

St. Paul's Mission School Kolkata



NIT. ROURKELA.

WINTER,

IN TERN SHIP.

DIARIES.

Name Kushanaw Rakshit.

Class 2nd Sec. B(y). Roll No. 2022CEB008.

Subject NIT. ROURKELA Year 2024-25

EMG signals

electromyography is a technique used to measure the electrical activity produced by muscles.

Muscle activity generates electrical signals called action potentials (APs), detected by non-invasive techniques like placing copper electrodes on the skin.

EMG output data is a superimposition of all the APs, coming from all the motor units that the electrode is detecting.

CNN detects APs, and then patterns in the APs, so features are automatically detected from the output data.

Henneman's law

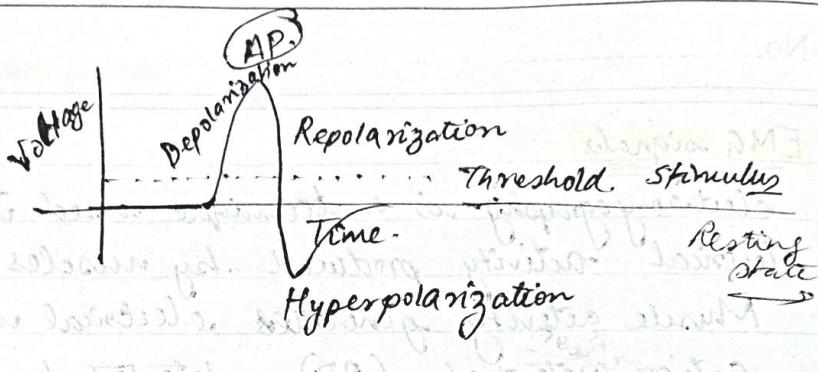
Motor units are recruited from smallest to largest based on the size of the motor neuron and the associated muscle fibres.



Extracting information about the type of muscle fibres that are firing, would give an insight on the amount of force we need to produce in our arm.

(APs of different muscle fibres have a certain range of frequencies)

EMG output data
is noisy hence
requires filtering
and feature
extraction to
get useful
information.



① Patterns in APs.

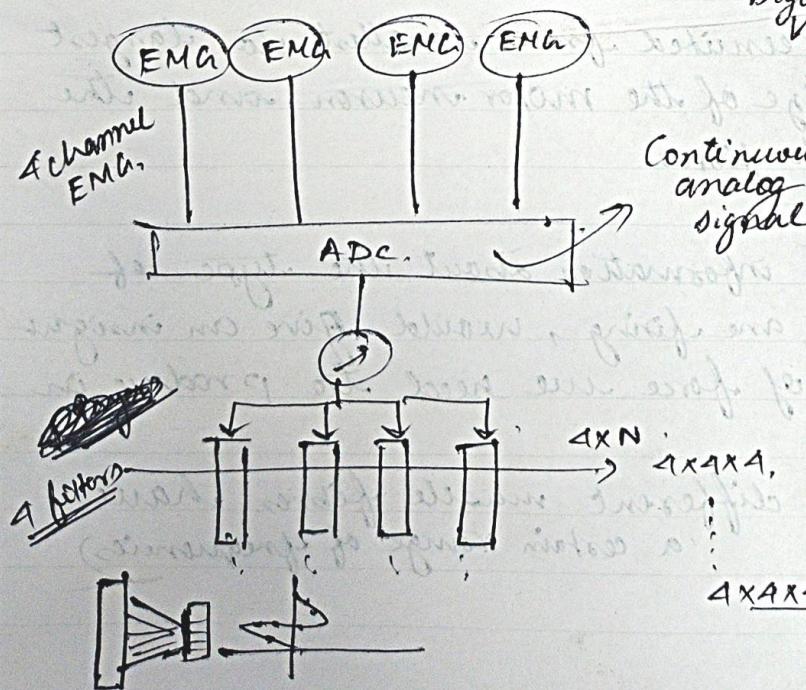
② Features of APs.

③ Location from where signal was taken

Determine
corresponding
motion.

Force required ↑.
AP frequency ↑.
Amplitude ↑.

Digital signals have binary
values and not affected by
noise hence used



Continuous
analog
signal.

digital

$$\text{Time} = \frac{N}{\text{Sampling rate}}$$

4 filters are created.

Each filter with 3 layers

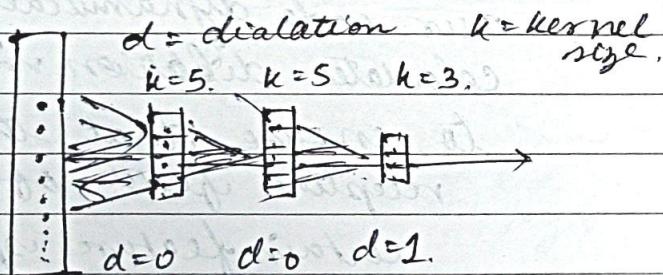
with each layer having
different dilations.

to capture data.

with a larger receptive field

Type I Motor units (slow stretch) → Low frequency.

low amplitude AP.



$$R = \frac{\text{Sampling rate}}{\text{frequency}} = 1 + \sum_{i=1}^L (k_i - 1) \cdot d_i.$$

L = no. of layers.

