Week 10 complete: (8-1-1 — 8-3) Advanced NN

Learning objectives:

- Write NN loss for intermediate ML problems
- write properties of natural, signals.
- List advantages of convolution & pooling layers
- Explain reason of exploding & vanishing gradient
- How LSTM solves exploding gradient problem.

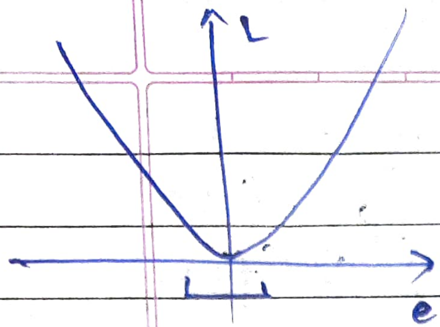
1.) Multi output regression :

$t_i = [t_{i1} \; t_{i2} \; \dots \; t_{id}]^T$ .. Height & width

y_i we want to reduce difference y_i & t_i

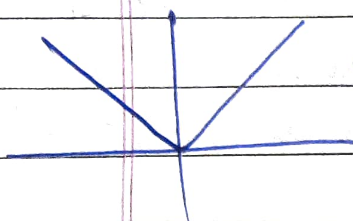
Loss : $\frac{1}{N} \sum_i \frac{1}{2} \| t_i - y_i \|^2$ (For number of training pts)

L1 & Huber Loss :-



Gaussian
Exponential is not a heavy tailed distribution

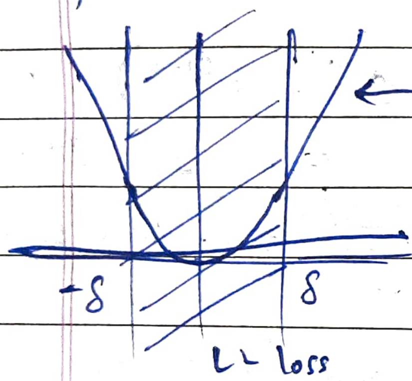
Heavy tailed ⇒ More chance of outlier



← L1 loss gives equal importance to outlier (MAE or L1 loss)

Near origin it gives importance to less error

∴ Compromise :



L1 & L2 piece wise Huber loss.

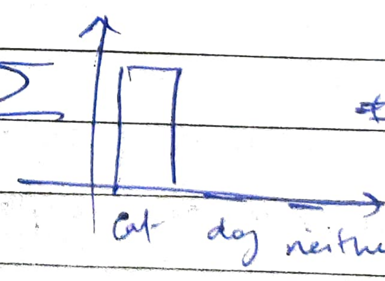
(Continuous & smooth expression)

(8-1-2) N-ary classification :

{cat, dog, neither} — 1-Hot encoding

$t_i \in \left\{ \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix} \right\}$ N vectors or N e_i's

Encodes a target pmf : Σ



$= 1$

$y_i = [y_{i1} \; y_{i2} \; \dots \; y_{ic}]$ for c classes.

$\sum_j y_{ij} = 1$ (PMF essentially)

∴ Minimize cross entropy between t_i & y_i's.

Softmax: $y_{ij} = \dfrac{e^{z_{ij}}}{\sum_j e^{z_{ij}}}$

PAGE No.

DATE / /

$y_{ij} \rightarrow t_{ij} \Rightarrow$ N ary cross entropy loss

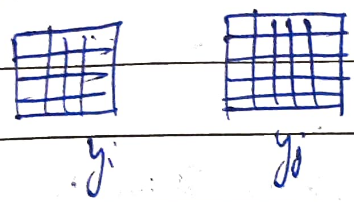
$= \frac{1}{N}\left(-\sum_i \sum_j t_{ij} \log(y_{ij})\right)$ ez pz

(8-1-3) Siamese Networks:

