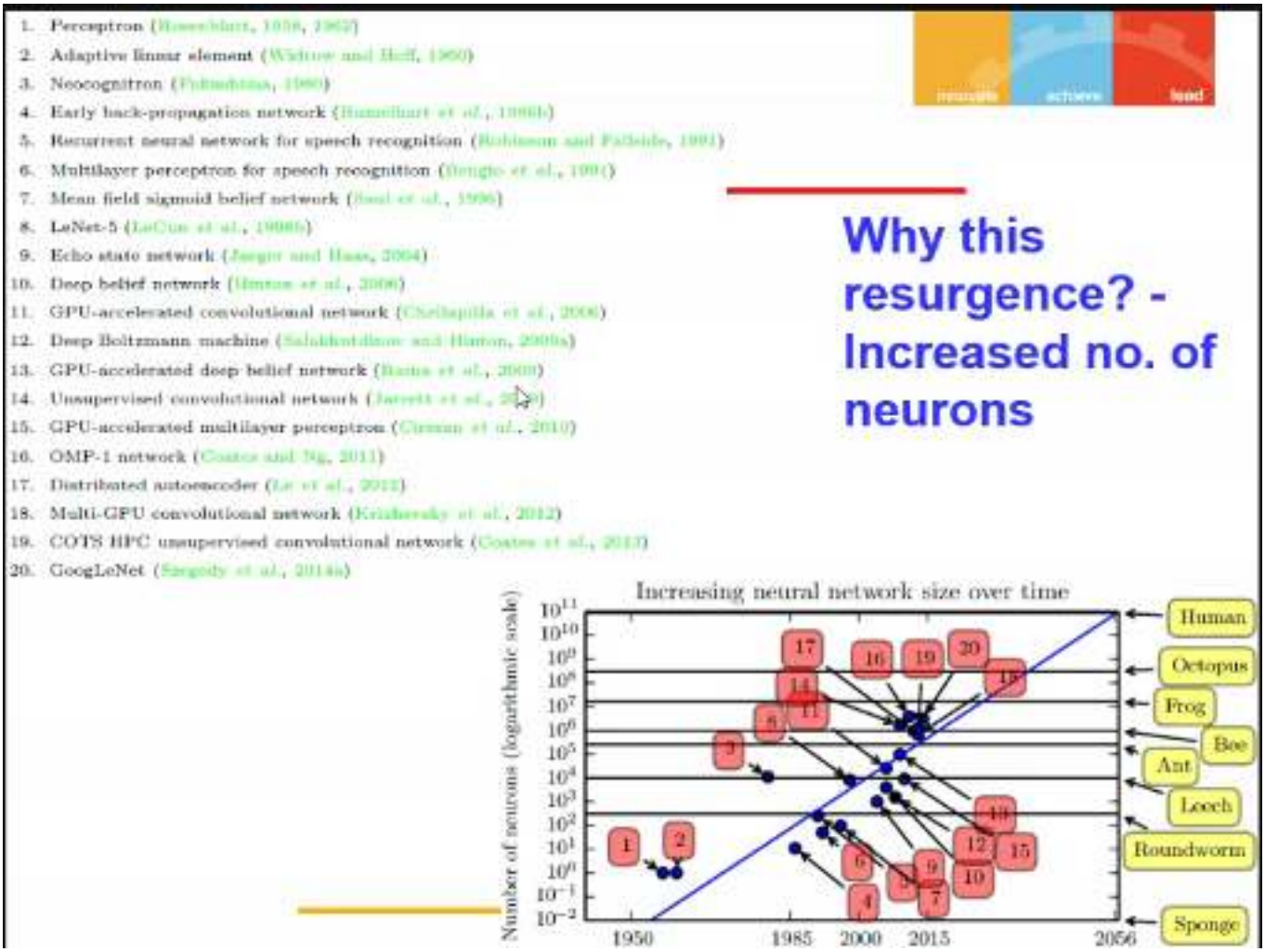
L2 - I guess(?)

- Deep learning can help discern / disentangle the accent from the voice and get just the pronunciation, through having layers for representing the accent
- Three historical waves
  - Cybernetics (1970s)
  - Connectionism/ Neural networks (peaked in 1995)
  - Deep learning (2006+, layerwise training, big data)
- Hardware didn't increase immensely until 2015, so that's why they were able to increase suddenly only then
  - so we will need a immense development in hardware for a machine to reach the number of neurons in humans. And even then it may not think just as well.
  - DL is *not* simulating a brain. It is inspired from the brain, not immitation



- number of connections per neuron for a human is 10^4, and we have reached that in a COTS HPC unsupervised CNN, but most of our models are still nowhere in comparison. Perhaps we don't even want to simulate the brain.

One learning algorithm Hypothesis

- Most perception (input processing) in the brain may be due to one learning algorithm so even if the vision cortex is damaged, the auditory cortex is forced to learn and can make up for it

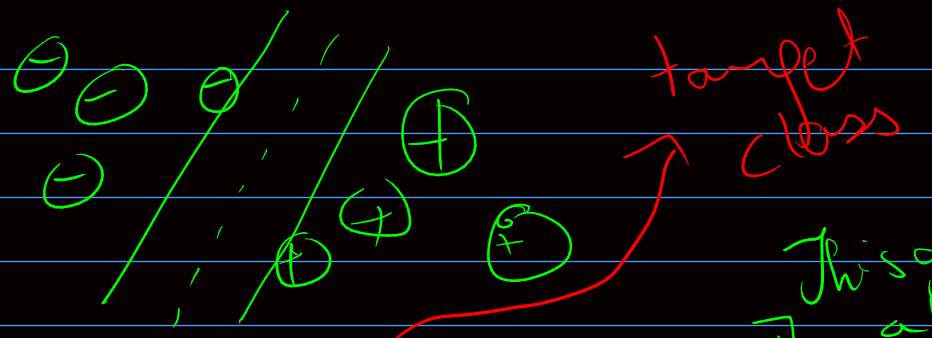
L3

- Vision Cortex
- Auditory Cortex
- Somatosensory cortex
  these guys can learn other senses when their primary functions are severed
  - So there may be just one algo behind this
- Perception based algorithms
  - Eg. recognising text and stuff
- focus onto the given problem is going down, but it's not
  - $\implies$ We can't say much about DL
  - We need to tune our DL algorithm, but we can't do a one-for-all algo

Pack learning , Bounce of learning etc.
No free lunch theorem

- averaging over all possible data generation distributions, Every classification algorithm has the same error rate, classifying previously unobserved points.
  - you should not look for a universal learning algorithm
- There is a proof

DL - did better than SVM for handwriting recognition
TL;DR - DL is kinda scam term only for nicer NN



$$y(x) = \sum_{i=1}^{n} a_i t_i k(x, x_i) + b$$

This gives a plane

finding the decision boundary (hyperplane)

is a function defined on x-testing with training example

you need to solve

$$\max \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j k_i k_j \, k(x_i, x_j)$$

$$s.t. \; a_i \geq 0$$

$$\sum_{i=1}^{m} a_i t_i = 0$$

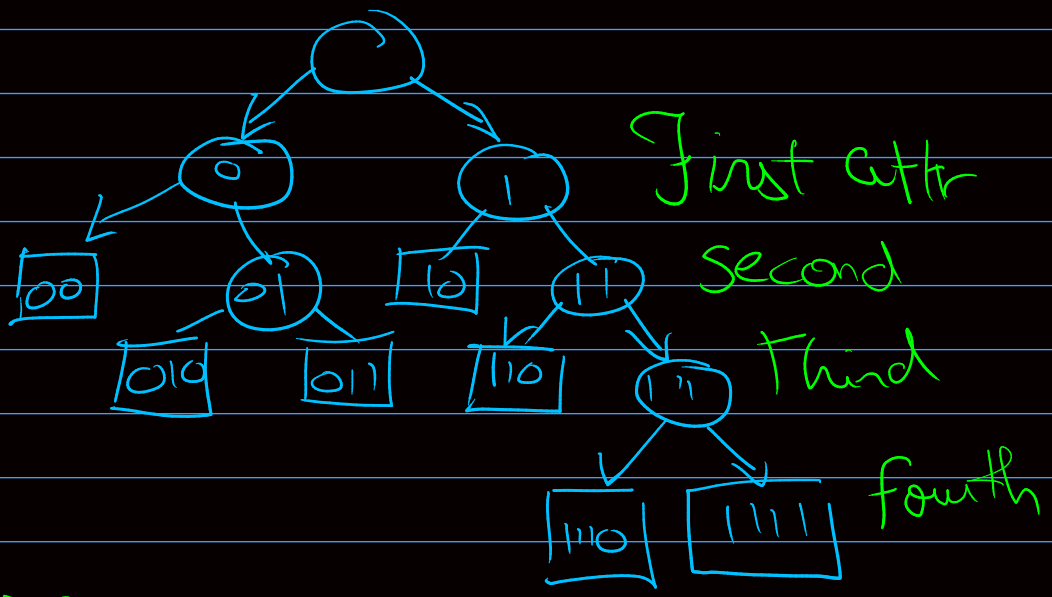
$$k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

$$b = \frac{1}{N_s} \sum_{i \in S} \left(t_i - \sum_{j \in z} a_j k_j (x_j, x_i) \right)$$

$e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$ k can be any famous kernel

disadvantage: Too many training examples - If we need to compute $O(n^2)$ (n = no of training examples)

Decision Tree

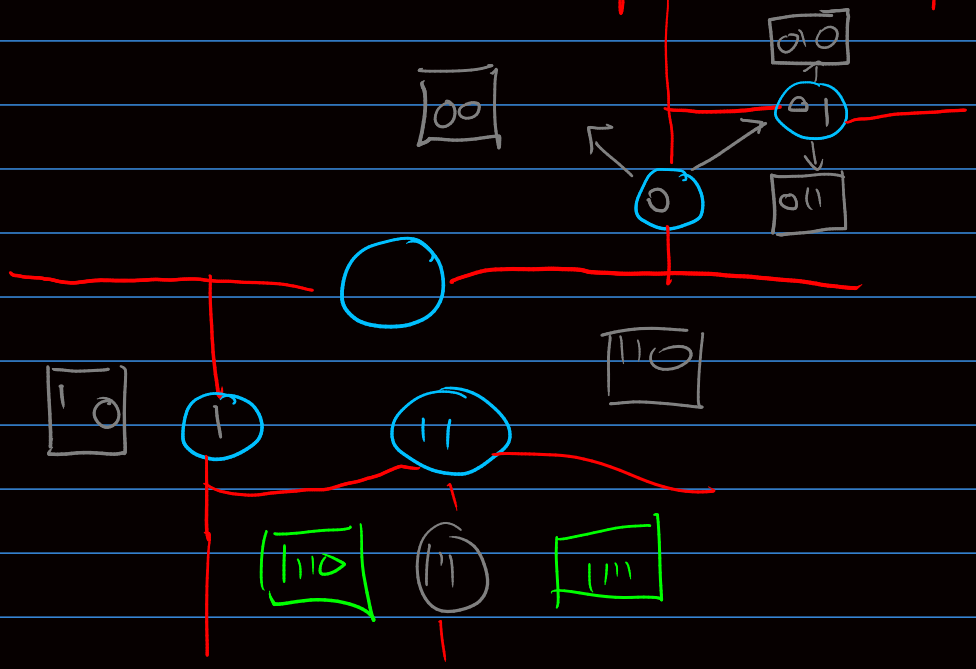


→ Considering a full blown tree, no overfitting

2000 training examples ⇒ 2000 leaves

to look at the decision boundaries

~~Or~~ divide the 2-D space into 2 parts



⇒ In this case, the max features a decision tree can take is based on the number of training examples.

⇒ In reality there will be exp. no of leaves

⇒ Decision tree doesn't work

→ you lose features

Curse of Dimensionality:

Imagine 1 attr taking 10 values

You see the samples for each 10

So now You need atleast 10 samples

if there are 2 attr. ⇒ 10x10 = 100 samples

3 attr = 1000 samples

30 attr ⇒ 10^{30} samples

⇒ Thats why local constancy assumption

for a good number of classical ML

the curse of dimensionality

holds

L7

Universal approximation theorem

$f : R^d \rightarrow R$

- Can be represented as a network with only one hidden layer
- There will exist an activation function
  - The problem is that this theorem states that there exists the function, and doesn't give the neural net itself
- also in any machine learning problem you don't even know the function

How many number of hidden units must be there to figure out the function?

- It is exponential (takes too long to understand)
- The weight matrix becomes too large to compute

l layers

- $O(\binom{n}{d}^{d(l-1)} n^d)$ decision regions, for d features, and n neurons per hidden later
- if l >=5 then it's called as a deep neural network

Linear Models

- Parameters are linear
  - Eg. Support vector machines
  - or use a linear activation also
- Classical ML can do nonlinear shit too btw.

if t is target
ϕ is transform
{ ($\phi(x1)$,t1), ($\phi(x2)$,t2), ($\phi(x3)$,t3).... ($\phi(xd)$,td) }
Phi can be exponential
we can use a linear model on top of this

How to make the raw features to matured features?

1. the specific basis functions can be applied by the domain expert, to get the matured featues, and then apply the linear model
   - Example, BMI
2. Kernels in SVMs, going with some kind of functions to transform. Eg. Gaussian Kernel

$(x^{(1)}, t_1)\ (x^{(2)}, t_2)\ \dots\ (x^{(n)}, t_n)$

$x = (x_1, x_2, \dots, x_d)$

$y(x, w)$

$y = w_1\phi_1(x) + w_2\phi_2(x) + \dots + w_m\phi_m(x) + b$

$y = w_1 x_1 + w_2 x_2 \dots w_m x_m$

y is linear in ϕ (matured features)

but not in x

SVM

$x_1\ x_2 \dots x_d$ $\quad k(x^i, x^j) = e^{-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}}$

$\underbrace{\phi_1\ \phi_2 \dots \phi_m}$

you may not know

Using SVMs with a kernel → each vector is a support vector ⇒ overfit

In SVMs, we fix our features

DL → we get a ton of features

→ we dunno how many are appropriate

Problem is, architecture is mostly trial & error

$x_{i1}, x_{i2} \cdots x_{id}$, apply some nonlinear functions $[\phi_1, \phi_2 \cdots \phi_m]$ to the vector

→ What we get is a matured feature vector
it may be a linear model of matured features
but non linear model of original features
if the basis functions $[\phi_1 \cdots \phi_m]$ are non linear

→

Feed forward Networks

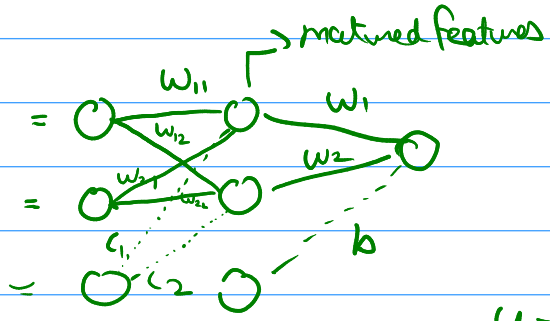
x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0

4 examples

$$\sum_{i=1}^{4} \left((w_1 x_1^{(i)} + w_2 x_2^{(i)} + b) - t_i \right)^2$$

Linear regression != neural network

$w_1 = 0 \quad w_2 = 0 \neq b = 0.5$



$$h = f^{(1)}(x; W, c)$$

$$y = f^{(2)}(h, w, b) = w^T h + b$$

$$y = f^{(2)} f^{(1)}(x; W, c, w, b)$$

What will be $f^{(1)}$?