R = Receptive field

k_i = Kernel size of i th layer.

d_i = dilation state of i th layer.

3 features for AP.

Type I. 30Hz

Type II. 100Hz.

Type IIIB. 225Hz

1 feature for silence $k=5$.

(NO RELU in 1st layer).

{ To capture negative parts of AP }

Teacher's Signature:

Type I.: Freq 30 Hz

$$k=15.$$

$$L=4.$$

$$\text{Stride} = 1.$$

Sampling rate → set as a variable in code so that the code can be adapted for different ~~use~~ hardware

$$d_0 = \frac{\text{sampling rate}}{\text{freq} \times k}$$

Trainable features

i.e. kernel values are calculated by back propagation.

∴ size of AP doesn't change our code dynamically calculates dilation values to ensure that the receptive field of a certain feature spans a fixed time period.

Type IIa. $k=10.$
Freq = 100 Hz.
 $L=4,$
stride = 1

Trainable feature
dilation calculated dynamically from formula

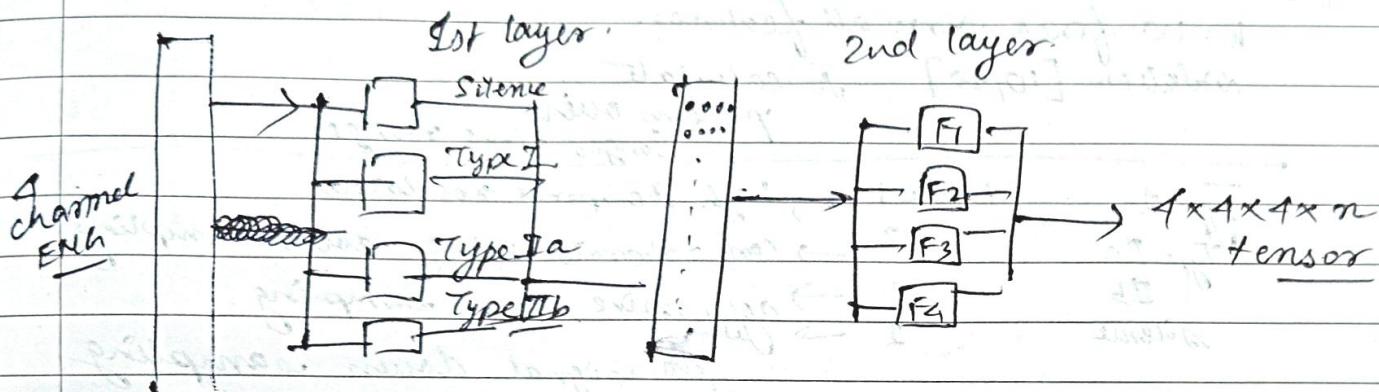
Type IIb $k=7.$ Freq = 225 Hz
 $L=4.$
stride = 1

Silence

$k=7.$
 $L=4$
stride = 1.
no dilation

fined feature

$$[-1 \ 1 \ -1 \ -1 \ -1 \ 1]$$



1D CNN is individually performed on each column of each channel and output matrix is filled by the output columns of 2nd layer.

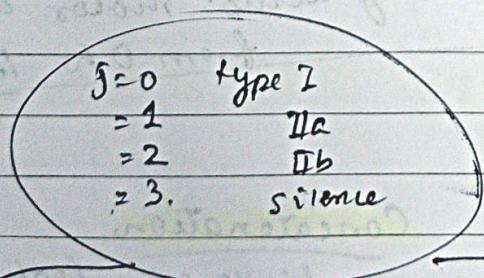
F_1, F_2, F_3, F_4 } identical 1D CNN.
 F_1, F_2, F_3, F_4 } multi layered trainable features.

RELU
present

If no stride.
output = $4 \times 4 \times 4 \times n$ tensor.

i = EMG signal
 f = 1st layer feature
 k = 2nd layer feature
 n = sample

$i \times j \times k \times n$ will represent
64 data points at the sample ' n '



for $i \times j \times k \times n$. $s=1$.

$i \times j \times k \times n$ $s=2$.

Total Receptive field

$$R = 1 + \sum_{i=1}^L (n_i - 1)(d_i)(s).$$

Teacher's Signature : _____

for 2nd layer

$k=10$ fixed covers all features.

dilatation [10, 20] to calculate

patterns over longer time ranges

Type 1

stride = 1

→ high temporal resolution

Type IIa

2

→ can tolerate minor down sampling

Type IIb

3

→ aggressive down sampling.

silence

1

no signal down sampling
for rest periods.

MLP

Input layer = output from 2nd CNN.

Hidden layer → 128 with ReLU Activation
→ 64 with ReLU activation.

Output layer = 5 neurons,

each corresponding to 1 servo motor.

