

# Deep Learning from scratch with python

Sophie Seary  
Data Scientist @ Metis



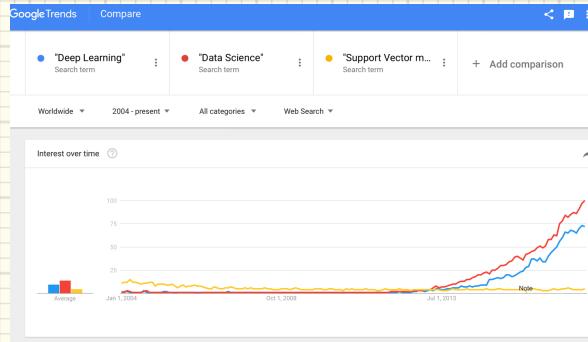
# My Deep Learning background



Both hot shots in  
Deep Learning now!

2008 Learned to do DL the hard  
way before it was cool!

2012 Switched fields to  
(proto-) Data Science



## Outline

- 1) Context
- 2) In theory
- 3) In practice

Context

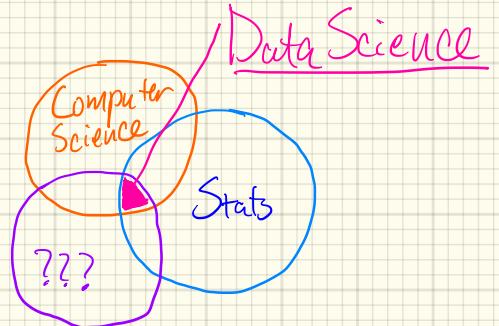
plus What is Data Science?

# What is Data Science?

- Statistics = Data Science ? (Wu, 1997)  
No!
- The sexiest job of the 21<sup>st</sup> century? (HBR)  
No!

# What Is Data Science?

- Extracting value (knowledge, insight, etc.) from data.
- Interdisciplinary
- Breadth of expertise  $\stackrel{?}{>}$  depth of expertise



# Batteries included python packages

## Tensorflow

- +/- Very python
- + many useful tools included (e.g Reinforcement learning)
- + extremely flexible
- + great docs
- +/- by Google™
- + GPU acceleration
- relatively slow
  - Replaces Theano

## Batteries included python packages

### 2) Caffe

- +/- great for vision, not intended for much else
- + model zoo w/ pretrained models
- poor documentation & support

## Batteries included python packages

### 3) Torch (Pytorch)

- + dynamic computational graphs (variable input)
- + fast
- more work / steeper learning curve
- not python but has an API

## Batteries included python packages (others)

4) Keras

5) MxNet

6) DL4J

7) Lasagne

8) ...

## Benefits of building from scratch

- depth of knowledge
- situation-specific modifications
- contribute to FOSS projects

Deep learning from scratch:

In theory

From the top: shallow learning



(Rosenblatt 1957)

- linear binary classifier
- supervised
- cannot learn XOR\*
- "cybernetics"

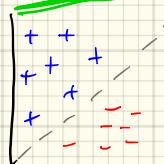
\* (Minsky + Papert 1969)

classify  $f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$

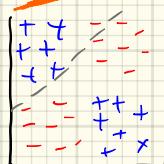
learn

$$w^{(t+1)} = w^{(t)} + (d^{(t)} - f_{\text{old}}^{(t)})x^{(t)}$$

Good

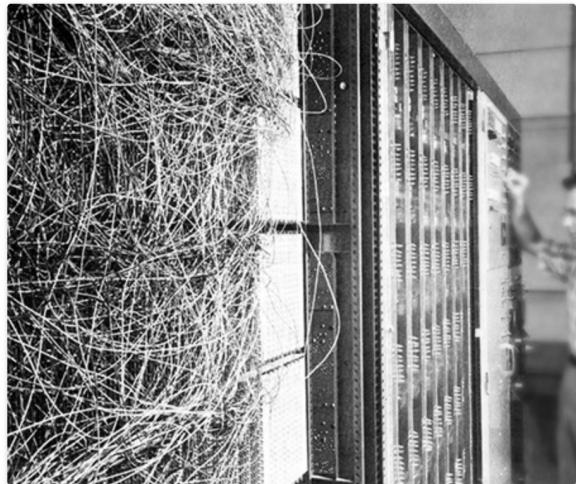


Bad





Rosenblatt, with the image sensor of the Mark I Perceptron (Source: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. *Neurocomputing* (Reading, Mass.: Addison-Wesley, 1990).)