L.T. $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \in \mathbb{R}^2$ to $\mathbb{R}^3$ can be $\begin{bmatrix} f_1\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) \\ f_2\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) \\ f_3\left(\begin{bmatrix} 2 \\ 1 \end{bmatrix}\right) \end{bmatrix}$

$$\begin{bmatrix} 3 & 1 & 5 \\ 4 & 2 & 6 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 10 \\ 14 \\ 16 \end{bmatrix}$$

$f_1 = 3x_1 + 4x_2$

$f_2 = x_1 + 2x_2$

$f_3 = 5x_1 + 6x_2$

} linear combination $\Rightarrow$ linear transformation

Affine transformation

Affine transform: Linear transformation + constant vector added to result

$$f^{(1)}(x; W, c) = \underbrace{W^T x + b}_{\text{Affine transform}}$$ } matured space

For brevity (and to make life easier), let us just consider a linear transformation

$$y = \omega^T (W^T x)$$
$$= (\omega')^T x \quad \text{where } (\omega')^T = \omega^T W^T$$

$$\sum_{i=1}^{4} (\omega^T x - t_i)^2$$

SO, The resultant matured features of a linear transformation is a linear model. (which can't fit a non linear function like XOR)

So as a result of this problem, all of deep learning and Some machine learning algorithms use something else on top of it

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix}^T \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} w_{11} + w_{21} + c_1 \\ w_{12} + w_{22} + c_2 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

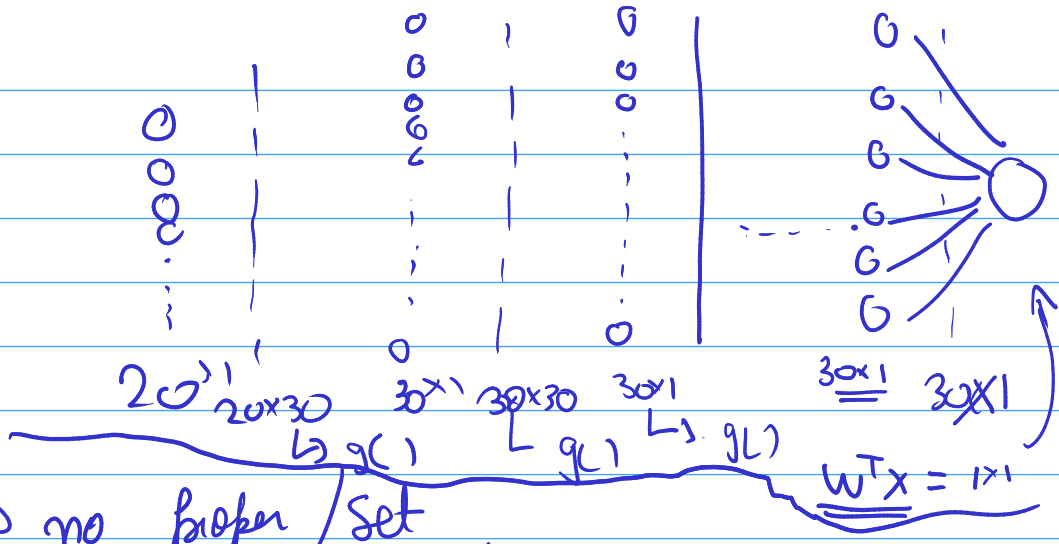
Add a non linear function

$$= \begin{bmatrix} g(w_{11} + w_{21} + c_1) \\ g(w_{12} + w_{22} + c_2) \end{bmatrix}$$

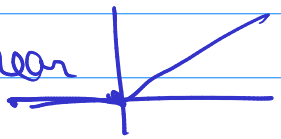
one of them could be

$g(z) = \max(z, 0)$ ⇒ This function is an activation function

applied to every hidden layer

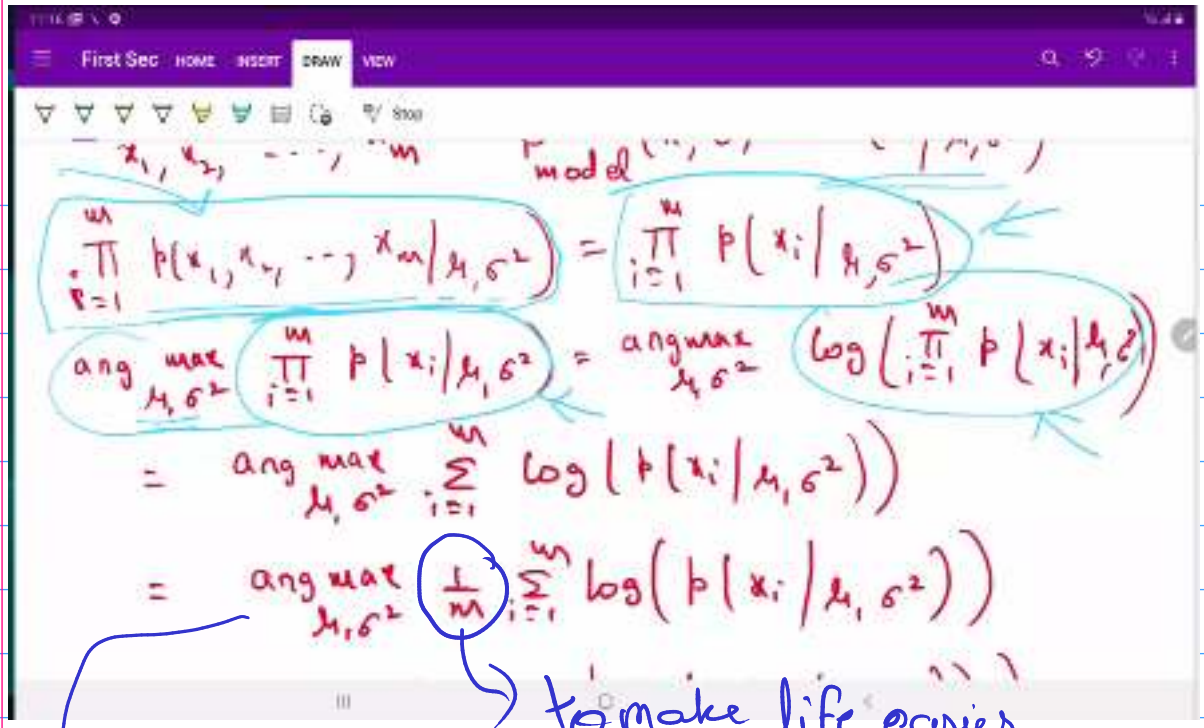


Theres no proper / set "algorithm" to choose an activation function, You have to experiments/make a judgement to choose

RELU: piecewise linear  but not actually linear → nonlinear

→ It gives nonlinear properties, and can be used as activation
⇒ but you need to do the math involved.
→ It even has some biological motivation.

Similarly, there is not set algo for Number of layers
we will need insights for the domain/ experimentation if the above insights are not there.



$X = x$ discrete

$$E(f(x)) = \sum_{i=1}^{m} f(x_i) P(X = x_i)$$

Lognaml distribution $X \sim \hat{P}_{data}$

$$\frac{1}{m} \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x_i-\mu)^2}{2\sigma^2}}\right)$$

$$= \frac{1}{m} m \log\frac{1}{\sqrt{2\pi}\sigma} + \frac{1}{m} \sum_{i=1}^{m} \left((-1)\frac{(x_i-\mu)^2}{2\sigma^2}\right)$$

$$= \log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{m} \sum_{i=1}^{m} \frac{(x-\mu)^2}{2\sigma^2}$$

$P_{data}(x)$ — actual distribution

$P(x;\theta)$ — model

$\hat{P}_{data}(x)$ (discrete)

$p_{model}(x;\theta)$ $\quad \hat{p}_{data}(x)$
(discrete)

$D_{KL}(\hat{p}_{data} \| p_{model})$ shows how dissimilar these are.

$$E_{x\sim\hat{p}_{data}}\left[\log \hat{p}_{data}(x) - \log p_{model}(x)\right]$$

$$= E_{x\sim\hat{p}_{data}} \log\left(\hat{p}_{data}(x)\right) - E_{x\sim\hat{p}_{data}}\left(\log p_{model}(x)\right)$$

↳ "from" / follows

$$= \sum_{i=1}^{m} \frac{1}{m} \log\left(\frac{1}{m}\right) - \cdots$$

↳ given (there are m things you can pick on)

$$= -\log m - \cdots$$

$$= \arg\min E\left[\frac{-1}{\sqrt{2\pi}\sigma} + \frac{1}{m}\sum_{i=1}^{m}\left(\frac{x_i-\mu}{2\sigma^2}\right)\right]$$

KL divergence is called cross entropy (where the points taken are the same as the data on sth)

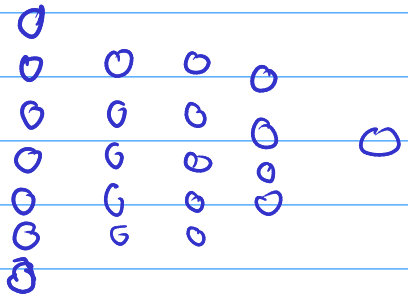
Cross entropy $(\hat{p}_{data}, \hat{p}_{model}(x;\mu,\sigma^2))$

$$= -E_{x\sim\hat{p}_{data}}\left(\log p_{model}(x)\right)$$

Whenever you are finding the parameters by MLE, the error function to that is the result of the likelihood function, you call the error function as a "cross entropy" function

This applies to regression as well! if MSE so can be got from a likelihood function too, it's also a cross entropy error!

Deep learning fixes the family of functions and uses that family!



$$p(y_1, y_2 \cdots y_m | x_1 \cdots x_m) = \prod_{i=1}^{m} P(y_i/x_i)$$

$$\max \log \left(\prod_{i=1}^{m} P(y_i/x_i) \right)$$

~~$\min -\log$~~ $\min \sum_{i=1}^{m} -y \log P(y_i/x_i)$

$$y(\theta) = -\log p(y/x)$$

$$\downarrow w_1 w_2 \cdots = -\log \sigma((2y-1)z)$$

$$= \zeta(-(2y-1)z)$$

not activation — Softplus — cost fun

$$= \zeta((1-2y)z)$$

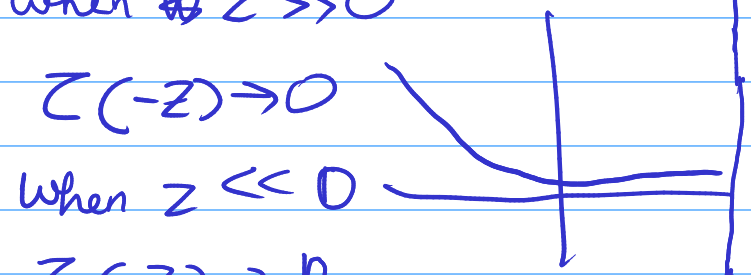
@ $y=1$ $\zeta((1-2y)z) = \zeta(-z)$

when $z \gg 0$

$\zeta(-z) \to 0$

when $z \ll 0$

$\zeta(-z) \to$ large

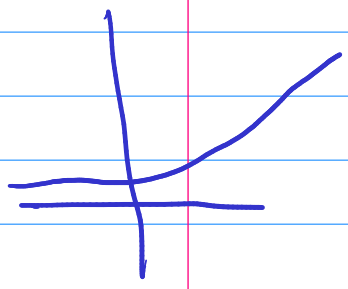


@ $y=0$ $z \gg 0$

$\zeta(z) \Rightarrow$ large ($w^{(k+1)}$ changes quite a lot) ▽ also large

$z \ll 0$

$\zeta(z) \to 0$ w's wont change



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

$$\frac{d}{dz}\sigma(z) = \sigma(1-\sigma)$$

$$\frac{\partial y(\theta)}{\partial z} = \sigma((1-2y)z)(1-2y)$$

(Take 1 example, treat as SGD)

Why is this cost function any better than MSE

Pf

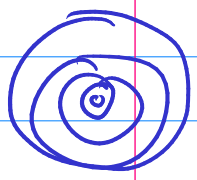
$$\max \log P(y_1, y_2 \cdots y_m \mid x_1, x_2 \cdots x_m)$$

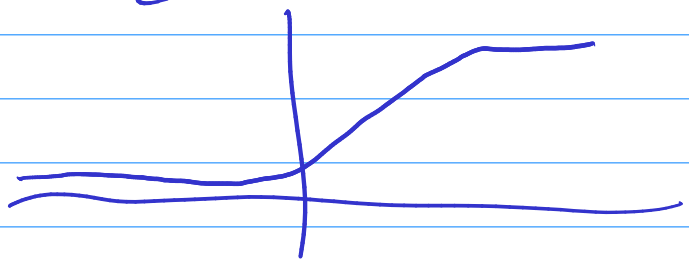
$$= \max \log \prod_{i=1}^{m} P(y_i \mid x_i)$$

$$= \max \log \prod_{i} \sigma(z_i)^{y_i} (1-\sigma(z_i))^{1-y_i}$$

$$= \max \sum_i \left(y_i \log \sigma(z_i) + (1-y_i) \log (1-\sigma(z_i)) \right)$$

$$= \min \left[\sum_i -\left(y_i \log \sigma(z_i) + (1-y_i) \log (1-\sigma(z_i)) \right) \right]$$





$$= \min -[y \log \sigma(z) + (1-y) \log (1-\sigma(z))]$$

Activation

Leaky Relu
PRELU
Maxout

RBF
Soft plus
Hard tail
sigmoid

$$\min_{w,b} L(w, b, x^{(i)} t^{(i)}) =$$

$$w^{(k+1)} = w^k - \eta \left(\frac{dL}{dw}\right)_{w_1 = w_1^{(k)}}$$

$$\left(\frac{dL}{dw_1} \quad \frac{dL}{dw_2} \quad \cdots \quad \frac{dL}{dw_{10000}}\right)$$

Let $z = f(y)$

$$\frac{\partial z}{\partial y} = \frac{\partial f(y)}{\partial y}$$

$$\frac{\partial z}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \cdot \frac{\partial (g(x))}{\partial x}$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \cdot \frac{dy_j}{\partial x_i}$$

$$\nabla_x z = \frac{\partial z}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & & \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix}^T \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \vdots \\ \frac{\partial z}{\partial y_m} \end{bmatrix} = \left[\frac{\partial y}{\partial x}\right]^T \nabla_y z$$

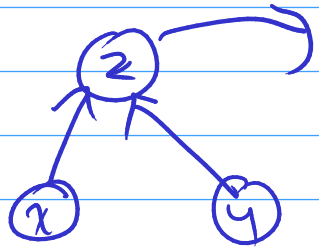
If x is not a vector, but of higher order (matrix or sth), then people see x as a vector as a whole

→ write the matrix as follows and actually expand this internally

[This is brainfuck ah]

To ease us in this process, we use a computational graph, which is a discrete structure

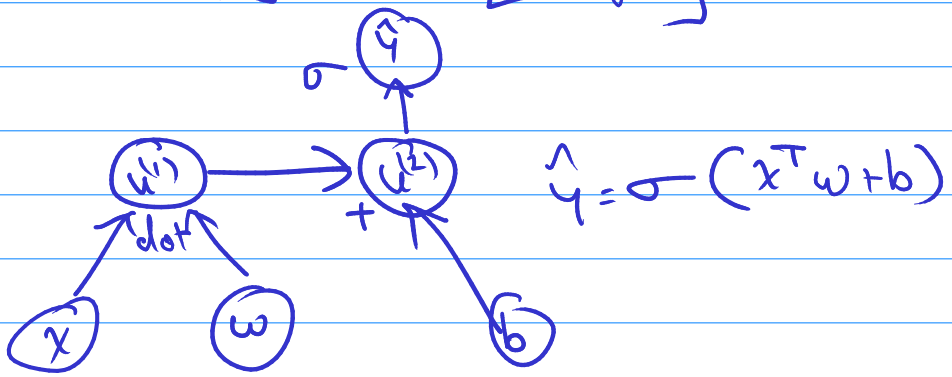
Ex Computational graph of xy



Operation = *, multiplication

$z = xy$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \quad w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

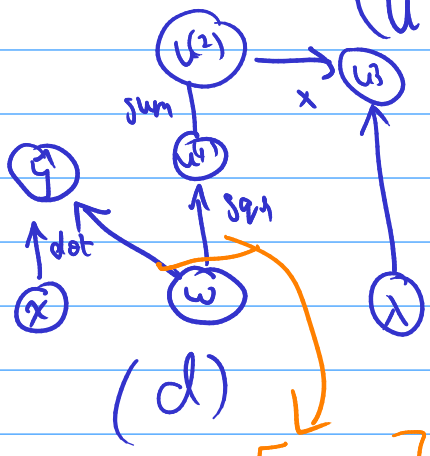


$\hat{y} = \sigma(x^T w + b)$

Two operations on 1 variable

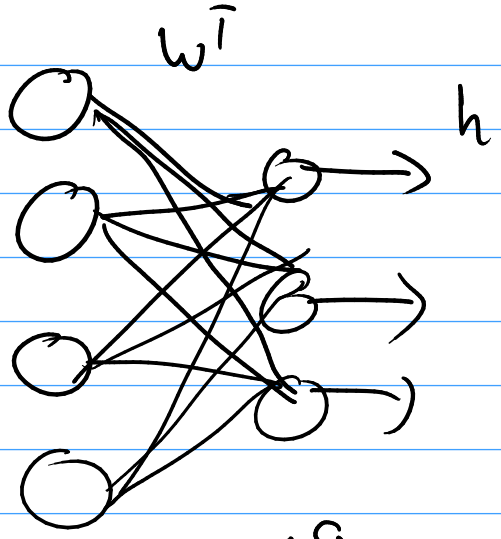
Weight decay (i) $\hat{y}$

(ii) weight decay penalty

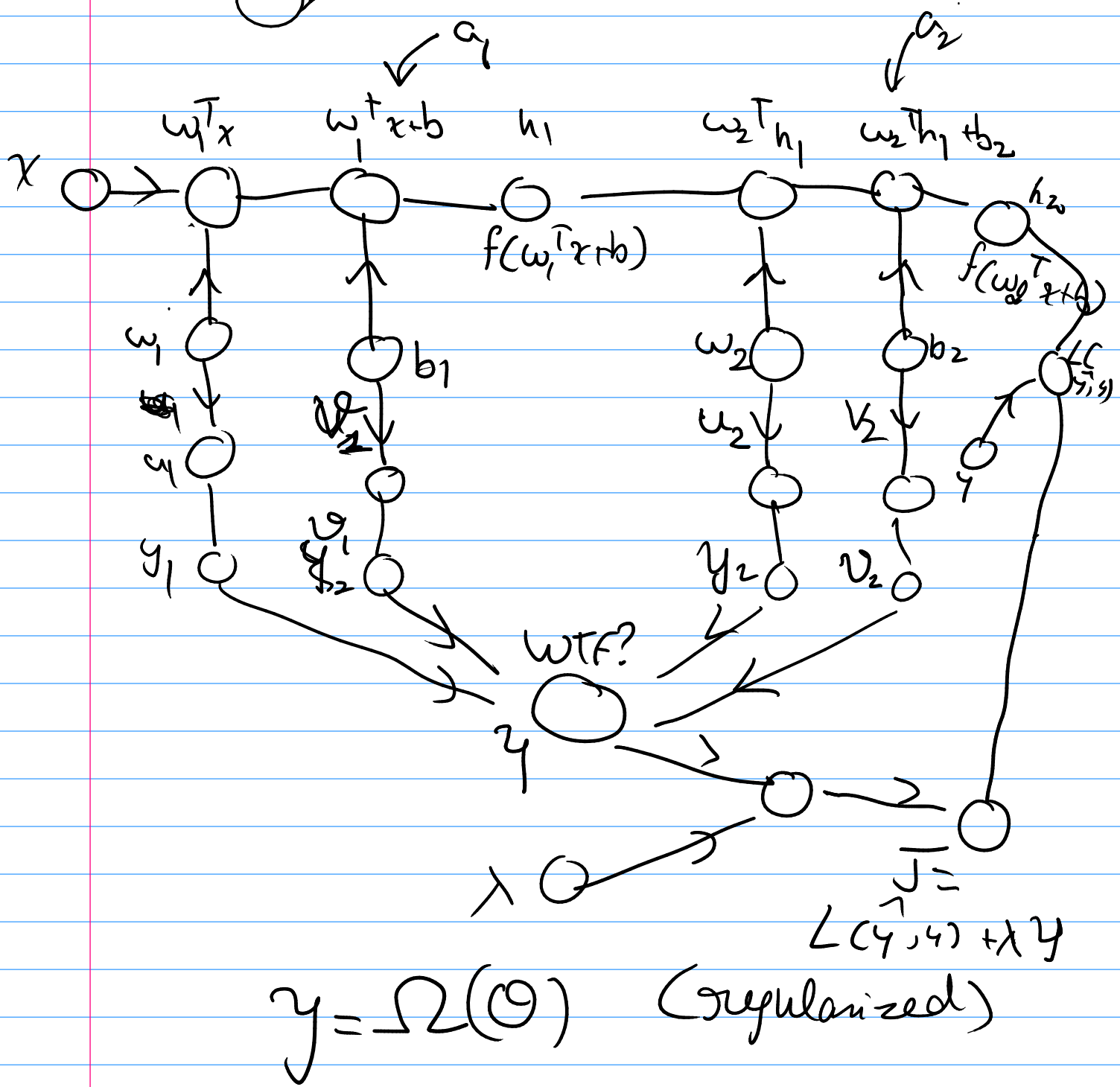


(d)

