ML - Week 8 - Advanced NN's

- Convolutional NNs (CNN)
- Recurrent NN (RNN)

CNNs

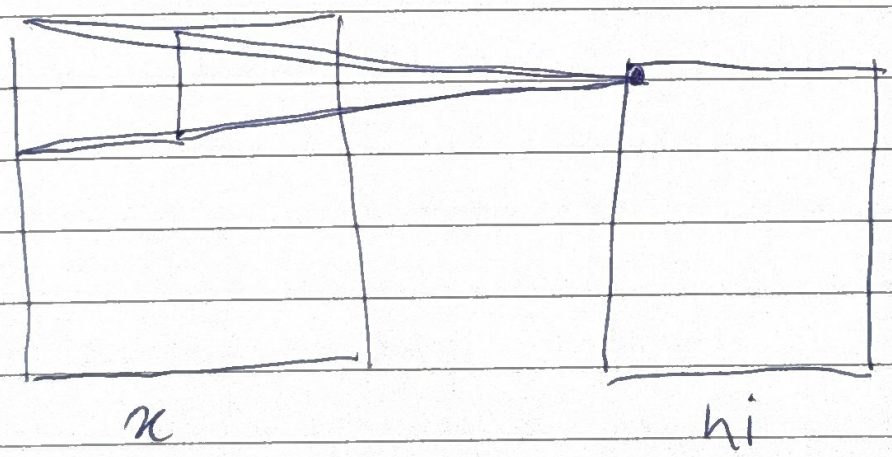
Continues the general concept of layers

Sections of the layers predict sections in the next layer

CNN created for ~~grid~~ structure data

learns ~~spta~~ spatial relationships

e.g image -> neighbouring pixels are more related than distant pixels



Square -> Point

Compared to MLP layer:

MLP = unit -> unit in the layers

Convolutio small weight matrix applied

- Convolutional has a neighbour structure
- MLP has unit to unit structure

Colour has 3 channels → RGB

Image = matrix array → colours = 3-D matrix

With matrix = the rectangle

~~Ae~~ input = width matrix, output = single pixel

With matrix = height, width, channel (colour, shade etc)
→ should be significantly smaller than input data (image) h, w, c

weights are called a kernal in CNN layers

~~each output~~ Does each Pixel have a unique kernal??

Stride = how much the width matrix moves

Padding → Allows better preds @ edge of image

Input data example: w x h x c = 7 x 7 x 3

each channel has a weight matrix

Activation funcs & CNNs:
- ReLu common
- Alternate layers between convolution & "Pooling"
- Pooling = aggregation to data
- lower size of image

Pooling allows invariance to minor transforms

MLP all dimensions are interconnected

CNNs uses smaller weights

Week8 lab = • Uk met data
• Apply convo to just 1 Dimension

Question: is activation (relu) applied to pixels?

the use of CNNs is very expensive and reqs GPUs

2D CNN is the standard for image data

↳ 3D for video where 3rd is time

Summary

Primary feature of weight sharing of CNNs

weight sharing reduces overfitting

hyperpars are kernel, stride, padding & channel

Reccurent Neural Networks (RNN)

- MCP has no connections within the same layer
- only between layers

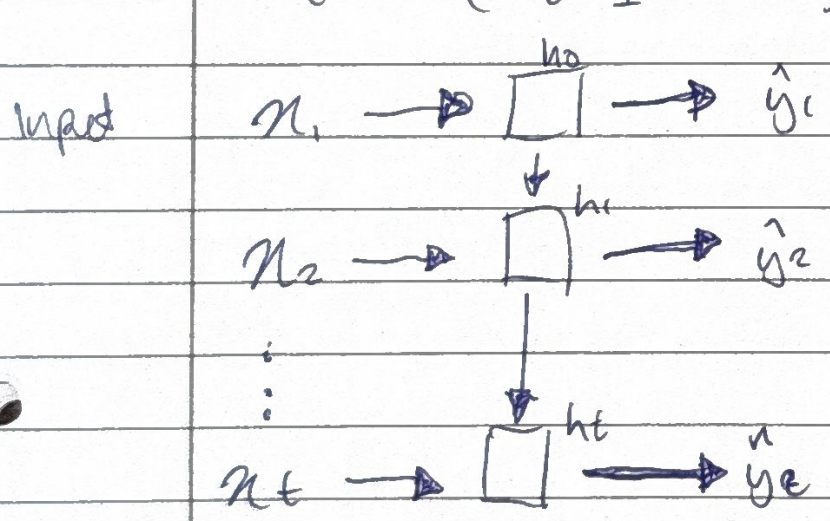
RNNs have connections between layers

this connection is feedforward

Designed for sequential data

reccurance function is always the same

$h_t = f(h_{t-1}, x_t, W)$



many-to-many mapping

~~video is a good ex~~

A common example is video due to time aspect

Another is prediction of the next sequence of words given an input sequence of words

RNN can be rechured to many-to-one in order to just predict the $\hat{y}_t$

or one-to-many, e.g. image ~~labelled~~ / summarise

Problem w/ RNN:

