Mo Tu We Th Fr Sa Su

Date 13 / 4 / 24

"DEAP NEURAL NETWORK"

① Fundamentals of NN.

→ Supervised / Unsupervised / Semi supervised RL

→ Why DL?

→ Perceptron

② → Multilayer Perceptron ← multiple layers input → hidden → output

→ & XOR Problem.

→ MLP as classifiers & Universal Proximators
↳ with single hidden layer can approximate any continuous function

→ Issues of Depth & width.

more complexity but vanishing gradient

can learn intricate pattern but computationally heavy

③ Deep Feed Forward NN.

hidden layers — depth vs width — complexity vs computation — vanishing gradients

→ Forward & Backw. Propagation.

→ propagate input forward for given input data

→ propagate weight backwards from output to input using loss function to evaluate

IMPOSSIBLE is I AM POSSIBLE

Mo Tu We Th Fr Sa Su

→ computation graph — nodes represent operation (+, ×) etc
DAG, visual representation of mathematical operations
— Edge
↳ used to debug neural network architecture

→ Gradient Descent.
↳ optimization algo for minimizing loss function to find optimal parameters
↳ Loss or cost function $\boxed{\theta = \theta - \alpha \times \text{gradient}}$

→ Impact of Depth in DNN.
↳ more complex
↳ hierarchical Abstractions
↳ vanishing / Exploding gradients
↳ overfitting potential

(4) Deep feed Forw. NN.

→ Activation Funct & → Soft Max Regression.
↳ get non linearity in network
↳ sigmoid $= \frac{1}{1+e^{-x}}$
↳ tanh $= \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
↳ Relu $\max(0, x)$
↳ Leaky Relu → sets stuck at 0
↳ softmax $\frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$ [common in classification]

(5) Optimization Algo. /
↳ choose depend on data size, model architecture, convergence speed requirements

→ Challenges - Saddle points & plateau
points where gradient is zero but not a min or maxima (Hessian matrix is mixed)
close to 0 for extended period

→ Non-convex Optimization Intuition

→ Stochastic gradient Descent (SGD)
↳ Uses single random sample in each iteration
↳ high variance

→ Mini Batch SGD.
↳ computes gradient using small batches of data

→ Overview of R Prop, Quick Prop.

Mo Tu We Th Fr Sa Su Date / /

→ Momentum, Nastov's Accelerated Momentum.

→ Algo. with Adaptive Learning Rates
↳ ADA Grad
↳ Adapt learning rate based historical gradients

↳ Momentum - uses an exponentially weighted avg of past gradient to accelerate convergence

→ RMS Prop.
↳ squared gradients
↳ prevents from decreasing learning rate

larger updates to infrequent parameters

→ Adam.
↳ Adaptive Moment Estimation
↳ combines ideas from momentum & RMSProp

(6) Regularization for Deep Models.
↳ prevent overfitting

→ Model Selectn, Underfitting, Overfitting.

→ L1 & L2 Regularisatn
↳ penalize large weights, preventing model complexity

→ Drop out.
↳ randomly set some input units to 0

→ Challenge - vanishing & exploding gradients.

→ Parameter Initializatn.

Mo Tu We Th Fr Sa Su Date / /

→ chanenge co-variants shift.

→ Batch Normalizatn
↳ standardize to mean 0 & sd 1. ↳ Adds regularization
↳ reduce vanishing / exploding gradient
↳ faster convergence

(7) Convolution NN
↳ specialized NN architecture for grid like data like images

→

→ Basics of computer vision & Invariants

→ Convolutn in Images.
↳ sliding filter / kernel on image to perform element wise multiplication & summation, creating feature maps local patterns

→ Learning a kernel
↳ learn filters in image extract hierarchical features like edges, textures

→ Padding & Stride
add border pixels
amount of movement of filter

→ Channels

→ Pooling
→ Max pooling: Extract maximum value from each patch reducing spatial dimensions
→ Average pooling: Mean of patch
down sampling
prevent overfitting

→ Designing ← CNN.