[two] connections



$f(W^T$



$J = L(y, \hat{y}) + \lambda \Omega(\theta)$ (regularized)

$$\min L(y, \hat{y}) + \lambda \Omega(\Theta)$$

$$\Downarrow$$

$$w, b$$

$$\Omega(\Theta) = w_1^2 + w_2^2 + \cdots + w_{10}^2 + w_{\Theta_n}^2$$

$$\begin{matrix} \nabla_{w_1} J & \nabla_{b_1} J \\ \nabla_{w_2} J & \nabla_{b_2} J \\ \nabla_{w_3} J & \nabla_{b_3} J \end{matrix}$$

a matrix of weights can be treated as a vector of vectors

How can we find the derivative of a scalar with a vector

3 layered NN

$$\nabla_{a_3}(L(\hat{y}, y)) = \nabla_{a_3}(h_3) \nabla_{h_3}(L(\hat{y}, y)) \quad (1)$$

$$= \nabla_{a_3}(f(a_3)) \odot g = f'(a_3) \odot g$$

$\odot$ → element-wise mul

$$\frac{\partial L}{\partial a_3} = \frac{\partial L}{\partial h_3} \cdot \frac{\partial h_3}{\partial a_3}$$

$$\nabla_{w_3}(J) = \nabla_{w_3}(L(\hat{y}, y) + \lambda \Omega(\Theta))$$

$$= \nabla_{w_3}(L(\hat{y}, y)) + \lambda \nabla_{w_3}(\Omega(\theta))$$

$$= \nabla_{a_3}(L(\hat{y}, y)) \nabla_{w_3}(a_3)$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial w_3} + \lambda \nabla_{w_3}(\Omega(\theta))$$

$$\left[\nabla_{a_3}(L(\hat{y}, y)) = f'(a_3) \odot g \right] \quad \ldots (1)$$

$$f'(a_3) \odot g \; \nabla_{w_3}(w_3^T h_2 + b_3)$$

$$\left.\frac{\partial J}{\partial w_3}\right|_{w_3 = w_3^{(k)}}$$

Ans $\quad \frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial b_3} = f'(a_3) \odot g \; \nabla_{b_3}(w_3^T h + b_3)$

$\vdots$

keep applying Chain rule

$\nabla_{W_1}(J)$

$\nabla_{b_1}(J)$

$\nabla_{W_3} J = \nabla_{W_3}(L(\hat{y}, y) + \lambda \Omega(\theta))$

$= \nabla_{a_3}(L(\hat{y}, y)) \nabla_{W_3}(a_3) + \lambda \nabla_{W_3}(\Omega(\theta))$

$= \delta^{(3)} f'(a_3) h_2^T + \lambda \nabla_{W_3}(\Omega(\theta))$

$\nabla_{b_3} J =$

$\delta^{(3)} f'(a_3) 1$

$+ \nabla_{b_3}(\Omega(\theta))$

Don't worry

I moved the text to the

next page

$$\nabla_{h_2}(L(\hat{y},y)) = \nabla_{h_2} a_3 \nabla_{a_3} L(\hat{y},y)$$

$$= \nabla_{h_2}(w_3^T h_2 + b_3)(g \odot f'(a_3))$$

$$= w_3^T (g \odot f'(a_3))$$

$$g \leftarrow \nabla_{h_2}(L(\hat{y},y)) = w_3^T (g \odot f'(a_3))$$

$$\nabla_{a_2}(L(\hat{y},y)) = \nabla_{h_2}(L(\hat{y},y)) \nabla_{a_2}(h)$$

$$= g \nabla_{a_2}(f(a_2)) = g \cdot f'(a_2)$$

$$\nabla_{w_2} J = \nabla_{w_2}(L(\hat{y},y)) + \lambda w_2 (\Omega(\theta))_{(\text{layer})}^{2}$$

$$= \nabla_{a_2}(L(\hat{y},y)) \nabla_{w_2}(a_2) + \lambda \nabla_{w_2}(\Omega(\theta))$$

$$= (g \cdot f'(a_2))(\nabla_{w_2}(w_2^T h_1 + b_2)) + \lambda \nabla_{w_2}(\Omega(\theta))$$

$$\nabla_{b_2}(J) = \nabla_{b_2}(L(\hat{y},y) + \lambda \Omega(\theta))$$

$$= \nabla_{a_2}(L(\hat{y},y)) \nabla_{b_2}(a_2) + \nabla_{b_2}(\lambda \Omega(\theta))$$

$$= (g \cdot f'(a_2)) \nabla_{b_2}(w_2^T h_1 + b_2) + \nabla_{b_2}(\lambda \Omega(\theta))$$

$$= (g \cdot f'(a_2)) + \lambda \nabla_{b_2}(\Omega(\theta))$$

$$\nabla_{h_1}(L(\hat{y},y)) = \nabla_{a_2}(L(\hat{y},y)) \nabla_{h_1}(a_2)$$

$$= (g \cdot f'(a_2)) \nabla_{h_1}(w_2^T h_1 + b_1)$$

$$= (g \cdot f'(a_2)) w_2^T$$

$$g \leftarrow \nabla_h(L(\hat{y},y))$$

III $\nabla_{w_1}(J)$

$\nabla_{b_1}(J)$ can be found thru chain rule

in textbook he takes gradient of loss function with regularization

$g \leftarrow \nabla_{\hat{y}}(J)$ this is incorrect!

$$\nabla_{\hat{y}}(J) = \nabla_{\hat{y}}\left(L(y,\hat{y}) + \lambda\,\Omega(w,b)\right)$$

$$= \nabla_{\hat{y}}\left(L(y,\hat{y})\right) + \lambda\,\nabla_{\hat{y}}\left(\Omega(w,b)\right)$$

both $\hat{y}$ ←

& Ω are functions of w_3!!

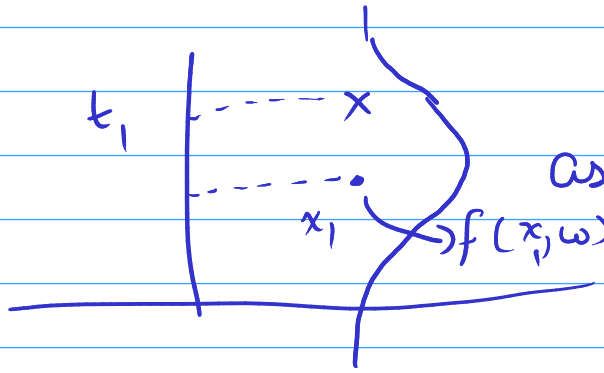
This gradient is not zero!

HOWEVER

You may say that TB considers regularization as part of the loss function i.e. $L(\hat{y}, y)$ has the regularized term inside

eg: $L(\hat{y}, y) = MSE + \lambda\,\Omega(\theta)$

$$y/x \sim N(y | f(x,w), \tilde{I})$$

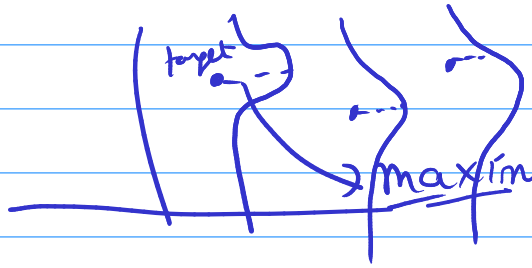


assume a normal distribution like this!

Consider covariance matrix to be I for now

now $f(X, w)$ is quite away from target

now lets have another w_2 such that



⇒ we need to find MLE (w)

Covariance matrix can also be determined

positive, semidefinite

you need a more complex neural network

What cost functions are good??

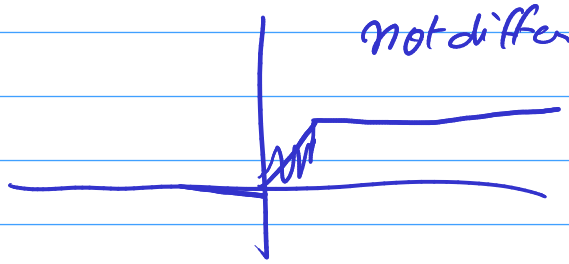
→ $P(y=1) = \sigma(w_0 + w_1 w + w_2 w \cdots + w_n w)$

what if

$p(y=1) = \max\{0, \min\{1, w^T h + b\}\}$

gradients are zero outside

not differentiable, also saturates



we need a better cost function !!

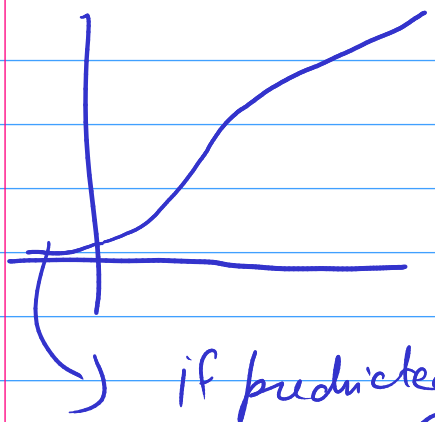
$-\log(P(\cdots))$ ...

$\Rightarrow$ we need a cost function that saturates @ correct classification & doesn't saturate @ incorrect classifications

$Y \mid X$	1	0
$P(Y \mid X)$	p	$1-p$

p is a bernoulli

Cost functions aren't forced, it's a consequence of an assumption of a normal distribution



if predicted output is low (-ve) but target is positive } Saturates

—(Z), if there is small change $Z \leq\!< 0$ then function won't change

i.e. $Z \to \infty$ or $Z \to -\infty$

and in an ~~it~~ interval, it changes a lot!

My error function should change when im not near a global or local minima

in SGD, since u're doing it example by example

you want to improvise vs drastically

⇒ Function shouldn't saturate

Deriving Cost

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

$$\frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$$

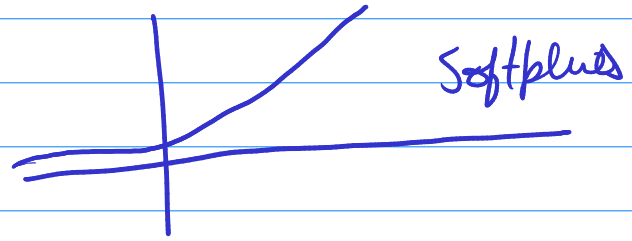
assume ← $\zeta(z) = \log(1+e^z)$ → softplus function

$x^+ = \max\{0, x\}$ Softer version of max function

$z \to +\infty \quad \log(1+e^z) \to z$

$z \to -\infty \quad \log(1+e^{-z}) \to 0$

very nice

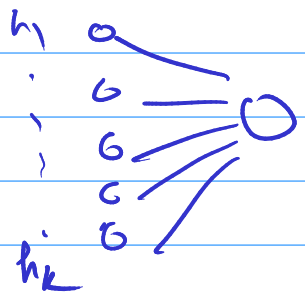


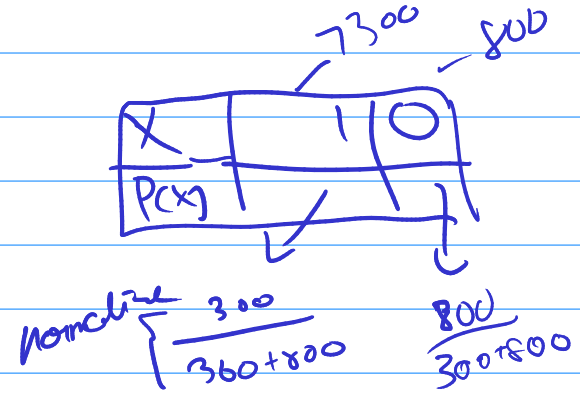
$$\zeta(1+e^{-z}) = \log\left(\frac{e^z+1}{e^z}\right) = \log\left(\frac{1}{\sigma(z)}\right)$$

$$= -\log(\sigma(z))$$

$$\log(\sigma(z)) = -\zeta(-z)$$

$$z = w_0 + w_1 h_1 + \cdots + w_k h_k$$





I want to combine z with target to arrive @ cost function that doesn't saturate

So they choose yz as that combination such that

$$\log(\hat{p}(y)) = yz = \hat{p}(y) = e^{yz}$$

$$p(y) = \frac{e^{yz}}{\sum_{y'} e^{y'z}} = \frac{e^{yz}}{1+e^{z}}$$

(Normalized)

(y takes values 0 & 1)

A "kind of" probability distribution

$$p(y=1) = \frac{e^{z}}{1+e^{z}} = \frac{1}{1+e^{-z}} \quad p(y=0) = \frac{1}{1+e^{z}}$$

$$\Rightarrow p(y) = \sigma((2y-1)z)$$

When $y=1$, $2y-1=1$, $y=0$, $\sigma(-z)$

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

Cost function of regular logistic regression also saturates

$$P(y_1 \cdots y_m | x_1 \cdots x_m) = \prod_{i=1}^{m} p(y_i/x_i)$$

$$\max \log\left(\prod_{i=1}^{m} p(y_i/x_i)\right)$$

$$\max \sum_{i=1}^{m} \log(p(y_i/x_i))$$

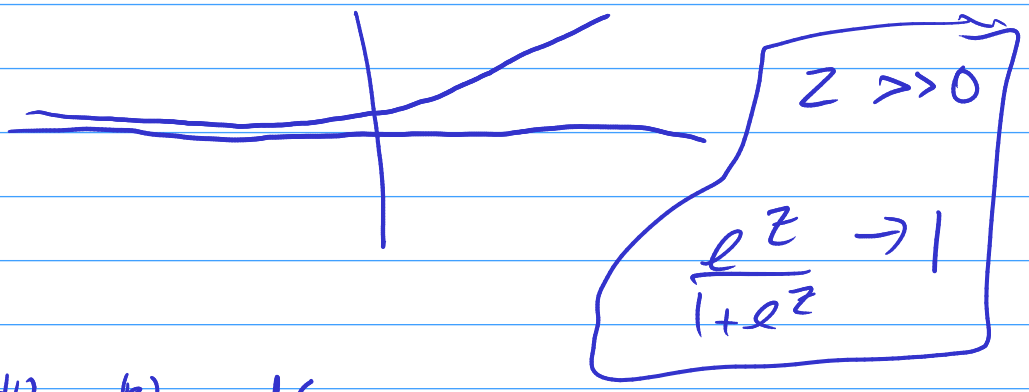
$$\min \sum_{i=1}^{m} -\log(p(y_i/x_i))$$

Cost func: $y(0) = -\log p(y|x)$

$$= -\log(\sigma((2y-1)z)) = \zeta(-(2y-1)z)$$

$= \zeta((1-2y)z)$

remember ζ is the softplus function



$$w^{(k+1)} = w^{(k)} - \eta \frac{d\mathcal{E}}{dw}$$

gradient should be small // $z \gg 0$, & prediction is & is correct

This ζ function achieves that, as, if $y=1$

$\zeta((1-2y)z)$

$\zeta(-z) \approx 0$ since $z \gg 0$

$z \gg 0$, $y=0$, you want high gradients

then $\zeta(z) = \to z$

you get

Why for

$z \ll 0$ $y=0$ then you get $\zeta(z) \to 0$

$z \ll 0$ $y=1$ $\zeta(-z) \to (-z)$

Why that cost function
WTF is wrong with what I have?!

$$P(y_1=1/x_1) = \sigma(z_1)$$
$$P(y_1=0/x_1) = 1-\sigma(z_1)$$
$$P(y_i) = (\sigma(z_i))^{(y_i)} (1-\sigma(z_i))^{(1-y_i)}$$
$$p(y_1,y_2,\dots y_m | x_1,x_2 \dots x_m) = \prod_{i=1}^{n} (p(y_i/x_i))$$
$$= \prod_{i=1}^{m} (\sigma(z_i))^{y_i} (1-\sigma(z_i))^{1-y_i}$$

~~ratio~~

$$= \log (p(y_1,y_2 \dots y_m | x_1,x_2 \dots x_m))$$
$$\max \sum_{i=1}^{m} \left[y_i \log(\sigma(z_i)) + (1-y_i) \log(1-\sigma(z_i)) \right]$$
$$= \min \sum_{i=1}^{n} -\left(y_i \log \sigma(z_i) + (1-y_i) \log(1-\sigma(z_i)) \right)$$

$z \gg 0$ $y=1$ → saturates
$z \gg 0$ $y=0$ also saturates ;-;

$\sigma(z)$ saturates
⇒ $\log(-\sigma(z))$ saturates (ultimately, when $\sigma(z)$ doesn't change much at all)

⇒ problem //

Up next: Softmax, Orders
& how to deal with them!