Predictions of servo motor angles
from 0 to 180

Processing pipeline

Flattening

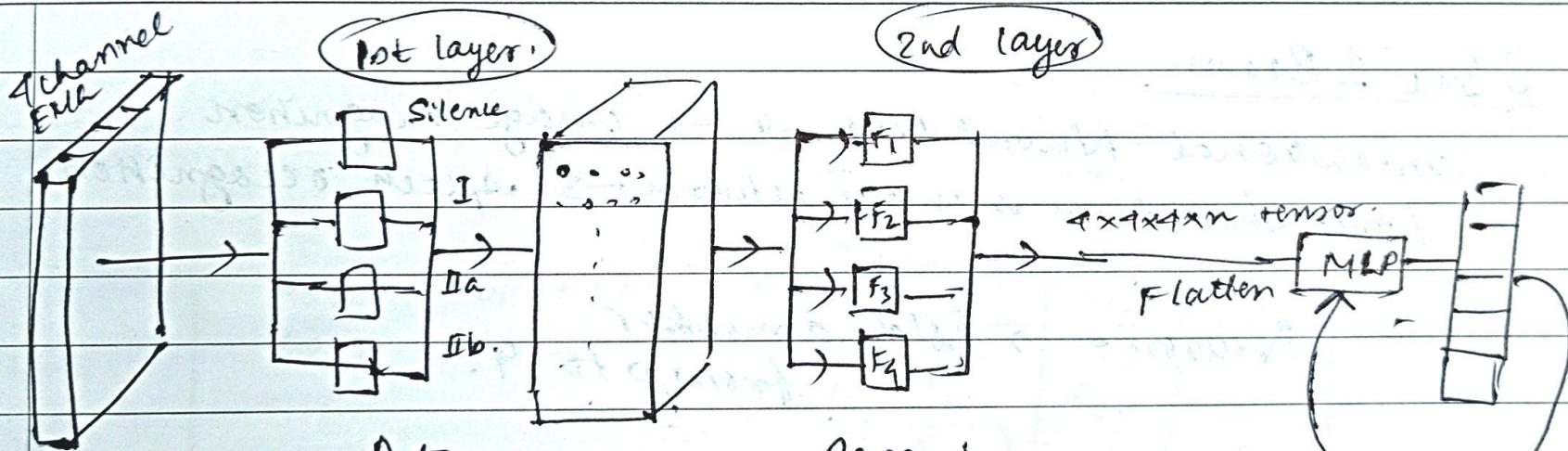
Output of 2nd CNN

2D vector

$$(4, 4, 4, 50) = 4 \times 4 \times 4 \times 50$$

Concatenation

flattened feature vector is concatenated with historical servo angle data ($A_{\{n\}}$) and (A_{n-1}) adding 10 neurons



Inputs
raw
emr data

Detests
APs.
of
different
rear
ranges

Recognises
patterns
in
APs

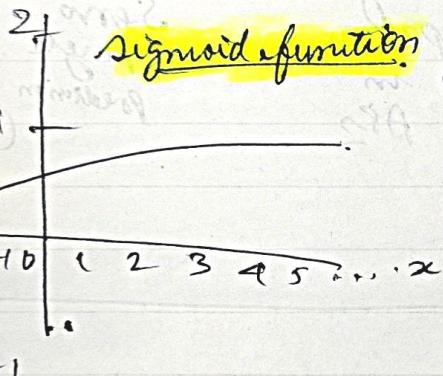
Servo
Angle
Prediction
Output
(5 servo
angles)

CNN Notes

3 Blue 1 Brown

Convolutional Neural Network \rightarrow image recognition
 Long short term memory network \rightarrow speech recognition.

Neuron \rightarrow holds a number from 0 to 9.

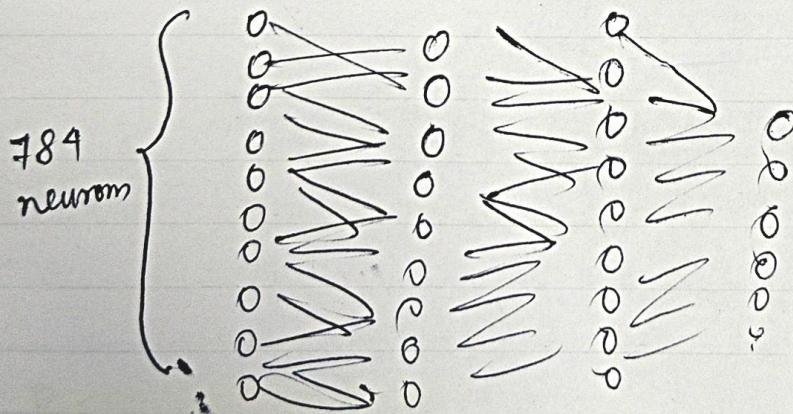


$$\sigma(x) = \frac{1}{1+e^{-x}}$$

(slow learns)

$$+ (w_1a_1 + w_2a_2 + w_3a_3 + \dots + w_na_n - 10)$$

\downarrow
bias



784 \times 16 weights
 16 biases.

16 \times 16 \times 10 biases
 $784 \times 16 + 16 \times 16$
 $+ 10 \times 16$.
 weights.

Learning \rightarrow finding the right
 weights & biases.