Metric learning using siamese neural network

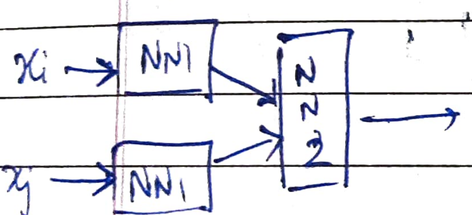
Eg. Ranking problem (x_i, r_i)

Verification $S(x_i, x_j) \in \{0,1\}$



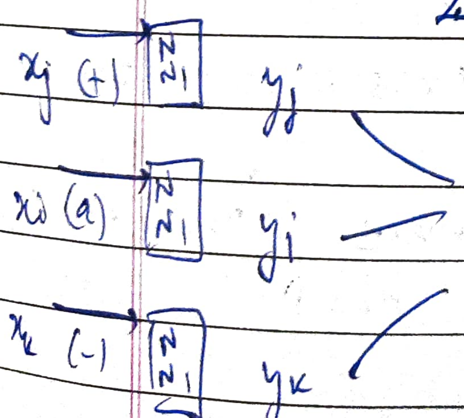
can't do the distance

Need to learn an embedding which captures the similarity of face



The twin NN1 are siamese twins

Triplet siamese $y(x_i)$ $y(x_j)$ closer than $y(x_i)$ $y(x_k)$ [x_k is -ve instance]



Hinge loss (Penalize if x_{ik} distance $<$ x_{ij} distance)

or $\max(0, \|y_i - y_j\| - \|y_i - y_k\|)$ + margin.

(8-2-1) Convolutional Neural Networks 1 :

1. Features are local
2. Their presence/absence is stationary.

2D convolution formula :

Given a input feature map x_{ij}

Take a small window $u, v \in \{-1, 0, +1\}$ (example)

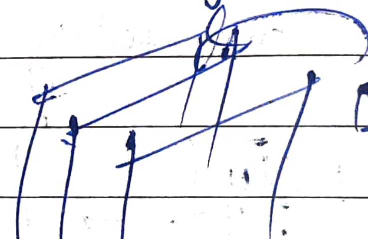
Compute $z_{ij} = \sum W_{uv} x_{i+u, j+v} + b$

Compute $a_{ij} = \sigma(z_{ij})$ or $\max(0, z_{ij})$

For multiple channel convolution :-

$$z_{i,j,m} = \sum_{u} \sum_{v} \sum_{k} W_{u,v,k,m}\, x_{i+u, j+v, k} + b_m$$

kernel index



kernel

Output is also multichanneled

(8-2-2) Convolution & Pooling :

Number of weights with & without convolution :

WO conv
- Input $32 \times 32 \times 3$
- Hidden $28 \times 28 \times 5$

Weights $\Rightarrow 32 \times 32 \times 3 \times 28 \times 28 \times 25 = 60,41,200$

W conv
- Input $32 \times 32 \times 3$
- Hidden $28 \times 28 \times 5$ $(32 - 28 + 1 = 5)$

Weights $5 \times 5 \times 3 \times 25 = 1875$ ONLY

∴ Can stack more layers & use property of features in different locations being all contributing.

Importance of increasing receptive field with depth:

Pooling layer: (Small local summary):

$y_{ij} = avg(x_{2i-1, j-1}; x_{2i-1, 2j}, x_{2i, 2j-1}, x_{2i, 2j})$

$y_{ij} = max($ " $)$

Static operation - Nothing to learn.

Reduces size of a feature map.

Typical CNN architecture:

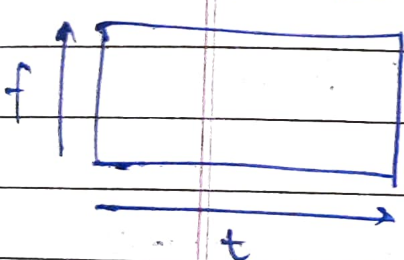
Input → Conv → Pool → Conv → Pool ...

- Till it becomes very small. ⟹ No more conv.

Make it a 1D vector ⟹ fully connected NN & then output (Dense or FC)

[Size of feature map decreases but no of maps ↑.]

