Deep learning from scratch with python

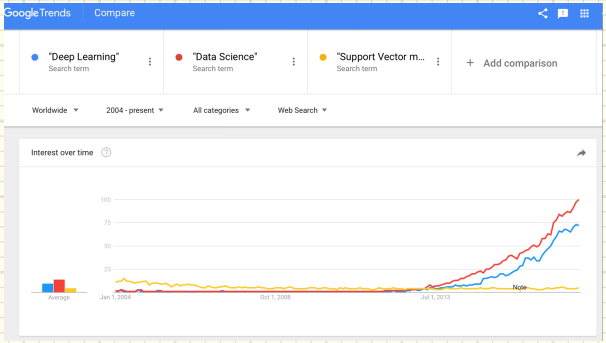
Sophie Searcy Data Scientist @Metis

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

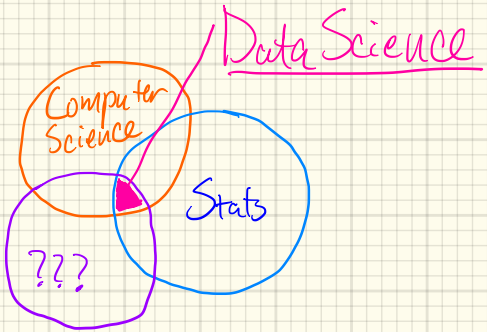
Context

plus What is Data Science?

What is Data Science?

- Statistics = Data Science ? (Wu, 1997)
  NO!
- The sexiest job of the 21st century? (HBR)
  NO!

What is Data Science?



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

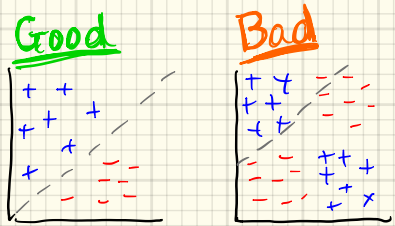
Perceptron 🤖 (Rosenblatt 1957)

- Linear binary classifier
- supervised
- cannot learn XOR*
- "cybernetics"

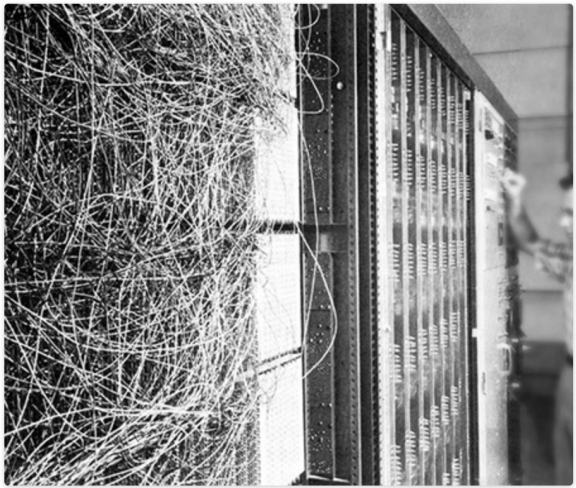
*(Minsky & Papert 1969)

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

learn $w^{(t+1)} = w^{(t)} + (d^{(t)} - f(x)^{(t)})x^{(t)}$



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).)



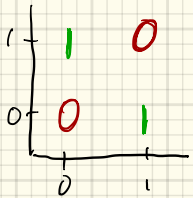
The Mark 1 Perceptron (Source: Arvin Calspan Advanced Technology Center; Hecht-Nielsen, R. Neurocomputing (Reading, Mass.: Addison-Wesley, 1990).)

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!)

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

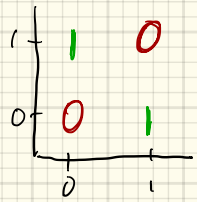
$f(x;\theta) = 0.5$ for all x

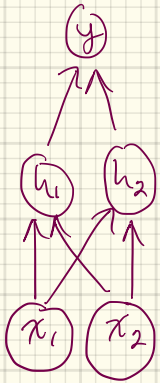
THIS IS BAD!

