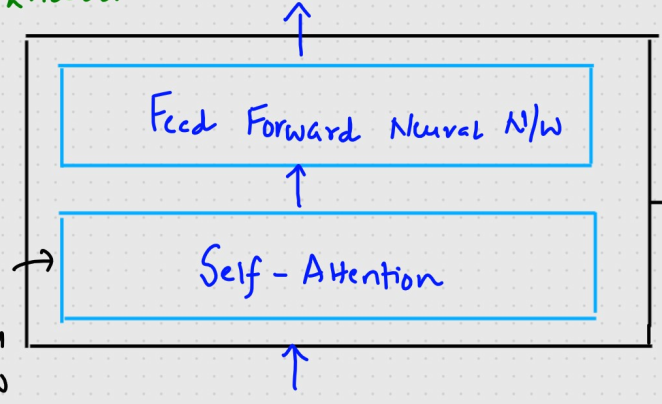


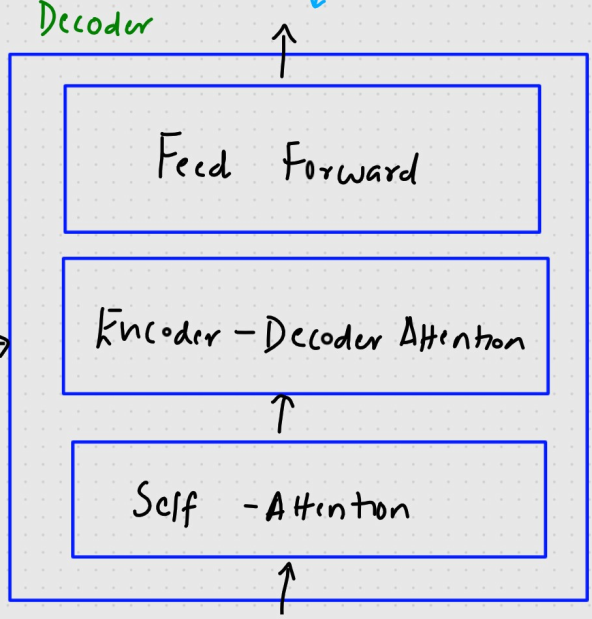




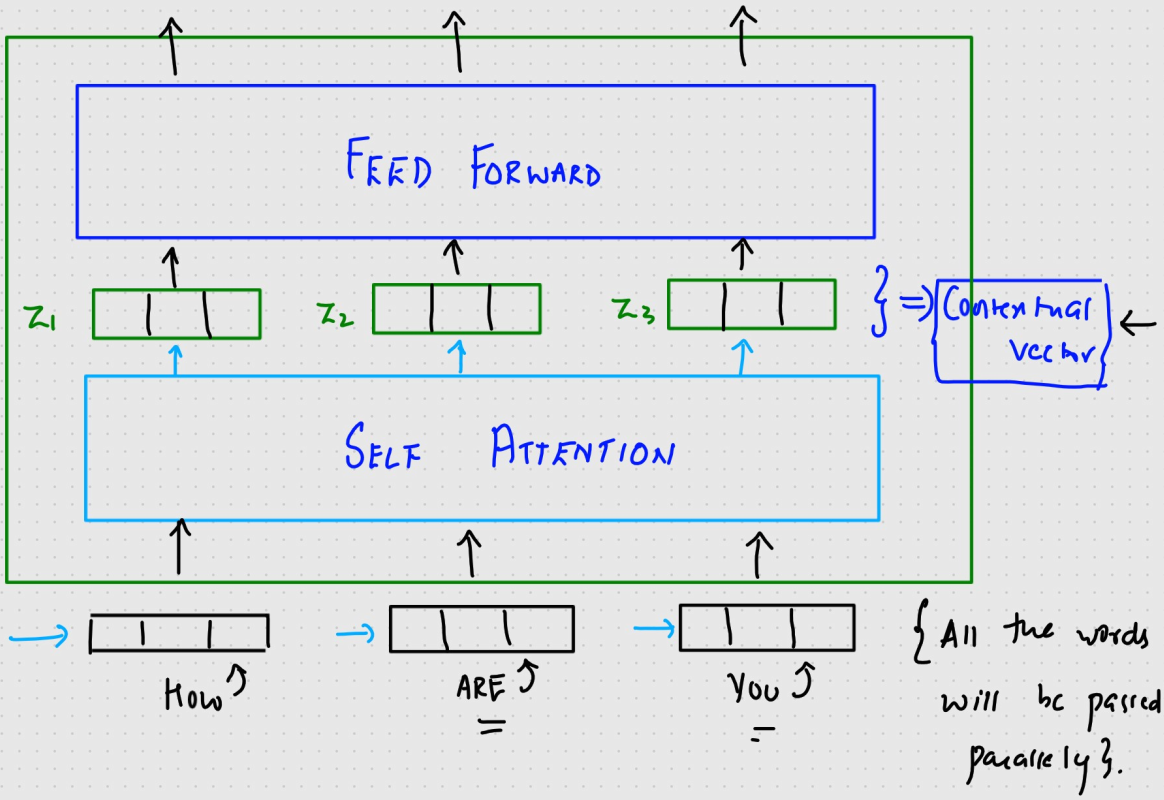








Encoder

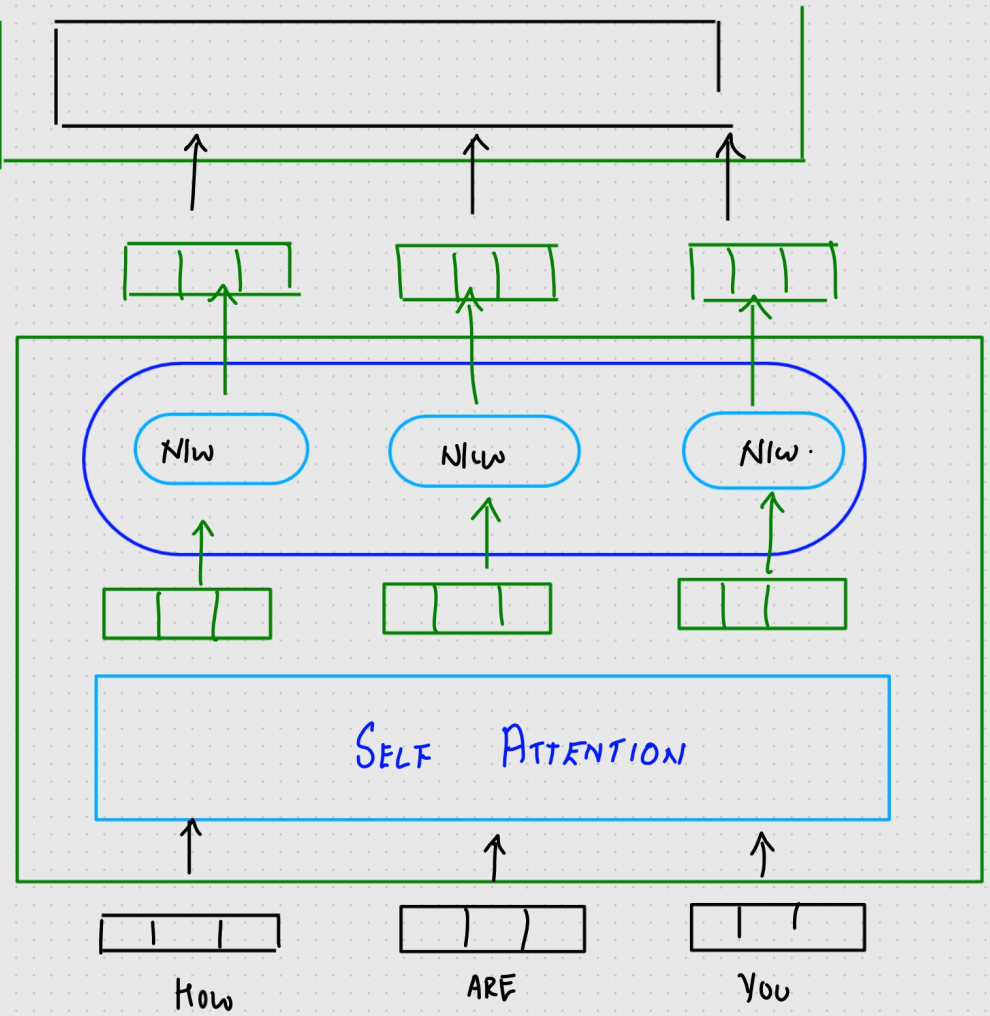


Vectors → Embedding Layer

All the words will be passed parallely.

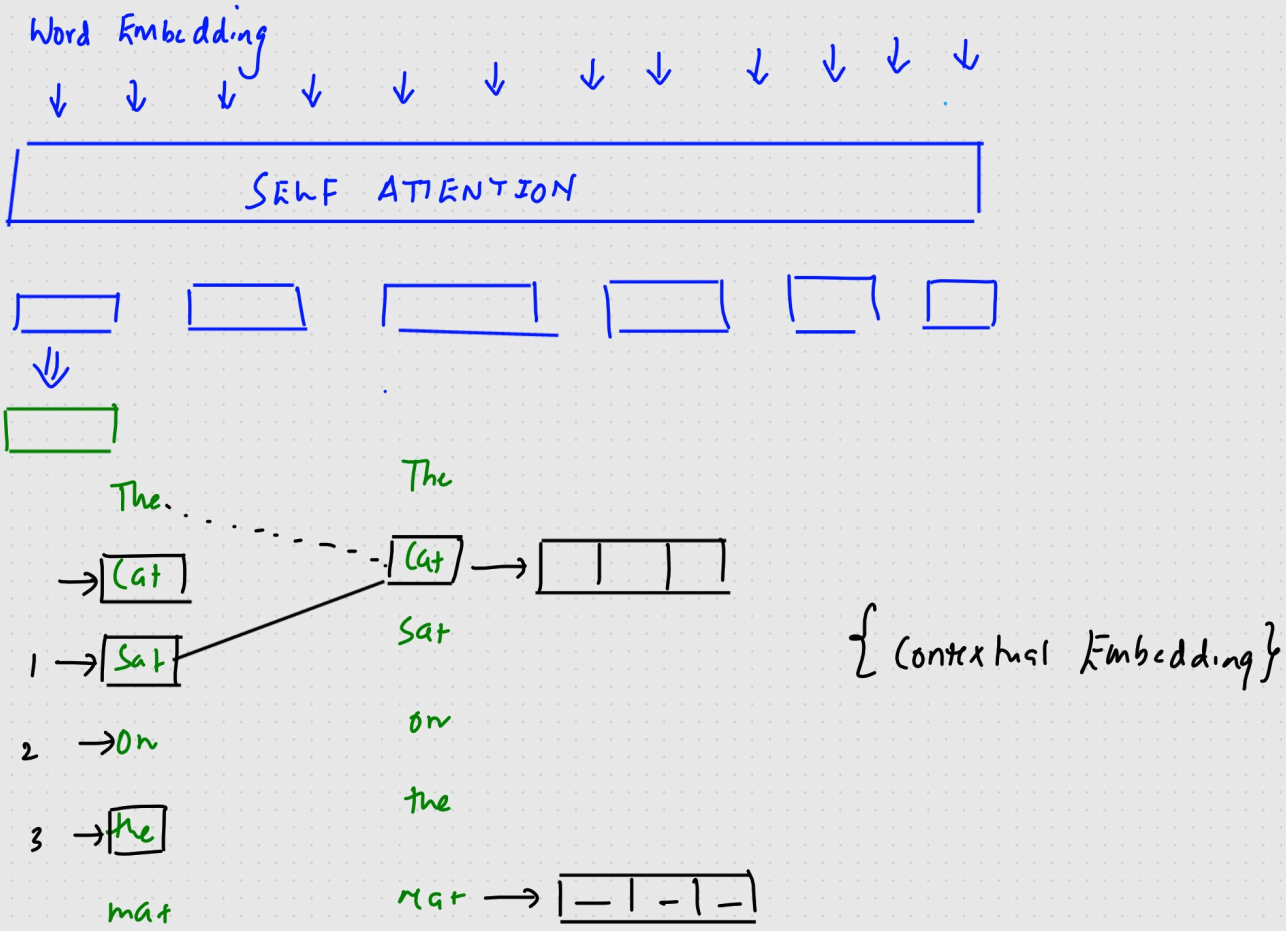
Encoder 2

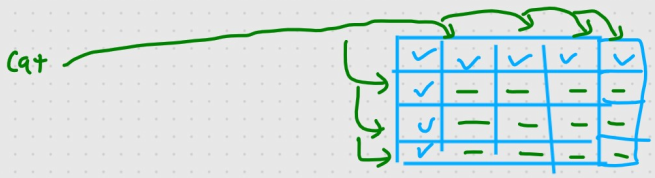
Encoder



Self Attention At a Higher Level

Eg: The cat sat on the mat, the cat lay on the rug.