13002

$$a_0^{(1)} = \sigma(w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)} + b^0)$$

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} =$$

$$t \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

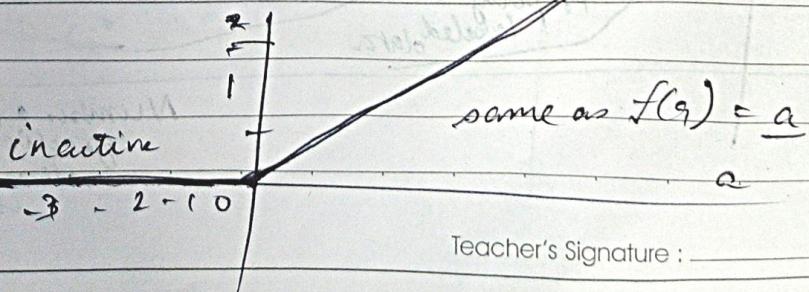
$$\sigma \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \sigma(x) \\ \sigma(y) \\ \sigma(z) \end{pmatrix}$$

$$\text{a}^{(1)} = \sigma(w_a^{(0)} + b),$$

Neuron → function.

Rectified Linear Unit (ReLU) (fast learning)

$$\text{RELU}(a) = \max(0, a)$$



Neural Network function

Input → 784 numbers/pixels.

Output → ~~10~~ 10 numbers

parameters → 13002 weights/biases

Cost function

Input → 13002 weights/biases.

Output → 1 number

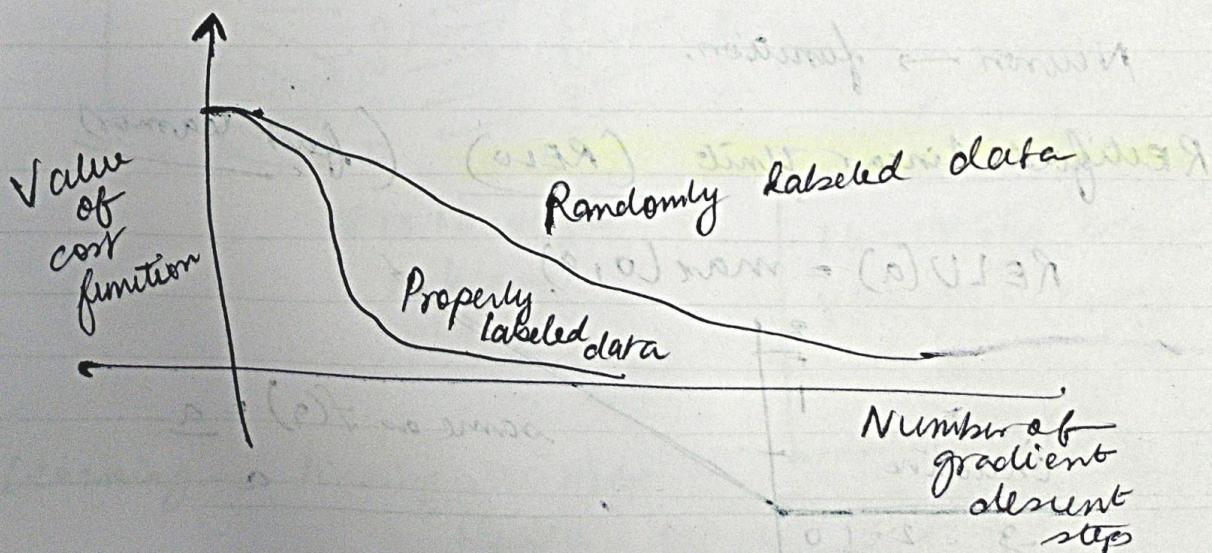
parameters → training examples

gradient descent → to reach local minimum
of cost function

$$-\nabla C(\vec{w})$$

Old technology → Multi-layer Perceptron (MLP)

New → CNN
LSTM



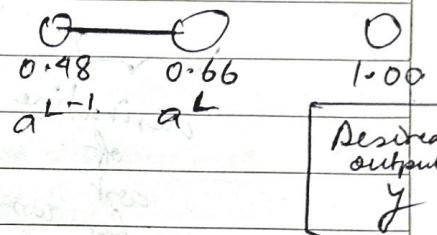
Backpropagation

a technique
to understand
how to modify
weights and biases.

in training data
to reduce.

cost function

by gradient descent.



Cost $\rightarrow C_0 (\dots)$

$$\underline{C_0 = \frac{(a^L - y)^2}{(0.66 - 1)^2}}$$

(MNIST Database)

$$a^L = \sigma(\underbrace{w^L a^{L-1} + b^L}_{z^L})$$

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w} \cdot \frac{\partial a^L}{\partial z^L} \cdot \frac{\partial C_0}{\partial a^L}$$

$$\underline{a^L = \sigma(z^L)}$$

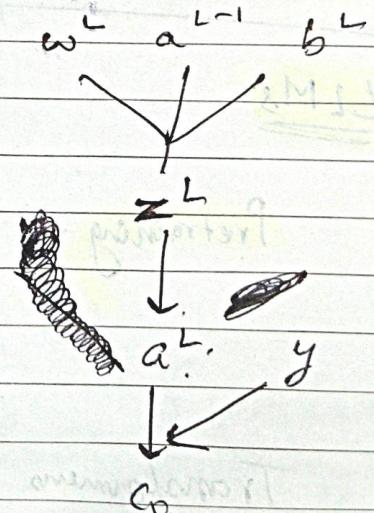
$$\frac{\partial C_0}{\partial a^L} = 2(a^L - y),$$