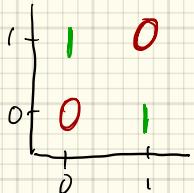
The Mark 1 Perceptron (Source: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. *Neurocomputing* (Reading, Mass.: Addison-Wesley, 1990).)

# Deep Learning, an example

(Goodfellow, Bengio, & Courville 2017)

Training Data

$$X = \begin{bmatrix} 0, 0 \\ 0, 1 \\ 1, 0 \\ 1, 1 \end{bmatrix} \quad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



Model

$$f(x; \theta) = w \cdot x + b$$

$$J(\theta) = \sum_i (f(x_i; \theta) - y)^2$$

Solution

(it's convex!)

$$w = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad b = 0.5$$

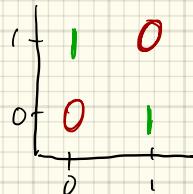
$$f(x; \theta) = 0.5 \text{ for all } x$$

THIS IS BAD!

# Deep Learning, an example

Training Data

$$X = \begin{bmatrix} 0, 0 \\ 0, 1 \\ 1, 0 \\ 1, 1 \end{bmatrix} \quad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



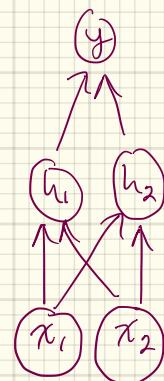
Model

$$f^{(1)} = \max\{0, w^{(1)} \cdot x + b^{(1)}\}$$

$$f^{(2)} = w^{(2)} f^{(1)} + b^{(2)}$$

$$w^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad b^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

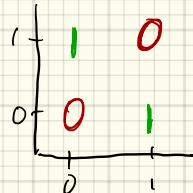
$$w^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \quad b^{(2)} = 0$$



# Deep Learning, an example

[Training Data]

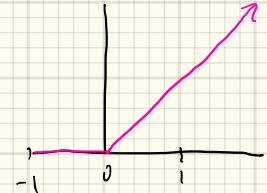
$$X = \begin{bmatrix} 0, 0 \\ 0, 1 \\ 1, 0 \\ 1, 1 \end{bmatrix} \quad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



[Example]

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\text{relu}(x) = \max(0, x)$$



[Model]

$$f^{(1)} = \max\{0, w^{(1)} \cdot x + b^{(1)}\}$$

$$f^{(2)} = w^{(2)} f^{(1)} + b^{(2)}$$

$$w^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$b^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$$w^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

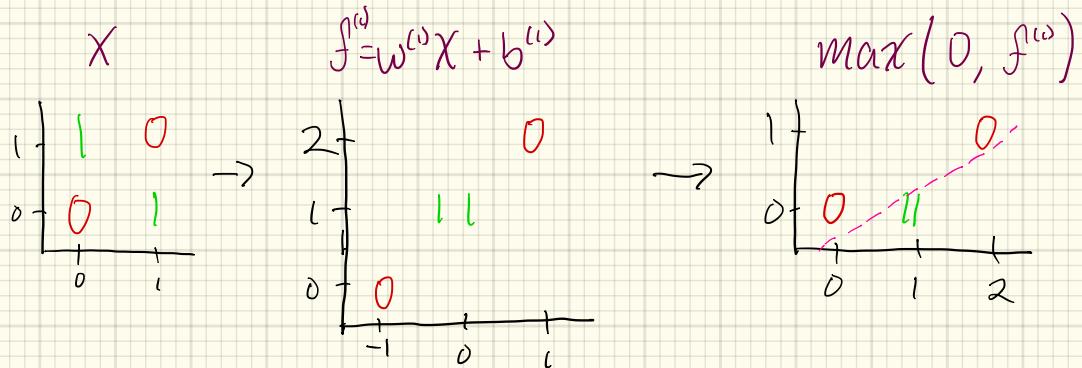
$$b^{(2)} = 0$$

$$w^{(1)} \cdot X + b^{(1)} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$f^{(1)} = \max\{0, w^{(1)} \cdot X + b^{(1)}\} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$f^{(2)} = w^{(2)} f^{(1)} + b^{(2)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