Self Attention In Detail

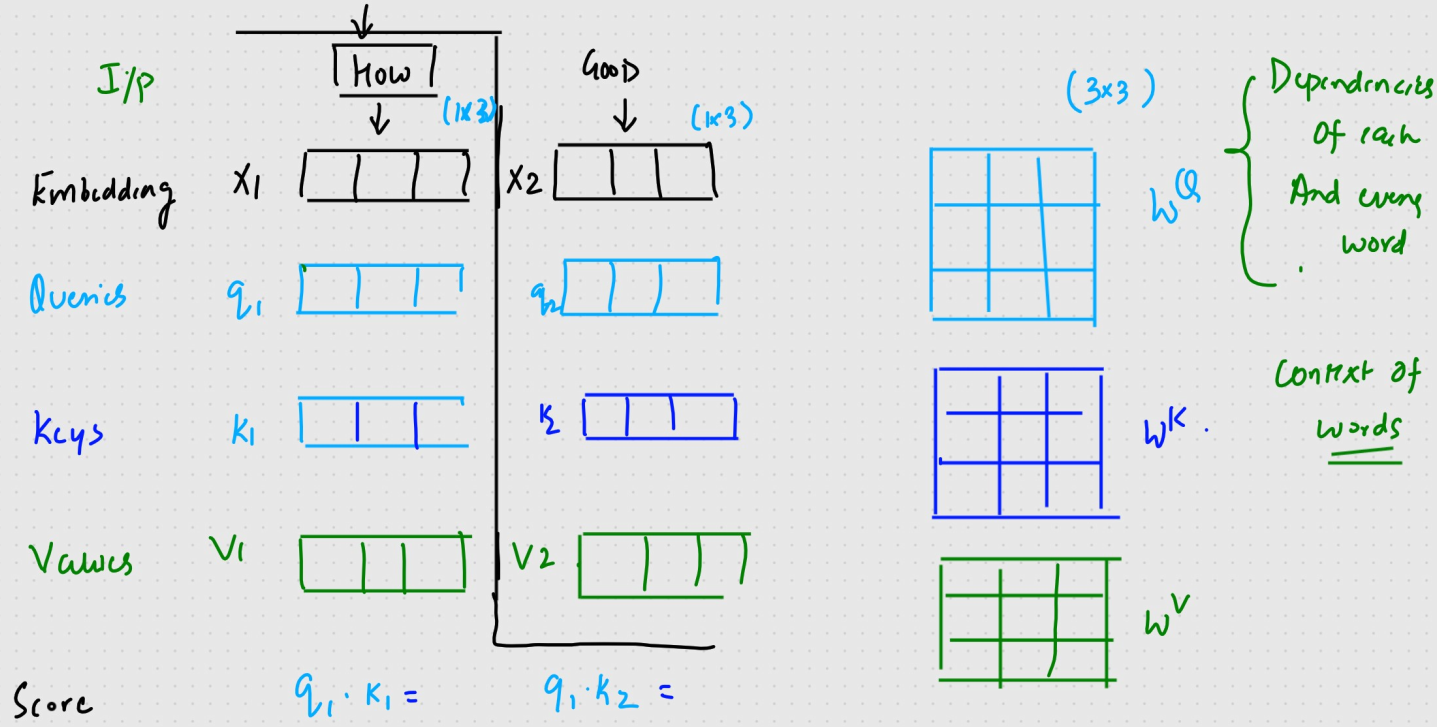
① To create 3 vectors from each of the encoder i/p: Query vector, Key vector, value vector. ⇒ Contextual Embedding

Eg: YT ⇒ Search keywords (Topic's) ⇒ {Query}.

Query ⟶ {Key} → Tags, Description, Title ⇒ Value ⇒ O/P Video

② Second step in calculating self attention is to calculate the score.

The score determines how much focus to place on the other part of the sentence.



(*) How much Focus to place on other part of the i/p sentence as we encode a word at certain position

Divide $\sqrt{dk}$

⇓

{ More stable gradients }

Dimension=3 ↓ Variance ≈ $\frac{\text{Variance}}{\sqrt{dk}}$

Dimension ↑↑↑ T12 O/p ↑↑↑ Variance ↑↑↑ ⇒ $\frac{\text{Variance}}{\sqrt{dk}}$

SoftMax

[0-1] [0-1].

0.88 + 0.12 = 1

Softmax => The Softmax score determines how much each word will be expressed at this position

X X X

Value Vectors V1 [1 1] V2. [1 0.5 1]

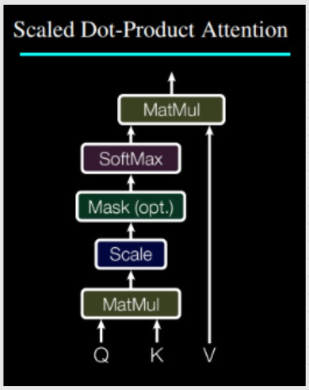
0.88 [1 1 1]

[0.88 | 0.88 | 0.88] + [0.12 | 0.6 | 0.12]

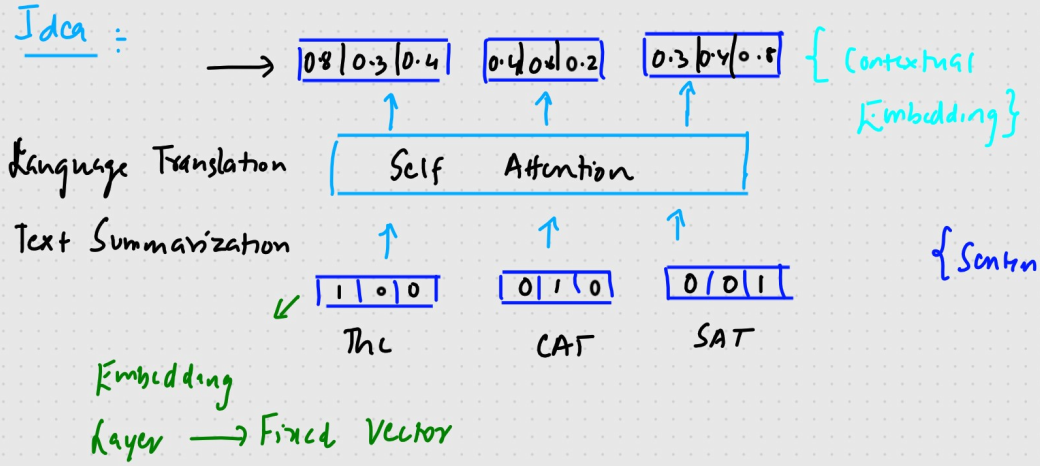
Contextual vector → Z1 [ ] ↑ Z2 [ ]

Self Attention At Higher And Detailed Level

Self-attention, also known as scaled dot-product attention, is a crucial mechanism in the transformer architecture that allows the model to weigh the importance of different tokens in the input sequence relative to each other



Idea :



{Sentence, Dataset}.

1) Inputs: Queries, Keys, And Values

Model → Queries, Keys And Values

1. Query Vectors (Q):
Role: Query vectors represent the token for which we are calculating the attention. They help determine the importance of other tokens in the context of the current token.

Importance:

Focus Determination: **Queries help the model decide which parts of the sequence to focus on for each specific token. By calculating the dot product between a query vector and all key vectors, the model assesses how much attention to give to each token relative to the current token.**

Contextual Understanding: **Queries contribute to understanding the relationship between the current token and the rest of the sequence, which is essential for capturing dependencies and context.**

2. Key Vectors (K):
Role: Key vectors represent all the tokens in the sequence and are used to compare with the query vectors to calculate attention scores.

Importance:

Relevance Measurement: **Keys are compared with queries to measure the relevance or compatibility of each token with the current token. This comparison helps in determining how much attention each token should receive.**
Information Retrieval: **Keys play a critical role in retrieving the most relevant information from the sequence by providing a basis for the attention mechanism to compute similarity scores.**

3. Value Vectors (V):
Role: Value vectors hold the actual information that will be aggregated to form the output of the attention mechanism.

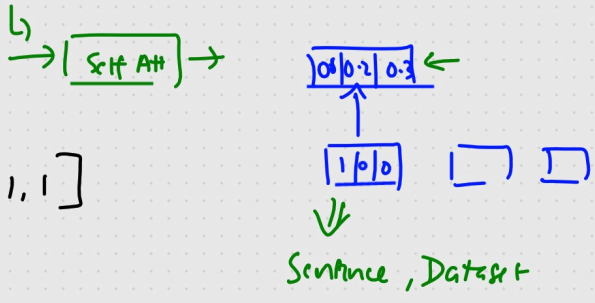
Importance:

Information Aggregation: Values contain the data that will be weighted by the attention scores. The weighted sum of values forms the output of the self-attention mechanism, which is then passed on to the next layers in the network.
Context Preservation: By weighting the values according to the attention scores, the model preserves and aggregates relevant context from the entire sequence, which is crucial for tasks like translation, summarization, and more.

Input Sequence = ["The", "CAT", "SAT"]

Embedding Size = 4

Q, K, V ⇒ 4



① Token Embedding

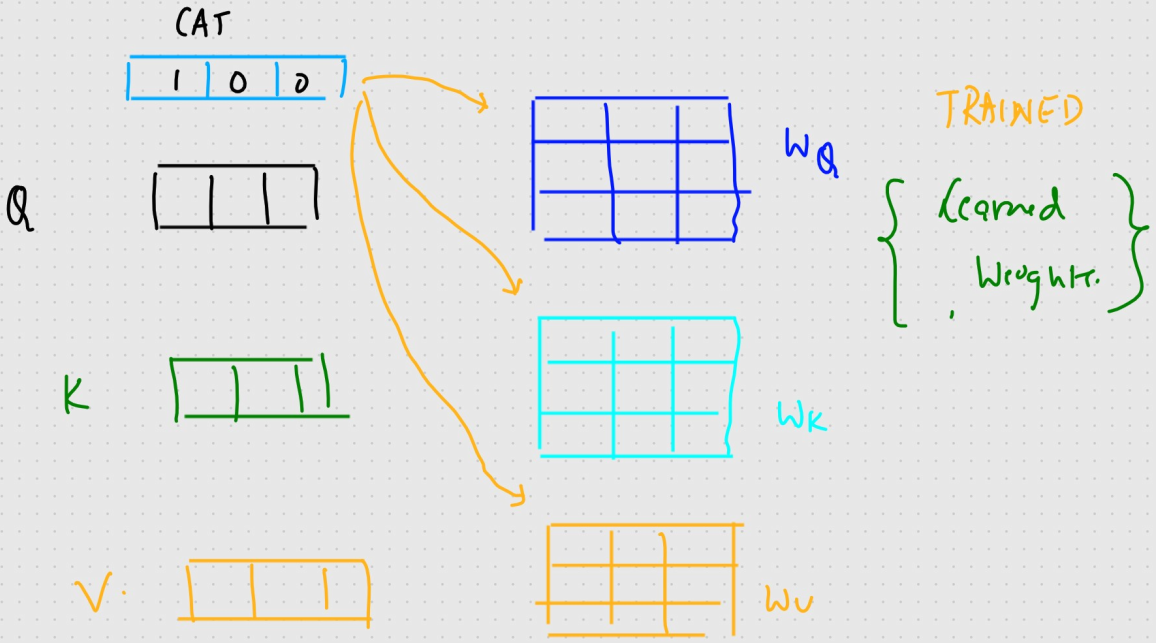
$E_{The} = [1\ 0\ 1\ 0]$

$E_{CAT} = [0\ 1\ 0\ 1]$

$E_{Sat} = [1, 1, 1, 1]$

② Linear Transformation

We create Q, K, V by multiplying the embeddings by learned weights matrices W_Q, W_K and W_V.