$$\frac{\partial a^L}{\partial z^L} = \sigma'(z^L),$$

$$\frac{\partial a^L}{\partial w^L} = a^{L-1},$$

$$\frac{\partial z^L}{\partial w^L} = (a^{L-1})^T \cdot 2(a^L - y)$$

$$\therefore \frac{\partial C_0}{\partial w^L} = (a^{L-1})^T \cdot (a^L - y)$$



$$\therefore \frac{\partial C}{\partial w^L} = \frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^L}$$

derivative
of
cost
function

Average of
all training
examples

$$\nabla C =$$

$$\begin{bmatrix} \frac{\partial C}{\partial w^1} \\ \frac{\partial C}{\partial b^1} \\ \vdots \\ \frac{\partial C}{\partial w^L} \\ -\frac{\partial C}{\partial b^L} \end{bmatrix}$$

LLMs

Pretraining \rightarrow Reinforced Learning

with
human
Feedback

RLHF

Transformers use feedforward mechanism as well
attention

GPT \rightarrow generative pre-trained transformer

input \rightarrow sentence/voice.

\downarrow
broken down into tokens/vectors $\xrightarrow{\text{high dimensional}}$

GPT3 \rightarrow first LLM model in world

$$\begin{aligned} \text{total weights} &= 175,181,291,520 \\ &= 27938 \text{ matrices} \end{aligned}$$

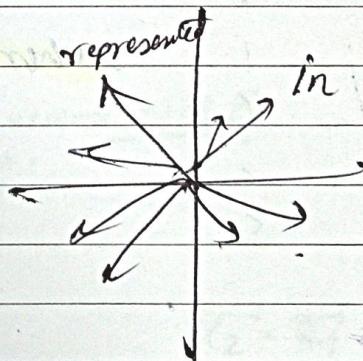
Embedding matrix (W_F) \rightarrow first layer.

Words $\xrightarrow{\text{embedding}}$ vectors.

Embedding space

12288-dimensional

All data in deep learning must be represented as vectors



Unembedding matrix \rightarrow last layer

$$[W_U \cdot [\quad]] = [\quad] \xrightarrow{\text{softmax}} \left. \begin{array}{l} \text{probability} \\ \text{distribution} \\ \text{of next words} \end{array} \right\}$$

Softmax

(technique to arrange numbers according to probability distribution from 0 to 1)

$$e^{x_1} > 0$$

$$e^{x_2} > 0$$

$$\vdots$$

$$= \sum_{n=0}^{N-1} e^{x_n}$$

$$e^{x_n} > 0$$

$$e^{x_1} / \sum_{n=0}^{N-1} e^{x_n}$$

$$e^{x_2} / \sum_{n=0}^{N-1} e^{x_n}$$

with temperature

Logits

$$e^{x_i/T} / \sum_{n=0}^{N-1} e^{x_n/T}$$

when T is

large

small values are given higher weight

so that probability distribution becomes more uniform

T is small:

T is large, large values have importance

[a] [fluffy] [blue] [creature] [roamed] [the] [verdant] [forest]

\downarrow E_1 E_2 - - - - - E_8

E_1' E_2' - - - - - E_8'

E_1'

$\downarrow wQ$ (query vector = 128 dimensional)

Q'

fluffy

$E_2 \xrightarrow{wK}$ (key vector = 128 D) K_2'

creature

E_4

$\downarrow wQ$

Q_4'

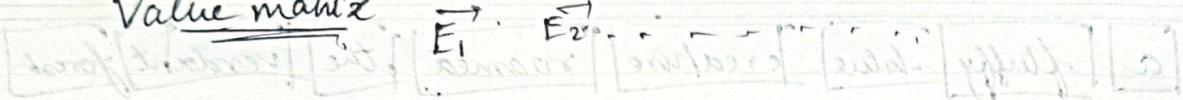
$E_2' \cdot Q_4' \geq 0$ (related)
 ≤ 0 (unrelated)

∴ Attention (Q, K, V) = softmax. $\left(\frac{k^T Q}{\sqrt{dk}} \right) v$.

Masking is a phenomenon by which $E_2' \cdot Q_4'$ values, being negative are treated as $(-\infty)$ so that it doesn't affect previous tokens.

Teacher's Signature: _____

Value matrix



$$\vec{E}_1 \xrightarrow{W_V} \vec{V}_1$$

$$\vec{E}_1 \xrightarrow{W_V} \vec{V}_1$$

$$\Delta \vec{E}_1 \Delta \vec{E}_2 \dots \dots$$

$$\begin{array}{c} \vec{E}_1 \\ + \\ \Delta \vec{E}_1 \\ || \\ \vec{E}'_1 \end{array}$$

one head of attention

value params

||,
(# query params) + (# key params)

self attention \rightarrow masking required
precede next words in a sentence

cross attention \rightarrow no masking

audio to transcript

translation b/w 2 languages

Value vectors inside an attention head would have same dimension as the embedding space.
 Involves taking weighted sums of large vectors, weighted by the attention pattern.