Alright, Lets say m outputs are there

Then how do we predict

$(w^T h + b)$ is input

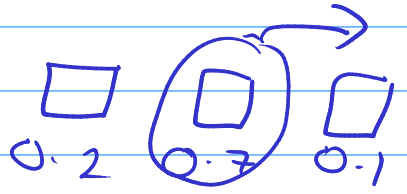
$(w^T h + b) = z_i = \log \hat{p}(y=i/x)$

unnormalized log probabilities

$\Rightarrow e^{w^T h + b} = \hat{p}(y=i/x)$

$$p(y=i/x) = \frac{e^{z_i}}{\sum_{i=1}^{m} e^{z_i}}$$ for m classes

Why distribution? This is a multinomial dist.



→ we can say it belongs here

→ So we can "turn" z_i into a probability this way.

→ we dont normally maximize THE probabilities

So we ~~not~~ maximize the (unnormalized) log probabilities

then we normalize

Softmax activation $= \frac{e^{z_i}}{\sum_{i=1}^{m} e^{z_i}}$

$$\min\left(-\log \prod_{i=1}^{m} \left(p(Y=2|x_i)\right)^{\text{one hot encoding}}\right)$$

$$\left(P(Y=2/x)\right)^{0} \left(P(Y=2/x)\right)^{1}$$

0, since this is class1

1, since this is class2

It is however not for sure that you'll hit local minima after some iterations

we might need early stopping

Other activation funcs

ReLU: differentiability problem @ '0' take either LHS, or RHS derivative

Leaky ReLU

$h_i = g(z_i) = \max(0, z_i) + \alpha_i \min(0, z_i)$

$\alpha = -1 \Rightarrow$ absolute value activation function

take max of each $w^T x + b$ in each division and all neurons in that division have max of division in that output (maxout)

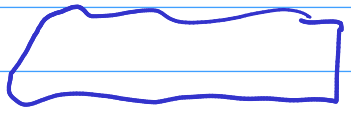
This is maxout

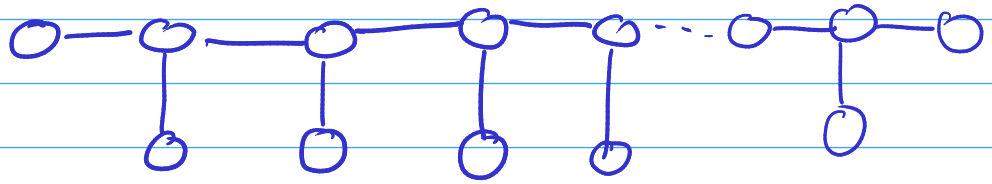
Don't use sigmoid, tanh(z) saturates lesser, use that instead
(make sure you shift origin of this graph)

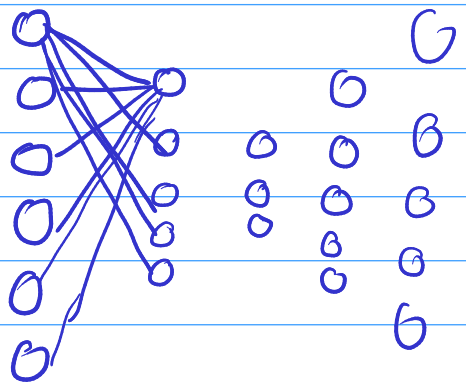
radial basis

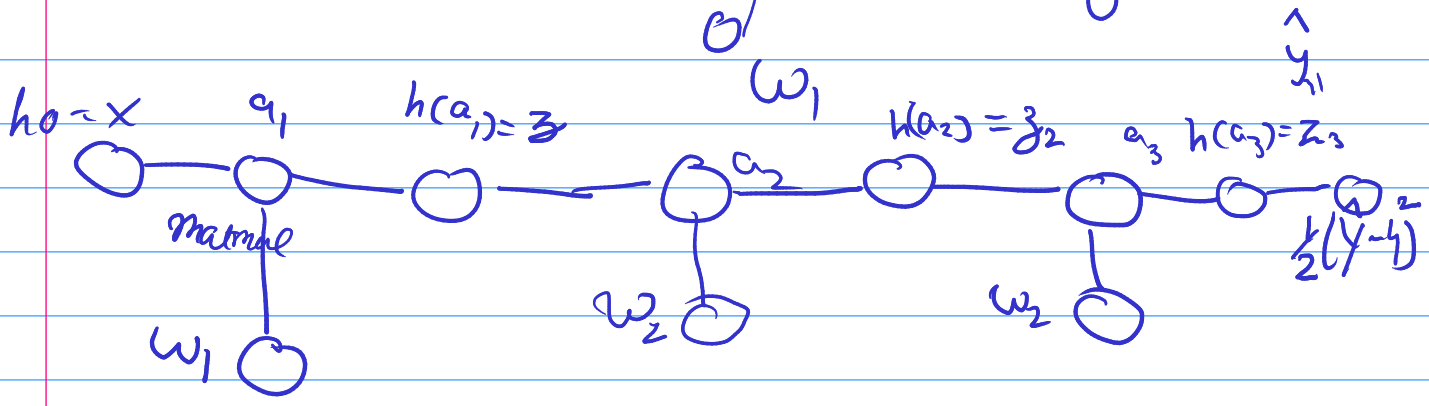
softplus $\log(1+e^z)$

hard tanh $\max(-1, \min(1, a))$

$E(\omega) =$ 







$$\omega^{(k+1)} \leftarrow \omega^{(k)} - \eta \left.\frac{\partial E}{\partial \omega}\right|_{\omega=\omega^k}$$

$$\hookrightarrow \nabla E(\omega)\big|_{\omega=\omega_k}$$

$$\nabla_A(z) = \nabla_c(z)\,\nabla_A(C) = GB^T$$

$$\nabla_c z_{20\times 1} = \begin{bmatrix} & \\ & \end{bmatrix}_{20\times 1}$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,10} \\ a_{21} & a_{22} & \cdots & a_{2,10} \\ \cdots & \cdots & \cdots & a_{20,10} \end{bmatrix}_{20\times10} \rightarrow \nabla_A(y)$$

So they flatten it $\rightarrow 200\times1$

find derivative

and then reshape

treat matrix as a vector (vectorize)

$$\frac{\partial}{\partial x}\sigma(x) = \sigma(x)\sigma(1$$

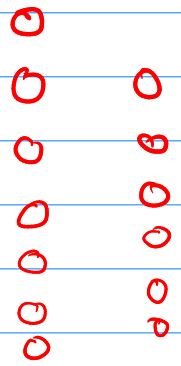
$(x_1, t_1)\ (x_2, t_2) \cdots (x_N, t_N)$

$y = w_0 + w_1 x^1 + w_2 x^2 \cdots + w_q x^q$

$$E(w) = \sum_{n=1}^{N} (t_n - y(x_n, w))^2$$

Squared sum of all weights $\leftarrow \|w\|^2 \leq \eta \Rightarrow$

$$J(\theta) = \sum_{n=1}^{N} (t_n - y(x_n, w))^2 + \frac{\alpha}{2} \|w\|^2$$



$$\frac{1}{2} (\hat{y}_n - y)^2 + \frac{\alpha}{2} \|w\|^2$$

↓ not biases

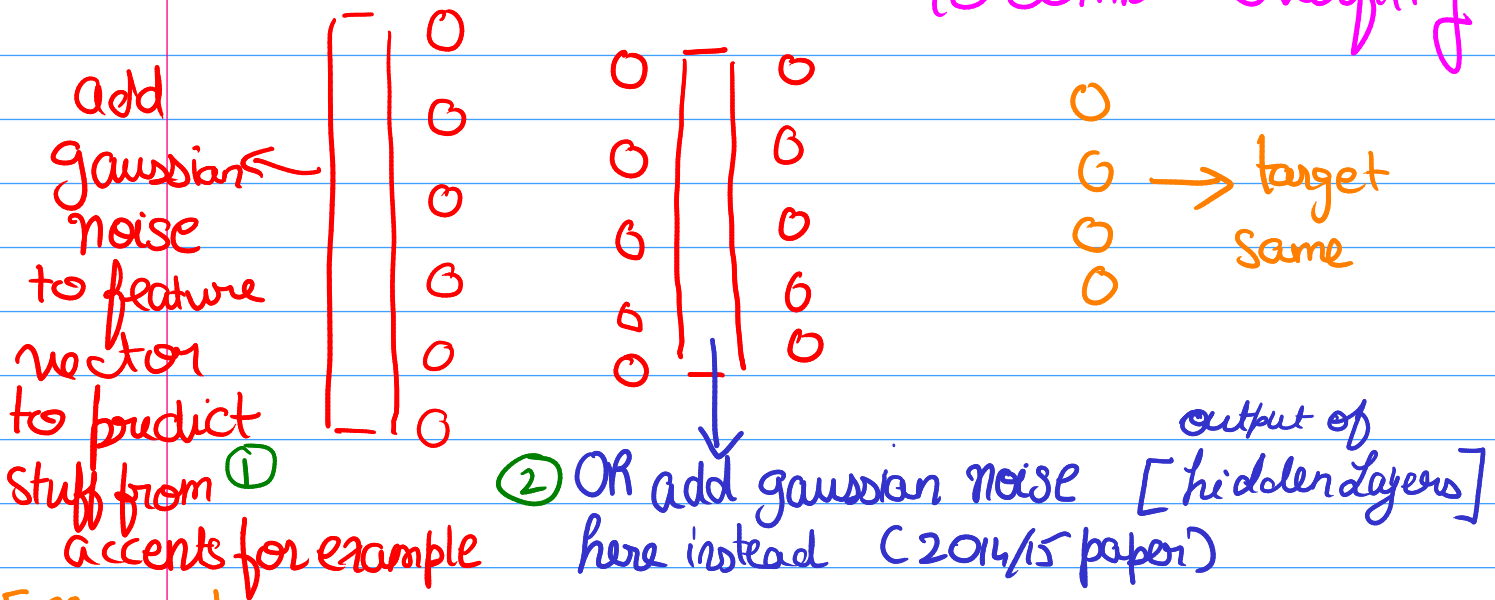
bias → extra node with '1' weight

We regularize weights, but not biases normally

(it is to provide a bias; it is hardwired!

can be updated but rarely is

Some few other techniques to combat overfitting



[mean at the datapoint] $\omega^{(k+1)} = \omega^{(k)} - \eta \left(\nabla E(\omega) \big|_{\omega=\omega^{(k)}} \right)$ [①&②]

target can be considered the same

⇒ more datapoints!

$\omega^{(k+2)} = \omega^{(k+1)} - [\text{noise}] - \eta \left(\nabla E(\omega) \big|_{\omega=\omega^{(k)}} \right)$ ③

(3) add noise to the weights themselves

① Seems nice
② Fine
③ Sounds like WTF but works

Another technique

$(x^{(1)}, t_1) \; (x^{(2)}, t_2) \ldots (x^{(m)}, t_m)$

$(0 \quad 1 \quad 0\,0\,0 \cdots \quad 0)$

Crossentropy $p^{(0)} \; p^1 \; p_3^0 \; p_4^0 \; p_5^0 \; p_6^0$

estimated probabilities functions of ω & other params

We want to maximise probability
⇒ we minimise error

but then if 0 1 0 0 0 0 is wrong (scope for error in target attribute)
we can't acc for it

if we acc for the probability of error to be ϵ

$p_1^{\epsilon/5}$ $p_2^{1-\epsilon}$ $p_3^{\epsilon/5}$ $p_4^{\epsilon/5}$ $p_5^{\epsilon/5}$ $p_6^{\epsilon/5}$ → Softens the error function

also hyperparameter

This is our new crossentropy

from this we get our error

Early stopping

In a way where computational power is reduced and fit is optimal

SGD $\theta_{k+1} = \theta_k - \eta \left(\nabla E(\theta) \right)\Big|_{\theta=\theta_k}$

find training & validation error

$\theta^* : \theta_{100}$

θ_{200} has lower validation error

$\Rightarrow \theta^* : \theta_{200}$

θ_{300} has higher validation error

$\theta^* : \theta_{200}$ (Don't replace)

θ_{400} also has higher than θ_{200}

→ 100 epochs

this 100 is called "patience"

if patience = 20 ⇒ 20 iterations to observe worsening of validation error

Don't refer to this!

not necessarily increase in worsening

worsening ⇒ worse than θ^*

Sliding window over epochs

keep a counter everytime next if error ↓ otherwise ↑ 1

if counter == patience

RIP

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let θ_0 be the initial parameters.
$\theta \leftarrow \theta_0$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\theta^* \leftarrow \theta$
$i^* \leftarrow i$
while $j < p$ **do**
  Update θ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\theta)$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\theta^* \leftarrow \theta$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are θ^*, best number of training steps is i^*

Algorithm determines the best amount of time to train. The meta algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Take epochs like for now

epochs

Update n times (iterations)

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let θ_o be the initial parameters.
$\theta \leftarrow \theta_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\theta^* \leftarrow \theta$
$i^* \leftarrow i$
while $j < p$ **do**
  Update θ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\theta)$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\theta^* \leftarrow \theta$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are θ^*, best number of training steps is i^*

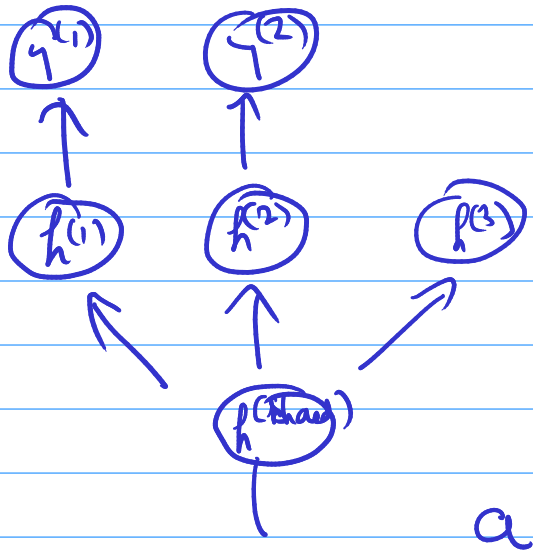
T iterations

20

Algorithm determines the best amount of time to train. The meta algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

θ_0 4,000 20,000 7

(provides)
Note: SGD does implicit regularization



no. of steps for early stopping is a hyperparameter
↳ find this by training a small subset, then use the number to train whole model (Shut up & TRAINNN)

Flavor #2

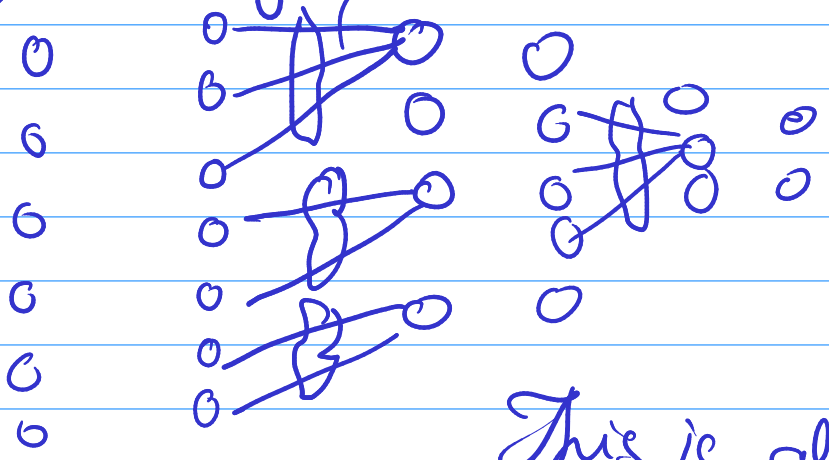
Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**
  Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for n steps.
end while

θ^* for the 's

don't throw θ^* away

70%.