Lets consider

$W_Q = W_K = W_V = I$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \leftarrow$$

$$Q_{The} = [1\ 0\ 1\ 0] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [1\ 0\ 1\ 0]$$

$$K_{The} = [1\ 0\ 1\ 0]$$

$$V_{The} = [1\ 0\ 1\ 0]$$

① $Q_{The} = K_{The} = V_{The} = [1\ 0\ 1\ 0]$

② $Q_{CAT} = K_{CAT} = V_{CAT} = [0\ 1\ 0\ 1]$

③ $Q_{SAT} = K_{SAT} = V_{SAT} = [1, 1, 1, 1]$

③ Compute Attention Scores

$[1\ 0\ 1\ 0]\begin{bmatrix}1\\0\\1\\0\end{bmatrix}$

The

$\text{Score}(Q_{The}, K_{The}) = [1\ 0\ 1\ 0] \cdot [1\ 0\ 1\ 0]^T = 2$

$\text{Score}(Q_{The}, K_{CAT}) = [1\ 0\ 1\ 0] \cdot [0\ 1\ 0\ 1]^T = 0$

$\text{Score}(Q_{The}, K_{SAT}) = [1\ 0\ 1\ 0] \cdot [1,1,1,1]^T = 2$

For the token CAT

$\text{Score}(Q_{CAT}, K_{The}) = [0\ 1\ 0\ 1] \cdot [1\ 0\ 1\ 0]^T = 0$

$\text{Score}(Q_{CAT}, K_{CAT}) = [0\ 1\ 0\ 1] \cdot [0\ 1\ 0\ 1]^T = 2$

$\text{Score}(Q_{CAT}, K_{SAT}) = [0\ 1\ 0\ 1] \cdot [1,1,1,1]^T = 2.$

For the Token SAT

$\text{Score}(Q_{SAT}, K_{The}) = [1,1,1,1] \cdot [1\ 0\ 1\ 0]^T = 2$

$\text{Score}(Q_{SAT}, K_{CAT}) = 2$

$\text{Score}(Q_{SAT}, K_{SAT}) = 4$

④ Scaling ÷

We take up the scores and scale down by dividing the scores by the

$\sqrt{d_k} \Rightarrow d_k = 4 \qquad \sqrt{d_k} = 2.$

Scaling in the attention mechanism is crucial to prevent the dot product from growing too large. ⇒ Ensure stable gradients during Training.

d_k is large →

① Gradient Exploding

② Softmax Saturation } → Vanishing Gradient Problem.

$Q = [2\ 3\ 4\ 1]$ $K_1 = [1\ 0\ 1\ 0]$ $K_2 = [0\ 1\ 0\ 1]$

Without Scaling

$Q \cdot K_1^T = 2\times1 + 3\times0 + 4\times1 + 1\times0 = 2 + 4 = 6.$

$Q \cdot K_2^T = 2\times0 + 3\times1 + 4\times0 + 1\times1 = 0 + 3 + 0 + 1 = 4$

*) Score $[6, 4]$ $\Rightarrow$ Scaling Not Applied

$$\text{Softmax}([6,4]) = \left[\frac{e^6}{e^6+e^4}, \frac{e^4}{e^6+e^4}\right] = \left[\frac{e^6}{e^6(1+e^{-2})}, \frac{e^4}{e^4(e^2+1)}\right]$$

(*) Property of Softmax W_Q, W_K, W_V

$$= \left[\frac{1}{(1+e^{-2})}, \frac{1}{(e^2+1)}\right]$$

$([10, 1]) = [0.99, 0.01]$

Dot product = Large values

$\approx [0.88, 0.12]$.

Most of the attention weight is assigned to the first key vector, very little to the second vector,

With Scaling

$\sqrt{d_k} = \sqrt{4} = \sqrt{2}$ $\Rightarrow$ Variance 2, 3

dimension ↑ Variance ↑2

① Compute Scaled Dot Product

$[6, 4] \Rightarrow \text{Scale} \Rightarrow \left[\frac{6}{2}, \frac{4}{2}\right] = [3, 2]$.

$\sqrt{d_k}$ ↑ same

$$\text{Softmax}([3,2]) = \left[\frac{e^3}{e^3+e^2}, \frac{e^2}{e^3+e^2}\right] = \left[\frac{e^3}{e^3(1+e^{-1})}, \frac{e^2}{e^2(e^1+1)}\right]$$

$= [0.73, 0.27] \Rightarrow$ Attention Weights

(*) Here, the attention weights are more balanced compared to the unscaled case

Summary of Importance
Stabilizing Training: Scaling prevents extremely large dot products, which helps in stabilizing the gradients during backpropagation, making the training process more stable and efficient.

Preventing Saturation: By scaling the dot products, the softmax function produces more balanced attention weights, preventing the model from focusing too heavily on a single token and ignoring others.
Improved Learning: Balanced attention weights enable the model to learn better representations by considering multiple relevant tokens in the sequence, leading to better performance on tasks that require context understanding.

Scaling ensures that the dot products are kept within a range that allows the softmax function to operate effectively, providing a more balanced distribution of attention weights and improving the overall learning process of the model.

(4) Scaling $= \sqrt{d_k} = \sqrt{4} \Rightarrow 2$

Similarly scaling will be done for all other Tokens.

$\text{Scaled\_score}(Q_{The}, K_{The}) = 2/2 = 1$

$\text{Scaled\_score}(Q_{The}, K_{CAT}) = 0/2 = 0$

$\text{Scaled\_score}(Q_{The}, K_{SAT}) = 2/2 = 1$

(5) Apply Softmax

$\text{ATTENTION WEIGHTS}_{\text{"The"}} = \text{Softmax}([1,0,1]) = [0.4223, 0.1554, 0.4223]$

$\text{ATTENTION WEIGHTS}_{\text{"CAT"}} = \text{Softmax}([0,2,2]) = [0.1554, 0.4223, 0.4223]$

$\text{ATTENTION WEIGHTS}_{\text{"SAT"}} = \text{Softmax}([2,2,4]) = [0.2119, 0.2119, 0.1762]$.

(6) Weight Sum of Values

We multiply the attention weights by corresponding value vectors

For the token The =

$$Output_{(The)} = 0.4223 * V_{The} + 0.1554 * V_{CAT} + 0.4223 \cdot V_{Sat}$$

$$= 0.4223 [1\ 0\ 1\ 0] + 0.1554 * [0\ 1\ 0\ 1] + 0.4223 * [1\ 1\ 1\ 1]$$

$$= [0.4223, 0, 0.4223, 0] + [0, 0.1554, 0, 0.1554] + [0.4223, 0.4223, 0.4223, 0.4223]$$

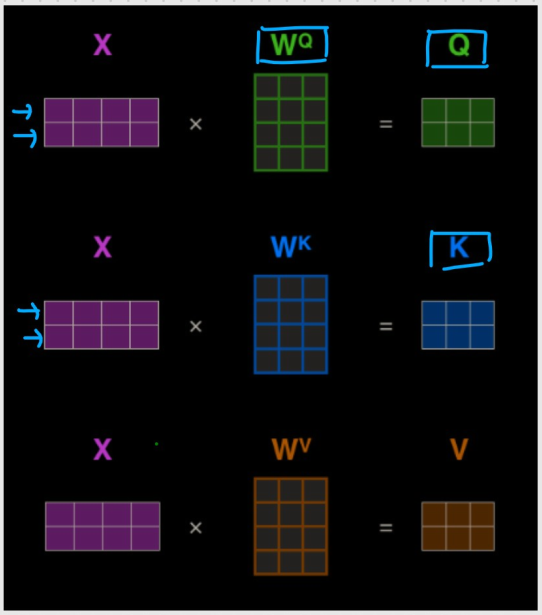
$$= [1.2669, 0.9999, 1.2669, 0.9999].$$

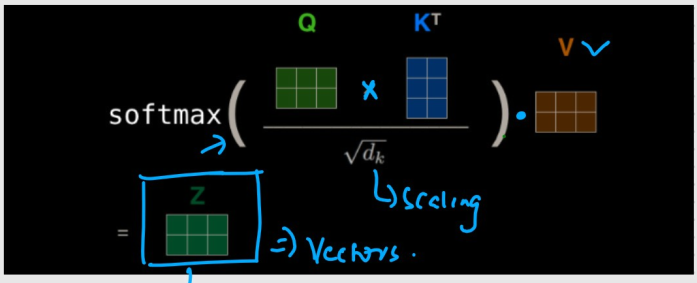
↓ Contextual vector

The [1 0 1 1] [0] ⇒ Self Attention ⇒ [1.2669, 0.9999, 1.2669, 0.9999].

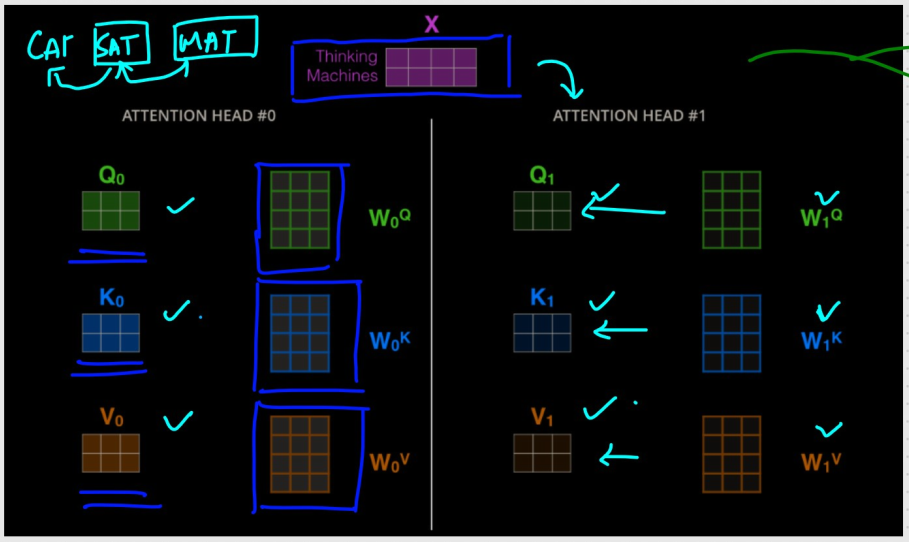
① ↳ Q, K, V [W^Q, W^K, W^V]
② ↳ Attention Score
③ ↳ Scaled
④ ↳ Softmax
⑤ ↳ Weighted Sum of Value (Softmax × V).