Inside each head, use only the value down matrix to produce 128-D vectors:

> Takes weighted sums of these with attention pattern.

Multiply each by heads' value up matrix to get 12,288 dimensional vector that can be added to the embedding in that position.

Input → Attention → MLP → Attention → MLP.

Output

MLP

non linear function

Linear \rightarrow RELU \rightarrow Linear \rightarrow Output

First Name \rightarrow Michael

$$\begin{bmatrix} \vec{R_0} \\ \vec{R_1} \\ \vec{R_2} \\ \vdots \\ \vec{R_n} \end{bmatrix} = \begin{bmatrix} \vec{E} \end{bmatrix} = \begin{bmatrix} \vec{R_0}, \vec{E} \\ \vec{R_1}, \vec{E} \\ \vdots \\ \vec{R_n}, \vec{E} \end{bmatrix} = \underbrace{\dots}_{\text{If } E \text{ encodes first name Michael}} \quad \begin{cases} \approx 1, \text{ If } E \text{ encodes first name Michael} \\ \leq 0, \text{ If not} \end{cases}$$

Bias

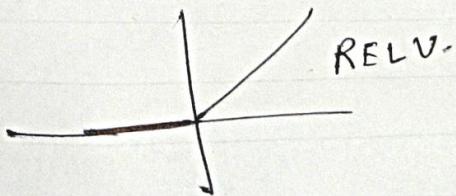
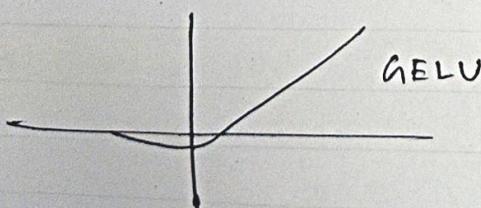
$$+ \begin{bmatrix} \quad \end{bmatrix} = (\vec{M} + \vec{J}) \cdot \vec{E} - \underline{1} = \vec{M} \cdot \vec{E} + \vec{J} \cdot \vec{E}$$

upprojection

$$W_i \vec{E}_i + \vec{B}_i = \text{output matrix}$$

after passing through RELU, positive values are retained.
zero/negative values are made zero = 0

Same as \Rightarrow AND gate



$$\left[\begin{array}{c} \vec{c}_0 \\ \vec{c}_1 \\ \vec{c}_2 \\ \vdots \\ \vec{c}_n \end{array} \right] \cdot \left[\begin{array}{c} n_0 \\ n_1 \\ n_2 \\ \vdots \\ n_m \end{array} \right] + \left[\begin{array}{c} \vec{B} \end{array} \right] \cdot \vec{c} = \text{output}$$

$$n_0 \vec{c}_0 + n_1 \vec{c}_1 + n_2 \vec{c}_2 + \dots n_m \vec{c}_m$$

down projection

$$W \downarrow \square + \vec{B} \downarrow = \text{output}$$

GPT 3

Johnson-Lindenstraum Lemma

\Rightarrow Maximum of vectors
 $\approx \exp(\epsilon \cdot N)$

Embedding

Key

Query

Value

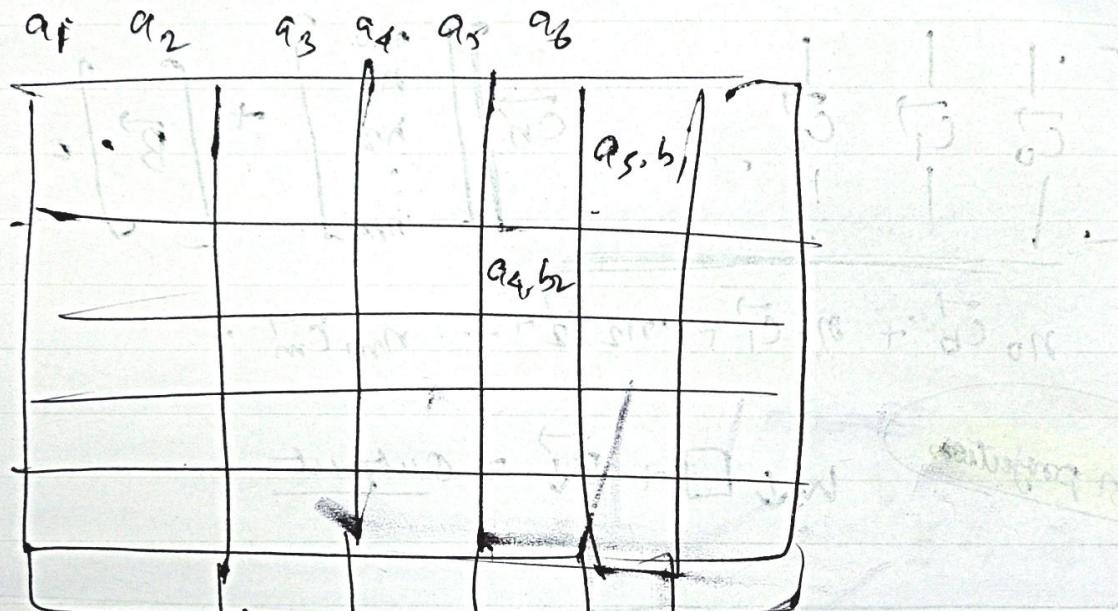
Output

Up projection.