Deep Learning, an example

Training Data

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





Model

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

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

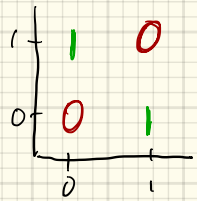
$$W^{(1)} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad b^{(1)} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

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

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

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

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

Model

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

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

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

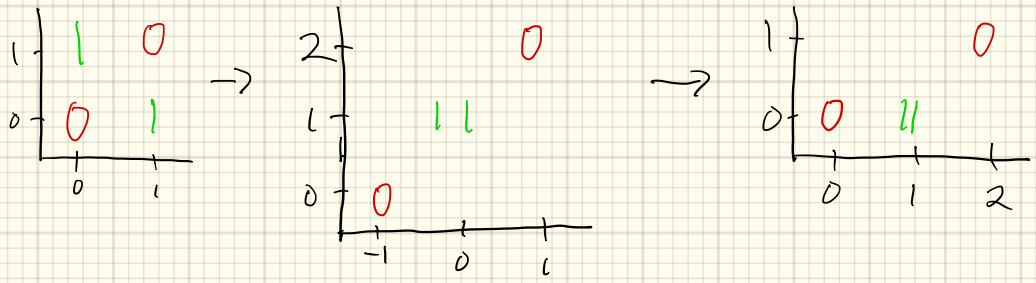
$$w^{(2)} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} \qquad b^{(2)} = 0$$

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

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

What's special about deep networks?

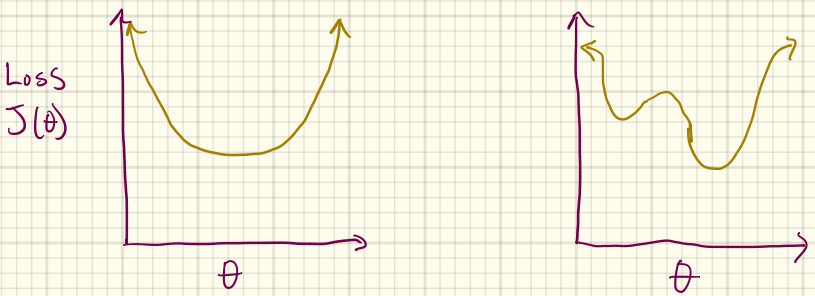
X $\quad f^{(1)} = W^{(1)}X + b^{(1)}$ $\quad \max(0, f^{(1)})$



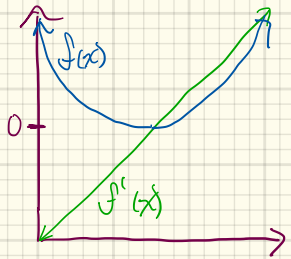
Non-linearities!

Non-linear transfer functions

- allow deep networks to approximate <u>any</u> function
- Convex optimization → non convex



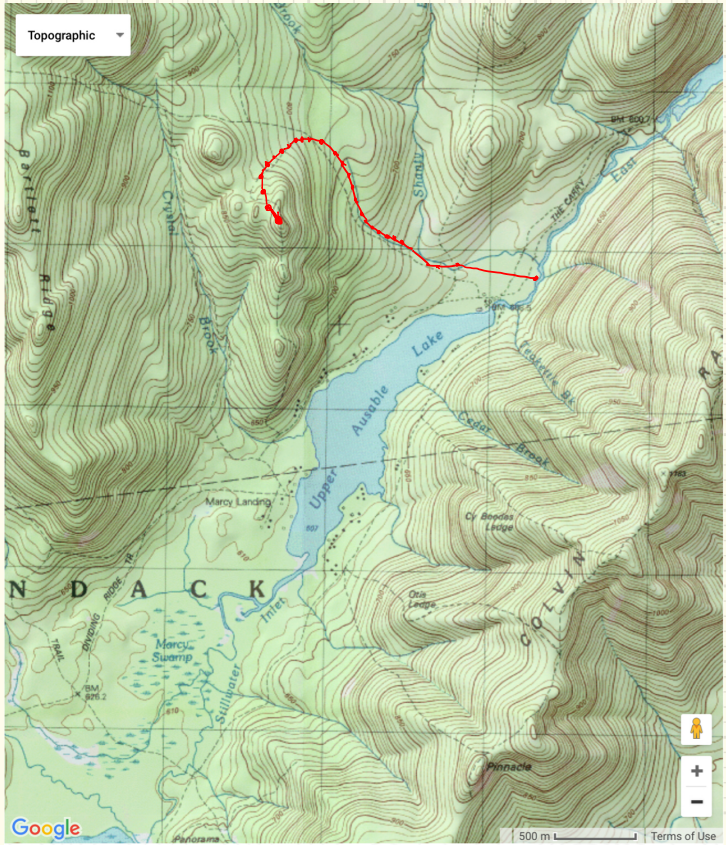
Gradient Descent (Cauchy 1847)



for a small enough ϵ,

$$f(x - \epsilon \operatorname{sign}(f'(x))) < f(x)$$

Gradient Descent



Gradient wrt x: (x is a vector)