④ Multi Head Attention





→ Self Attention with Multi Heads



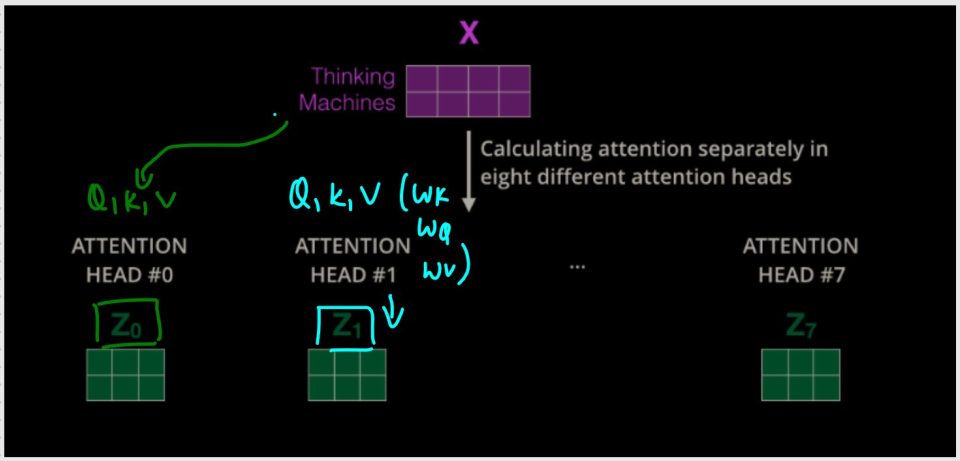
It expands model's ability to focus on different position of tokens.

Score, Softmax X V

[_ _ _] → Z0 (Vectors)

