Chapter 1 : What is Deep Learning

AI = effort to automate intellectual tasks normally performed by humans

Symbolic AI : a sufficiently large set of explicit rules for manipulating knowledge stored in explicit databases
↳ dominant paradigm 1950 - 1980

Although the Turing Test has sometimes been interpreted as a literal test Turing merely meant it as a conceptual device in a philosophical discussion about the nature of cognition

ML → a very hands-on field driven by empirical findings and deeply reliant on advances in software and hardware

"Deep" → for deep learning means successive layers of representation

loss function = objective function = cost function
↳ Backpropagation algorithm

There were already 2 AI winters → 1360, 1990

Probabilistic ML → application of the principles of statistic to data analysis
Ex. Naive Bayes (it assumes that all the data points are independent)
Logistic regression

Kernel Methods → classification algorithms best known for SVMs
1) the data is mapped to a new high-dimensional

representation where decisions boundaries can be expressed as hyperplanes

2) Find the decision boundary by maximising the distance between the hyperplane and the closest datapoints from each class

The thing is that it can be computationally expensive to map all the points to a higher dimensional space
=> kernel trick allows you to find the distance between two points from the initial space into the target space

Decision Trees -> flowchart-like structure that lets you classify stuff
=> Random forest.
=> Gradient Boosting Machines

Deep learning -> one of the reasons why it is so cool is because it automates feature engineering

Best libraries according to Kaggle Teams:
Keras, LightGBM, XGBoost, PyTorch, TensorFlow, Sklearn, FastAI, Caffe

2009 - 2010 -> better activation functions
↳ better optimisation schemes (RMSProp & Adam)

2011 - 2016 -> improved gradient descent
↳ batch normalisation, skip connections

Chapter 2: mathematical building blocks of neural networks

Softmax classification layer -> return an array of 10 probability scores (summing to 1)

optimiser -> the mecanism through which the model will update itself based on the training data it sees, so as to improve its performance

loss function -> how the model will measure its performance on the training data, and how it will be able to steer itself in the right direction

metrics to monitor during training and testing

tensor = container for data

number -> rank 0 tensor
vector rank-1 tensor
matrix rank-2 tensor
...

Tensor
- number of axes (rank)
- shape: tuple of integers that describes how many dimensions the tensor has along each axis
- datatype (or dtype)

Deep learning -> usually process data in batches (chunks)

Common data tensors

1) Vector data
usually rank 2 tensors (samples, features)

2) Time series data
usually rank 3 tensors (samples, timestamp, features)

3) Image tensors (samples, H, W, colors)
(128, 256, 256, 3)
samples -> height -> width -> RGB

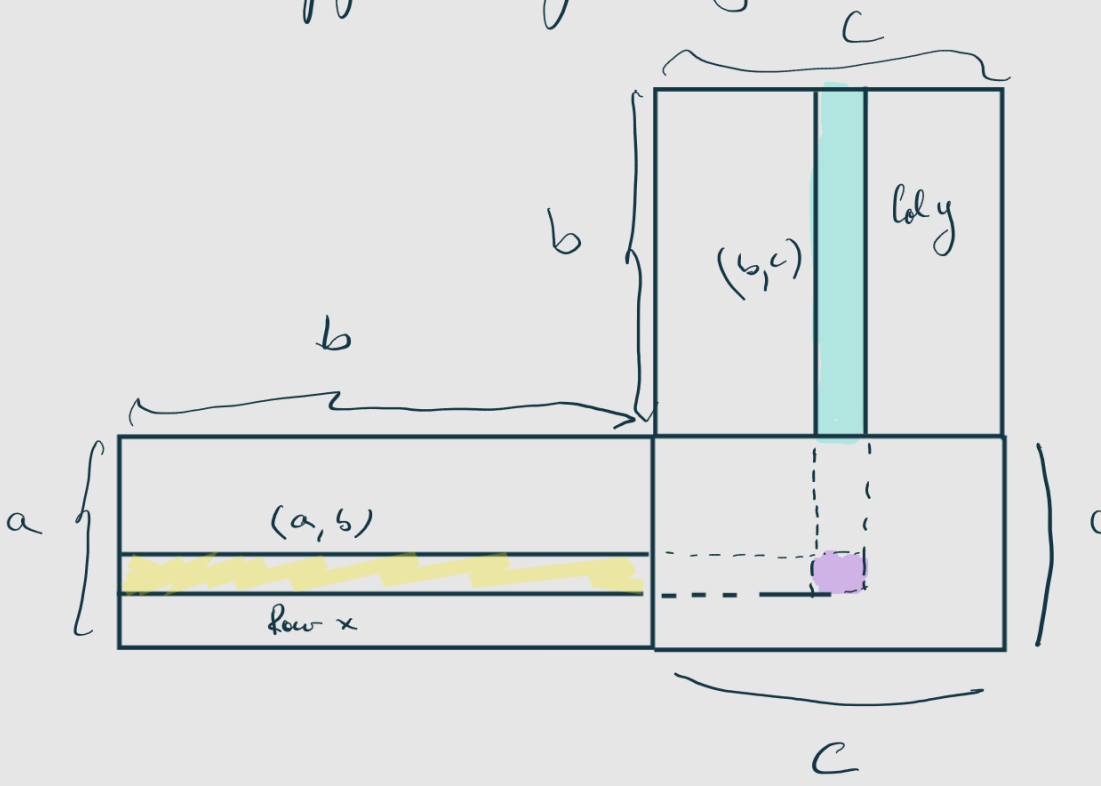
4) Video data
(samples, frames, H, W, colors)