Down projection

Unembedding

Convolution of (a_i) and (b_i) → Gaussian distribution



$$(a * b)_n = \sum_{\substack{i, j \\ i+j=n}} a_i \cdot b_j$$

$n=0 \text{ to } n=n$

$$\begin{matrix} a_0 & a_1 & a_2 \\ 1 & 2 & 3 \end{matrix}$$

$$\begin{matrix} b_0 & b_1 & b_2 \\ 4 & 5 & 6 \end{matrix}$$

$$= (c_0, c_1, c_2, c_3, c_4, c_5) \\ = (4, 13, 28, 27, 18)$$

$n=0$
 $= i_{\min} + j_{\min}$

$n=4$
 $= i_{\max} + j_{\max}$

$$(1 \cdot 4) \quad \text{for } n=0$$

$$(1 \cdot 5 + 2 \cdot 4) \quad \text{for } n=1$$

$$(1 \cdot 6 + 3 \cdot 4 + 2 \cdot 5) \quad \text{for } n=2$$

$$(2 \cdot 6 + 3 \cdot 5) \quad \text{for } n=3$$

$$(3 \cdot 6) \quad \text{for } n=4$$

Fast convolution algorithm

$$a = [a_0, a_1, a_2, \dots, a_{m-1}]$$

$$b = [b_0, b_1, \dots, b_{n-1}]$$

$$f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{m-1} x^{m-1}$$

$$g(x) = g_0 + g_1 x + g_2 x^2 + \dots + g_{n-1} x^{n-1}$$

$$a * b = f(x) \cdot g(x)$$

$$\underline{\underline{O(N^2)}}$$

Const. N^2 + (stuff asymptotically
smaller than N^2)

Fast Fourier Transform (FFT)

$$\hat{a} = [\hat{a}_0, \hat{a}_1, \hat{a}_2, \dots, \hat{a}_{m+n-1}]$$

$$\hat{b} = [\hat{b}_0, \hat{b}_1, \hat{b}_2, \dots, \hat{b}_{m+n-1}]$$

$$\hat{a} * \hat{b} = [\hat{a}_0, \hat{b}_0 + \dots]$$

inverse FFT

$a * b$

$$\underline{\underline{O(N \log(N))}}$$

CNN notes

DVLV

input dimensions are very big (for an MLP)

1 channel of an image $1920 \times 1080 \approx 2M$ features.

1 second of sound at 44kHz = 44k features.

So, a video frame rate + image features + sound

10 seconds

$$\rightarrow (60 \text{fps} \times 3 \times 2M + 44k) \times 10$$

music as input

user preference

classifier

pleasant music/not. ($1 \rightarrow 100$)

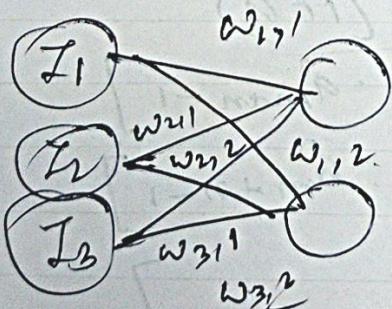
regression

Traditional manual feature extraction

(problem)

Noise variations.

Fragments missing



MLP \rightarrow ignores ordering of inputs.

Large MLP required with lots of weight.

We represent our filter as a vector which is equivalent to the weight matrix of an MLP.

The filter can act as a feature extractor.

filter can be used at all locations to detect criteria.

→ we call sliding a filter over all positions convolving

The operation is called a convolution operator.

The output of the convolution operator is itself again a vector.

A timeshift in the input causes the same shift in the output : Convolution is translation equivariant.

This operation can be understood as a small MLP that wanders across

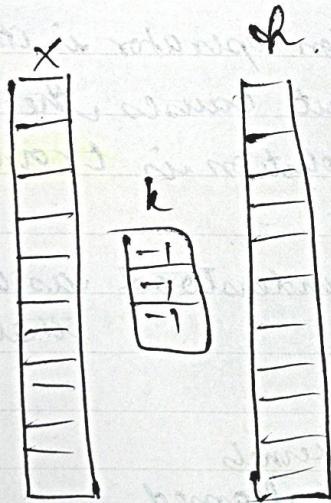
In practice only, the conv kernels are learned.

To understand how the conv kernels are learned, we require a formal definition.