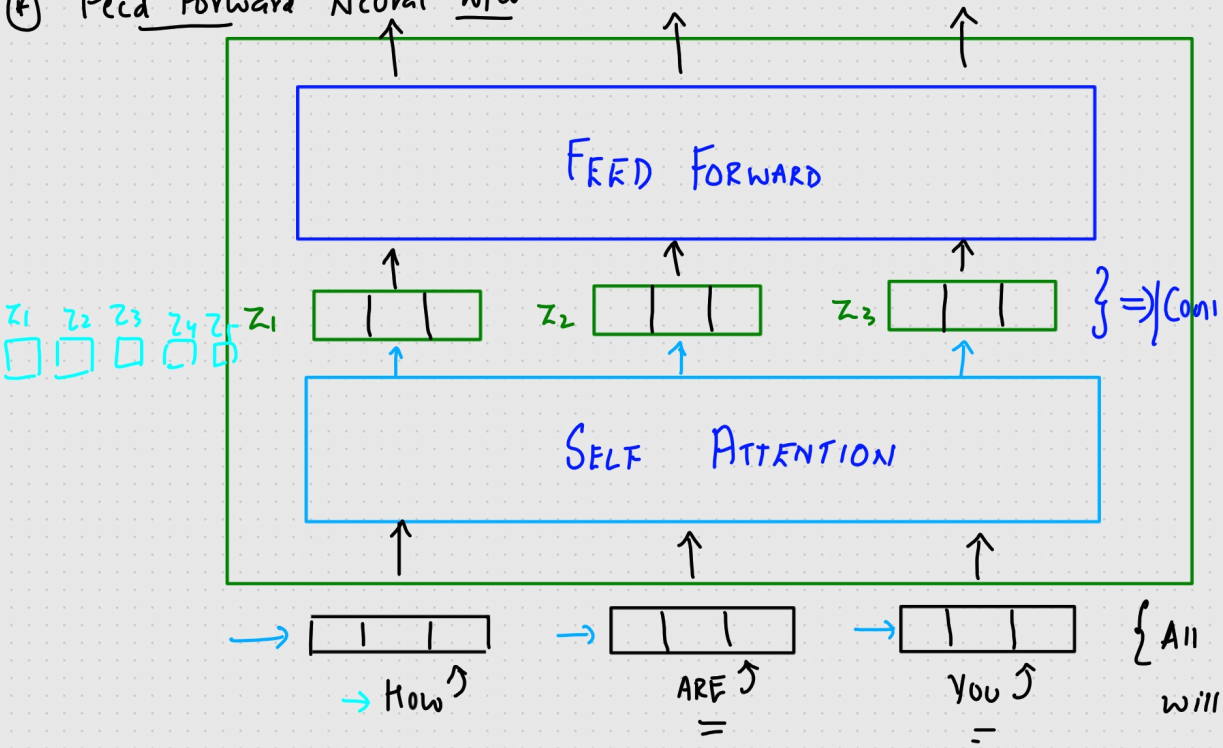
[_ _ _] → Z1

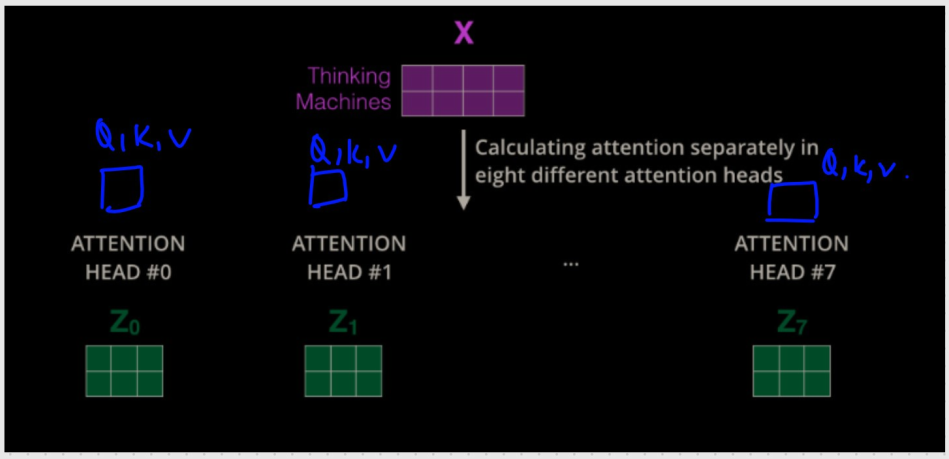
Multi Head Attention

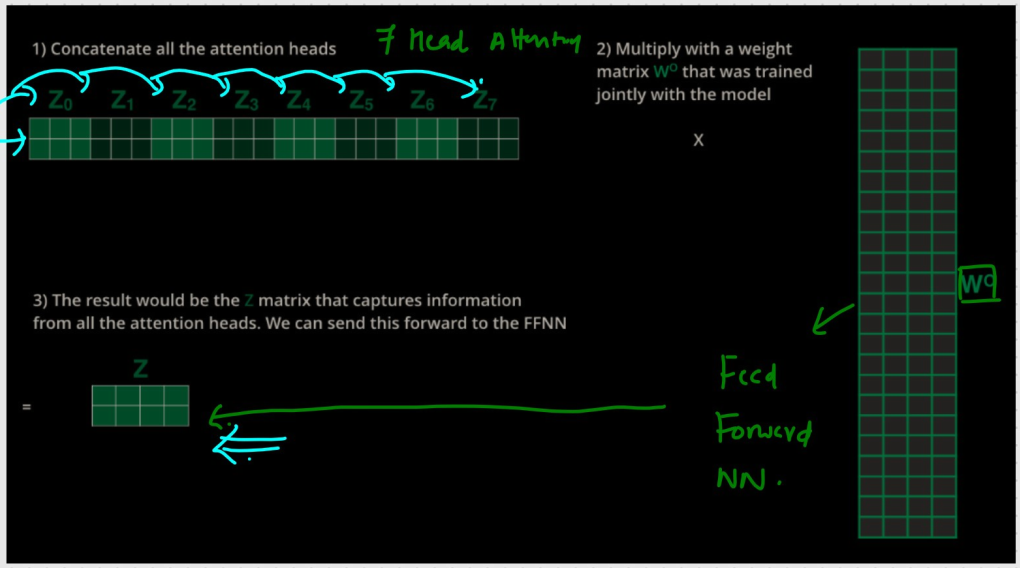


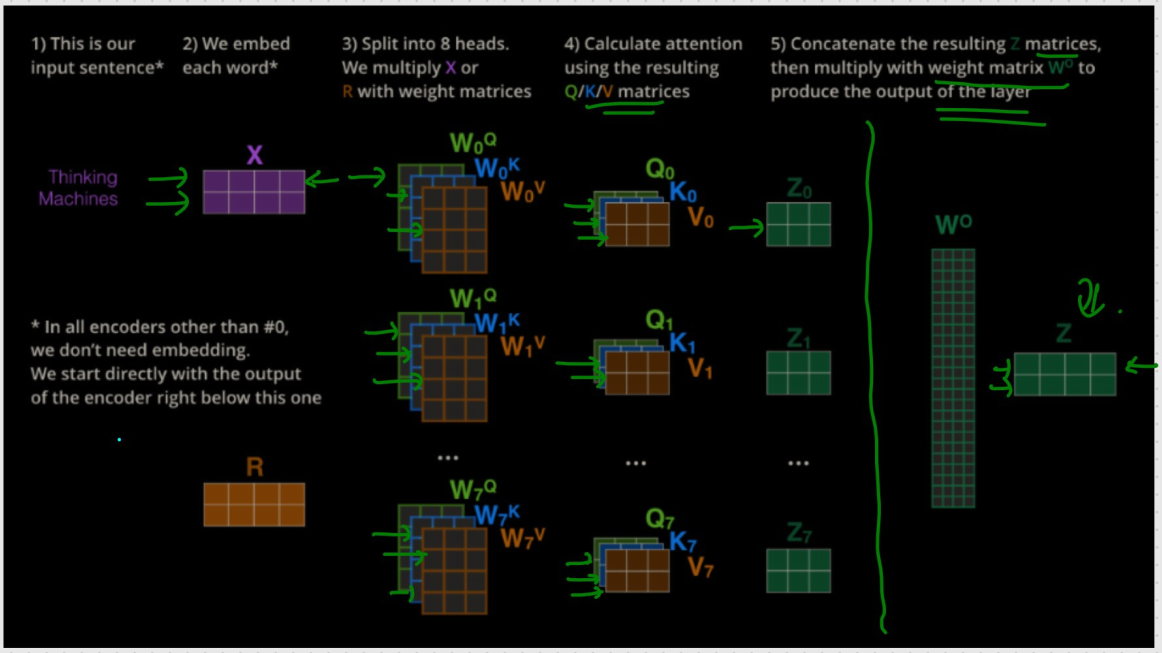
Z Z Z

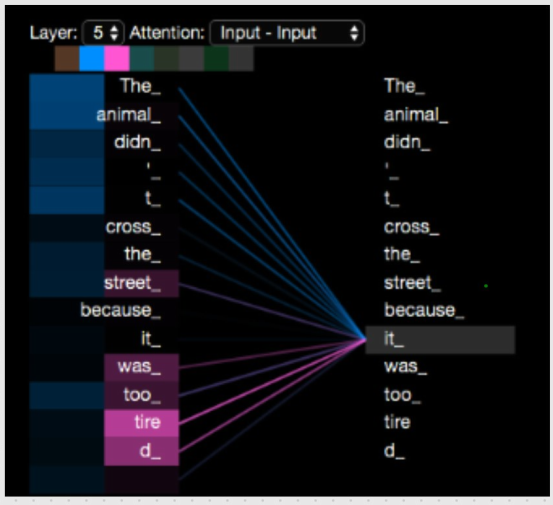
(F) Feed Forward Neural N/W

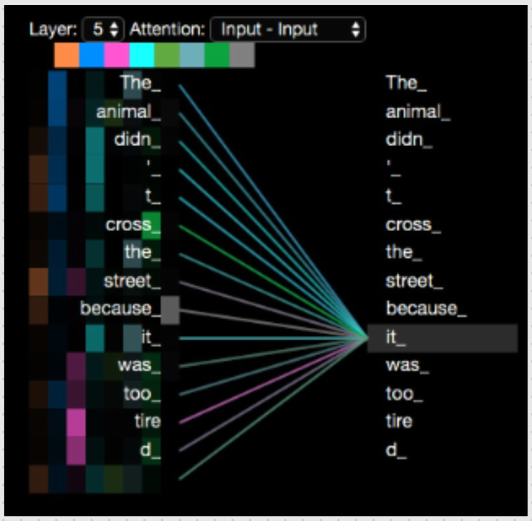










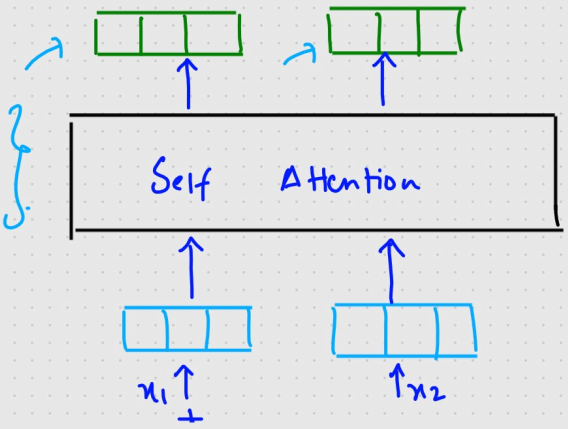


⑥ Positional Encoding – Representing Order of Sequence

① Lion KILLS TIGER
② TIGER KILLS LION

⇓

Missing



Advantage

① Word Tokens it can process parallely

⇓

DRAW BACK

Lack the sequential structure of the words {order}

Attention Is All You Need

↓ Positional Encoded Vector

Book, Journal, Novel
↳ 1 lakh words
↓ Backpropagation

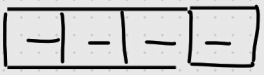
Types of Position Encoding

✓1) Sinusoidal Position Encoding →

2) Learned positional Encoding ⇒ Positional Encoding Are learned during Training ←

① Sinusoidal Positional Encoding : It uses sine and cosine functions of different frequencies to create positional encodings

Formula :



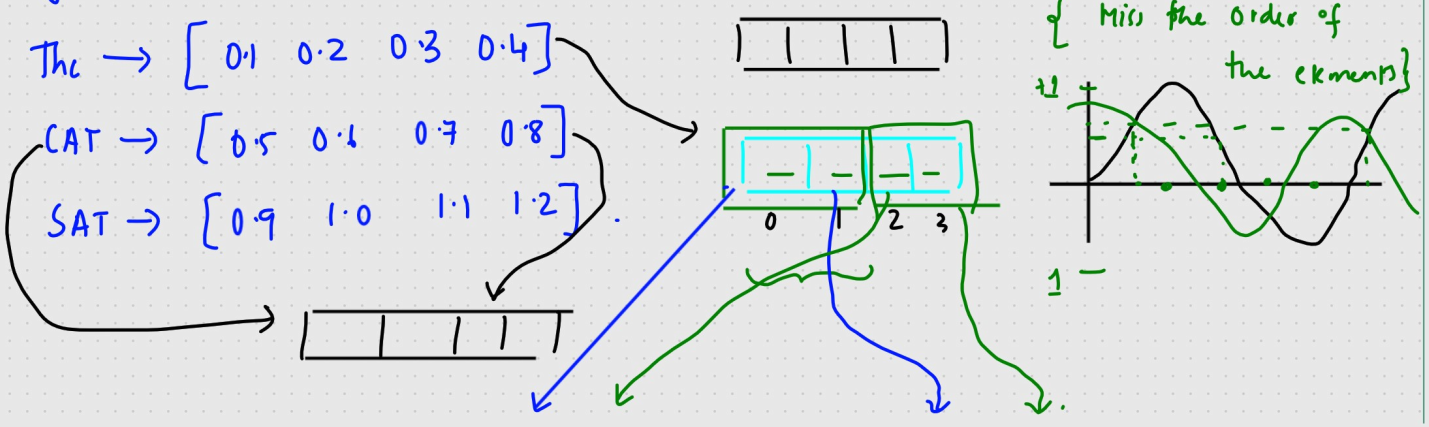
$$P.E(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

Where pos is the position
i is the dimension
d_{model} is the dimensionality of the embeddings.

Eg: The Cat Sat

The → [0.1 0.2 0.3 0.4]
CAT → [0.5 0.6 0.7 0.8]
SAT → [0.9 1.0 1.1 1.2].



{ Miss the order of the elements }

$$P.E_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \qquad P.E_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

For our example $d_{model} = 4$

For position $pos = 0$

$$P.E(0,0) = \sin\left(\frac{0}{10000^{0/4}}\right) = \sin(0) = 0$$

$$PE(0,1) = \cos\left(\frac{0}{-}\right) = \cos(0) = \underline{1}$$

$$P.E(0,2) = \sin\left(\frac{0}{10000^{2/4}}\right) = \sin(0) = 0$$

$$P.E(0,3) = \cos(0) = 1$$

$P.E = [0, 1, 0, 1]$

$$P.E_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \qquad P.E_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

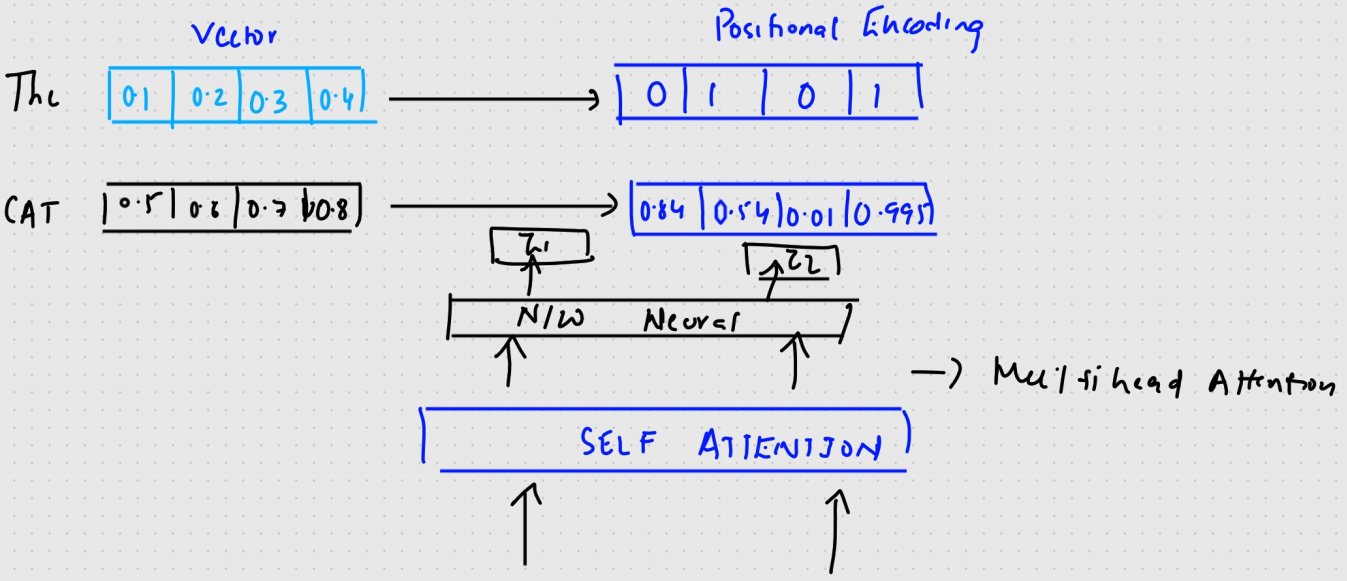
For $pos = 1$

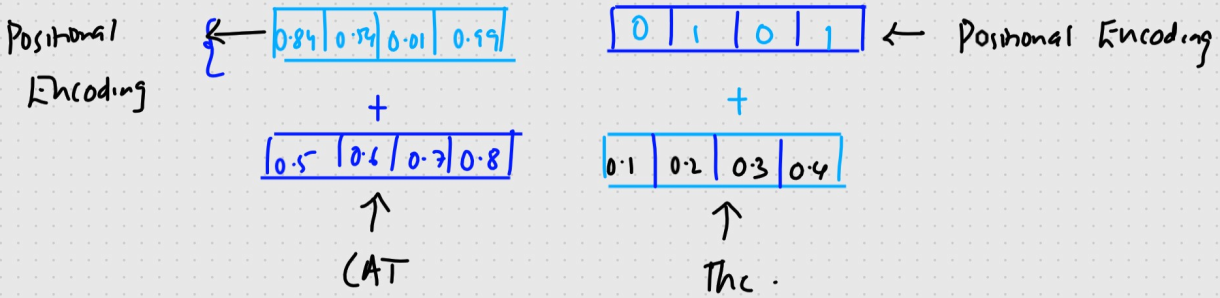
$$P.E(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1) = 0.8415$$

$$PE(1,1) = \cos\left(\frac{1}{10000^{2/4}}\right) \approx 0.5403$$

$$PE(1,2) = \sin\left(\frac{1}{10000^{2/4}}\right) \approx 0.01$$

$$PE(1,3) = \cos \approx 0.99995$$

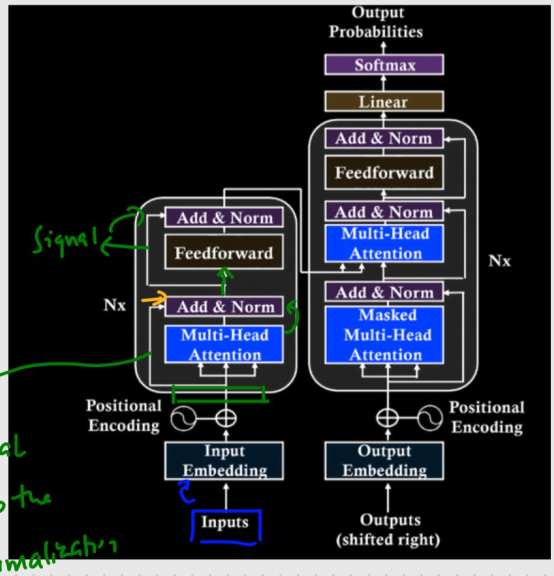




⑨ Layer Normalization In Transformers

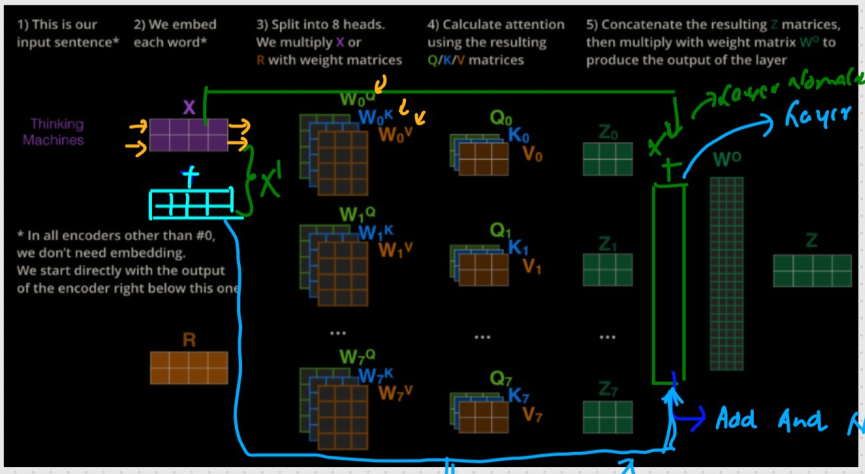
Transformers

Residuals



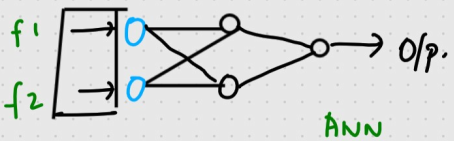
Residuals
↳ Additional signal to the Layer Normalization

