# DMML, 19 Mar 2019

## Neural networks

### Structure

### Backpropagation algorithm

Activation function : sigmoid

Cost function : MSE

Training :

Mini-batches – stochastic gradient descent

Epoch – full training sample

# Hyperparameters

Structure – no. of layers, size of each layer

Batch size
Epochs

Backpropagation

Activation function

Cost function

By default, sigmoid + MSE

Is this always a good choice?

Expectation of learning weights & biases

- Get closer to good value, rate slows down
- Very far away – progress is rapid

Nielsen example

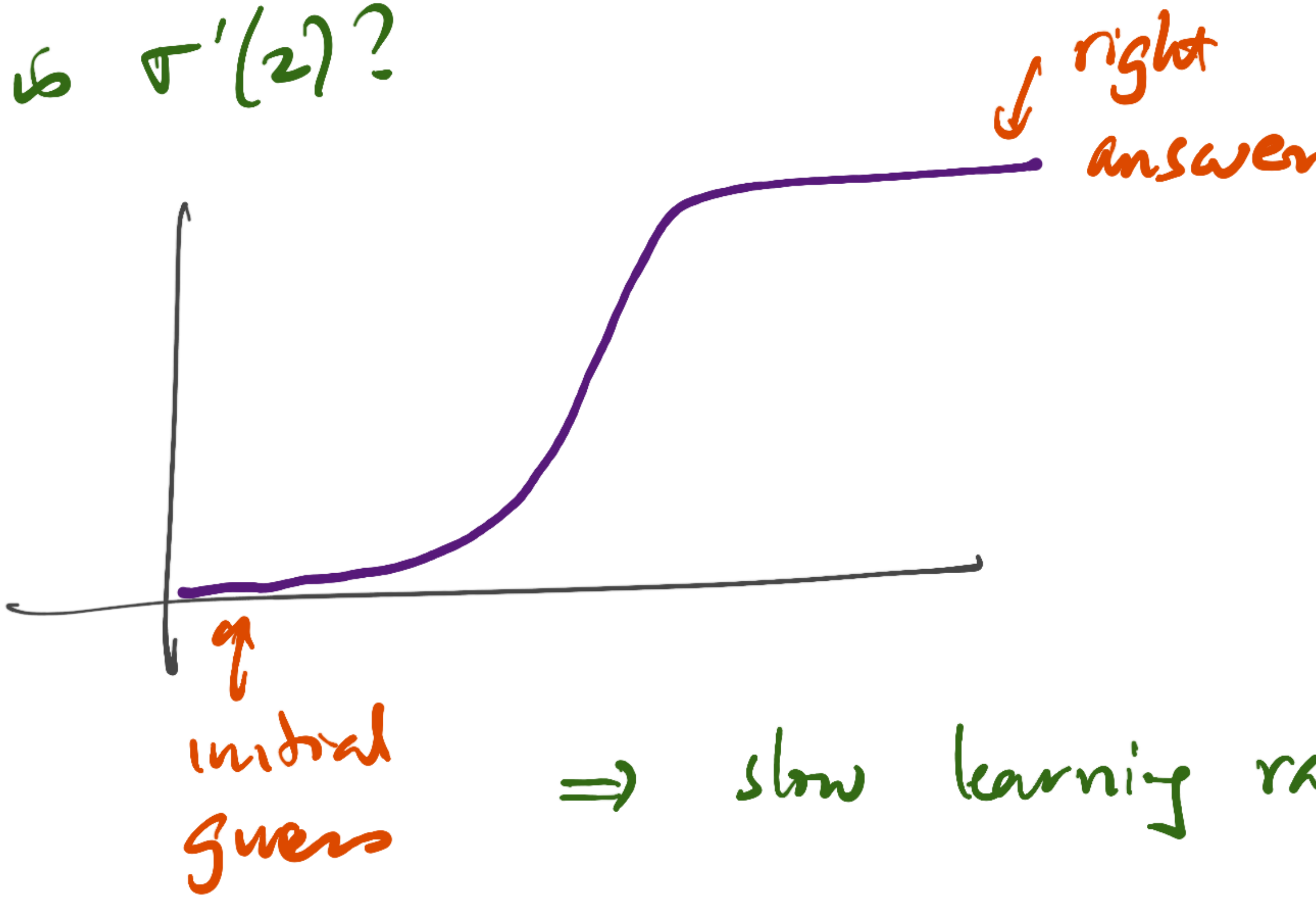
- Second assumption is not valid

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

$$\text{Cost} = \frac{1}{2n}\sum (y-a)^2$$

$\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$ are proportional to $\sigma'(z)$

What is $\sigma'(z)$?




$\Rightarrow$ slow learning rate

Want: $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$ proportional to $\|y-a\|$

By reverse engineering

Cross Entropy (not the "real" cross entropy)

$$\text{Cost} = \frac{1}{n} \sum_{x} y \ln a + (1-y) \ln (1-a)$$

Why? By reverse engg

Justify? $y=1$, $a \approx 1$ $\Rightarrow C=0$  $y=0$, $a \approx 0$ $\Rightarrow C \to 0$

Sigmoid $\quad \dfrac{1}{1+e^{-z}} = \sigma(z)$

Calculate $\dfrac{\partial C}{\partial w_j}, \dfrac{\partial C}{\partial b}$ for $C$ = cross entropy

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x \frac{\sigma'(z)\, x_j}{\sigma(z)(1-\sigma(z))} \quad \underline{\underline{(\sigma(z)-y)}}$$

$$a - y$$

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

cancels out

# Sigmoid + Cross Entropy

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j \left( \sigma(z) - y \right)$$

$$\frac{\partial C}{\partial b} = \frac{1}{n} \sum_x \left( \sigma(z) - y \right)$$