Weight sharing. → same



This is also a regularizer, overfitting doesn't happen

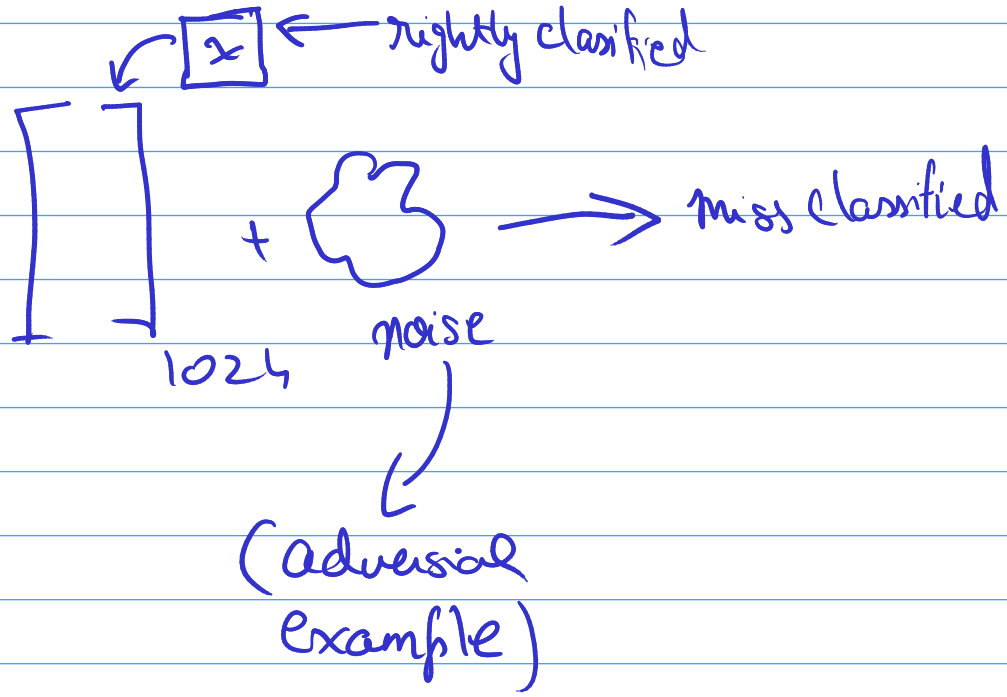
(adversarial)

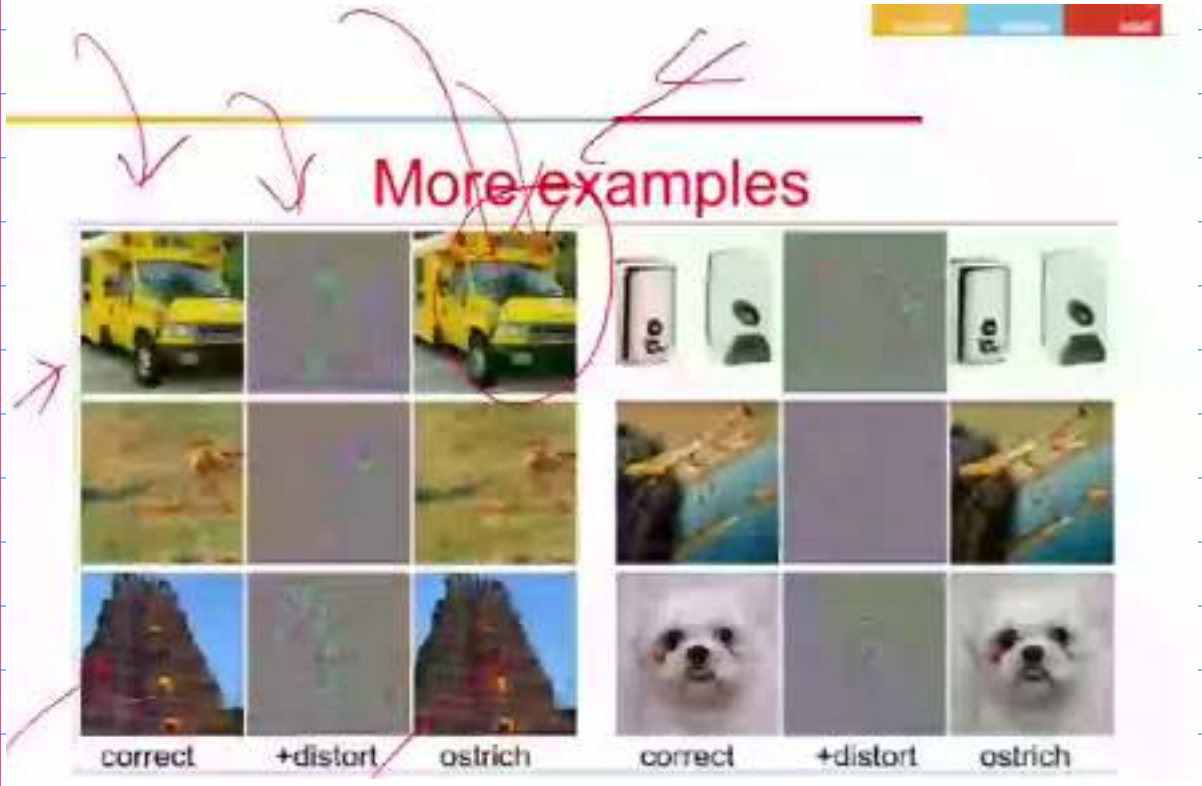
Adversian Training

2014

Szegedy

[x] ← rightly classified





goodfellow 2014, explaining adversial -- a composition of functions• but relu type functions composed can be incredibly linear most of the time

a heavily linear function will look like a lin. comb. of weights

a close neibourhood can be W.(x + epsilon) (x,w and e are vectors)

$f(x+\varepsilon)$

$\frac{1}{x+\varepsilon} \simeq \frac{1}{x}$

if f is quadratic, ε will become ε^2 etc. and vanish

Every layer

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30000} \end{bmatrix} \left(\begin{bmatrix} x_1 \\ \vdots \\ x_{30000} \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_{30000} \end{bmatrix} \right)$$

ε is non linear

$\Rightarrow$ x becomes high non linear

$f\left(\begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1100} \end{bmatrix} + \begin{bmatrix} \epsilon \\ \epsilon \\ \epsilon \\ \epsilon \end{bmatrix} \right)$ changes quite a lot

$$f(x) - f(x+\varepsilon)$$

Apply 'f' on this

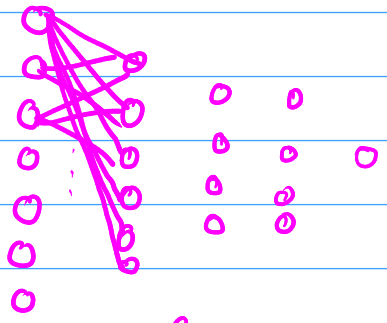
$$\vdots$$

$$g = f_0 \circ f_1 \circ f_2 \cdots f_n(x)$$

$g(y) - g(y+\varepsilon)$ will be way more than

$f(x) - f(x+\varepsilon)$

$$f = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_{100} x_{100}$$



$$f \simeq \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_{100} x_{100}$$

Add tiny noise

$$\omega^T x + \omega^T \delta$$

$$\omega^T x + \omega^T \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_{100} \end{bmatrix}$$

the quantity

Let
δ_1 be $+\epsilon$ if $\omega \geq 0$
δ_1 be $-\epsilon$ if $\omega < 0$

(Whitebox attack)

$x = x + \delta$

$\delta = \epsilon [\text{sign}(\text{grad})]$

(White box)

$$\omega^T x + \epsilon \underbrace{(\|\omega\|)}_{L_1} = \omega^T x + \epsilon (|\omega_1| + |\omega_2| + \cdots |\omega_{100}|)$$

$x \to (x + \delta)$

$f(x) \to f(x + \delta)$

Since f is linear, even if x is close to x'

$f(x')$ can be completely different

BUT Why!? (This seems contradictory)

You can't crack this lol

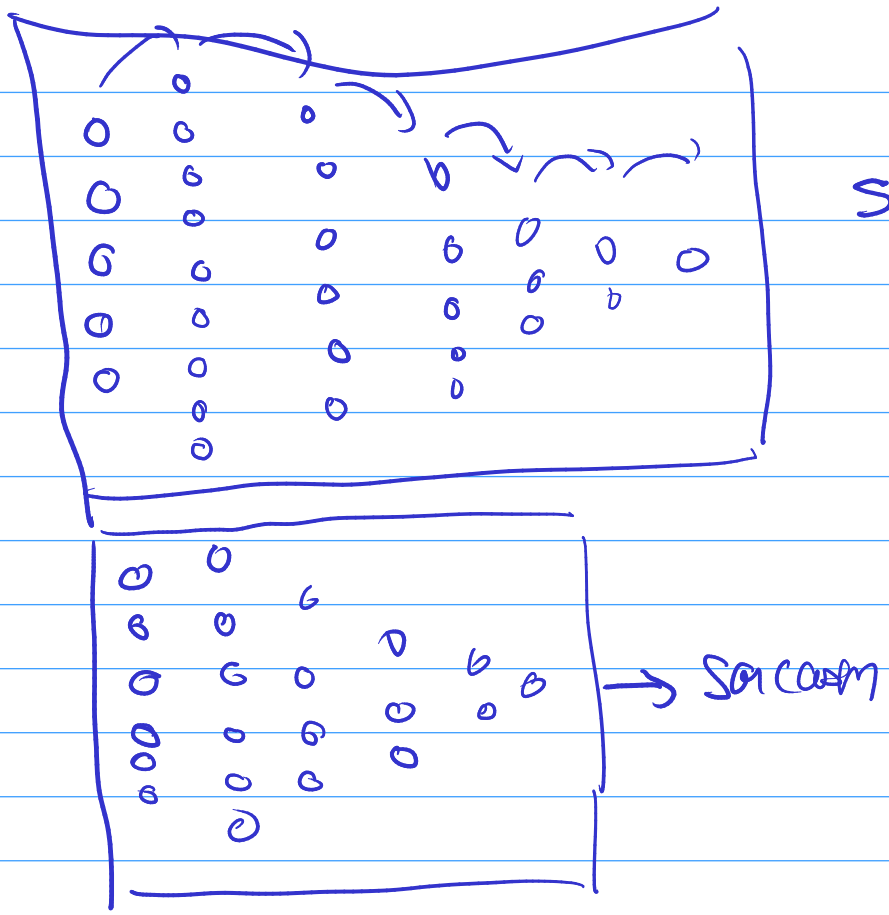
Basically, this is an explanation, not a proof for adversarial examples

It works for not all examples, f is normally piecewise linear

We are also NOT comparing with polynomial & exponential; we are just saying linearity itself makes values grow! Compared to sigmoid etc.

SVMs are also linear but have large decision regions, so they don't face this problems.

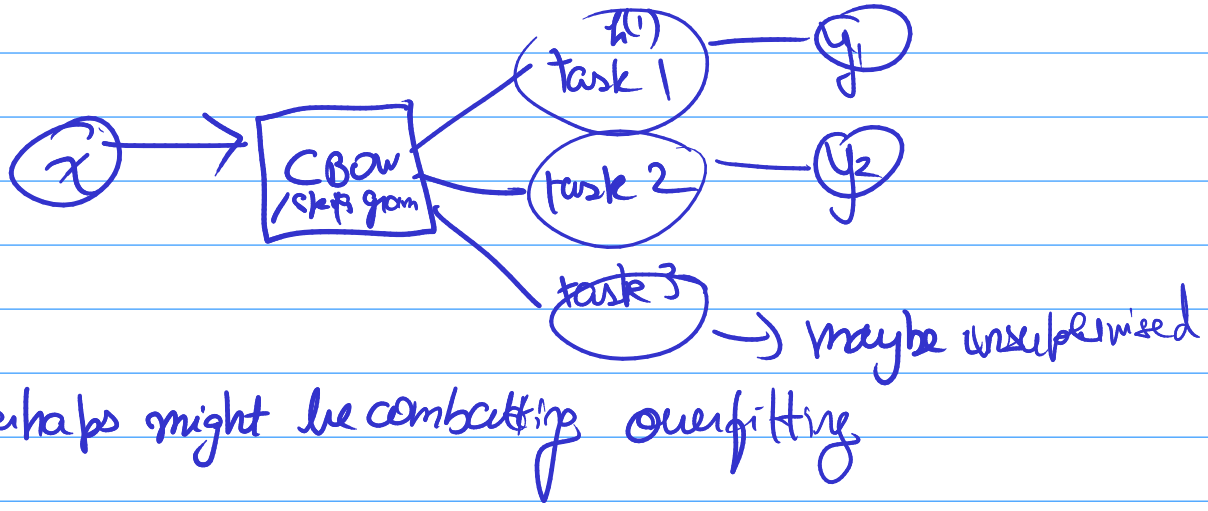
⇒ Training on adversarial examples makes model more robust



Whether it is a sentiment analysis or sarcasm predictor, to get a very nice numerical equivalent that is same

Since otherwise, you might overfit, since the word vector representations may be prejudiced!

Do make a word embedding commonly and then split



perhaps might be combatting overfitting

$$\min \underbrace{J(\theta; X, y) + \lambda g(\theta)}$$

$\theta = (\theta_1, \theta_2 \cdots \theta_n)$

$g(\theta) = \|\theta\|^2$

$g(\theta) = \|\theta\|_1$

Now rather than directly using weights, we use some other function

$+\lambda \underbrace{\Omega(h)}$

Some functions have been proposed

C_1 C_2 ---------- C_{20}

20 classifiers

R_1 R_2 ---------- R_{20}

Let ϵ_1 ϵ_2 ϵ_k

errors

↓

Let them follow a normal distribution $N(0, V)$

Ensembler $C_1\ C_2 \cdots C_k$ → classifiers

majority vote

→ improvises your testing error since majority of models aren't wrong.

→ Bagging ensembler :

↳ Model is just the same eg: Decision tree

Consider a sample set $T_1\ T_2 \cdots T_{10000}$

C_1 → sample with replacement $T_1\ T_{500}\ T_1\ T_{10}\ T_{6253}\ T_{3231} \cdots$

C_2 Similarly

⋮

C_k (it's also bagging ensembler)

Or

develop k regression models (varying them)

$M_1\ M_2 \cdots M_k$

Errors → $\epsilon_1\ \epsilon_2 \cdots \epsilon_k$

$\frac{\sum \epsilon_i}{k}$

Let these ϵ_i's follows $N((0,0,0\cdots 0), Cov(\epsilon_1, \epsilon_2, \epsilon_3 \cdots \epsilon_k))$

$E(\epsilon_1) = 0$

$Var(\epsilon_1) = E(\epsilon_1 - E(\epsilon_1))^2$

$= E(\epsilon_1^2) = V$

An assumption

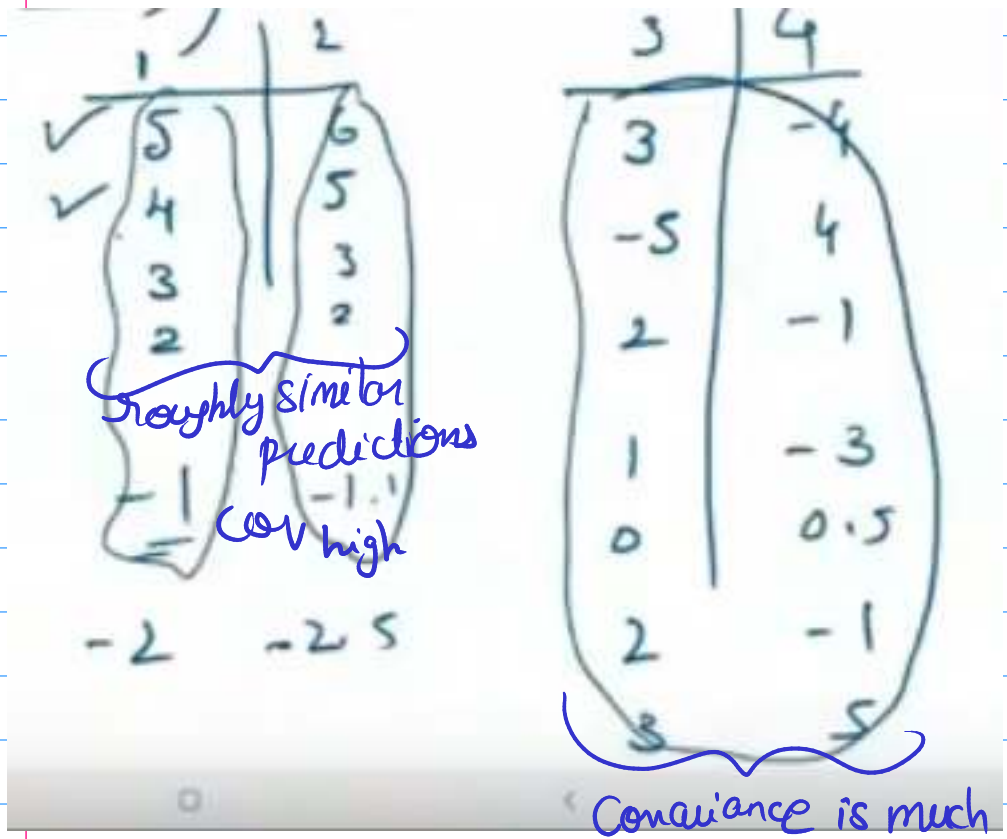
$$Var(\epsilon_1) = Var(\epsilon_2) = \cdots = Var(\epsilon_k) = \sigma$$

$$Cov(X,Y) = E((X - E(X))(Y - E(X)))$$

↳ How the variation of X from E(X) is varying w.r.t to variability/variation of Y from E(Y)

$Cov(\epsilon_1, \epsilon_2)$

1



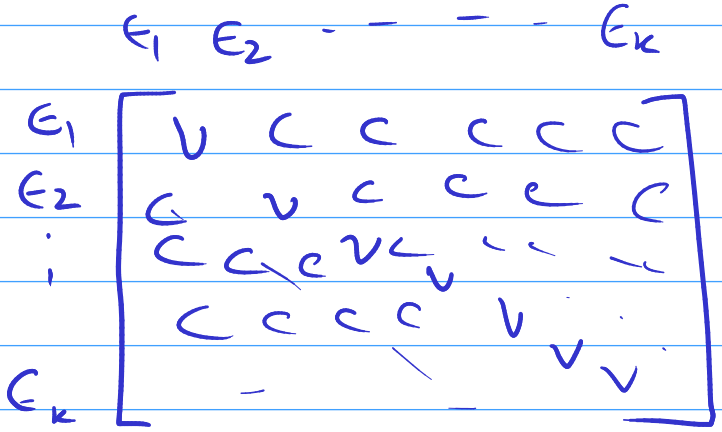
roughly similar predictions

Cov high

Covariance is much lesser, they vary independently

$$Cov(\epsilon_1, \epsilon_2) = E(\epsilon_1 - E(\epsilon_1))(\epsilon_2 - E(\epsilon_2))$$
$$= E(\epsilon_1 \epsilon_2)$$
$$= C$$

We assume all covariances are same



M_1 M_2 M_k

Errors → ϵ_1 ϵ_2 ϵ_k

$$\frac{1}{k}\sum_{i=1}^{k}\epsilon_i$$

$$\frac{1}{k}\sum_{i=1}^{k}\epsilon_{1i} \quad \text{(1st sample)} \quad \cdots \quad \frac{1}{k}\sum_{i=1}^{k}\epsilon_{ni}$$

Errors: Can you average these?