① Self Attention layer
② Multi head Attention
③ Positional Encoding
④ Layer Normalization
↑
ADD AND Normalize



Information

Normalization
→ Batch Normalization
→ Layer Normalization



f1 House Size	f2 No. of Rooms	Price
1200	2	45
1500	3	70

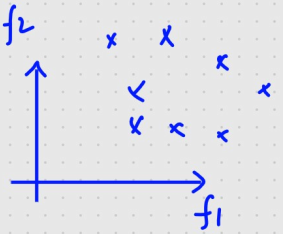
Normalization : Standard Scaling

μ, σ ← | 2000 | 3.5 80

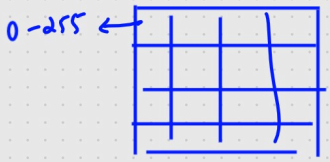
$$z_{score} = \frac{x_i - \mu}{\sigma} \Rightarrow$$

$\boxed{\mu = 0, \sigma = 1}$

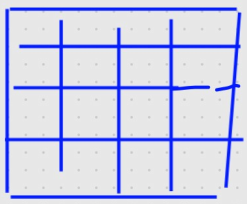
$f_1 \longrightarrow f_1' \Leftarrow \mu = 0, \sigma = 1$

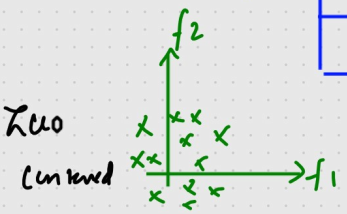


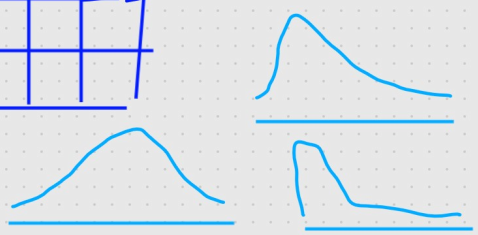
Deep learning : i/p Image ⇒ Min Max Scaler



⇒ Min Max Scaler ⇒





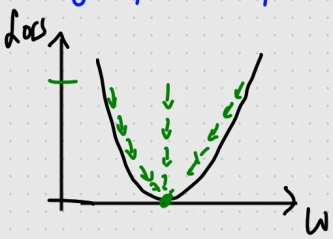


Advantages

① Improved Training Stability ⇓ Vanishing and exploding Gradient problem

$\{\mu = 0, \sigma = 1\}$ $\mu = 0, \sigma = 1$

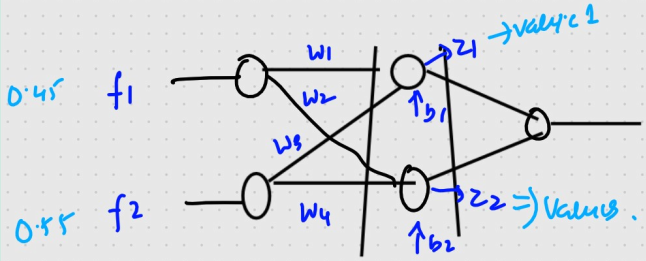
② Faster Convergence



⇒ Back propogation ⇒ Stable updates.

{Batch Normalization}

Normalize Normalized



	f_1 House size	f_2 Rooms	Price	z_1 =	z_2 =
→	0.45	0.55	45	—	—
	0.60	0.20	—	—	—
	—	—	—	—	—

$z_1 = \sigma\left[(0.45 * W_1 + 0.55 * W_3) + b_1\right]$ = Value 1 } Does this distribution

$z_2 = \sigma($ $+ b_2)$ = Value 2 }. $\boxed{\mu = 0, \sigma = 1}$

μ_1, σ_1 μ_2, σ_2

Batch Normalization Vs Layer Normalization

f_1	f_2	z_1	z_2	
—	—	—	—	→ μ_1, σ_1
—	—	—	—	→ μ_2, σ_2
—	—	—	—	→ μ_3, σ_3

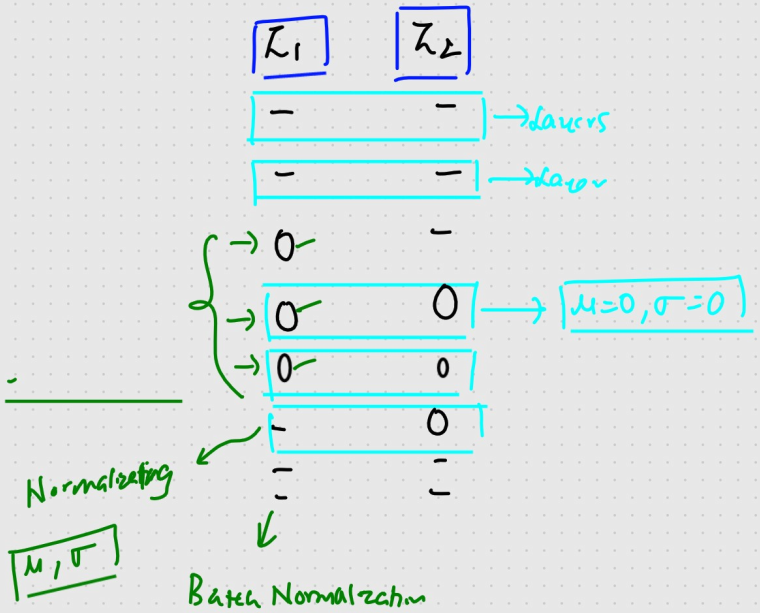
$z_{score} = \frac{x_i - \mu_1}{\sigma_1}$

$z_{score} = \frac{x_i - \mu_2}{\sigma_2}$

$z_{score} = \frac{x_i - \mu_3}{\sigma_3}$ ..

Layer Normalization

$\gamma, \beta \rightarrow$ learnable parameters



Normalization

γ, β

$z_1 = \sigma[w_1^T x + b_1]$

$y = \gamma\left[\frac{z_1 - \mu_1}{\sigma_1}\right] + \beta$

learnable parameters

Scale And Shift parameters

1) "CAT" = [2.0, 4.0, 6.0, 8.0] ← {Vectors} (normalized)

2) Parameters = γ = [1.0, 1.0, 1.0, 1.0] → Learned Scale

β = [0.0, 0.0, 0.0, 0.0] → Shift

⇒ Scale And Shift param

$z_{score} = \frac{x_i - \mu}{\sigma}$

i) Compute the mean

$\mu = \frac{1}{4}(2.0 + 4.0 + 6.0 + 8.0)$

$= \frac{20.0}{4} = 5.0$

ii) Compute the variance (σ^2)

$\sigma^2 = \frac{1}{4}\left[(2.0-5.0)^2 + (4.0-5.0)^2 + (6.0-5.0)^2 + (8.0-5.0)^2\right] = 5.0$

iii) Normalize the i/p

$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$ $\epsilon = 1e^{-5}$ ⇒ Avoid division by 0

$\sqrt{\sigma^2 + \epsilon} = \sqrt{5.0 + 1e-5} \approx \sqrt{5.00001} = 2.236$

$$\hat{x}_1 = \frac{2.0 - 5.0}{2.236} \approx -1.34$$

$$\hat{x}_2 = \frac{4.0 - 5.0}{2.236} \approx -0.45$$

$$\hat{x}_3 = \frac{6.0 - 5.0}{2.236} \approx 0.45$$

$$\hat{x}_4 = \frac{8.0 - 5.0}{2.236} \approx 1.34.$$

Normalized vector

$$\hat{x} = [-1.34, -0.45, 0.45, 1.34]$$

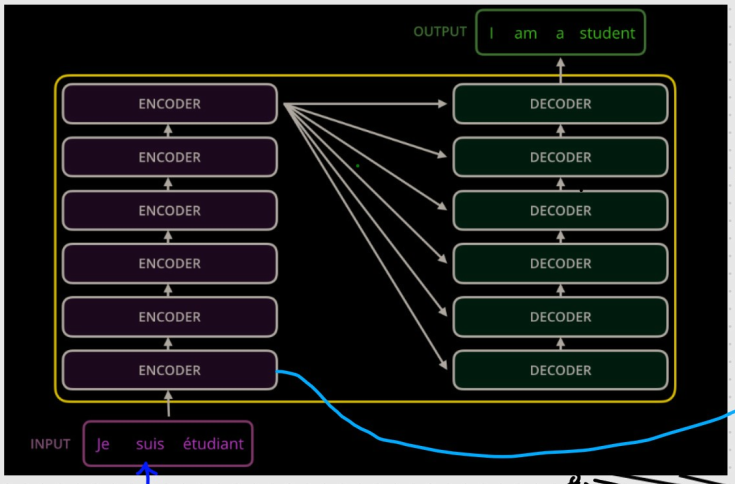
4) Scale And Shift

$$\gamma = [1.0, 1.0, 1.0, 1.0] \quad \beta = [0.0, 0.0, 0.0, 0.0]$$

$$y_i = \gamma_i \hat{x}_i + \beta_i$$

$$y = [-1.34, -0.45, 0.45, 1.34].$$

⑥ Encoder Architecture [Research Paper]



Sequence to Sequence

(complex)

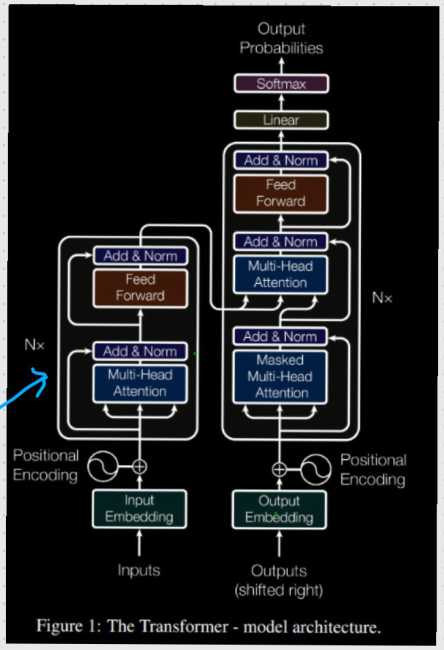


Figure 1: The Transformer - model architecture.

Feed Forward NN. → 0 0 0 0 0 0 0 0 0 0 0 0

(Add And Norm) → • • • • •

Multi Head Attention → $Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8$ {Research paper}.

Residuals

Text Embeddings + Positional Encoding ⇒ 512. → {Research paper}.

I/P Sequence

every word = 512.

