Linear regression Formula

Model equation

$$y = \beta_0 + \beta_1 x + \varepsilon$$

y is predicted value/output

β_0 is the y intercept (constant term

β_1 (coeffcient for the input feature (slope)

ε is the error term (difference between Predicted value & actual value)

- Cost function (Mean squared Error) MSE

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

MSE is the Mean squared error to minimize

n is number of data Points

y_i is actual output, $\hat{y}_i$ is a Predicte output

Logistic regression

- Hypothesis function: (sigmoid)

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

$h_\theta(x)$ is the probability that the output is 1 given x.

θ is parameter vector (weights)

x is feature vector (input data)

T is the linear combination of input features.

- Cost function (Binary-Cross Entropy):

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

→ Flip to next Page

Explanation of Binary Cross-Entropy

$J(\theta)$ is Cost function to minimize

m is number of traning examples

$y^{(i)}$ is Actual Label (0 or 1)

$h_\theta(x^{(i)})$: Predicted Probability of the ith data-Point.

Gradient Descent

- update rule:

$$\theta = \theta - a\frac{\partial J(\theta)}{\partial \theta}$$

θ is the model Parameters (weights)

a is the learning rate (step size)

$\frac{\partial J(\theta)}{\partial \theta}$ is the Gradient of the Cost function $J(\theta)$.

- L2 Regularization (Ridge):

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 + \lambda\sum_{j=1}^{p}\theta_j^2$$

λ is the Regularization strength it controls overfitting

$\sum_{j=1}^{p}\theta_j^2$ is the sum of squared model Parameters

* The rest of the equation is MSE (Mean Squared Error)

- L1 Regularization (LASSO):

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 + \lambda\sum_{j=1}^{p}|\theta_j|$$

$|\theta_j|$ is the absolute value of the parameters (includes sparsity).

- Distance Formula (Euclidean distance)

$$d(x, x') = \sqrt{\sum_{i=1}^{n} (x_i - x_i')^2}$$

$d(x, x')$ is the distance between Points x and x'.

x_i, x_i is the ith feature of Points x and x'.

Support vector machines (SVM)

- Desicion boundary:

$$f(x) = w^T x + b = 0$$

$f(x)$ is the desicion function

w is the weight vector (defines the hyperplane)

x is the input feature vector

Support vector machine

- Hinge loss:

$$L = \sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b))$$

L is hinge loss

y_i is the true label (+1 or -1)

x_i is the input feature vector for the ith data point.

w^T is the weight vector.

Principal Component Analysis: PCA

Variance Explained

$$Var(z) = \frac{1}{n-1} \sum_{i=1}^{n} (z_i - \bar{z})^2$$

Var(z) is the variance of the transformed Principal components

n is the number of data points

z_i is the value of ith Principal component

→ Continued for Principal Component Analaysis

$\bar{z}$ is the Mean of the principal components

Cross-Validation

K-Fold Cross-Validation:

$$CV_{error} = \frac{1}{k} \sum_{i=1}^{k} MSE_i$$

CV_{error} is the Average validation error across K folds

K is the number of folds

MSE_i is the MSE (Mean Squared Error) on the ith fold (MSE is explained on pg. 1)

Evaluation Metrics

Accuracy:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

TP is True Positives
TN is True negatives
FP is False Positives
FN is False negatives

Presicion:

$$\text{Presicion} = \frac{TP}{TP+FP}$$

- Presicion measures the Proportion of true Positives to sum of True Positives & False Positives

- Recall:

$$\text{Recall} = \frac{TP}{TP+FN}$$

Recall measures the Proportion of true Positive predictions out of all actual Positives.

F1 score:

$$F1 = 2 \times \frac{Presicion \times Recall}{PResicion + Recall}$$

* x means multiplication

F1 is the Harmonic meen of Presicion & Recall, used when there is an uneven class distribution

Confusion Matrix

True Positive: Correctly classified Positive

True Negative: Correctly classified negative

False Positive: Incorrectly classified as Positive

False negative: Incorrectly classified as negative

Decision Trees

Gini Impurity: (for classification)

$$G = 1 - \sum_{k=1}^{K} p_k^2$$

p_k is the proportion of classes in class K

K is number of classes

Entropy: for classification

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

p_k is the proportion of data points in class K.

Information gain:

$$IG = H_{parent} - \sum_i \frac{|D_i|}{|D|} H(D_i)$$

IG is the information gain from splitting the dataset

H_{parent} is the Entropy of the parent node.

$H(D_i)$: Entropy of the child node (i)

$|D_i|$ is the number of samples in child node (i)

$|D|$ is the total number of samples in the parent node

Random Forests

Final Prediction: (for classification)

$\hat{y}$ = majority vote of individual trees

Final Prediction: (for regression)

$$\hat{y} = \frac{1}{T} \sum_{t=1}^{T} \hat{y}_t$$

T is the number of trees in the forest

$\hat{y}_t$ is the Prediction from the t-th tree

Bayes theoroem:

$$P(y|x) = \frac{P(x|y)\,P(y)}{P(x)}$$

$P(y|x)$ is the Posterior probability of class Y → given feature x

$P(x|y)$ is the Posterior Probability of observing feature x given class y.

$P(y)$ is the Prior Probability of class y

$P(x)$ is the Probability of Observing feature x

Gaussian Naive Bayes (for Continous features):

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

X_i is the feature value

μ_y is the mean of the feature x_i for class y.

σ_y^2 is the Variance of the feature x_i for class

Computer Vision Formulas

Convultion operation (in CNN's)

Formula (2D convolution):

$$(I * K)(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot K(i,j)$$

$I(x,y)$ is the input image pixel at position (x,y)

$K(i,j)$ is the Kernel (filter) value at position (i,j)

$*$ is the convultion operator

(x,y) is the position in the output feature map

The kernel (Filter) slides over the image, applying the dot product between the image Patch and the Kernel to produce the output feature map

Pooling operation (Max Pooling)

Pooling is used to downsampre the image while still retaining important information.

Max Pooling formula:

$$P(x,Y) = \max\left(I(x+i, Y+j)\right), \text{ for } i,j \in [0, f-1]$$

$P(x,Y)$ is the Pooled output at position (x,Y)

$I(x+i, y+j)$ is the image pixel values in the $f \times F$ neighborhood

f is the size of the Pooling window (usually 2×2 or 3×3)

The maximum value within each Pooling window is taken to reduce the image size.

Average Pooling

$$y_{i,j} = \frac{1}{K} \sum_{m,n} x(i+m), (i+n)$$

check other page for variables.

ReLU (Rectified Linear Unit) Activation function

ReLU function:

$$f(x) = \max(0, x)$$

x is the input value

f(x) is the output value after applying ReLU

ReLU replaces negative values with 0, retaining only positive values which helps in learning complex patterns.

Softmax activation function (for image classification)

Softmax formula:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

z_i is the logit or score for class i

σ is the Probability of class i (zi)

n is the total number of Classes.

The softmax function normalizes the output so that the Probabilities for all classes sum up to 1.

Cross Entropy Loss (for classification tasks)

(cross-Entropy loss formula

$$L = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

L is the loss value

y_i is the Actual Label (0 or 1 for binary classification

n is the total number of classes

- the loss increases as the predicted probability for the true class decreases.