NO $\quad E\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)$ can cancel

But then isn't error squared already (for argument's sake, lets assume they are not)

$$E\left(\left(\frac{1}{k}\sum_{i=1}^{k}\epsilon_i\right)^2\right)$$

$$= \frac{1}{k^2}E\left(\left(\sum_{i=1}^{k}\epsilon_i\right)^2\right)$$

$$= \frac{1}{k^2}E\left(\sum_{i=1}^{k}\epsilon_i^2 + \sum_i\sum_j \epsilon_i\epsilon_j\right)$$

$$= \frac{1}{k^2}E\left(\sum_{i=1}^{k}\epsilon_i^2\right) + \sum_i\sum_j E(\epsilon_i\epsilon_j)$$

$$= \frac{1}{k^2}\left[E\left(\sum_{i=1}^{k}\epsilon_i^2\right) + \sum_i\sum_j C\right]$$

$$= \frac{1}{k^2}\left[\sum_{i=1}^{k} E(\epsilon_i^2) + \sum_i\sum_j C\right]$$

$$= \frac{1}{k^2}\left[k\,v + k(k-1)C\right]$$

$$= \frac{1}{k}v + \frac{k-1}{k}C$$

$\boxed{C=0}$

if $M_1, M_2 \cdots M_k$ all behave differently

then $E = \frac{1}{k}v$

M_1 M_2 $\vdots$ M_2 — every variance error is v (which is $E(\epsilon_i^2)$)

But the ensembles together have a variance of $\frac{1}{k}$

But V is a function of ϵ

$\Rightarrow$ even error reduces

$\Rightarrow$ If all models are independent, V comes down by a factor of k

$\Rightarrow$ impressive

if $C \neq 0$

$C = E(\epsilon_i \cdot \epsilon_j) \simeq E(\epsilon_i^2) = v$

$\epsilon_i = \epsilon_j$ if $\epsilon_i \simeq \epsilon_j$

if all ensemblers are same then you get

You common variance to be: $\frac{1}{k}v + \frac{k-1}{k}v$

$$\Rightarrow \quad v \geq E(\epsilon_i^2) \geq \frac{v}{k}$$

$\Rightarrow$ Worst case Ensemblers perform as good as individual models

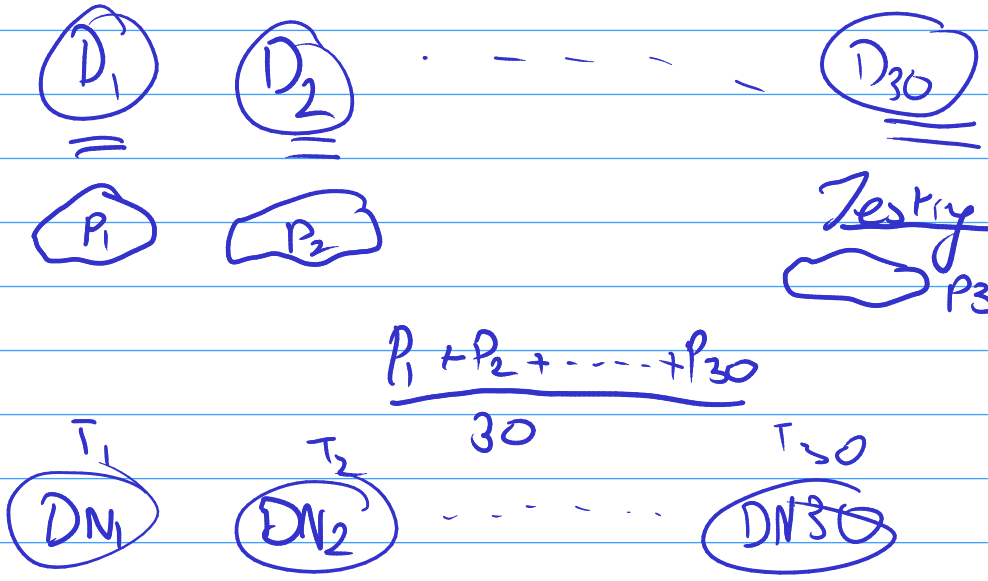
Similarly $C_1 \quad C_2 \quad \cdots \quad C_k$

$E_{Ensemble}(C_1, C_2, \cdots, C_k)$

will perform better, paper mathematical bc

* It enhances the performance of the model
  - ➡ acts as a "kind of" regularizer without being a regularizer in the strict sense.

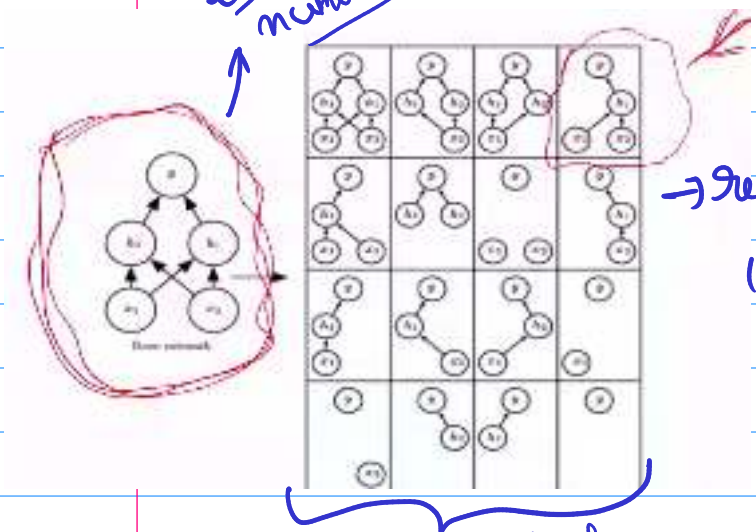
30 models



6 classifiers train — Szegedy ILSVRC

An Ensembles built Bagging

exponential number of models



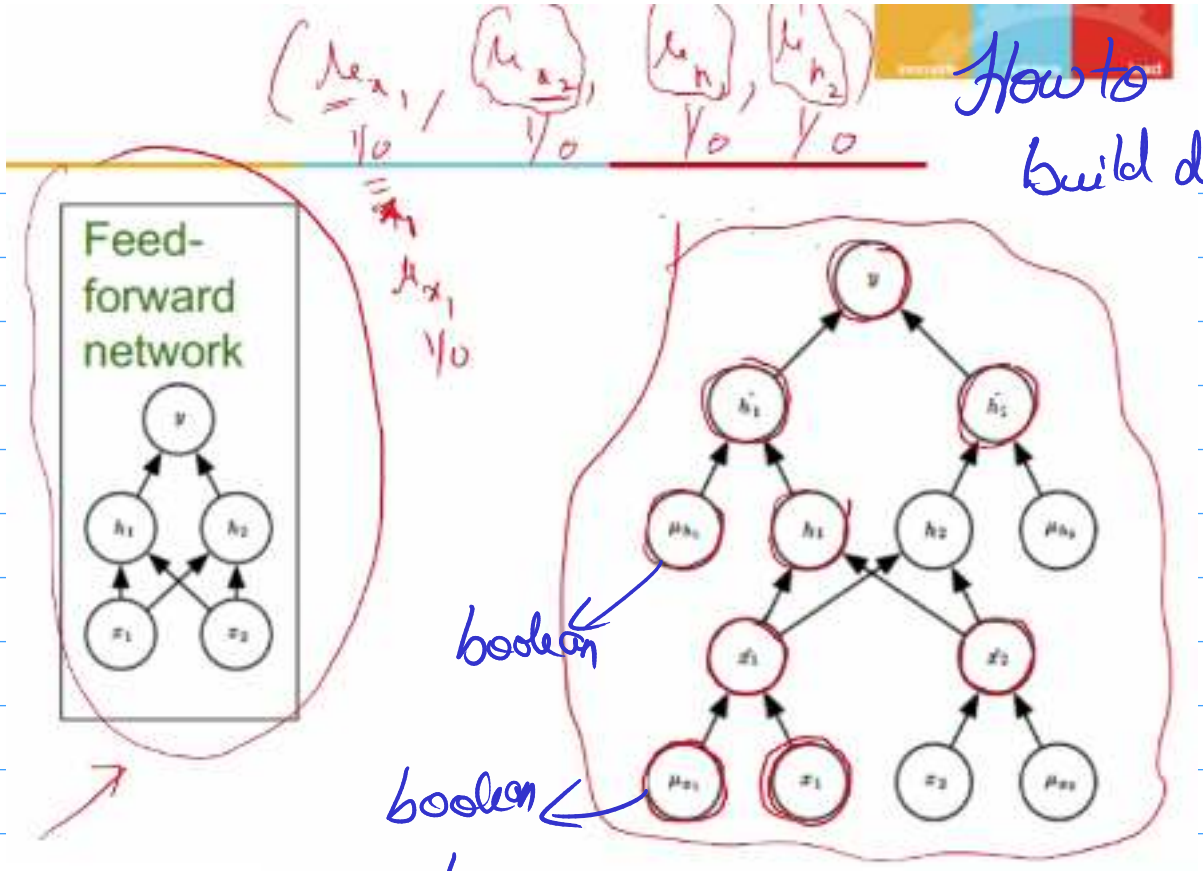
Srivastava → Alter the structure and build an ensembles out of it! #getrekt

→ Remove nodes and/or connections

instead of same model different splits, diff models (with same backbone) and same split

But in this train all these core in parallel, and use their aggregate ⇒ ensembles

Apparently this is Dropout!



vary this vector.

it acts as a regularizer

and we want look into it next class by

Finding the gaps in ML/DL

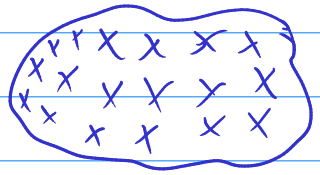
(min 0-1 loss) ← discrete function
intract optimisation
0-1 (not even integer)
optimization problem

very difficult to solve

So they want the surrogate optimisation problems!!

$$\left[E_{(x,y)}\left(\mathcal{L}(f(x;\theta),y)\right)\right]$$

$(x,y) \sim P(\text{data})$



* Lets assume that the dataset, and testing are coming from the same probability distribution

*
→ We are never minimizing expectation before

$$\iint (p(x,y))\,L(f(x,\theta),y)\,dx\,dy$$

) We are never doing this (were never talking before about gradient descent)

Instead, we are doing

$$\min \frac{1}{n} \sum_{i=1}^{n} L(f(x_i;\theta), y)$$

(not a KL divergence no joint distribution)

$$\min_{(x,y)\sim \hat{p}data} E(L(f(x_i;\theta), y))$$

$$\frac{\partial \epsilon}{\partial \theta_k} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial \epsilon_i}{\partial \theta_k} \quad \leftarrow \; n = 10,00,00,000 \text{ so much}$$

$\Rightarrow$ SGD / minibatch

{we estimate $\frac{\partial \epsilon}{\partial \theta_k}$ by $\frac{\partial \epsilon_{100}}{\partial \theta_k}$}

or

estimate $\frac{\partial}{\partial \theta}(\epsilon_2 + \epsilon_5 + \cdots + \epsilon_{75} + \epsilon_{2000})$

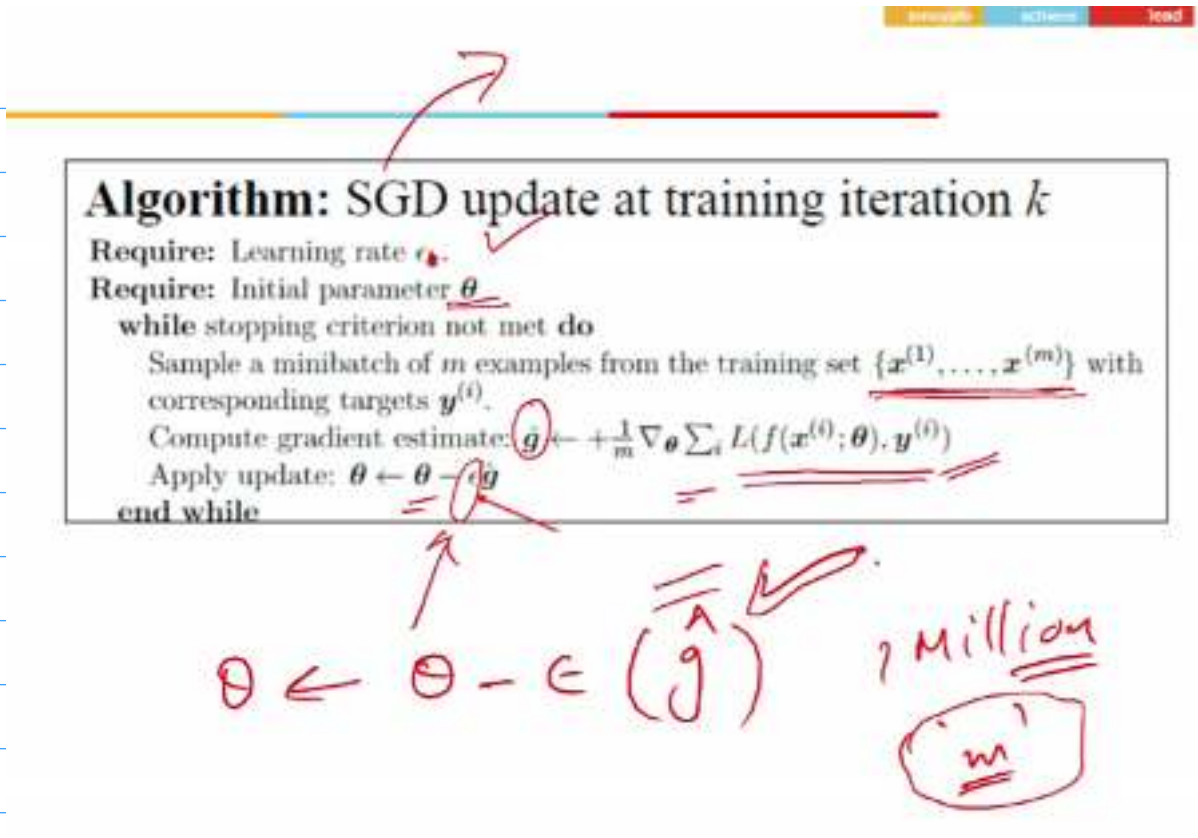
minibatch gradient descent

batch gradient descent typically means classical

cc bigger batch size, better estimate?? (if yes)

minibatch, stochastic, (online)

uses "most recent" stuff (samples) in timestamp or sth for example



a weight with larger derivative will have larger updates

$\epsilon_i \rightarrow w_i$

(actually we see the opposite)

$$O_i^{(k+1)} \leftarrow O_i^{(k)} - \epsilon\left(\nabla_i\left(\Sigma(L(f(x_i^{(i)}, \theta)), y)\right)\right)$$

The learning parameter should not be constant for all iterations

(reason is out of scope)

also, from parameter to parameter

Adagrad

(Lets do this before Adam)

ϵ → global

Initial θ

$\delta = 10^{-7}$

$r = 0$ (accumulation of gradient)

while

minibatch m

$g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x_{(i)}, \theta), y^{(i)})$

$r \leftarrow r + g \odot g$ → element wise product

$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$ but r is a vector

How do we get $\sqrt{r}$

$\theta \leftarrow \theta + \Delta\theta$

$\frac{\epsilon}{\delta + \sqrt{r}}$

$$\frac{\epsilon}{\delta + \sqrt{r}} = \begin{bmatrix} \frac{\epsilon}{\delta + \sqrt{r_1}} \\ \frac{\epsilon}{\delta + \sqrt{r_2}} \\ \frac{\epsilon}{\delta + \sqrt{r_3}} \\ \vdots \end{bmatrix} \quad \text{where } r = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \end{bmatrix}$$

L_1, L_2

Dropout

Early stopping

Data augment

Multitasking

Adversarial training

RMSProp

- Modifies AdaGrad for a nonconvex setting
  - Change gradient accumulation into exponentially weighted moving average
  - Converges rapidly when applied to convex function

Geoffrey Hinton

The RMSProp Algorithm

Require: Global learning rate ϵ, decay rate ρ.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

Algorithm: SGD with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

→ We aren't comparing adagrad with SGD+ momentum

→ SGD + momentum may perform better than adagrad at times

But it is better than SGD!