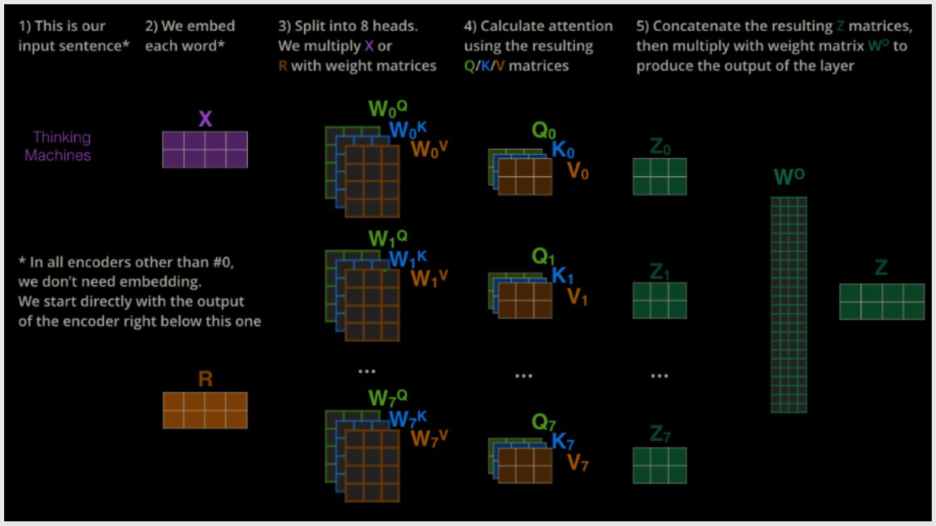
$Q_1 = 64$

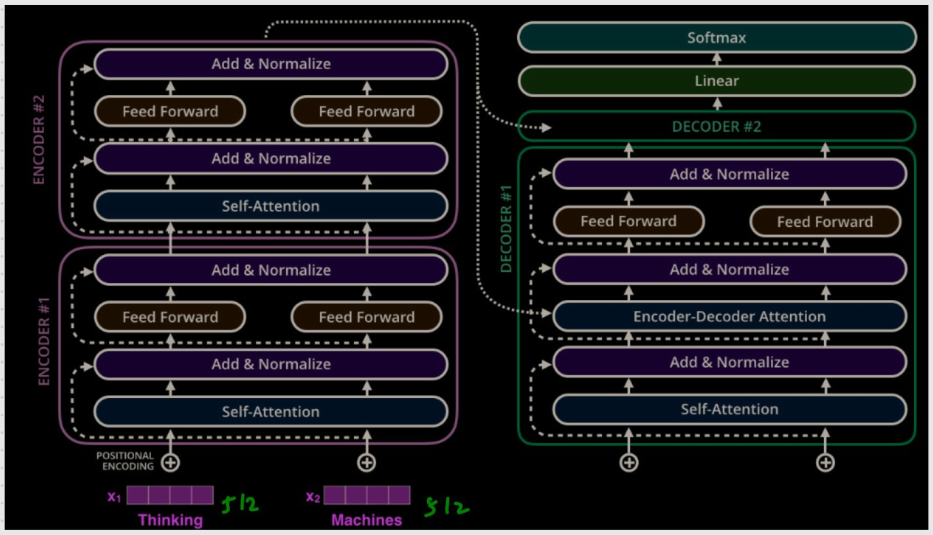
$K = 64$

$V = 64$

$\sqrt{64} = 8$ //.

Self Attention to Feed Forward Neural N/W





Embedding Vector: 512

Q, K, V = 64

Head Attention: 8

④ Residual connection : Skip connection NN.

1) Addressing the Vanishing Gradient Problem

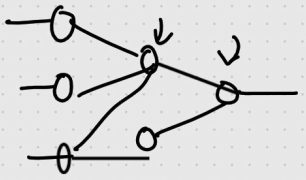
Residuals : Residual connection create a short paths for gradients to flow directly through the n/w. Gradient remains sufficiently large.

2) Improve Gradient flow

Convergence will be faster.

3) Enables Training of Deeper Networks.

④ Feed Forward NN



linear function problem

{Non linear functions}.

① Adding Non linearity

② Processing Each position Independently.

Sof Attention → (apture relationships)

FFN → Each token representation Independently.

⇓

Transforming these representation furthers

and allows the model to learn

Richer Representation.

} ANN ⇒

③ FFN → Deeper ⇒ Adds Depth to the Model.

Depth ↑↑ ⇒ More learnings → DATA

⑤ Decoders In Transformers

3 main Components

The transformer decoder is responsible for generating the output sequence one token at a time, using the encoder's output and the previously generated tokens.

① Masked Multi Head Self Attention ✓.

② Multi Head Attention (Encoder Decoder Attention)

③ Feed Forward Neural Network.

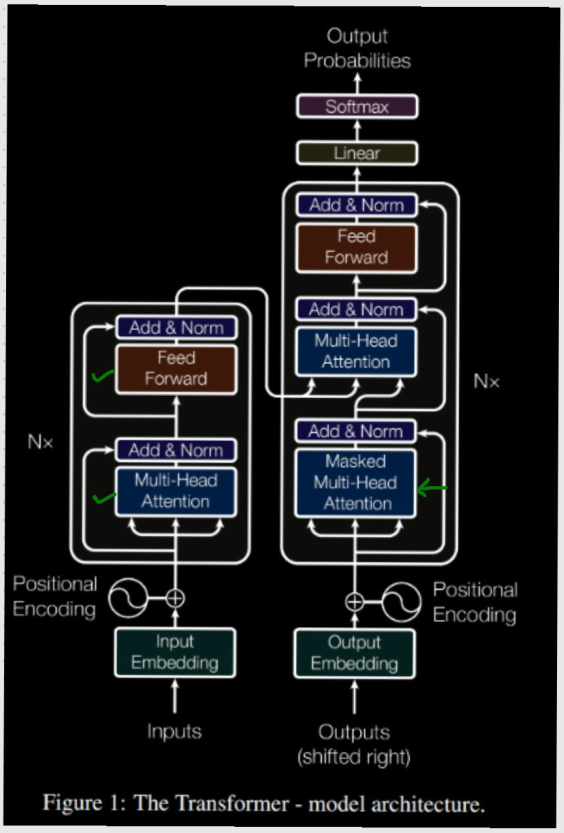
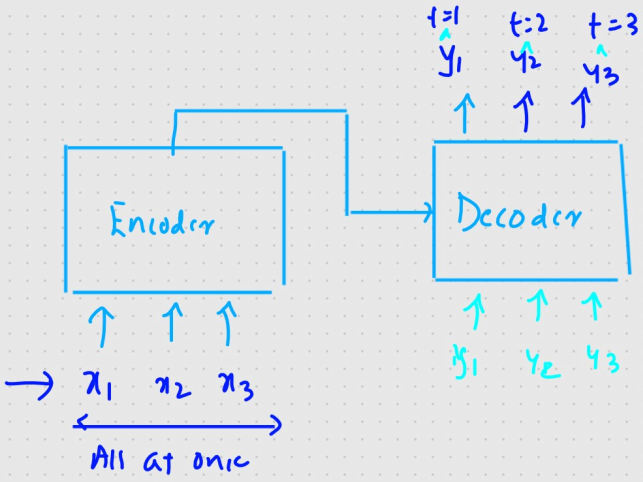


Figure 1: The Transformer - model architecture.



$\langle y_1, y_2, y_3 \rangle$

① Training Mechanism

② Inference Mechanism

⊛ Masked Multi Head Self Attention

① I/P Embedding And Positional Embedding ✓ → Zero padding → Sequence length Equal

② Linear Projection for Q, K, V

③ Scaled Dot Product Attention

✓④ Mask Application } ⇒ Try to understand the imp.

⑤ Multi Head Attention

⑥ Concatenation And Final Linear Projection

⑦ Residual connection And Layer Normalization

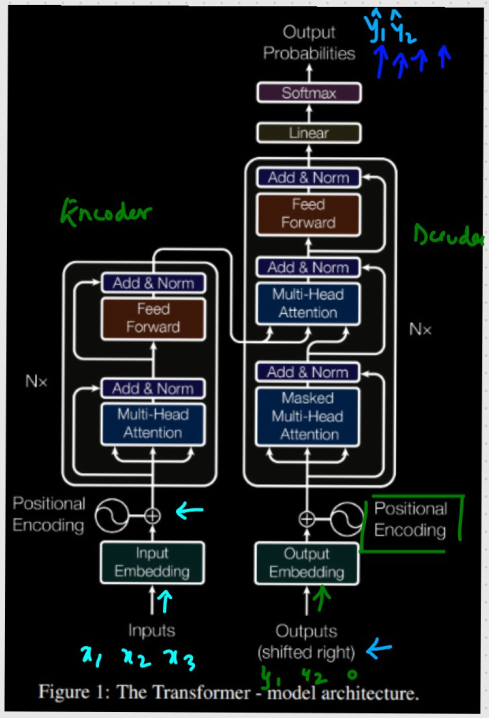


Figure 1: The Transformer - model architecture.

Dataset

Eng: $\langle x_1, x_2, x_3 \rangle$

Hindi: $\langle y_1, y_2 \rangle$ ← ⇓ $\langle y_1, y_2, 0 \rangle$ ⇓ Zero padding

① Linear Projections Q, K, V

② Scaled Dot Product Attention

③ Mask Application → Look Ahead Mask
→ Padding Mask

Masked Multi Head Attention.

I/P [4 5 6 7] O/P [1 2 3]

[1 2 3 0]

↓

4 dimension vector

i) Input Embedding and Positional Encoding

Output Embedding [STEP 1]

$$\begin{bmatrix} [0.1, 0.2, 0.3, 0.4], \\ [0.5, 0.6, 0.7, 0.8], \\ [0.9, 1.0, 1.1, 1.2], \\ [0.0 \quad 0.0 \quad 0.0 \quad 0.0] \end{bmatrix} + PE \Rightarrow 0$$

Step 2 : Linear Projection for Q, K, and V.

$W_Q = W_K = W_V = I$

Create query (Q), Key(K) and value(V) vectors

Q = Output Embedding * W_Q = Output Embedding

K = " * W_K = "

V = " * W_V = "

$$Q = K = V = \begin{bmatrix} [0.1, 0.2, 0.3, 0.4], \\ [0.5, 0.6, 0.7, 0.8], \\ [0.9, 1.0, 1.1, 1.2], \\ [0.0 \quad 0.0 \quad 0.0 \quad 0.0] \end{bmatrix}$$

$$K^T \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.6 \\ 0.7 \\ 0.8 \end{bmatrix} \cdot \begin{bmatrix} 0.9 \\ 1.0 \\ 1.1 \\ 1.2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

③ Scaled Dot Product Attention Calculation

$$\text{Scores} = Q * K^T / \sqrt{dk}$$

$$= Q * K^T / 2.$$

$$\begin{bmatrix} [0.1*0.1 + 0.2*0.2 + 0.3*0.3 + 0.4*0.4, \\ 0.1*0.5 + 0.2*0.6 + 0.3*0.7 + 0.4*0.8, \\ 0.1*0.9 + 0.2*1.0 + 0.3*1.1 + 0.4*1.2 \\ 0.1*0 + 0.2*0 + 0.3*0 + 0.4*0] \end{bmatrix}$$

Scores : [[0.3, 0.7, 1.1, 0.0]
[0.7, 1.9, 3.1, 0.0]
[1.1 3.1 5.1 0.0]
[0.0 0.0 0.0 0.0]]

[ ]
[ ]
[ ]]

(4) Masked Application

It helps manage the structure of the sequences being procesed And ensures the models behaves correctly during training And Inferencing

Reasons

① Handling Variable length Sequences with Padding MASK

Purpose

① To handle Sequences of diffuent length in batch

② To ensure that padding tokens, which are added to make sequences of uniform length, do not affect the model prediction

Eg: i/p ← Sequence 1 [1, 2, 3]

o/p ← Sequence 2 [4, 5, 0] 0 is the padding token

→ Influence the Attention Mechanism

⇓

lead to Incorrect or biased predictions.

o/p

→ 4, 42 0 0 0 0 0

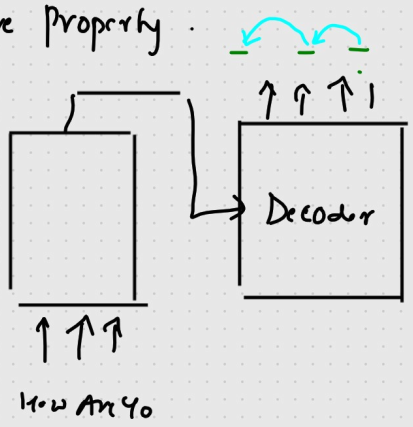
→ 100.

A padding mask ⇒ The tokens Are ignored.

Masking → Padding Mask
→ Look Ahead Mask.

Padding Mask [1 1 1]
[1 1 0]

② Look Ahead Mask → Maintain Auto Regressive property.



① To ensure that each position in the decoder output sequence can only attend to the previous position, but no future position

② Sequence → Language Modelling, Translation

Eg: [4, 5, 0] → [1, 1, 0] → 1D Mask.