CNNs for speech:



frequency f & time t

CNN can be used here as well.

CNNs for text: 1 hot bit is too cumbersome

Dense encoding done of dictionary (GLOVE etc)

CNN called transformers used to operate on this.

(8-3) LSTM:

Other layers to know about:

- Drop out
- Batch normalization

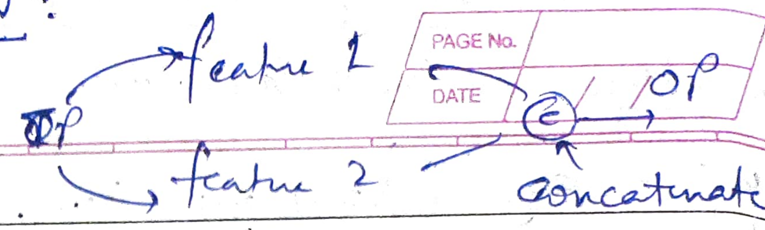
Randomly take Neuron with p & drop it for that training.

(Neurons learn indep. useful features)

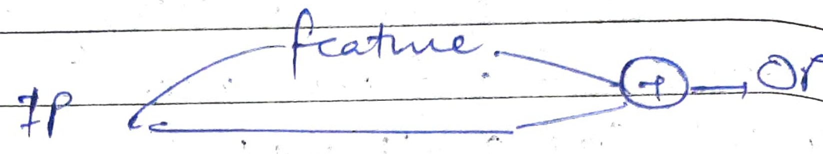
Takes ∇ of all batches and normalizes and stops from getting stuck due to small ∇.

Side branches in NN :

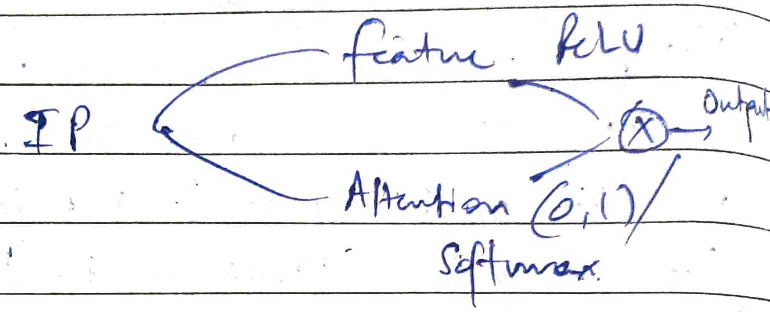
1. Parallel layers (Concatenation)



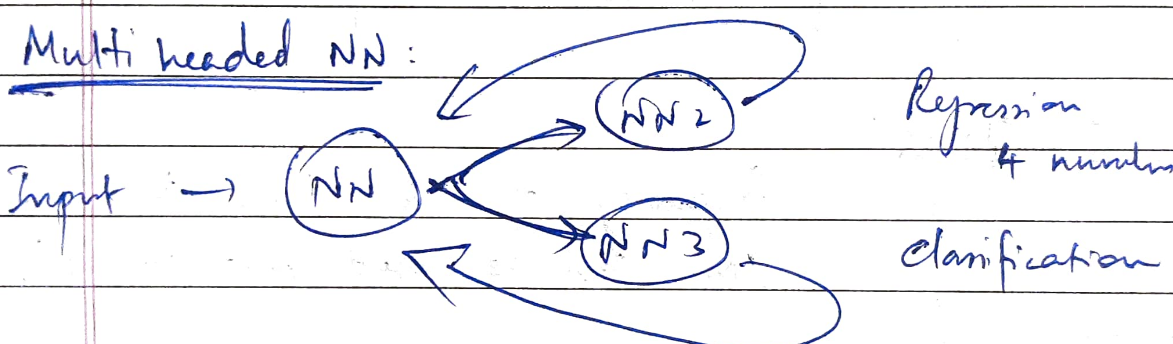
2. Residuals (Alternate path for ∇)