Bounding Box regression (for object detection):

Bounding box coordinates:

$(x_{min}, Y_{min}, x_{max}, Y_{max})$

x_{min}, Y_{min} : Coordinates of the top left corner for the bounding box

x_{max}, Y_{max} : Coordinates for the bottom right corner of the bounding box.

IoU (intersection over Union):

$$IoU = \frac{\text{Area of intersection}}{\text{Area of Union}}$$

IoU is the overlap between the Predicted and ground truth bounding boxes.
This metric is used to evaluate the accuracy of object localization.

Bounding box Loss L_2 loss:

$$L_{bbox} = (x_{pred} - x_{True})^2 + (y_{pred} - y_{True}$$

x_{pred}, y_{pred} is the Predicted center coordinates of the bounding box

x_{true}, Y_{true} is the Ground truth center of the bounding box.

the loss Penalizes the difference between Predicted an

Affine Transformation

for image augmentation

these are transformations applied to images.

Affine transformation Matrix;

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

(x, y) are the original coordinates

(x', y') are the transformed coordinates

a, b, c, d are the Transformation Matrix Elements (e.g Rotation, scaling)

e, f are the translation values

Representation (for multi-scale Prossesing)

Image Pyramid Scaling Formula:

$$I'(x,y) = I(sx, sy)$$

$I(x,y)$ Original Image

$I'(x,y)$ Rescaled Image

S is the scale factor (e.g 0.5 for downscaling)

This allows detection of objects in various sizes in images.

Optical Flow (for motion detection)

Optical flow tracks motion in videos by estimating movement of pixels between frames.

Optical Flow Equation:

$$I_x u + I_y v + I_t = 0$$

I_x, I_y are the image intensity gradients in the x,y directions

I_t is Temporal Intensity Gradient (change in Pixel intensity over time)

u,v are the optical flow velocities (displacement in x and y)

Histogram of oriented gradients (HOG)

HOG is used for feature extraction by capturing the distribution of gradient directions

Gradient Magnitude and direction:

$$M(x,y) = \sqrt{I_x^2 + I_y^2} \quad \Theta(x,y) = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

$M(x,y)$ is the Gradient magnitude at position x, y

$\Theta(x,y)$ is the gradient direction.

I_x, I_y are the image gradients in the x and y directions

HOG captures these gradients and groups them into orientations bins to describe

SSD (Single Shot multibox detector)

SSD multibox Loss function:

$$L = \frac{1}{N}(L_{conf} + \alpha L_{loc})$$

L_{conf} is how well the Predicted Class probabilities match the ground truth.

L_{loc} is the difference between the Predicted and ground truth bounding box coordinates.

α is the balancing factor between confidence loss and localization loss.

N is the number of default bounding boxes

$\hat{c}$ is the confidence score for the object

Predicted box = $(\hat{c}x, \hat{c}y, w, h, \hat{c})$

$\hat{c}x, \hat{c}y$ center coordinates, w, h width & height

Similar are: YOLO and R-CNN

Language Processing

Term frequency (TF)

The term frequency measures how frequently a term appears in a document

$$TF(t,d) = \frac{f_{t,d}}{n_d}$$

t is the term being evaluated

d is the specific document

$f_{t,d}$ is the count of term t in document d

n_d is the Total number of terms in document d

Inverse Document Frequency (IDF)

Measures the importance of a term across multiple document

$$IDF(t, D) = \log\left(\frac{N}{f_{t,D}} + 1\right)$$

t is the term

D is the set of documents

N is the total number of documents

$f_{t,D}$ is the number of documents containing the term t

TF-IDF (Term frequency - Inverse Document frequency)

Combines TF & IDF to evaluate the importance of a term in a document relative to a corpus

$$TF\text{-}IDF(t, d, D) = TF(t, d) \times IDF(t, D)$$

Measures the similarity between the two vectors (e.g document representations)

$$\text{Cosine similarity}(A,B) = \frac{A \cdot B}{\|A\| \|B\|}$$

$A \cdot B$ is the dot product of vectors A and B

$\|A\|$ is the length of vector A

$\|B\|$ is the length of vector B

Dot Product

$$A \cdot B = \sum_{i=1}^{n} A_i B_i$$

This calculates the sum of

Magnitude of a vector

$$||A|| = \sqrt{\sum_{i=1}^{n} A_i^2}$$

this computes the euclidean length of the vector

Range of values

the value of cosine similarity ranges from -1 and 1

1 shows that the two vectors are similar

0 shows that both vectors are orthogonal (no similarity)

-1 shows that both vectors are opposed

word embeddings (e.g word2vec)

This is used to convert words into vectors so numbers, so computers can process them

Skip-Gram model formula

$$J = -\sum_{t=1}^{T} \sum_{j=-c}^{c} \log P(w_{t+j} | w_t)$$

J is the object function to minimize

T is the total number of words in the corpus

w_{t+j} is the context of words within a window size c.

$P(w_{t+j} | w_t)$ is the Probability of context word given the target word, typically modeled with softmax activation function.

Skip gram's are used to generate word embeddings in NLP.

Sequence-to-sequence models (seq2seq)

this converts one sequence of data to another.

Loos function (cross entropy)

$$L = -\sum_{t=1}^{T} \sum_{k=1}^{V} y_{t,k} \log(p_{t,k})$$

L is the loss function to minize

T is the length of the sequence

V is the size of the vocabulary

$y_{t,k}$ is the ground truth output (1 if the t-th output is k, else 0)

$p_{t,k}$ is the predicted probability of word k being the t-th output.

Hidden state update:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$$

h_t is the hidden state at time t

h_{t-1} is the hidden state at the previous time step.

W_h is the weight matrix for the hidden state

W_x is the weight matrix for the input x_t.

b is the bias vector

σ is the activation function (e.g tanh, ReLU)

RNN's are used to process word input as it can keep track of pas words and update current state.

BERT (Bidirectional Encoder representation from transformers)

Self-attention mechanism:

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

Q is the query matrix

K is the key matrix

V is the value matrix

d_k is the dimensonality of the key vectors for scaling.

The softmax function ensures that the attention weights sum to one.

It is helpful for operations with text input and is a more effective model of RNN's.

Cross entropy Loss

Formula:

* refer to other page for variable meaning

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$

Reinforcement Learning Algorithms

Markov Desicion Process (MDP

States (S) Possible situiations the agent can be in

Actions (A) choices avaible to the agent

Transition function (T) Probability of moving from one state to another given an actio

Reward function (R) Immidiate reward recieved after transitioing from one state to another

Discount Factor (γ) A value betwee 0 & 1 that determines the importance of future rewards.

MDP's Provide a formal framework for modeling Desicion making Prohelms where outcomes are

Value function

Formula

$$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_t = s\right]$$

use. This estimates the expected return (total future rewards) starting from state s. helps us evaluate long-term value of states in a Policy.

Action Value function (Q-Function)

Formula:

$$Q(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_t = s, a_t = a\right]$$

use: Estimates the expected return of taking action a in state S and following afterward.