Algorithm: SGD with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

This line is added from plain momentum

Algorithm: RMSProp with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

$v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot g$

Das has left the meeting

The Adam Algorithm

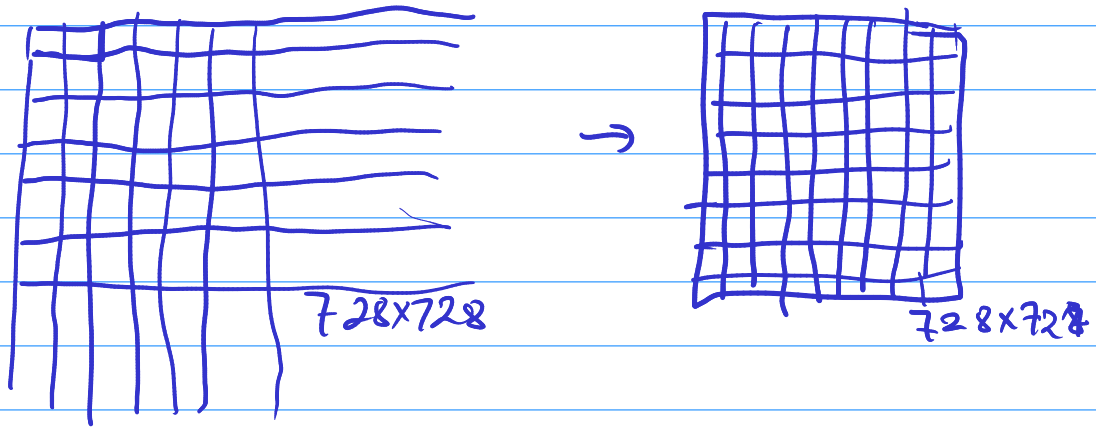
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}, \boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
end while

* Influential but sparse features give lower updates

GP (CS7015)

} to bring the moments as expected (they are lower) since $s=r=0$

specific way of updating



in an ANN

if we have an i/p matrix of 200x200 layer and a hidden layer of 200x200

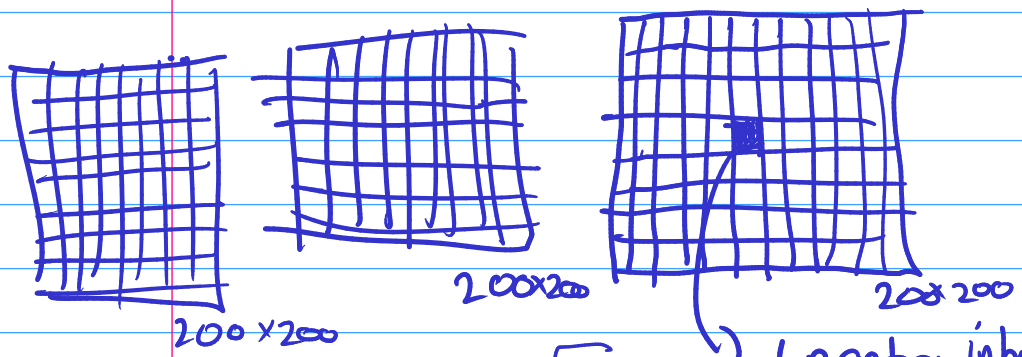
Weight matrix: $((200 \times 200) \times (200 \times 200))$

for a fully connected network

Since we're accounting for the dependency of every pixel

But

When we're discussing something like edge detection we dont need anything aside from the neighbouring pixels



We get an input here

for that, we need to multiply the 200x200 => 40000x1, with the let's say the column of the saved pixel

So we just need the neighbouring points to have some weight & rest all zero

Since there are a LOT of weights that are zero, we dont need so much

So we can just

1D case

Convolution

NOTE: This is not predicting future. We are trying to reduce noise we have

Lets say we want to know where a spaceship is given that we know some info that is noisy

(9.28.37)
(9.28.38)
⋮
(9.29) → to find

→ we have these readings too (Lets say microseconds)

Lets consider that the only measurement device is noisy

To find the true current position

now lets find a weighted average of all the previous 60 seconds

So we have $x(a)$, $w(t-a)$ @ $t = 9.29$, $a = 9.28.37$

9.29

$$s(t) = \int x(a)\, w(t-a)\, da$$

→ time is 't' & your weight w(t-1) will be the largest and then w(t-2), w(t-3) etc...

→ t-a is different for all 'a'.

so you can treat w as a probability distribution in order to get an "average"

$$s(t) = (x * w)\, t$$

↙ x convolved

$$(x-w)(t) = \int_{t_1}^{t_2} x(a)\, w(t-a)\, da$$

this is very similar to a convolution formula

x(a) will be some kind of function which we don't know if continuous

So since measurements are discrete

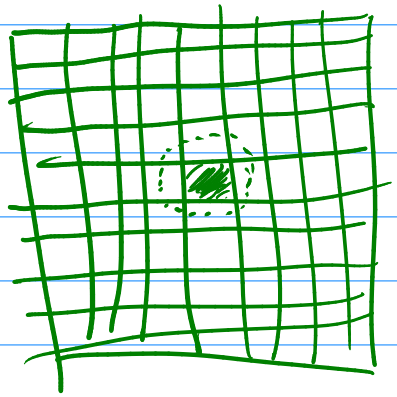
we get $\sum_{a=t_1}^{t_2} x(a)\, w(t-a)$

$$= \sum_{a=9.28.01}^{9.29} x(a)\, w(t-a)$$

this way we can reduce the effect of noise

We are not predicting future!

2D case



→ for edge detection

So we take a linear combination of nearby pixels

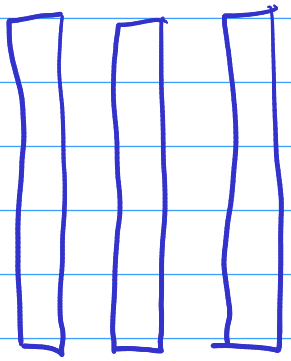
→ here, the linear combination expression will be the same for all pixels

$$S(t) = (x * w)\,t = \sum_{a} x(a)\, w(t-a)$$

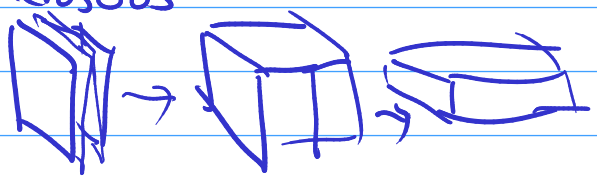
$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)\, k(i-m, j-n)$$

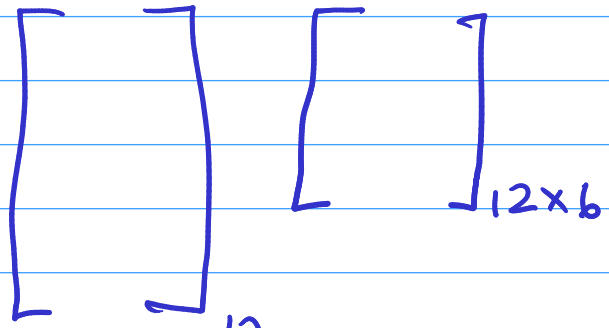
$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)\, k(m,n)$$

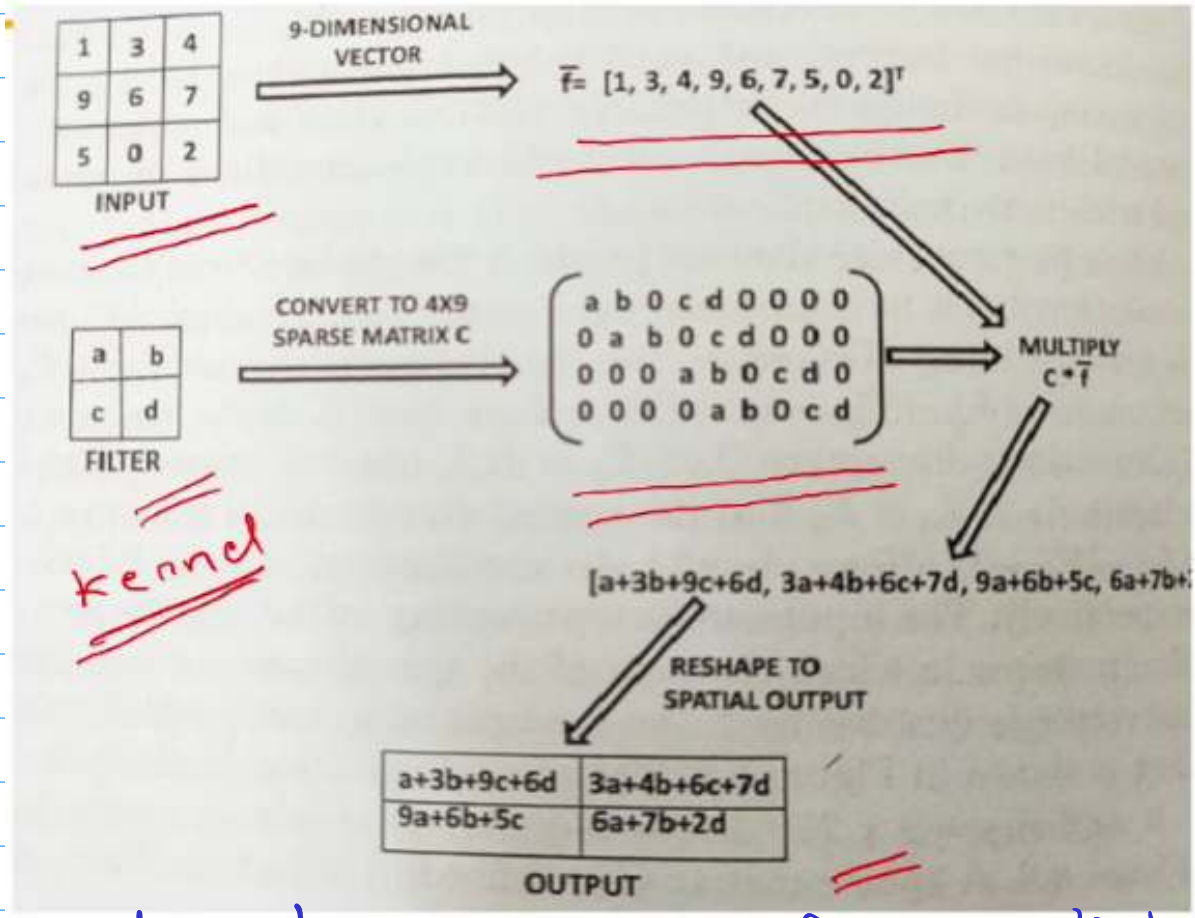
this is more a convolution w.r.t ω



You can give higher dimensional input as tensors







during backprop, mask the weights that are zero

some weights have to be the same

So we minimize loss function

$\min(L)$ subject to some weights being same

This is a very complex optimization problem (?)

So lets say $\boxed{\omega_{11} = \omega_{22} = \omega_{34} = \omega_{44} = \omega_1}$

$$\frac{\partial L}{\partial \omega_1} = \sum \left(\frac{\partial L}{\partial \omega_{11}} \cdot \frac{d\omega_{11}}{\partial \omega_1} \right)$$

$$= \frac{\partial L}{\partial \omega_{11}} \cdot \frac{\partial \omega_{11}}{\partial \omega_1} + \frac{\partial L}{\partial \omega_{22}} \frac{\partial \omega_{22}}{\partial \omega_1}$$

$$+ \frac{\partial L}{\partial \omega_{34}} \cdot \frac{\partial \omega_{34}}{\partial \omega_1} + \frac{\partial L}{\partial \omega_{44}} \cdot \frac{\partial \omega_{44}}{\partial \omega_1}$$

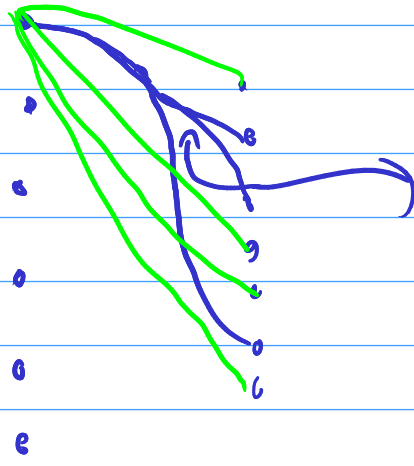
but $\omega_{11} = \omega_1 \Rightarrow \frac{\partial \omega_{11}}{\partial \omega_1} = 1$

$$\Rightarrow \frac{\partial L}{\partial \omega_1} = \frac{\partial L}{\partial \omega_{11}} + \frac{\partial L}{\partial \omega_{22}} + \frac{\partial L}{\partial \omega_{34}} + \frac{\partial L}{\partial \omega_{44}}$$

$$\omega_1^{(2)} = \omega_1^{(1)} - \eta \left. \frac{\partial L}{\partial \omega_1} \right|_{\omega_1 = \omega_1^{(1)}}$$

This way all of them will be updated the same way! LMAO

This is the concept of weight sharing



→ Shared weights, they get updated same way

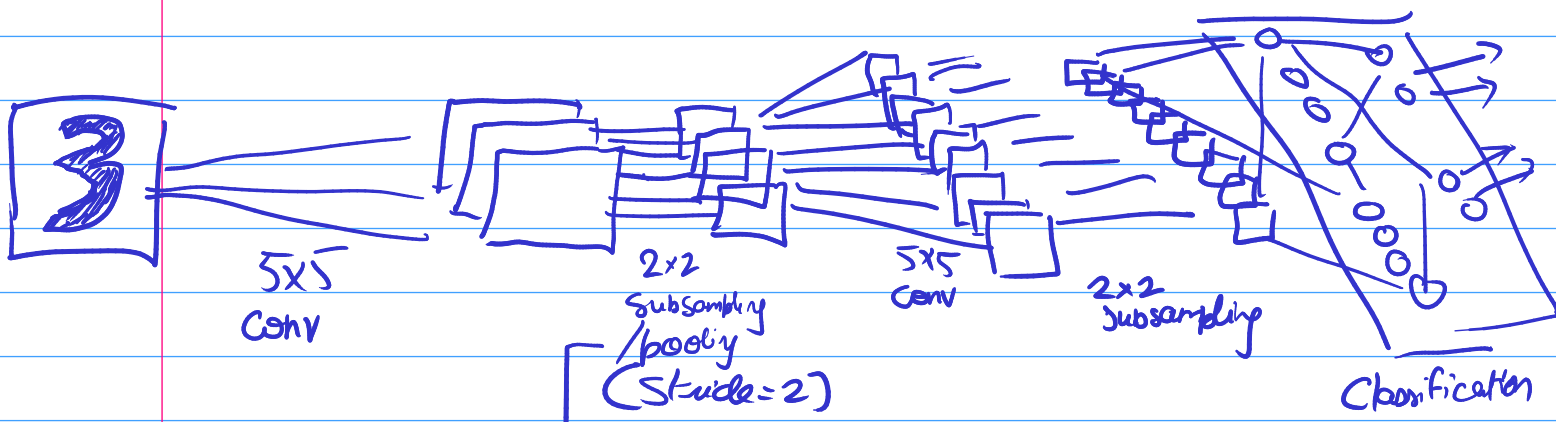
→ In this case, by expanding it to a sparse matrix we increase space complexity and reduce time complexity

→ Another way is to just aggregate over a sliding window, which is time intensive but less space

But internally we dont even actually store as sparse matrix, so most implementations choose sparse conversion

A computational graph does the trick as well

weights as nodes, and multiple edges from weights ⇒ weight sharing



pooling (makes model less sensitive to changes such as shifting)

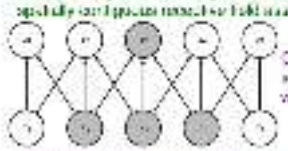
since you are taking max/avg of the values in the window, it doesn't matter where the window is.

Convolution → Strong prior? Low entropy

weights of one hidden unit → close to the neighbor, identical but shifted

Convolutional net is similar to a fully connected net but with an infinitely strong prior over its weights

- I says that the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space
- Prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to each hidden unit



Convolution with a kernel of width 3 is a locally connected unit. It has 3 weights which are the same for all

- Convolution introduces an infinitely strong, prior probability distribution over the parameters of a layer

low variance
⇒ local interactions

$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

(w_0, w_1, w_2, w_3)

Pooling also has strong prior

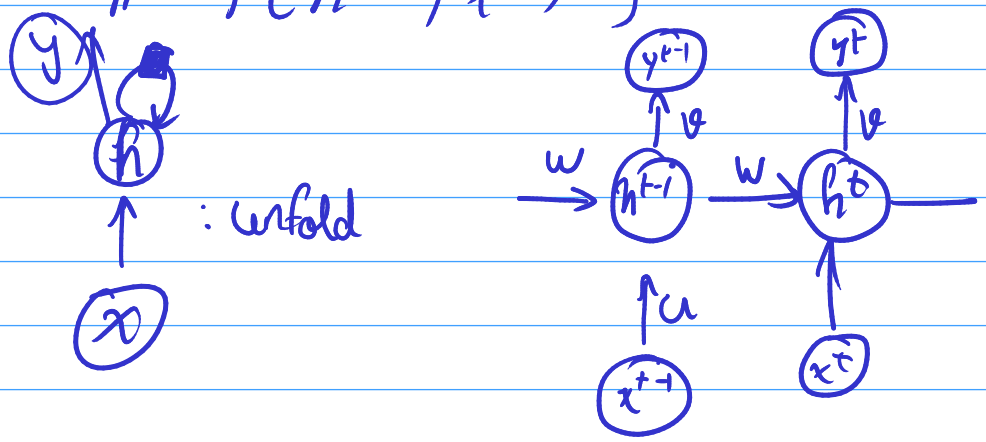
N convolutions → 1 pooling

(needn't be after 1 conv 1 pooling)

wed 4 till 9-4 : 9-10 → Neuroscientific basis (read when free)

Recursive function

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$



$$\begin{cases} s^{(1)} = g^{(1)}(x^{(1)}) \\ s^{(2)} = g^{(2)}(x^{(1)}, x^{(2)}) \\ s^{(3)} = g^{(3)}(x^{(1)}, x^{(2)}, x^{(3)}) \\ s^{(k)} = g^{(k)}(x^{(1)}, x^{(2)}, \cdots, x^{(k)}) \end{cases}$$

today's infosys stock price → today's condition
yesterdays MSE
↳ dependent on
Day before's ...