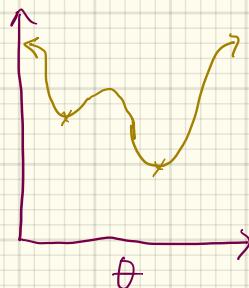
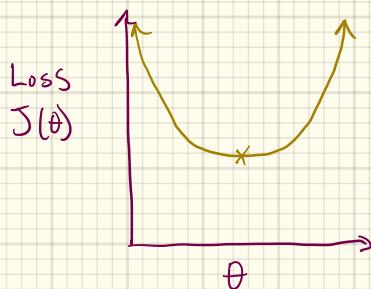
# What's special about deep networks?



Non-linearities!

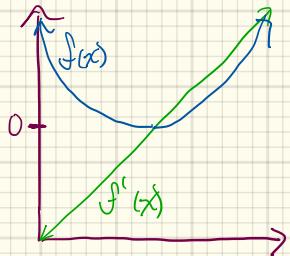
# Non-linear transfer functions

- allow deep networks to approximate any function
- Convex optimization  $\rightarrow$  non convex



multi-layer networks  
with linear transfer  
functions are equivalent  
to single-layer networks

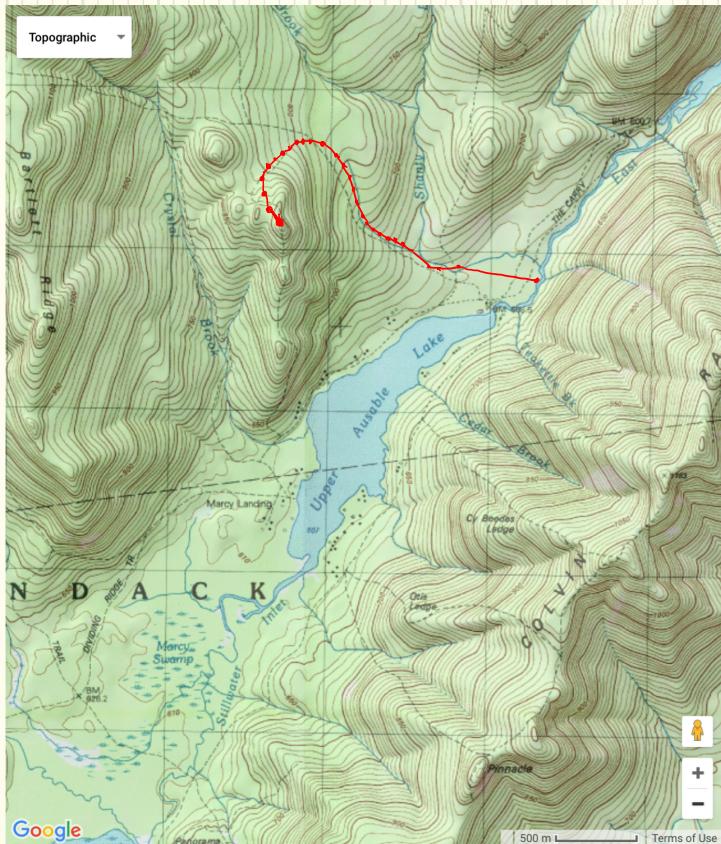
# Gradient Descent (Cauchy 1847)



for a small enough  $\epsilon$ ,  
 $f(x - \epsilon \text{ sign} f'(x)) < f(x)$

- Greedy
- only looks at first derivative (e.g. not at curvature)
- Not limited to neural nets