Bellman Equation ☆ signals gradie

Value function Bellman equation

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V \right.$$

Action-value function Bellman equat

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

Use: This provides a recursive relationship to compute value functions enabling iterative methods for policy evaluation and improvement

Policy

What: It is balancing the trade-off between exploring new actions (to discover their effects) and exploiting known action that yield high rewards

Use of Policies:
This is essential in RL to ensure the agent learns effectively over time and avoids local optima.

Temporal Difference Learning (TD Learning)

Formula:

$$V(s) \leftarrow V(s) + \alpha\left(R + \gamma V(s') - V(s)\right)$$

Use: combines ideas from monte carlo methods & dynamic Programming to update value estimates based on other learned estimates enabling learning from incomplete episodes.

Q-Learning update rule

Formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Use of Q-Learning:
This allows an agent to learn the value of actions directly from interactions with the enviroment enabling Off-Policy Learning.

Deep Q-Networks (DQN)

use: Combines Q-learning with deep neral networles to approximate the Q-Function, enabling the handling of high-dimensional state-spaces like images.

Policy Gradient Methods

Formula:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla \log \pi_\theta(s_t, a_t) R(\right.$$

use: Directly optimize the policy by adjusting its parameters (θ) to maximize expected returns, useful for complex action spaces and

Deep Learning formulas

Gradient Descent

This is a optimization algorithm used to minimize the loss function by updating model parameters iteratively.

Formula:

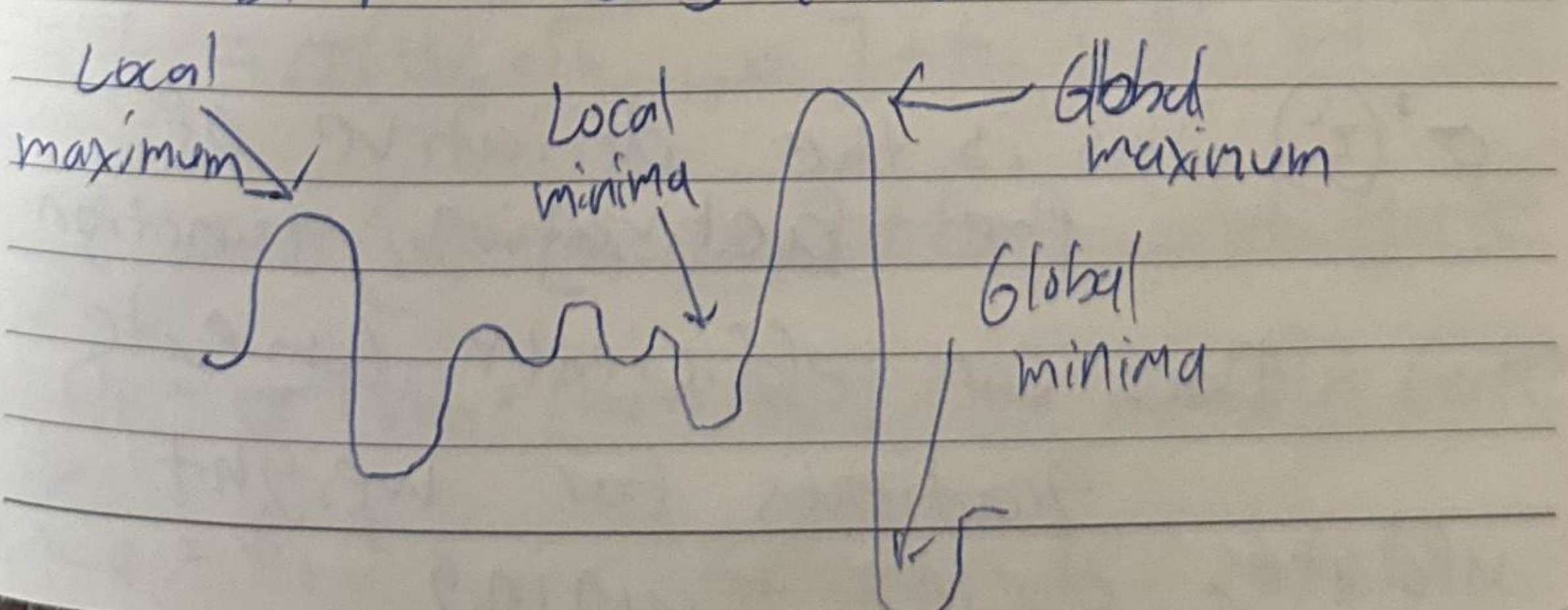
$$\theta = \theta - a\nabla J(\theta)$$

θ is the model Parameters

a is the learning rate

$J(\theta)$ is the loss function

this algorithm guides the model by minimizing errors.



Back Propagation through Neural Network

An algorithm for calculating gradients of the loss function with respects to weights using Chain rule.

Formula

For a layer l,

$$\delta^l = \nabla_a L \circ \sigma'(z^l)$$

δ^l is the error for layer l.

$\nabla_a L$ is the Gradient of the loss with respect to activations

$\sigma'(z^l)$ This is the deriative of the activation function

This can efficienty compute gradients for weight

Activation function

* check other Pages for ReLU & sigmoid

TANH activation function:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

These introduce non linearity enabling the model to learn complex relationships

---

Long short-term memory (LSTM) for MLP

LSTM cell equations meaning is on next page →

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

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ input gate

$O_t = \sigma(W_O \cdot [h_{t-1}, x_t] + b_O)$ output Gate

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

$h_t = o_t * \tanh(C_t)$ hidden state

These LSTM manage long term dependencies in sequencial data effectively.

Auto encoders

Structure

encoder:

$$z = f(x) = \sigma(W_e x + b_e)$$

decoder:

$$\hat{x} = g(z) = \sigma(W_d z + b$$

Useful for dimensionali
and anomaly dete

optimizers

Adam optimizer:

Importance: Different
help navigate
effectively, improving training

Adam optimer formula:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta = \theta - \frac{a\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

LSTM Cell equation meanings

f_t is the forget gate activation at time t (a value between 0 and 1)

N_f is the weight matrix for the forget gate

h_{t-1} is the hidden state from the previous time step (t-1).

x_t is the input at time t.

Explanation of

b_f is the bias term for forget gate.

σ is the sigmoid activation function

$O_{t-1}, f_{t-1}, I_{t-1}, \tilde{C}_{t-1}$ are same meaning for different gates

W_o, W_f, W_i, W_c are also same meaning

b_o, b_f, b_i, b_c are also same meaning

tanh is the hyperbolic tanget function which outputs values between -1 and 1.

C_t is the cell state at time t.

C_{t-1} is the cell ~~state from previous~~

M Gates.

* is the symbol for element wise multiplication

h_t is the hidden state at output at time t.

C_t is the current cell state at time t

Autoencoders Explanation

W_e is the weight matrix for encoder.

b_e is the bias vector for encoder.

h is the encoded representation (latent space of input)

$h = f(x) = \sigma(W_e x + b_e)$ is the equation for the encoder where f is the activation function.

* These mean the same thing for the Decoder's as well.

Adam Optimizer explained

Parameters

θ are the Parameter (weights) of the model that are being Optimized.

Learning rate

α is the learning rate, which controls how much to update the Parameters during training (it is set to a small value e.g 0.001)

Exponential Decay rate:

β_1 decay rate for first moment (e.g 0.9)

g_t is the Gradient of the loss function with respect to parameters at time step t.

