#2 Learn AI - Deep Learning 101.

1. Convolutional NNs
2. RNNs
3. LSTM
4. Cross entrophy
5. Stochastic Gradient Descent
6. Autoencoder

CNNs

Works very well for image recognition & Pattern recognition

CNNs learn by extracting & learning spatial ~~use~~ features of image data

Steps:

1. convolutional layer (kernal)
2. Pooling layer
3. fully connected layer

kernel = Filter layer, elementwise, Sliding, Makes matrix smaller

Image → kernal → Feature map

Diff types of kernal can highlight Diff types of lines in an image - (horiz, vert, curved, etc)

kernel size ~~impacts~~ Dictates the size & Detail of the feature map ineresr

small kernal = bigger, more detailed feature map
→ but more computationally expensive

Stride = Slide of kernal

bigger strides = less detail & smaller Feature map

Convolutional layers introduce an activation func to handle non-linearity

→ ReLu most typical

Pooling layer

Job is to reduce the size of the feature map

Mains = Max Pooling or Avg Pooling

use kernals w/ no stride

Max = Select max in kernal

Avg = Average of all values in kernal

Dense / Connected layer

Creates final output based on extracted features

Conv → Pool → Conv → Pool → Flat → Dense

training of a convo involves changing the weights of the kernal in the convo layer(s) to reduce error of output probs

Backprop & GD used

Over time/training the kernal is optimized to distinguish between the local feat of X & Z

RNNs

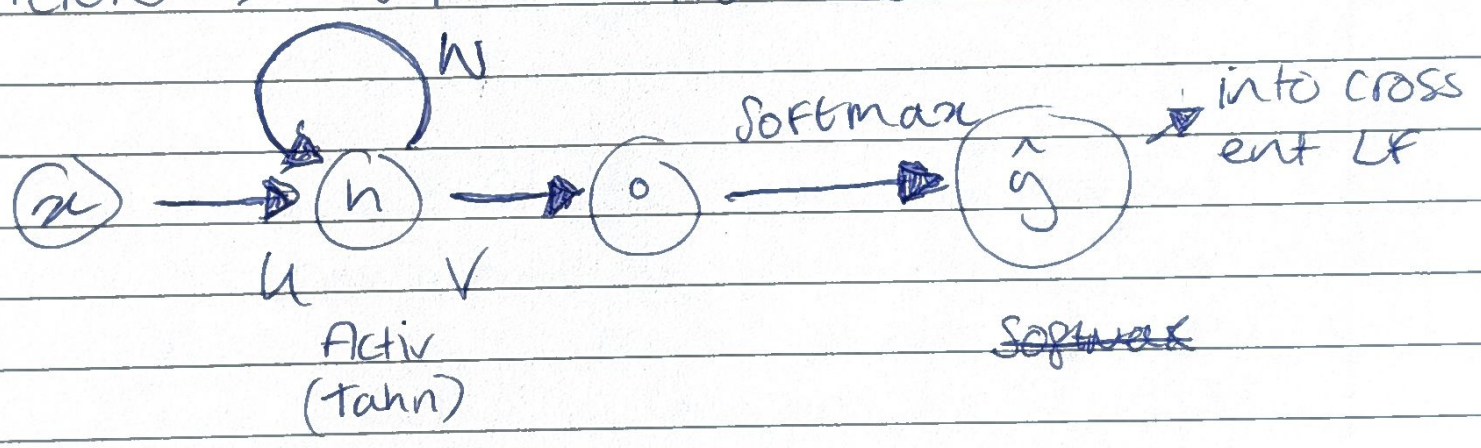
Process sequential data
- time series data
- continuous information

- Stock Prices
- Words/Sentances

if CNNs extract ~~temporal~~ spatial features from images

RNN extracts temporal features from time series data

RNN -> LSTM -> transformers



h_1 gets calced & passed into h_2

U, V, W are weights

LTSM

RNN weakness = longterm dependancy
the longer the time, the more chain rule
* calculations
  - if < 1 = diminish } early values also
  - if > 1 = explode } get lost

to overcome = LSTM
how? = Cell State
4 gates
  - Forget ~~gate~~
  - input ~~age~~ gate
  - Candidate gate
  - output gate

Forget gate = Decides what to forget
- takes input as prev state } concat
- At new/current state } matrices
- Sigmoid ($w \cdot$ concat), 0-1
  - meet cell state & updates element wise