{1}	{2}	{3}	{4}	{6}
1	4	9	16	36

$$f_n = f_{n-1} + f_{n-2}$$

0	1	2	3	4	5	6
1	1	2	3	5	8	13

Let the data be $S_1 \quad S_2 \quad S_3 \quad \cdots \quad S_{t-1} \quad S_t$ (0)

$S_t = f(S_{t-1}, \theta)$ where θ $\not\equiv$ $S_t = 3S_{t-1} + 5$

↳ how do we discover this

Day 1 2 3 .... 900 901 902 } Stock market

$S_1 \; S_2 \; S_3 \qquad S_{900} \; S_{901} \; S_{902}$

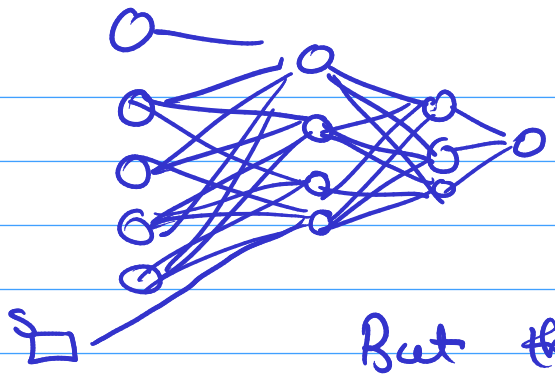
now Lets say that $f(x_1, x_2 \cdots x_{50})$ be the feature vector for 1 day

$S_{903} = f(x_1, x_2 \cdots x_{50})$ is wrong

S_{903}: $f(S_{902}, x_1, x_2 \cdots x_{50})$

↳ consider this also as a feature

Develop a model



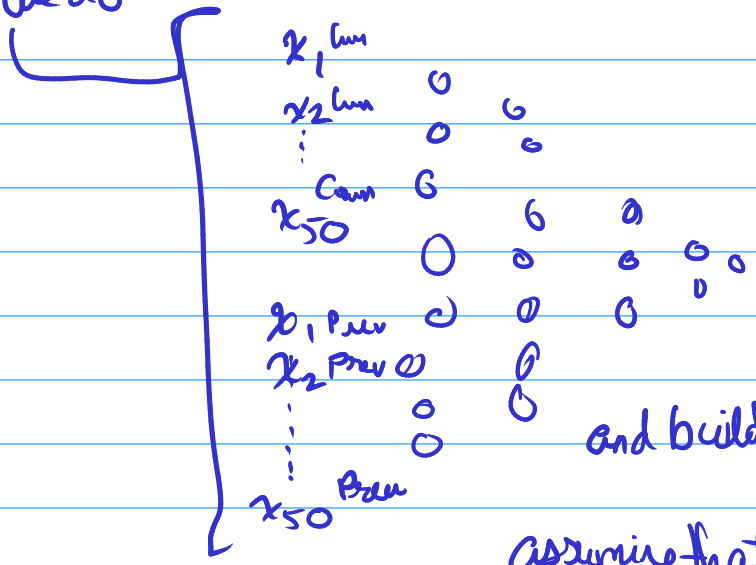
But that's stupid

S_{902} depends on the feature vector of the day 902

So we get

$$g(x_1^{(902)}, x_2^{(902)}, x_3^{(902)} \cdots x_{50}^{(902)}, x_1^{(903)}, x_2^{(903)} \cdots x_{50}^{(903)})$$

So we do



and build a model

assuming that today's condition doesn't depend on day before's state

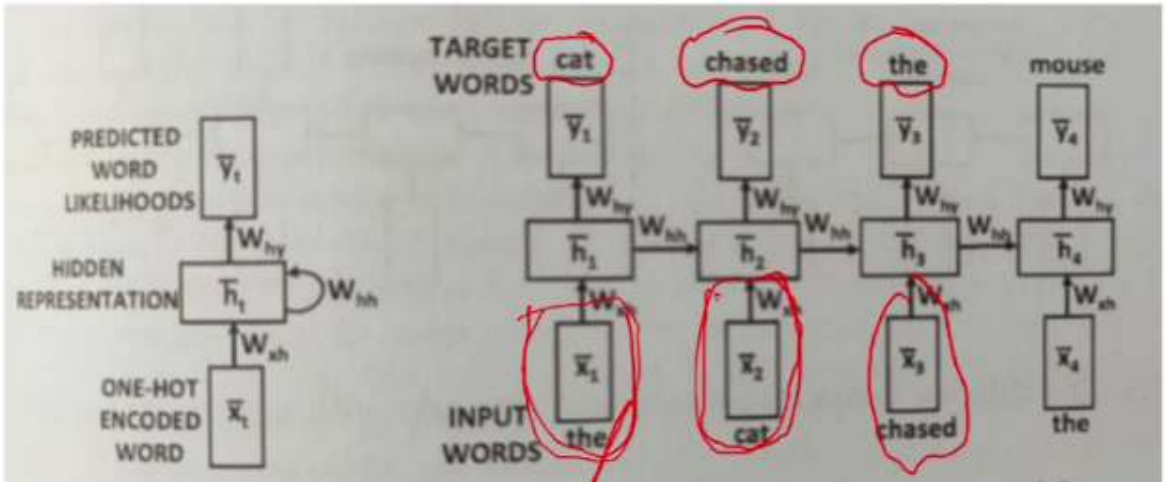
But as per Financial experts, diff number of days are required!

⇒ Each model for # of days! TF?

Another problem

The cat chased the .....

Now the issue is, the later words can influence the sentence



( ) + chased = the

The essence of the previous input & the current input } that function remains the same over the sequence

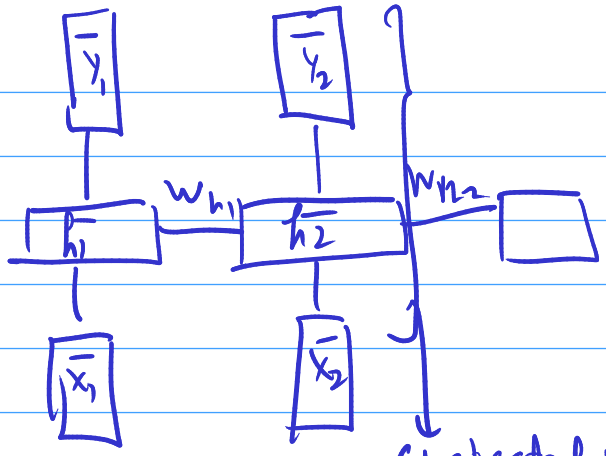
$f(h_2, x_4)$

f is the same

$f(h_0, x_1)$
$f(h_1, x_2)$
$f(h_2, x_3)$

→ if you believe that this f must be the same

almost.. f needn't depend only on position } So the output is independent of the position of input in the sequence by assuming that the function is same



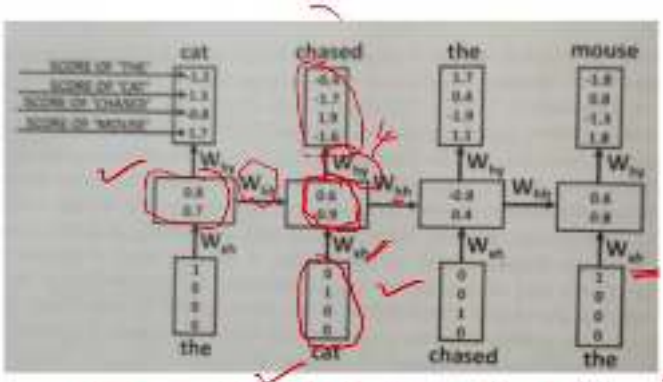
Abstracted info is sent to the next guy (Same dimension as X by the way)

So our abstraction 'h' is as follows:

$$h_2 = \sigma\underbrace{(W_1 h_1 + W_2 h_2)}_{\text{encode}}$$

Why $h_2 \Rightarrow$ word

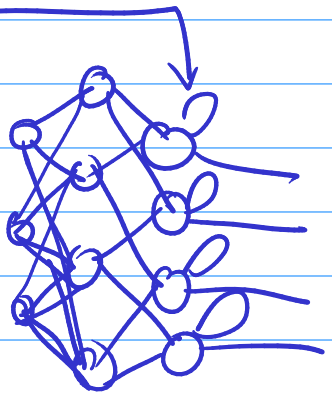
decode, get scores/word vector



$$\bar{h}_t = \tanh(w_{xh}\bar{x}_t + w_{hh}\bar{h}_{t-1})$$

$$\bar{y}_t = w_{hy}\bar{h}_t$$

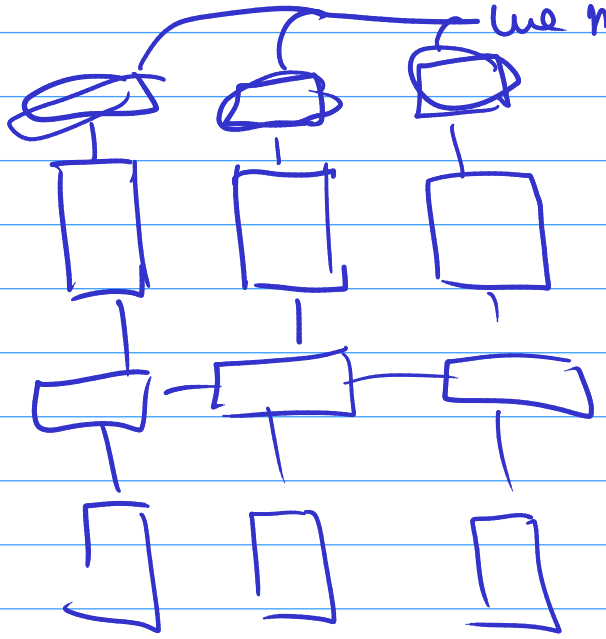
This is loop unrolling



$$\bar{h}_t = \tanh(w_{xh}\bar{x}_t + w_{hh}\bar{h}_{t-1})$$

now we know the ground truth whilst training

if ground truth is [1 / 0 / 0 / 0] → then '1' has to be maximised.



we minimise the loss for entire sequence

The sequence length can vary but that's not a problem, since we built the model with that in mind as seen below

$$L = -\log\left(\hat{p}_i^{y_i}\right)$$

for gradient descent

$$\frac{\partial L}{\partial w_{xh}} \quad \frac{\partial L}{\partial w_{hh}} \quad \frac{\partial L}{\partial w_{hy}}$$

1·8	$\hat{y}_1$
1·3	$\hat{y}_2$
0·8	$\hat{y}_3$
1·7	$\hat{y}_4$

$\frac{\partial L}{\partial \hat{y}_1}$ we can find

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{bmatrix} = w_{hy} h_2$$

$\frac{\partial L}{\partial w_{xh}}$: g RNX

↳ accumulated error

also $g^{(2)} x^{(2)}$ (so we get multiple gradients)

$g^{(3)} x^{(3)}$

$L = -\log(\quad w_{xy} h_h)$

how to maintain w_{xh} to be the same?

Similar to CNNs (Refer notes) $\frac{\partial L}{\partial w_{xh}^{(1)}} \frac{\partial L}{\partial w_{xh}^{(2)}} \frac{\partial L}{\partial w_{xh}^{(3)}} \frac{\partial L}{\partial w_{xh}^{(4)}}$

$$\frac{\partial L}{\partial w_{xh}} = \frac{\partial L}{\partial w_{xh}} * \left(\frac{\partial w_{xh}}{\partial w_{xh}^{(1)}} + \frac{\partial w_{xh}}{\partial w_{xh}^{(2)}} + \frac{\partial w_{xh}}{\partial w_{xh}^{(3)}} \right)$$

$$w_{xh} = w_{xh}^{(1)} = w_{xh}^{(2)} = w_{xh}^{(3)} \Rightarrow \frac{\partial w_{xh}}{\partial w_{xh}^{(1)}} \Rightarrow 1$$

RNN → Recurrence

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

$$= f(f(h_{t-2}, x_{t-1}), x_t)$$

$$f = (\tanh(U x_t + W h_{t-1}))$$

Problem

(1) Every big sentence, answer question
forget stuff
carry stuff

$$h_{t+1} = \tanh(b + Wh_t + Ux^{(t+1)})$$

$$\frac{\partial L}{\partial h_{t+1}} \omega C$$

$$\left[\because \frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_t} \right]$$

$$= \frac{\partial L}{\partial h_{t+1}} \omega \square$$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \omega \square \qquad = \frac{\partial L}{\partial h_3} \omega^2 \square$$

$$\hookrightarrow \frac{\partial L}{\partial h_3} \omega \square$$

$W = Q A Q^T \Rightarrow W^k = Q A^k Q^T$

(positive semidefinite)

orthogonal matrices

$$A = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & & \lambda_k \end{bmatrix}$$

$\left[\because W^2 = Q A \boxed{Q^T Q} A Q^T = Q A^2 Q^T \right]$

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_k} \cdot Q A^k Q$$

λ_1 $\lambda_2 \cdots \lambda_k$

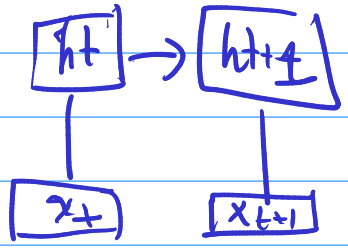
$0.01 \quad \lambda^{40} \Rightarrow 0.000\cdots01$

$\lambda = 2$

$\lambda^{40} \Rightarrow 2^{40}$

→ So gradients become very small (vanish) very large (explode)

$h_{t+1} = \tanh(W h_t + U x_{t+1})$



now we want to forget some stuff as well

But NOW?

$f: \sigma(b^f + U^f x_t + \omega^f h_{t-1})$

multiply it by h^{t-1}

Now we get how much we forget

Now we need a remember function

(But why ? Can't we just add $b + Ux^t + \omega h^{t-1}$)

$g = b^g + U^g x + W^g h^{t-1}$

→ Since it's always good to get an idea of it apparently

→ we take this and multiply it with $(b + Ux^t + Wh^{t-1})$

$f^{(t)} h^{(t-1)} + g^t (b + Ux^t + Wh^{t-1})$

But thoda wait kaale, there's a change now

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right)$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right)$$

$$h_i^{(t)} = \tanh\left(s_i^{(t)}\right) q_i^{(t)}$$

$$q_i^{(t)} = \sigma\left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\right)$$

But here, we dont do $\tanh(b + (Ux_t + Wh_{t-1}))$ here, unlike RNN

we do

$$h^{t+1} = \tanh(s^t)\,\sigma(b + Ux_t + Wh_{t-1})$$

(fraction of previous person also taken)

now $s^{(t)} =$

$$f^{(t)} s^{(t-1)} + g^{(t)} \sigma(b + U x_t + W h_{t-1})$$

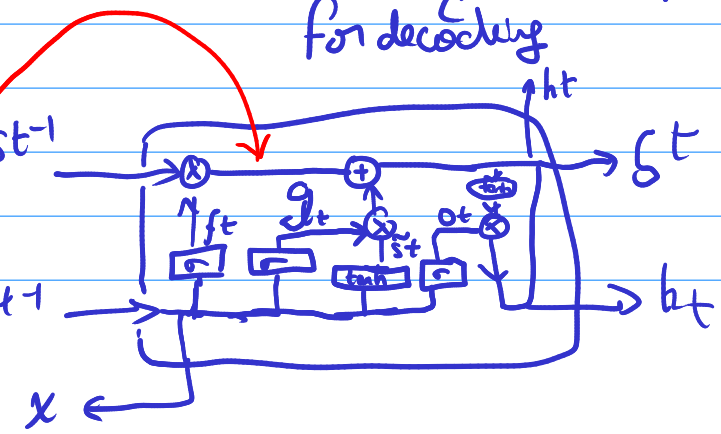
So now, we need to find out

$b^f\ U^f\ W^f$ $\quad b, W, V, e$ $\quad b^g\ U^g\ W^g$

$b^g\ U^g\ W^g$ — fraction for remembering

b, W, V, e — for decoding

our backprop is now on 's' and is very simple!!



the derivative of the product will give a sum though instead of products maybe that's why gradients don't explode

weighted average

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma\left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

resets prev input

- Where **u** stands for the update gate and **r** for reset gate. Their value is defined as usual:

$$u_i^{(t)} = \sigma\left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)} \right) \quad \text{and} \quad r_i^{(t)} = \sigma\left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)} \right)$$

GRU