Moment estimates

m_t the first moment estimates (mean of the gradients)

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

v_t is the second moment estimate (the uncentered variance of the gradients)

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias corrected moment estimates

$\hat{m}_t$ the bias correcred first moment estimate

$\hat{v}_t$ is the bias corrected second moment estimate

Parameter update:

The Parameters are updated using:

$$\theta = \theta - \frac{a\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

ϵ, is the small constant added for numerical stability usually around (10^{-8}).

DIAGrams OF NETS in ML

Feed forward Neural Networks (FNN)

- single-layer Perceptron
- multi layer Perceptron

Convuntuional neural networks (CNN)

- Standard CNN
- Residual Networks (ResNet)
- Inception Networks
- U-Net (used for image segmentation)
- SqueezeNet

Recurrent Neural Networks (RNN)

- Standard RNN
- Long short-term memory (LSTM)
- Gated Recurrent Unit (GRU)

Generative models

- Generative adversarial Networks (GAN)
- Variational Autoencoders (VAE)

Transformer Models

- Vanilla transformer
- BerT (Bidirectional Encoder Representations from transformers)
- GPT (Generative Pre-trained transformer)
- T5 (Text-to-text transformer)
- Vision transformer (ViT)

Autoencoders

- Denoising Autoencoder
- Sparse Autoencoder
- Variational Autoencoder (VAE)

Graph Neural Network

- Graph convontual network (GCN)
- Graph Attention Networks (GATN)

Reinforcement Learning networks

- Deep Q-Networks (DQN)
- Policy Gradient Methods
- Actor-Critic methods

Specialized networks

- Capsule networks
- Attention networks
- Neual Turing machines
- self-organizing Maps (SOM)

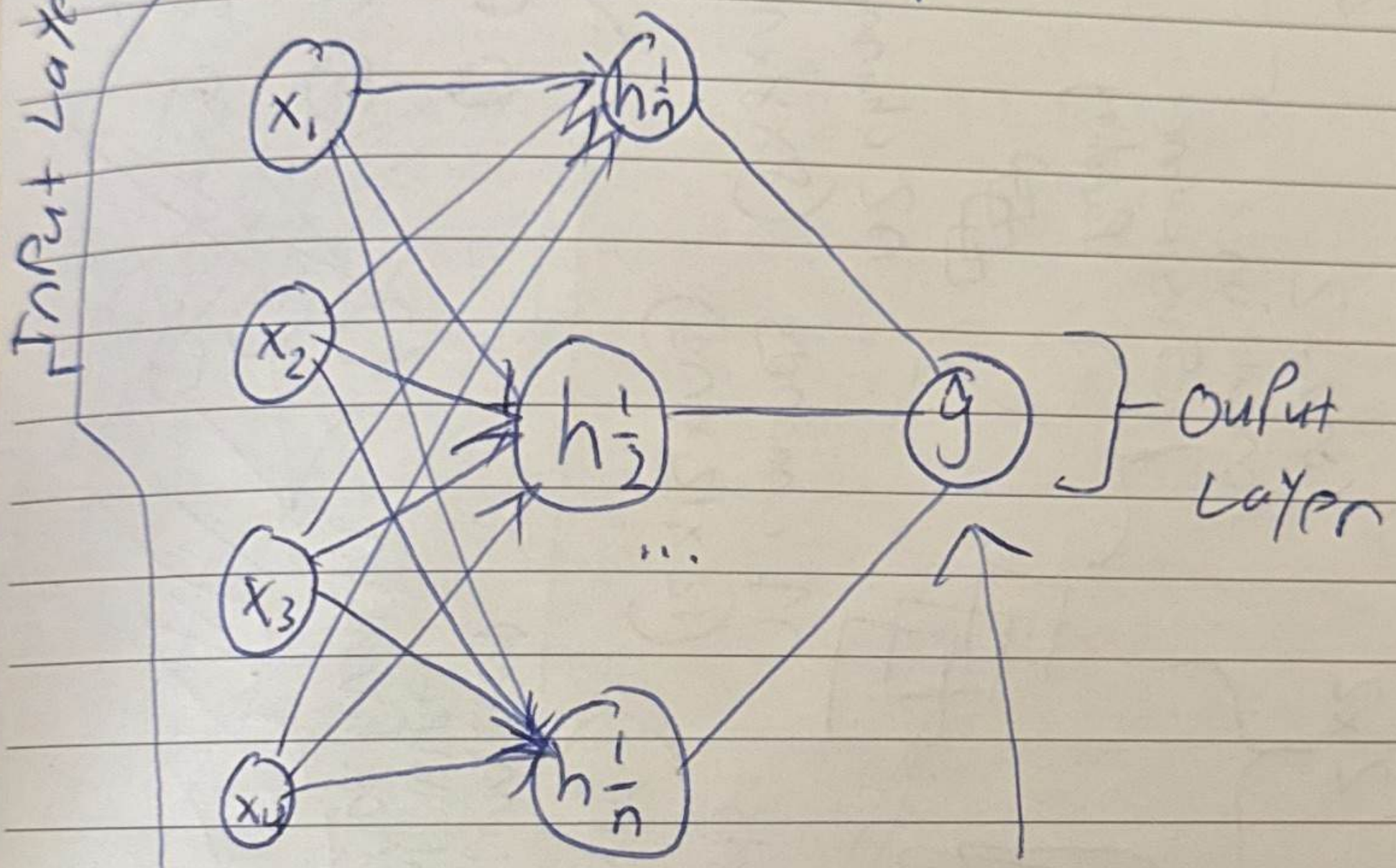
HYbrid models

- convultional Recurrent Neural Networks (CRNN)
- Neral Architecture Search (NAS) Networks