→ (1) Convolution layer
(2) Pooling layer
(3) Flattening
(4) Fully activated layer

Mo Tu We Th Fr Sa Su

Date / /

(8) Popular CNN Architectures

→

→ LE NET → Yann LeCunn in 1998
→ MNIST data
→ Conv layers + average pooling & fully connected layers

→ ALEX NET → Alex, Ilya in 2012
→ alternating conv & max pooling layers
RELU activations, dropout & fully connected layers

→ VGG 16 → 2014
→ small 3X3 conv filters
→ stacked conv layers (19 layers in VGG 19), max pooling (fully connected layers)

→ RCNN & fast RCNN
↳ Region based CNN
↳ selective search to generate region proposals → apply CNN to each region
→ pass through SVM classifier for object classification
Fast RCNN → Introduce Region of Interest (RoI) to reduce computation
share computation across regions

→ Network in Netw. (NIN)
↳ introduce micro networks (MLP's)
↳ 1X1 conv layer → incr. non linearity, ReLU

→ Inception Net (Google Net)
↳ Inception modules
↳ diff. filter sizes

→ Res NET → Microsoft, 2015
→ (100 + layers)
→ Address vanishing gradient problem
skipping connections

→ Dense NET. (Dense CNN)
↳ feature reuse, mitigate vanishing gradient
↳ dense blocks

→ Transfer Learning
↳ inductive transfer → from source & target share patterns, behavior, concepts
↳ Transductive transfer → unrelated domain

→ App. of CNN
↳ obj detection
→ ↳ Image classification
↳ NLP
↳ Driving, Robots
→ RSS, VGG. pretrained models

Mo Tu We Th Fr Sa Su Date

→ handle sequential data
→ understand context & temporal patterns
→ audio, video, text

(9) Sequence Models

→ Recurrent NN.
↳ sequence data
↳ RNNs maintain internal hidden state that captures previous information

→ Types of Sequences & RNNs.
↳ LSTM's ↳ capture long term dependencies
↳ ↳ maintains info over long sequences

→ Back Propagation
↳ RNN's are trained through back-propagation in time steps to update weights (BPTT)

→ Gates & Exploding / Vanishing Gradient

→

(10) Popular RNN Architectures

→ Gated Recurrent Units. (GRU).
↳ simplified version of ~~GRU~~ LSTM
↳ have reset & update gates that control info flow

→ Long Short Term Memo (LSTM)
↳ Long term sequences in memory
↳ handle vanishing gradient problem

→ Bi directional Models
(Bi RNNs or Bi LSTMs)
↳ use both past & future information
↳ two separate recurrent layers (LSTM layers)

(12) Attention Mechanism

→ Self Attent.

→ Positional encoding

Mo Tu We Th Fr Sa Su Date / / Vaswani, 2017

→ Transformer Architecture

→ App. of Transformers

(11) Attent. Mechanism

→ Attent. Pooling
↳ Max, Average, Global

→ Attent. Scoring Funct.

→ Multi Head Attent.

(13) NN Search Overview.
↳ Find similar datapoints in dataset

→ Search Space

→ " Algo. → Brute Force Search
Index structures like KD-trees, Ball trees, LSH (Local sensitive hashing)
FAISS

→ Examin Strategy.
↳ Curse of Dimensionality

→ Siamese Networks
→ Triplet Networks
→ Graph Neural Nets
→ Memory Augmented Networks

← long range dependencies
↳ RNN, LSTM struggle
↳ Parallel computation
↳ attention mechanism
each input token is associated with 3 vectors
→ Query, Key, Value
→ compute attention scores
→ weighted sum of values based on attention scores
↳ capture dependences long range between words

Mo Tu We Th Fr Sa Su Date / /

14) Time Series Modelling & Forecasting

→ Using CNN → use input window cfg
→ 1D CNN
→ More pooling
→ Fully connected Dense layers

→ " LSTM
(Trend & Seasonality)
↳ cell state, forget state, Input, Gates, Output Gate
↳ Gates & Activations: Sigmoid & tanh

15) Federated Learning

→ FL of out of vocab. words
Meta Learning,

16) Online (Incremental) Learning

→ Continual Life Long Learning with NN

→ 3 scenarios for Continual Learning

Mo Tu We Th Fr Sa Su Date /

* LABS

1. Tensor flow, Keras.
2. Pytorch
3. Back prop. & Optimization. ANN
4. CNN
5. RNN
6. LSTM
7. Transformers
8. Time series forcasting.