Another option for activation

0 - 9 digit recognition - 10 outputs,
take max

Softmax

$a_0$
$a_1$
$\vdots$
$a_9$

$$\frac{a_i}{\sum a_j}$$

add up to 1

Instead Softmax: $\dfrac{e^{a_i}}{\sum e^{a_j}}$

Use a different cost function

$- \ln(a)$ "log likelihood"

In practice

Sigmoid + Cross Entropy

Softmax + log likelihood

# Regularization

Smoothen out the learning process

Penalize large weights

Add -- $+ \underbrace{\lambda \sum w^2}_{\text{regularization term}}$

to cost function

# Deep learning

Theoretically, one hidden layer can approximate any continuous function

- High fan in / fan out of signals

Impractical

Build layers that make incremental progress

Image recognition

- layer detects horizontal edges
- layer " vertical edges

⋮

features

Combine layers to classify

- Multiple layers → deep networks

# Deep learning

- Neural network with "many" layers

By hand, "many" $\leq 10$

Brute force, $> 100$

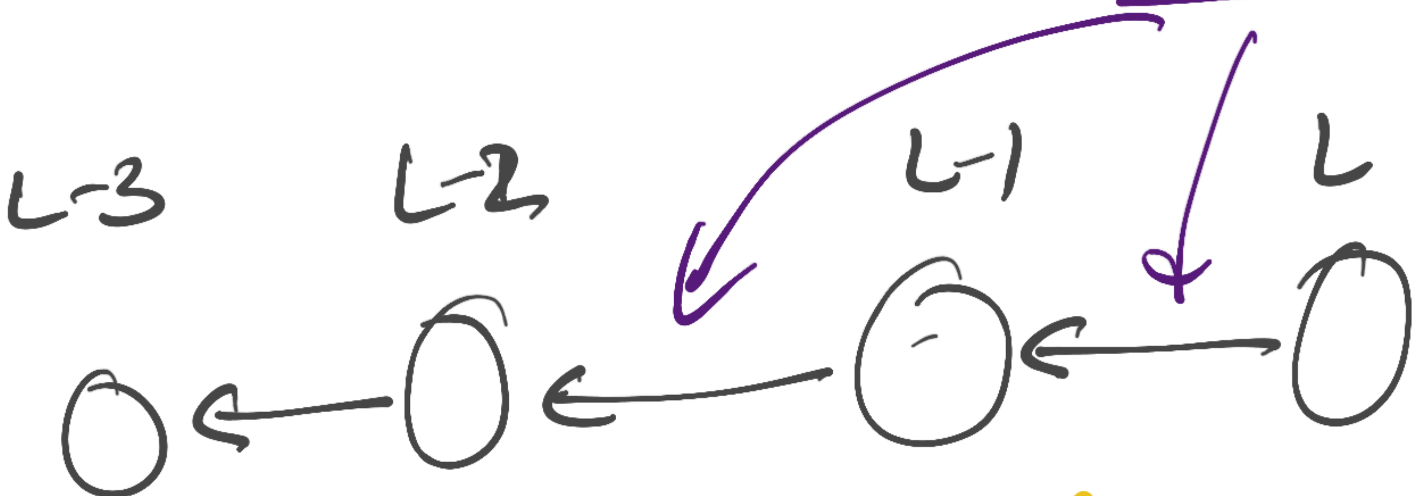
# Slowdown in learning rate

Intrinsic to backpropagation

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

Express $\delta_j^l$ in terms of $\delta_j^{l+1}$

$$\delta_j^l = \sum_k \delta_k^{l+1} \cdot \underline{w_{kj}^{l+1} \, \sigma'(z_j^l)}$$



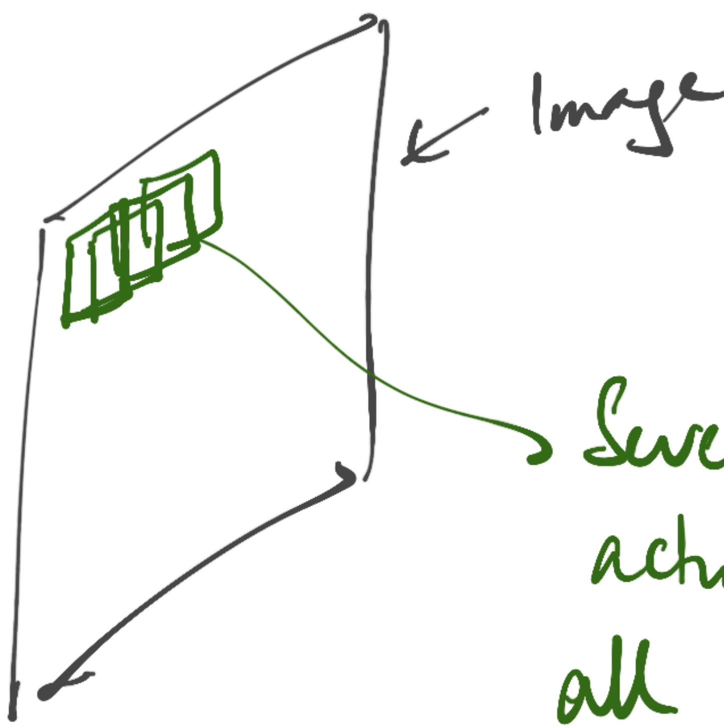

As we move left, $\delta_j^l$ "vanishes"

"Vanishing gradient" problem

## Solution?

Add structure to the network

Design a small "filter" that checks for a feature in a small part of input




Several copies in actual network, all identical

All copies of our filter have exactly same weights & biases

Convolution - Passing a filter over an input

Convolutional Neural Networks

Layer is a collection of identical small filters