Models / visual representations

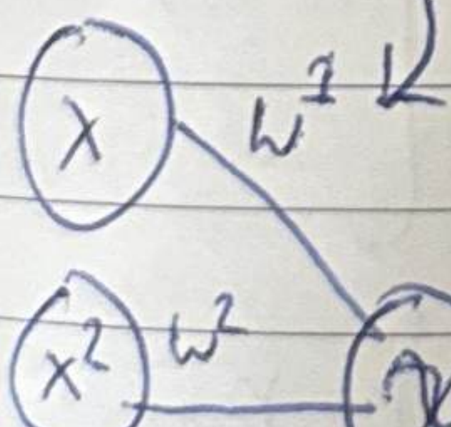
Feed forward Neural Network

Multi-LAYER PercePtron



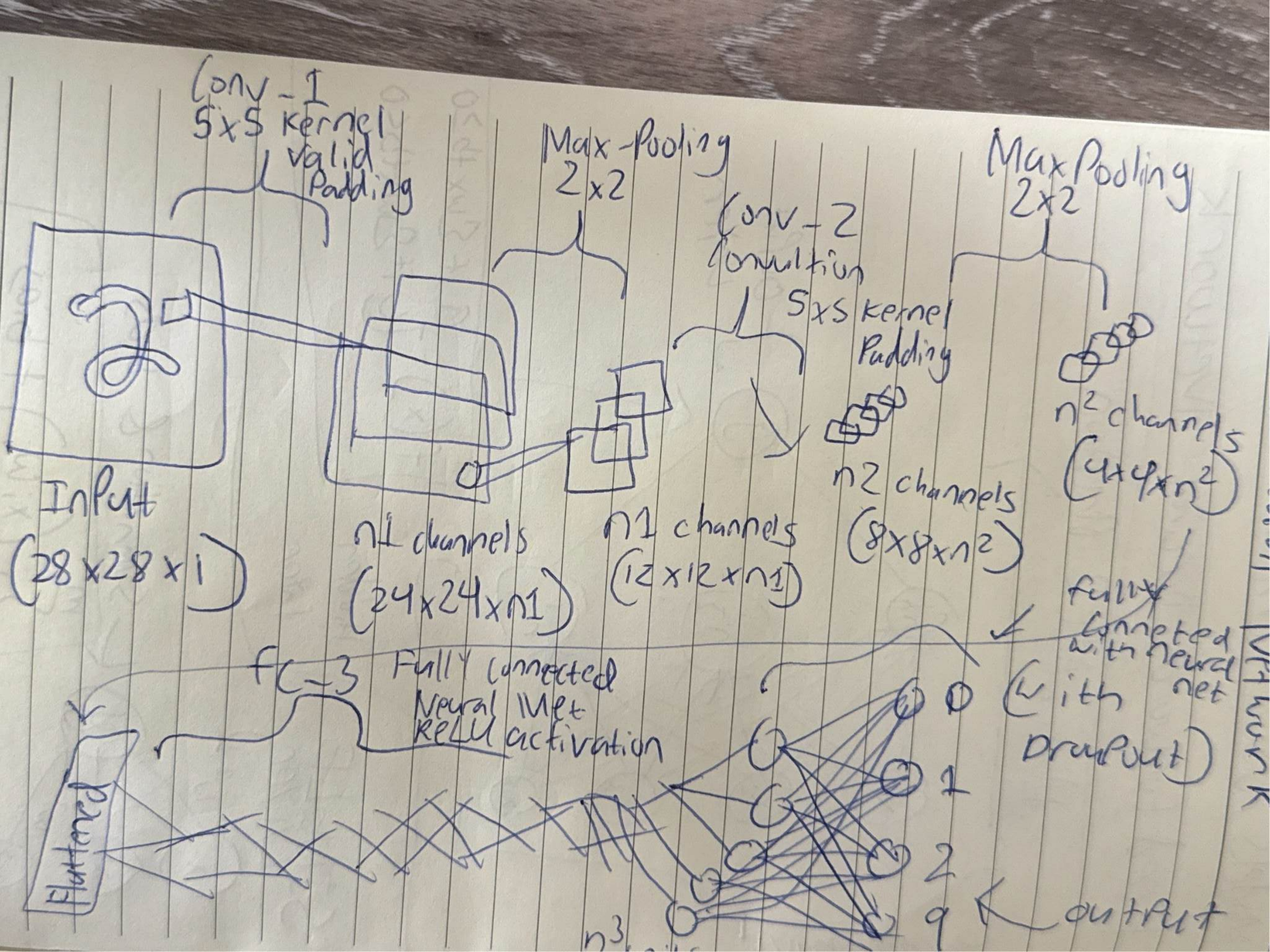
SINGLE LAYER

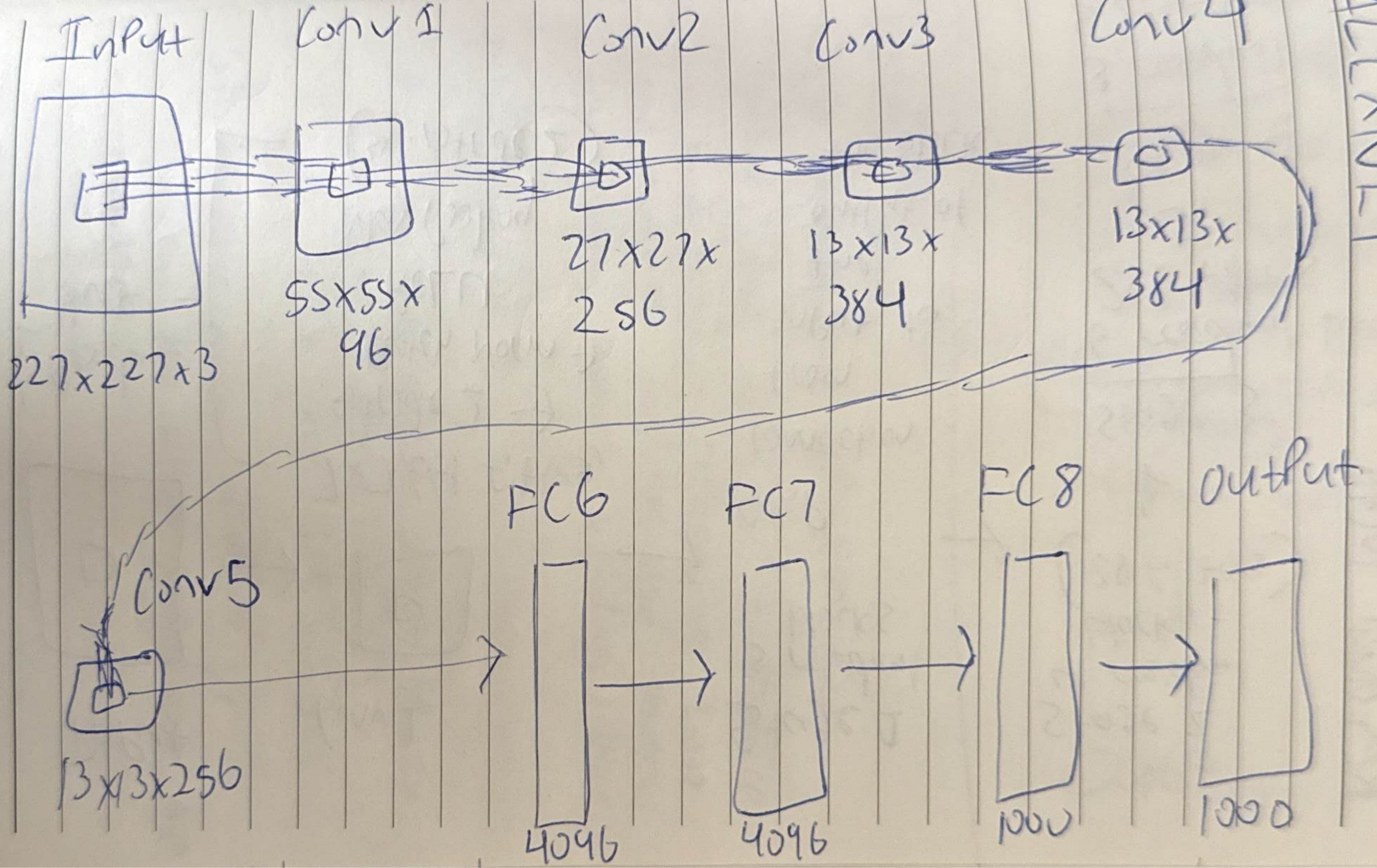
$$f(x) = \begin{cases} 1 \text{ if } \sum wx + b \geq 0 \\ 0 \text{ if } \sum wx + b \leq 0 \end{cases}$$