3. Basic attention



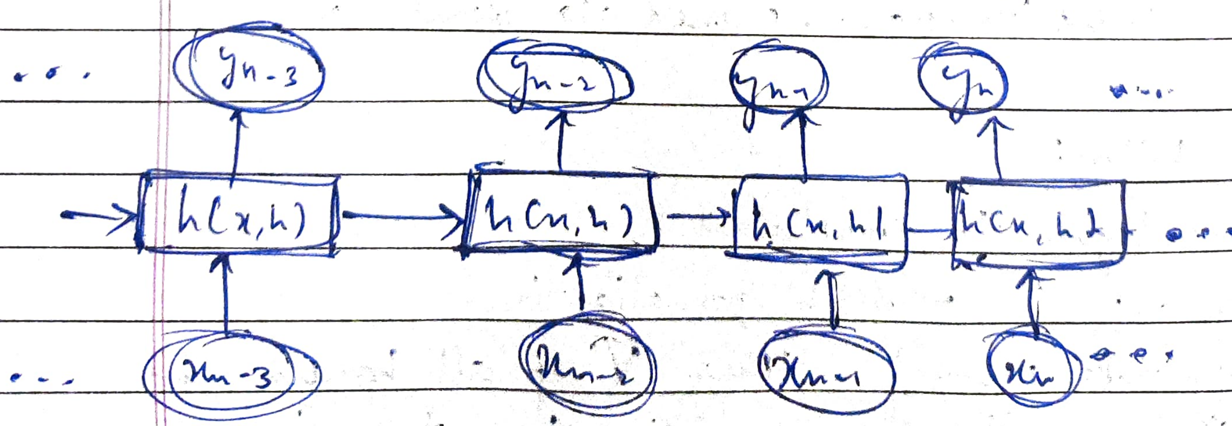
Multi headed NN :

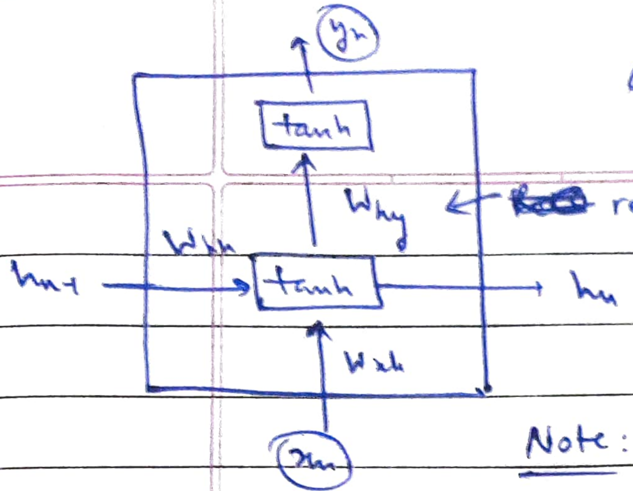


Introducing memory (recurrence or state) in neural networks

Hidden op sent forward into time (remembered state)

Recurrent NN :





One unit of block

Backpropagation: ∇ sent back (Switch all arrow dir^n)

Note: We need to limit the gradient steps back in time (cannot have ∞)

Limited for a window of 50-100 or something

Fwd: $y_n = g(W_2 h_n)$
$= g(W_2 f(W_{11} h_{n-1} + W_{12} x_n))$

Bckwd: $\dfrac{\partial y}{\partial W_{11}} = g' W_2 h_n' = g' W_2 f'(h_{n-1} + W_{11} h_{n-1}')$

W_{hh} large $\Rightarrow$ Exploding ∇ | W_{hh} small $\Rightarrow$ Vanishing ∇

Exploding & Vanishing ∇:

- Chain rule in backprop - Gradient requires mult of weights
(Careful weight initialization Xavier & He)

LSTM:-

- CEC is constant error carousel
  - No vanishing ∇
  - Always on
- Introducing gates
  - Allow or disallow input/output
  - Remember or forget state