# Gradient Descent



Gradient wrt  $x_i$  ( $x$  is a vector)

$$\nabla_x f(x)$$

$-\nabla_x f(x) \rightarrow$  direction + Magnitude of  
steepest descent

Update :

$$x' = x - \epsilon \nabla_x f(x)$$

$\overbrace{\quad\quad\quad}^{\text{Learning rate}}$

# Gradient Descent More data $\rightarrow$ more problems

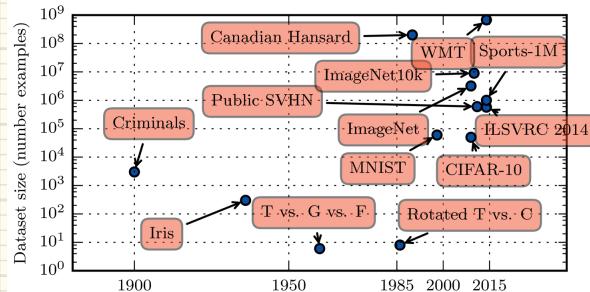
Batch full set of training data,  $X = \{x_0, x_1, \dots, x_n\}$

$$J(\theta) = \overline{\mathbb{E}_X} L(X, Y, \theta) = \frac{1}{|X|} \sum_{x \in X} L(x, y, \theta)$$

$$\nabla_{\theta} J(\theta) = \frac{1}{|X|} \sum_{x \in X} \nabla_{\theta} L(x, y, \theta)$$

$$\theta' = \theta - \epsilon \nabla_{\theta} J(\theta)$$

Time complexity:  $O(|X|)$



## Stochastic Gradient Descent

minibatch: some subset of the training data  
 $B \subset X$

$$g = \frac{1}{|B|} \nabla_{\theta} \sum_{x \in B} L(x, y; \theta)$$

$$\tilde{\theta} = \theta - \epsilon g$$

Time complexity:  $O(1)$  wrt  $X$  (big caveat)

- depends on complexity of model, not  $|X|$

# Computational graphs and forward propagation

$$y \rightarrow J(\theta)$$

$$\uparrow$$

$$h$$

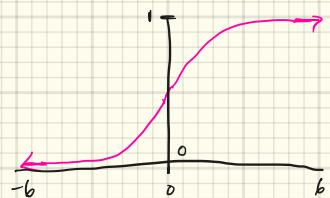
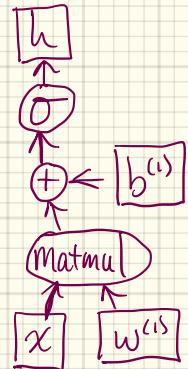
$$x$$

$$h = f_h(x) \rightarrow o($$

$$\text{plus } (b^{(i)},$$

$$\text{Matmul}(x, w^{(i)}) ) )$$

$$\text{sigmoid } o(x) = \frac{1}{1+e^{-x}}$$



## The chain rule

$$y = g(x), \quad z = f(g(x)) \quad \boxed{x} \rightarrow \boxed{g(\textcircled{l})} \rightarrow \boxed{y} \rightarrow \boxed{f(\textcircled{l})} \rightarrow \boxed{z}$$

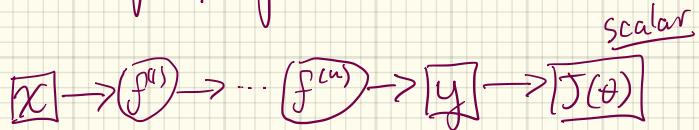
adjust  $x$  in order to descend along  $z$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} \quad (\text{scalars})$$

$$\nabla_x z = \left( \frac{\partial y}{\partial x} \right)^T \nabla_y z \quad (\text{vectors})$$

Jacobian, like a gradient between Matrices

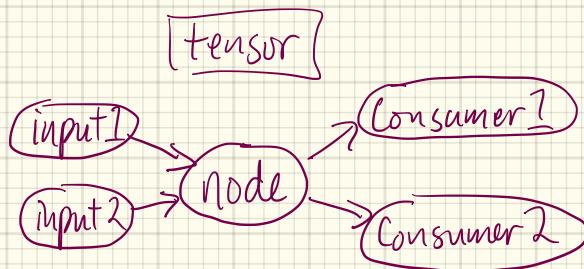
# Back propagation (Rumelhart et al 1986)



Graph: 1) operations (relu,  $\sigma$ ,  $+$ ,  $\cdot$ )

2) variables ( $w, b$ ) things to be updated

3) placeholders ( $x, y$ )



slides at [soph.info/scratch.pdf](http://soph.info/scratch.pdf)  
code on github: [sophiaray/sophscratch](https://github.com/sophiaray/sophscratch)