↳ Summation & bias

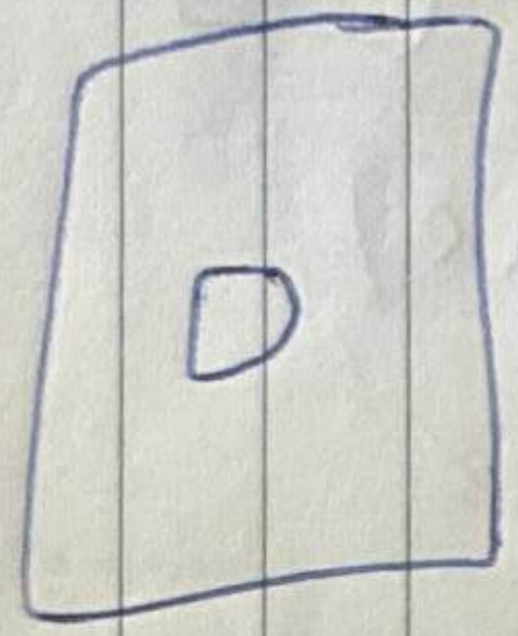
$$\sum_{i=1}^{m} (x_i w_i) + \text{bias}$$


Conv-1
5x5 kernel
valid
Padding
Max-Pooling
2x2
Conv-2
Convultion
5x5 kernel
Padding
Max Pooling
2x2
Input
(28x28x1)
n1 channels
(24x24xn1)
n1 channels
(12x12xn1)
n2 channels
(8x8xn2)
n2 channels
(4x4xn2)
fully
connected
with neural
net
(with
DropOut)
FC-3
Fully connected
Neural Net
ReLU activation
Flattened
0
1
2
9
output


Input
Conv 1
Conv2
Conv3
Conv 4
Conv5
FC6
FC7
FC8
output
227x227x3
55x55x
96
27x27x
256
13x13x
384
13x13x
384
13x13x256
4096
4096
1000
1000

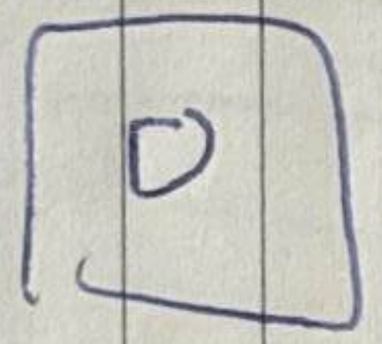
Residual Net

Input



→

Conv 1



→

7x7/64 filters,
stride 2 →
Batch Norm →
ReLU
Max Pooling
(3x3, stride 2)

→

Stage 1
3 residual blocks

Skip
connection
from
input to
the
output of
the block

→

Stage 2
4 residual blocks
(128 filters)

↓

Stage 3
6 residual blocks
256 filters

↓

Stage 4
3 residual blocks
512 filters

$\frac{1}{2}$ Residual block

→ Final Layer
Average Pooling
(Global average
Pooling)
(Fully connected
Layer)
Softmax output
→
output

Convultion

- Is a mathematical operation that combines two functions to produce a third function, representing how one function modifies the other. In CNN it inuves a sliding filter.

Filter (kernel)

A small matrix used to detect specific features (edges & textures) in the input (e.g 3x3, 5x5)

Stride: The number of Pixels by which the Filter slides over the image.

Padding

The addition of extra Pixels around the input image to control Spatial dimensions of the output feature map.

Valid Padding: No Padding, output is smaller than input

Same Padding: Padding is added to keep the output size the same as input size.

Feature map: the output produced by applying a filter to the input.

Pooling: A downsampling operation is used to reduce the Spatial dimensions of the image.

Max Pooling: Takes the maximum value from a region.

Average Pooling: Takes a average value

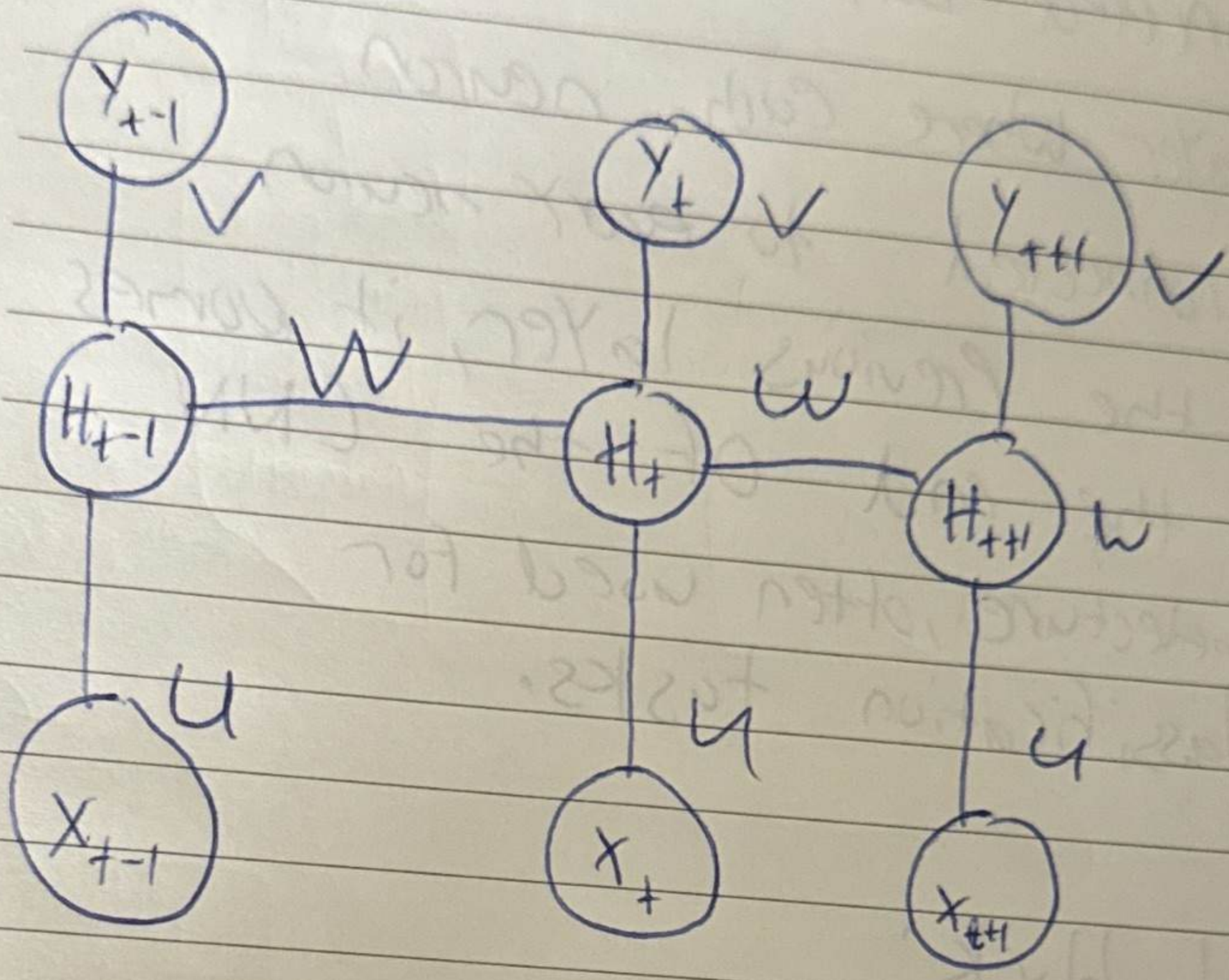
Fully connected Layer:

A layer where each neuron is connected to every neuron in the previous layer, it comes at the end of the CNN archetecture, often used for classification tasks.