Perceptron Example

OR logic Function

x_1	x_2	OR (x_1, x_2)
0	0	0
0	1	1
1	0	1
1	1	1

Mo Tu We Th Fr Sa Su Date / /

Inputs x_1 & x_2

Weights w_1 & w_2

Bias b

Activation Function : Step Function (1 if >0, 0)

Initial weights & bias

$w_1 = 0.5$, $w_2 = 0.5$, $b = -0.5$

sum $= w_1 \times x_1 + w_2 \times x_2 + b$

(1) $0.5(0) + 0.5(0) + (-0.5) = -0.5$, 0

(2) $0.5(0) + 0.5(1) + (-0.5) = 0$, 0

(3) $0.5(1) + 0.5(0) + (-0.5) = 0$, 0

(4) $0.5(1) + 0.5(1) + (-0.5) = 0.5 > 0$, 1

MLP Examples

Data : MNIST, 28x28 pixels ; output : 0-9 number

MLP architecture

Input layer = 784 neurons (28x28)

Hidden layer = Multiple layers

output layer = 10 neurons (0-9 digit)

Data prep : Convert Image to pixel (0-255) & normalize

Model design : choose Activation Function : ReLU for hidden & softmax for output

Issue of depth & width = Add more hidden layers can capture more features like edges, shapes

More Width. More nuanced representation

Number of neurons in hidden : ~~log n~~ < 10 (eg. 100)

Hope is like clouds, some pass by others bring rain.

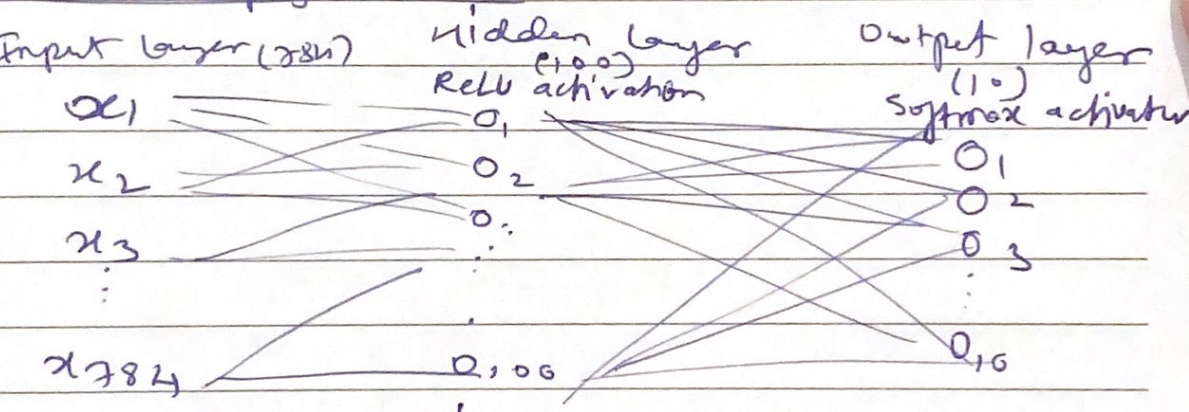
→ Too many neurons in Hidden layer can overtrain
Too few neuron " " " will underfit

Mo Tu We Th Fr Sa Su Date / /

→ Techniques like Regularization can mitigate overfit

→ Number of hidden layers: 1 is good starting point
More than 1 → hierarchical representation of data
↳ consider vanishing / Exploding gradient
↳ complexity vs performance trade off
↳ Architecture CNN, RNN etc require more hidden layers
↳ Regularization needed in deep networks
like dropout & weight decay to prevent overfitting

Forward Propagation



ReLU activation

$$= z_j^{(1)} = \text{ReLU}\left(\sum_{i=1}^{784} w_{ij}^{(1)} x_i + b_j^{(1)}\right)$$

Backward Propagation

① Output predictions are computed ② Loss function is computed
③ Errors are propagated through network to calculate gradients
④ Chain rule is applied to compute gradients layer by layer
⑤ weights of network are updated using algo like Gr. Descent
Batch training, Supervised learning

Be daring, be different, be first to be a success.