Input gate
- same calc, $\sigma(w_t, x_t)$, diff weights

Card gate
- work w/ input, elementwise
- tahh(w, x) (-1, 1)

Add & update Cell State

C_t = long term memory

H_t = short term (Hidden)

two flows of information

Note: video has good example of setting up & initing weights to calc the feedforward (main)

H_t is derived from the C_t layer + the output/forget gate

Output = z_t comes from only H_t

$z_t = W \cdot H_t$ (raw logit output)

↳ (Softmax) → $\hat{y}_t$

C_t & H_t shape do not need to equal the shape of output vector

↳ the z_t weights will convert to the right size

5 sets of weights to Backprop & Grad Desc

- forget G
- Input G
- Candi G
- Output G
- z_t (raw output)

for each Section of the Model Seperate out the weights calc & activation for Back prop to take place

Softmax & cross ent are best pair as

$$\frac{\partial L}{\partial Z_t} = \hat{y} - y$$

Note: video has good breakdown of calc the derivatives w/ memorization caveats to consider

Cross-Entrophy loss

Frequently used loss func

$$H(p,q) = -\sum p(x) \log q(x)$$

Aspects of cross ent:

- Informatoin
- Expectation value
- Entrophy

information (suprise)

Prob & info are inversely related

$$P(x) \rightarrow 1/p(x)$$

in info theory the actual formula is =

- $\log_2\left(\frac{1}{p(x)}\right)$

In info theory, information is a measure of suprise / unexpectedness expressed as an objective number

Degree of suprise $\log\left(\frac{1}{p(x)}\right)$

or $\log(1) - \log(p(x))$

$0 - \log(p(x))$

$-\log(p(x))$

Expectation value

Skill level * probability = Expected Value
Objective

EV considers the objective data & the daily (?) variabs

EV = value Expected today

$$E(x) = \sum x \cdot P(x)$$

Entrophy

linked to EV & Suprise

take ev calc & replace w/ suprise

$$H(x) = \sum \log\left(\frac{1}{P(x)}\right) \cdot P(x)$$

"Expected Degree of suprise"

why & when to loose?
→ consider football example where both E(x) is the same for both teams
EV

then consider Entrophy
- if Prob is near 0 or 1 = Predictable
- near 0.5 = unpredictable

- More predictable ≠ more chance of win

it can be an important datapoint in det the outcome of the match

Entrophy is used to calc this type of predictability

More predictable = low entrophy

- high Ent = unexpected events occur frequently

Cross entrophy

$\sum \log\left(\frac{1}{p(x)}\right) \cdot p(x)$ = entrophy

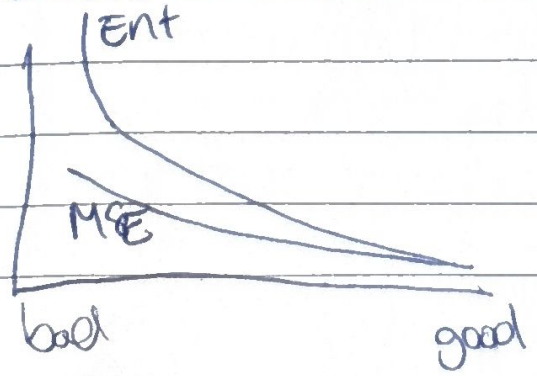
→ cross ent change p to q

$q(x)$ = Suprise of x

Multiply Suprise of x by prob of x

Why X-ent better than MSE?

Shows loss better than MSE



@ bad predict the gradients are much steeper

better for learning

Stochaistic Gradient Descent

3 types:

- Batch
- Stochastic
- Mini-batch

NNs inherently have the local minima problem

→ no true solution but these 3 are methods explored

known as data scheduling for training

Batch = use all data points to calc the derives ~~for the loss func~~ for a singe weight update

~~Adat~~ updates in stable ~~many~~ manor but is very inefficeint if many data points

Stochastic = Calc error ~~For~~ just 1 point & update the weight

- More efficent
- Very sensitive to each point
  ↳ increase local minima problem

Batch is more stable & less prone to local minima

Mini Batch

uses set num of data points for each datapoint

- → More stable than Stoch
- → Mac efficient than Batch

Note: has a code example