Residual block:

A building block in archetectures like ResNet that include skip connections, allowing the gradient to flow through the network more effectively

Recurrent Network



U = weight evector for hidden layer

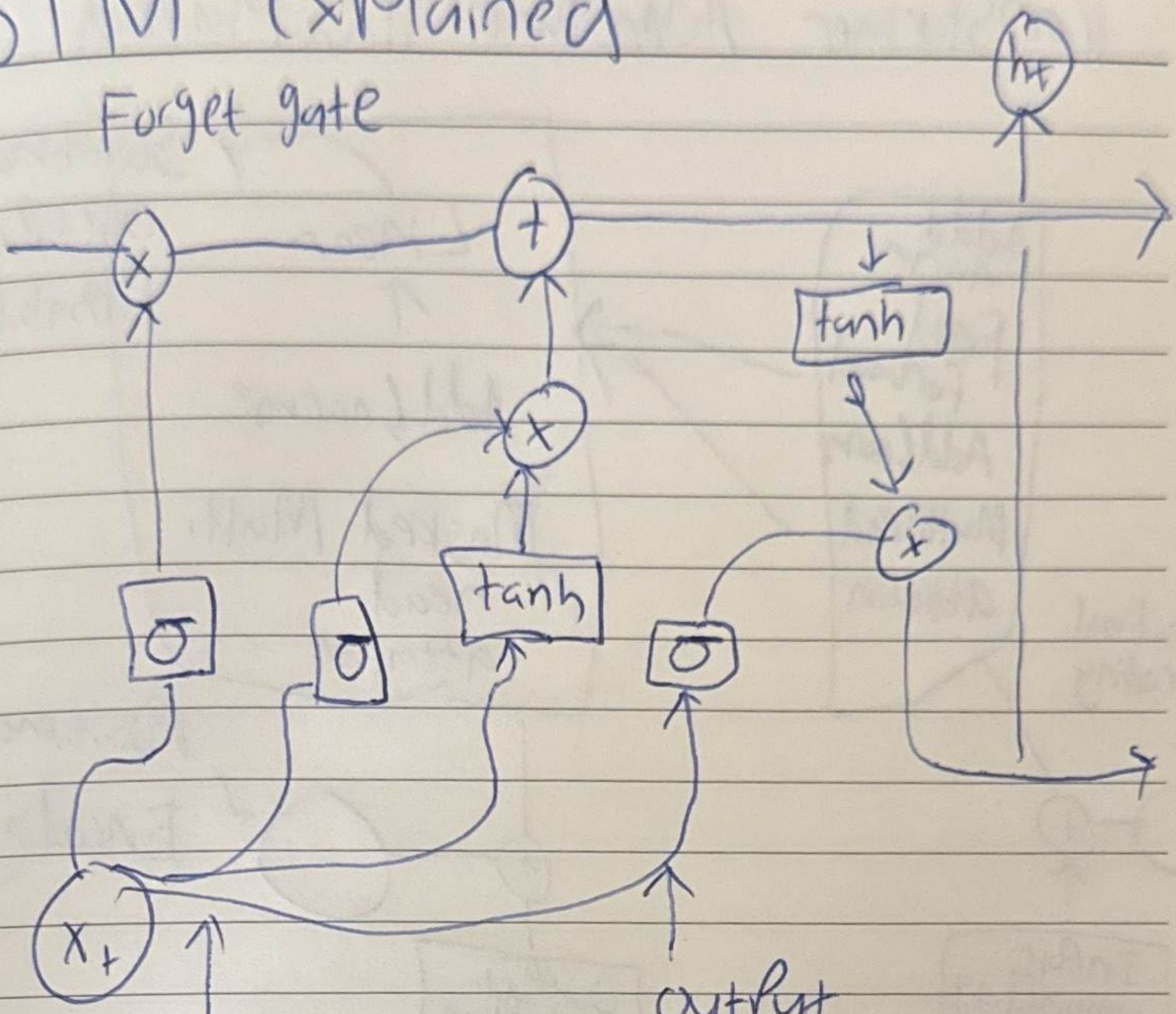
V is weight vector for output layor

W is the same weight for many timesteps

X is the word vector for input word

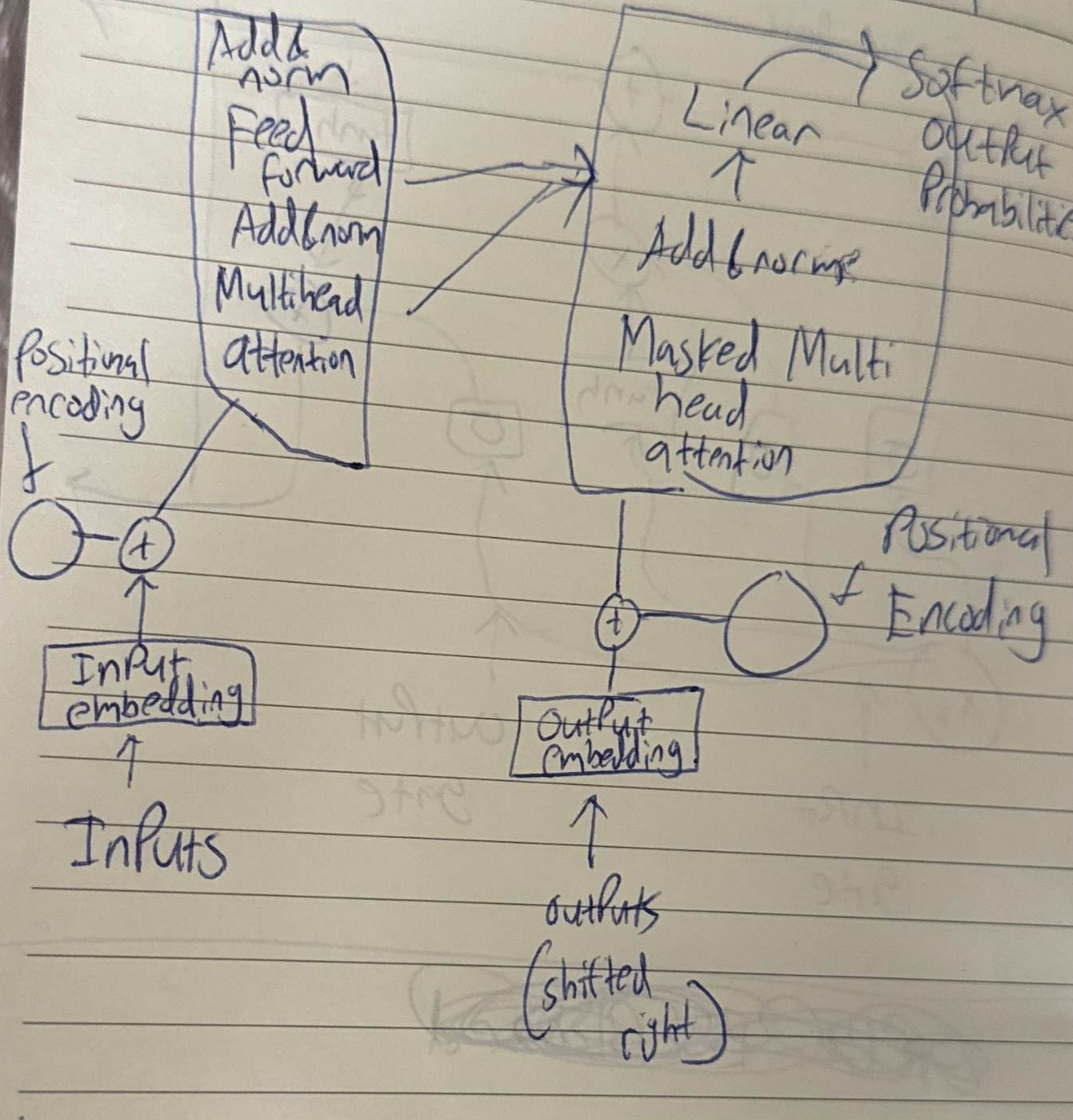
Y is the word vector for output

LSTM explained



GRU explained

Transformer Achreture Explained



DATA Science Fundamentals for ML

1. DATA COLLection

Gather relevant data from various sources

Sources:

Databases: SQL or NOSQL, where it is Stored in Structured formats

API's: Interfaces to retrieve data from external websites

Web scraping: extracting data from Web Directly

CSV/excell files: import data from flat files

2. DATA Cleaning

Ensure Data quality & integrity

Handling missing values

- Removing records:

  Delete rows or colums with excessive missing values

- Imputation techniques:

  - Mean, Median, Mode & ranye
  - K-Nearest Meighbors
  - Inter Polation

use heat maps for visualizing missing data Patterns using libaries like seabom.

keep format of data Consistent.

Handling outLiers in DATA

use IQR Method to identify outliers outside of the range:

$[Q1 - 1.5 \times IQR, Q3 + 1.5 \times IQR]$

Capping: Limit the value of outliers by capping them at certain thresholds.

Transformations: APPLY transformations to reduce the effect of outliers
eg: Log transformation.

Cross-Reference Data to make sure it is accurate.

Documentation of cleaning Processes in data.
Automate reptitive Data cleaning tasks.
Involve domain experts.
Always test the data after cleaning the data.

Data Transformation

Normalization
this rescales the data to a specific range typically (0,1). this is important for various tasks.

Methods:

Min-Max scaling:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

used, when the data does not follow a gaussian distrubution & when it is important to have features on same scale.

Ex: converting range of ages from a range of (0,1).

tandarization of DATA

this transforms data to have a average of 0 and a standard deviation of 1. This is important for algorithms that use assume normally distributed data, such as

LINEAR REGRESSION,
LOGISTIC REGRESSION,
Principal component analysis.

Methods:

2-score Normalization:

$$x' = \frac{x - \mu}{\sigma}$$