$\nabla_x f(x)$

$-\nabla_x f(x) \rightarrow$ direction & magnitude of steepest descent

Update:

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

(ϵ: learning rate)

Gradient Descent — More data → more problems

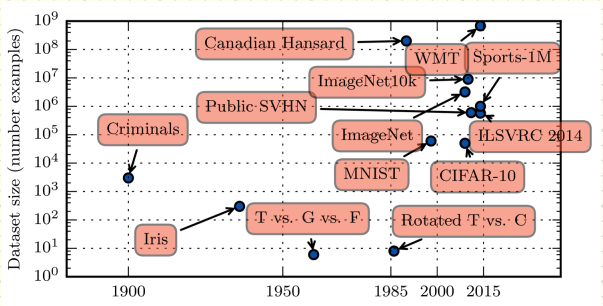
Batch full set of training data, $X = \{x_0, x_1, \ldots x_n\}$

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

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

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

Time complexity: $O(|X|)$



Stochastic Gradient Descent

minibatch: some subset of the training data

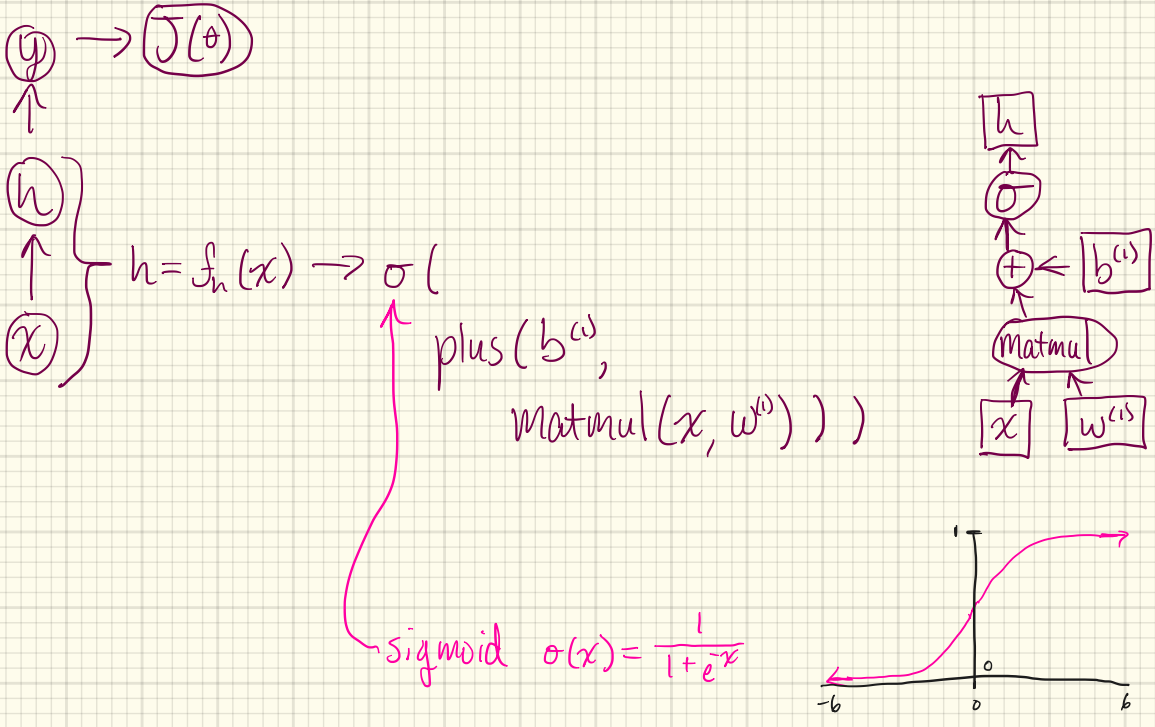
$B \subset X$

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

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

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

Computational graphs and forward propagation



The chain rule

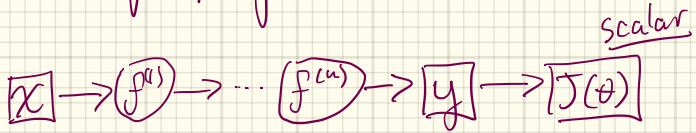
$y = g(x)$, $z = f(g(x))$ $\quad x \to g() \to y \to f() \to z$

adjust x in order to descend along z

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

Jacobian, like a gradient between Matrices

Back propogation (Rumelhart et al 1986)



Graph: 1) operations (relu, σ, +, ·)

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

3) placeholders (x, y)