Convert 1D to 2D Mask

For each token in the sequence, the mask should indicate which tokens it can attend to.

Token 1 attends to token 1, 2

$$\rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Attention Mechanism ⇐ 1, 2

③ Look Ahead Mask → Decoder Output

$$\begin{bmatrix} [1 & 0 & 0] \\ [1 & 1 & 0] \\ [1 & 1 & 1] \end{bmatrix}$$

④ Combine Padding And Looking Ahead Mask

Element Wise multiplication of 2 mask

$$\text{Combine Mask} = \begin{bmatrix} [1, 0, 0] \\ [1, 1, 0] \\ [0, 0, 0] \end{bmatrix}$$

④ Mask

Scores : [[0.3, 0.7, 1.1, 0.0]
[0.7, 1.9, 3.1, 0.0]
[1.1 3.1 5.1 0.0]
[0.0 0.0 0.0 0.0]]

Look Ahead Mask

[[1 0 0 0]
[1 1 0 0]
[1 1 1 0]
[1 1 1 1]]

Padding Masking [extended to 2D Format]

[[1, 1, 1, 0]
[1, 1, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]]

Combined Mask : Look Ahead Mask + Padding Mask

[[1*1, 0*1, 0*1, 0*0]
[1*1, 1*1, 0*1, 0*0]
[1*1, 1*1, 1*1, 0*0]
[1*1, 1*1, 1*1, 1*0]]

=

[[1, 0, 0, 0],
[1, 1, 0, 0],
[1, 1, 1, 0],
[1, 1, 1, 0]]

⇒

[[1, -∞, -∞, -∞],
[1, 1, -∞, -∞],
[1, 1, 1, -∞]
[1, 1, 1, -∞]]

Masked Score

[[0.3, -∞, -∞, -∞] ← 0
[0.7, 1.9, -∞, -∞]
[1.1, 3.1, 5.1, -∞]
[0.0, 0.0, 0.0, -∞]].

Zero out the influence when the Softmax is applied.

Attention weights.

(*) Softmax

Softmax Score = Softmax (Masked Scores)

$$= [[1.0, 0.0, 0.0, 0.0],$$
$$[0.3, 0.7, 0.0, 0.0],$$
$$[0.1, 0.3, 0.6, 0.0],$$
$$[1.0, 0.0, 0.0, 0.0]]$$

(*) Weigth Sum of Value

Attention O/p = Softmax Scores * V.

Masking

Masking in the transformer architecture is essential for several reasons. It helps manage the structure of the sequences being processed and ensures the model behaves correctly during training and inference. Here are the key reasons for using masking:

1. Handling Variable-Length Sequences with Padding Mask

Purpose
To handle sequences of different lengths in a batch.
To ensure that padding tokens, which are added to make sequences of uniform length, do not affect the model's predictions.

2. Maintaining Autoregressive Property with Look-Ahead Mask

Purpose
To ensure that each position in the decoder output sequence can only attend to previous positions and itself, but not future positions.
This is crucial for sequence generation tasks like language modeling and translation, where the model should not have access to future tokens when predicting the current token.

④ Encoder Decoder Multi Head Attention

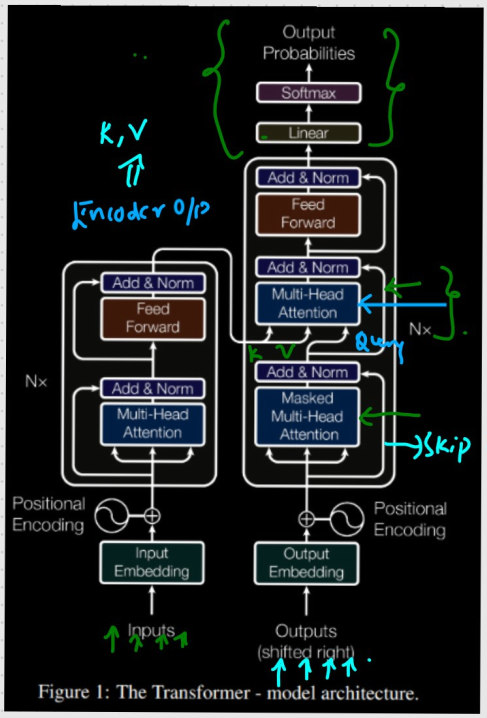


Figure 1: The Transformer - model architecture.

① Encoder O/p → Set of Attention vector K & V

② Masked Multihead → Attention vector Q {Query Vector}

These are to be used by each decoder in its "encoder-decoder" attention layer

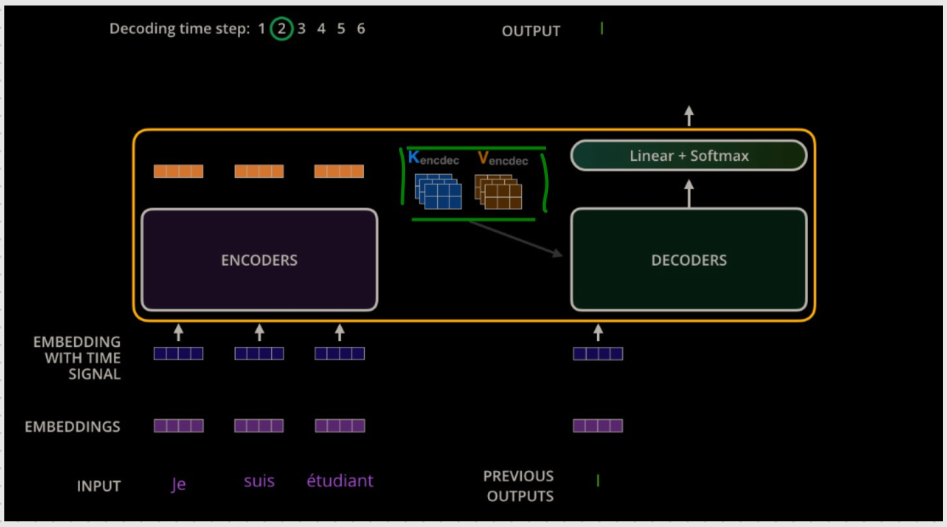
⇓

Helps the Decoder to focus on appropriate places in the i/p sequence

Dataset

i/p $\langle x_1, x_2, x_3 \rangle$

O/p $\langle y_1, y_2, y_3 \rangle$.



④ The Final Linear And Softmax Layer { Vectors ⟶ o/p word }

{ BLOG } ⇒ Transformers

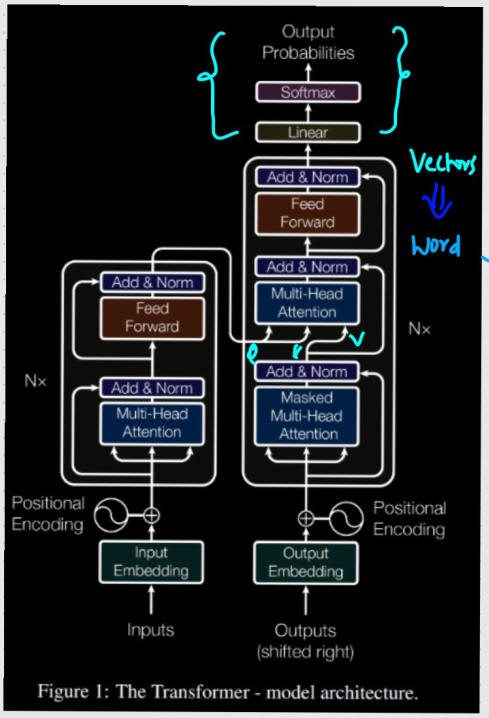
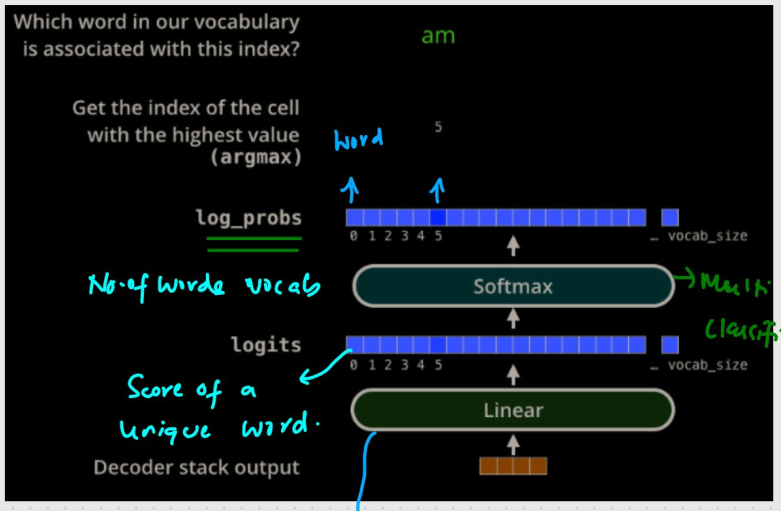


Figure 1: The Transformer - model architecture.



Linear ⇒ The linear layer is a simple fully connected neural n/w that projects the vector produced by the stack of Decoder ⇒ logits vector

⇒

Model ⇒ 10,000 ⇒ vocabulary ⇒ logits vector = 10000 cells wide

② Softmax layer turns those scores into probabilities (all add upto 1.0).

The cell with the highest probability is chosen, and the word associated with it is produced as the o/p. ⇒ time stamp.

Recap of Training

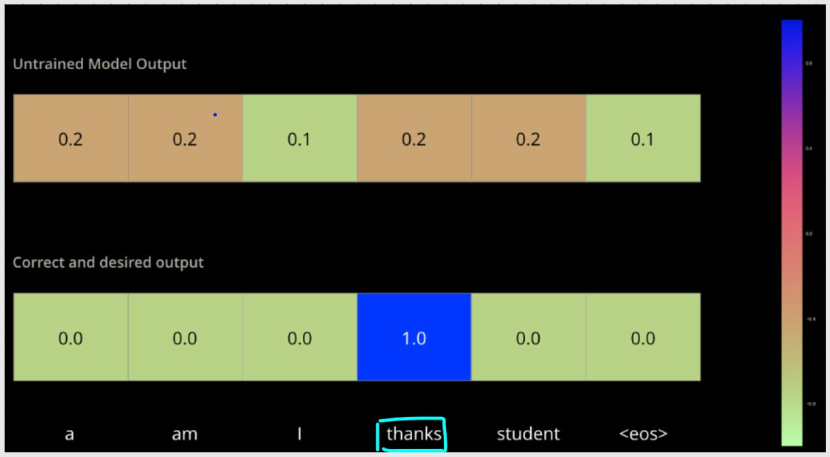
Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

OHE ←

Merci → Thanks

I am a student <eos>.



⇒ Loss function ↓↓.

Back Propogation

i/p

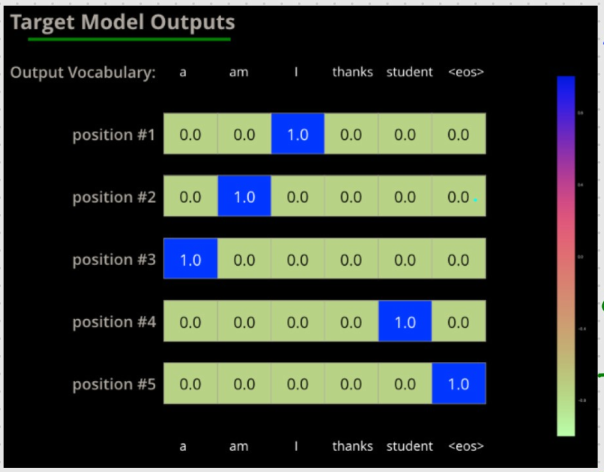
a, am, i, student

o/p

I am a student

↓ ↓ ↓ ↓

OHE OHE OHE OHE



Loss