μ is the mean of the data and σ is the standard deviation of the data

this is effective when features have different units & scales.

Categorical Variables

Categorical variables need to be converted into numerical format.

Methods:

- one-hot encoding:

  Creates binary colums for each value, useful for data without intristic ordering (e.g color, type)

  Ex. ['Red', 'blue', 'green'] Create three colums is_red, is_blue, is_green.

- Label encoding:

  This is good for data which categories have a clear oder (e.g 'low', 'medium', 'high'.

  Example: ["low" = 0, "Medium" = 1, "high" = 2]

Other important tips:

Feature Extraction:

- create new features from data to enhance the model's ability to capture patterns.

use TF IDF for text data.

Feature engineering

use knowledge to create new features that can improve a model's performance.

DIMENSIONALITY REDUCTION

- Reduce the number of features while preserving essential information, which can help prevent overfitting.

Methods: PCA, t-distributed Stohastic neighbor embedding

DATA Integration

- Combining data from varias sources to create one coherent dataset.

DATA SPLITING in ML

Generalization: This can help the model make predictions on new unseen data.

Model evaluation: This can help model evaluation on new unseen data.

Avoiding overfitting: This can help avoid overfitting as model's need to train on other data than the train set.

training set: This set of the data is used to train the ML Model.

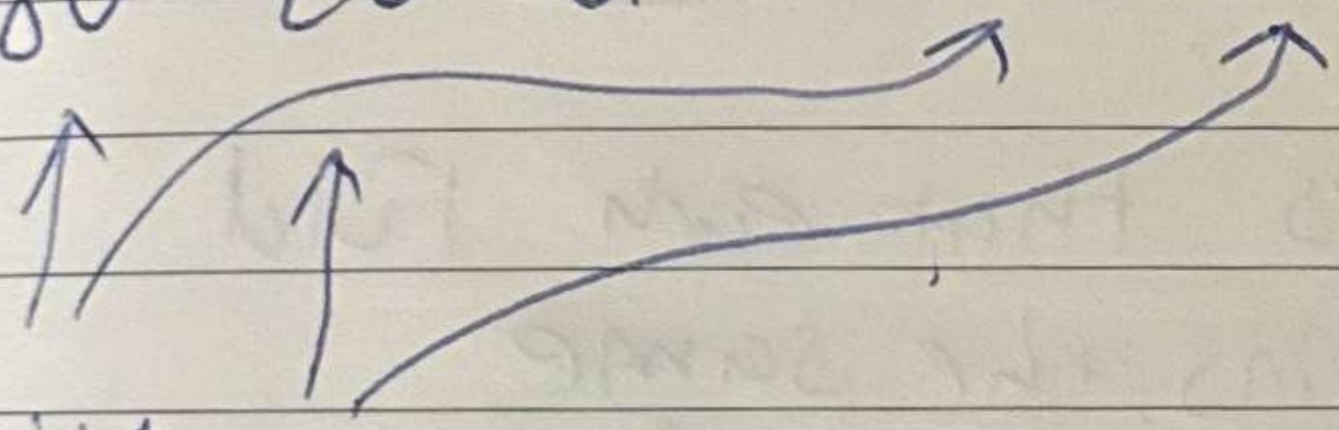
Validation set: This is used to tune hyperparameters and make choices about the model architecture.

Test set: this is used after the model has been trained & tuned. this provides a final evaluation on how well the model is doing in real-life.

SIMPLE Train-Test split:

Divide the set like this:

80-20 or 70-30



☆ Do not use this if the dataset is small.

K-Fold cross validation

Dataset is divided into K-Folds, then it is trained k times using K-1 Folds for training and Rest for validation.

* The issue is that this is computationally expensive.

Reduces variance in performance estimates.

- All datapoints are used for training & validation.

Stratified K-Fold cross validation

This ensures that each Fold maintains, the same proportion of class labels as the original dataset.

Essential for imbalanced datasets.

Pros: Preserves class distribution

Cons: slightly more complex to implement that standard K-fold.