Vanish gradient

Recall Stotastic gradient Descent

in MLP, gradient is Backpropiageted ~~to the~~ from the loss func to the input & update

RNN w/ time data = Backprop through time

Problem is when you go further enough back in time, the gradient becomes very small & ~~also 0~~ almost 0

this means the model stops learning

long short-term memory (LSTM)

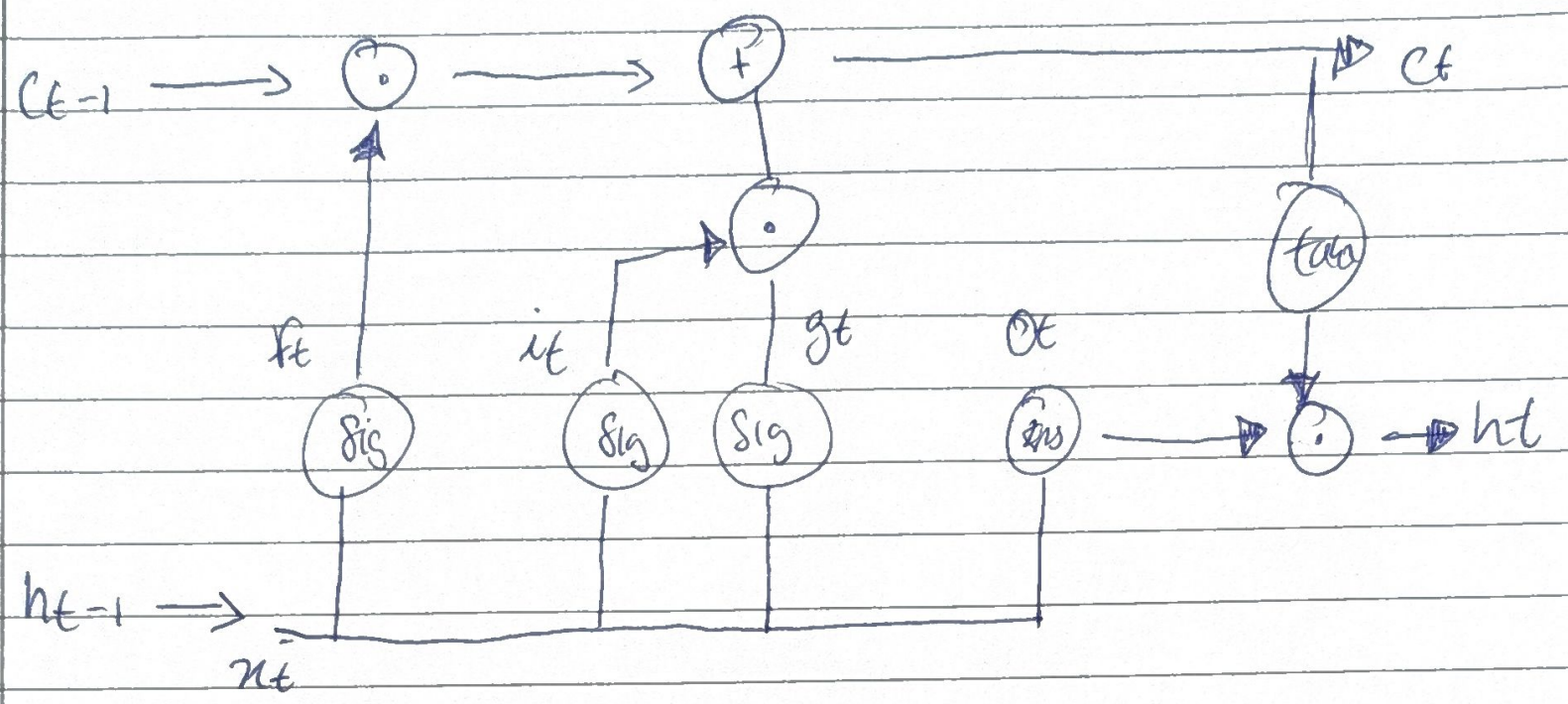
As a solution to vanishing gradients

before: $h_t = f(h_{t-1}, x_t, W)$

After: $h_t = f(c_t, o_t)$

uses gates to control flow through time

$c_t = f^*(c_{t-1}, f_t) + f^*(i_t, g_t)$



Back

forget gate = f_t = ~~$f_t(h_{t-1}, x_t, w_t)$~~ $f_t(h_{t-1}, x_t, w_f)$

input gate = i_t = $f_i(h_{t-1}, x_t, w_i)$

Cell update = g_t = $f_c(h_{t-1}, x_t, w_c)$

output gate = O_t = $F_o(h_{t-1}, x_t, w_o)$

How does it does it correct vanish grads?

- LTSM has blocks (usually 1 but can be multi)
- each Block has multi-cells
- cells in Block share gates

Error flow in LSTM:

- Backprop within cells & Blocks
- No Backprop through time

↳ i.e Problem addressed