For, normal 1D input sequence $x \in \underline{\mathbb{R}^n}$,
and a filter $k \in \underline{\mathbb{R}^{2m+1}}$

convolution operation is

$$h(t) = \cdot(x * k)(t) = \sum_{\tau=-m}^m x(t-\tau) \cdot k(\tau)$$



If k is learned, we update the filter weights based on some loss

gradient utilized
to update a kernel weight $k(t_0)$
is given by

$$\frac{\partial L_h}{\partial k(t_0)} = \frac{\partial L_h}{\partial h} \cdot \frac{\partial h}{\partial k(t_0)}$$

depending on the conv. response at all places.

e.g. \rightarrow cross correlation for classification

$$\begin{aligned} \frac{\partial L_h}{\partial k(t_0)} &= \sum_{t=0}^m \frac{\partial L_h(t)}{\partial h(t)} \sum_{\tau=-m}^m x(t-\tau) \cdot \boxed{\frac{\partial k(t)}{\partial k(t_0)}} \\ &= \end{aligned}$$

\rightarrow always zero except when sum zero = 1

The weights are shared across the entire input.

(MLPs learn independent weights at every position)

$$h(t) = W(t, :) \cdot x(t).$$

Since weights are shared for every position,

Convolutional Neural Networks CNN are ~~more~~ smaller than MLPs

parameter efficiency

Convolutions can learn a powerful pattern recognizer.

e.g. for silence, based on "silences" appearing everywhere in the input.

(MLPs must learn an independent silence recognizer.

for every position)

Data efficiency

Convolutions can recognize a "silence pattern". regardless of where it appears.

(MLPs must have seen silence at a given position before in order to recognize it.)

generalization improvements

consequence
of
CNN

being
translation
equivariant.

Near the boundaries of the data, we cannot apply the convolution as we normally do:-

Solutions

ignore boundaries.
Pad boundaries → add $(\text{filter length} - 1)/2$.

around the data to preserve the dimension.

→ Fill this with a fixed value, often 0,

we need to reduce dimension for the final classification.

#

→ larger steps with our filter.

the size of the step is called the stride.

The dimension reduces with a factor equal to the stride.

→ Input dimension must be a multiple of the stride.

Solutions

→ use a larger stride.

use a pooling layer

→ goes over the data similar to a convolution.

→ applies a deterministic function like max or average on the input.

→ usually has stride = pool size

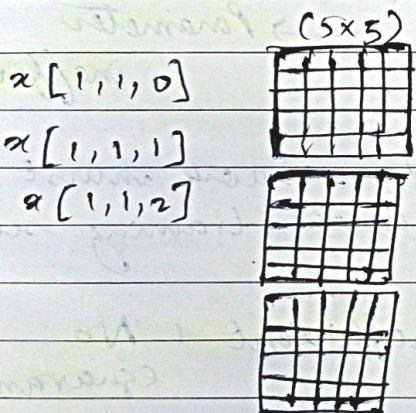
2D-CNN

If we work with a 2D image \rightarrow becomes a 3D tensor.
~~3D~~ \rightarrow becomes a 4D tensor.
 video

We have a 5×5 image.

3 channels.

2 filters \rightarrow 3D



filter w_0 w_1
 $(3 \times 3 \times 3)$ $(3 \times 3 \times 3)$

'Bias. b_0 b_1
 $(1 \times 1 \times 1)$ $(1 \times 1 \times 1)$

b_0

b_1

b_0

Input volume + (pad 1)

convolving near the borders.

$\rightarrow (7 \times 7 \times 3)$

Output volume $(3 \times 3 \times 2)$

A translation of the input produces an equivalent translation in the output.

for a translated input $x(t - t_0)$:

$$(x * k)(t - t_0) = \sum_{\tau=-m}^m x(t - t_0 - \tau) k(\tau).$$

CNN is equivariant to translation
non-equivariant to rotation / scale mapping

Convolution - Equivariance analysis

Problem:- Each of these filters are independent weights:-

The network wastes a lot of parameters learning transformed versions of the same

Parameter
inefficiently

To learn these filters, the network must see these transformations in the training set.

→ Data inefficient + No equivariance guarantees

Solutions ?? (Group convolutions)

→ not just share parameters for translation but also other transformations!!

→ Extreme parameters sharing + equivariance guarantees.

→ Active field of research.

Recurrent architectures eg. - RNNs, LSTMs, ...

are able to handle arbitrarily long sequences.
→ via recurrence.

However, suffer from vanishing/exploding gradients problem.

(difficult to train and to learn from past)

CNNs offer interesting alternative for sequence modelling

Causality is easily obtained by padding asymmetrically,
for a convolutional kernel of size K .

→ add padding " $K-1$ "

~~convolution~~

→ the output at position " t "
can be dependent on input values
upto " $K-1$ " steps in the past.

"The space it sees is called
receptive field"

→ dilation factor d

We can stack several convolutional layers to form
dilated causal convolutional networks.

→ Temporal Convolutional Networks
(TCNs)

TCN's do not have recurrent connections .,

they do not use Back-Propagation through
time.

→ Trained in parallel.

Much faster training + Optimal GPU usage.

- TCNs do not exhibit exploding/vanishing gradient descent problems along time axis
- They learn from past by input values within their receptive fields.

(Input values outside receptive field cannot be considered for the calculation of output at a particular position)

In order to avoid vanishing gradients and improve learning, TCNs use batch normalization, residual connections and dropout.

Residual block (κ, d)