Keras layers Dense(512, activation="relu")

=> identic output = relu(dot(input, W) + b)

Element wise operations -> naive = step by step on each

↳ python style using tensors

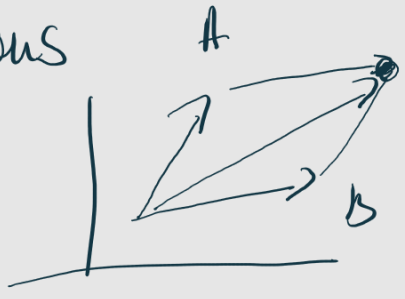


=> Reshaping Tensors

Geometric interpretations

1) adition

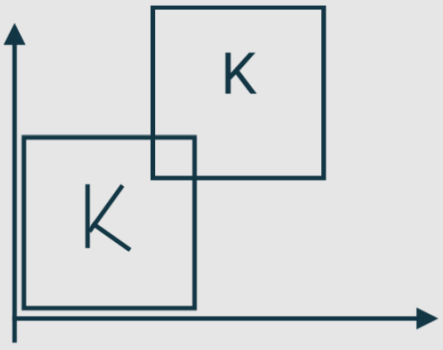
= translating an object



$\begin{bmatrix} \text{Horizontal} \\ \text{Vertical} \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$

2) Translation

= apply adition to a set of points



3) Rotation

=) counter clockwise rotation by angle θ =) via dot product

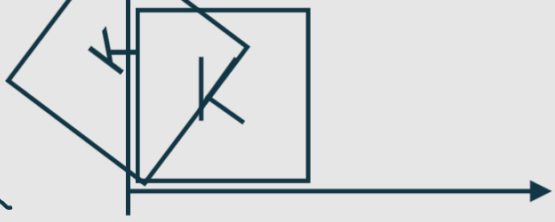
$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

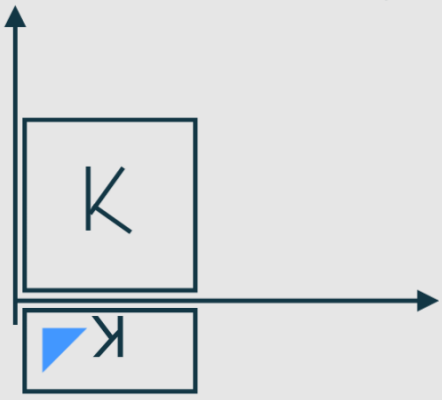
4) Scaling

=> vertical & horizontal

=> with a dot product with a diagonal matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & -0.5 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



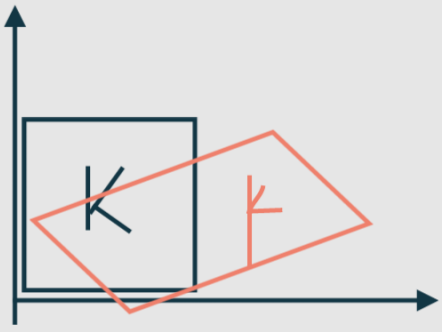


5) Affine transformations

=> combination of linear transformation and translation

$$y = W \cdot x + b$$

This is what a dense layer is performing. A dense layer without an activation function is an affine transformation



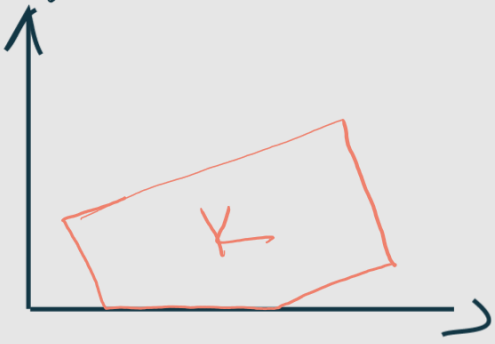
A dense layer with Relu activation

$\text{affine}_1(\text{affine}_2(x))$ Equivalent $\longrightarrow$ $\text{affine}_3(x)$

where affine 3 can be computed

=> no need for multiple layers

affine + Relu $\longrightarrow$



Gradient based optimisation

- weight
- bias

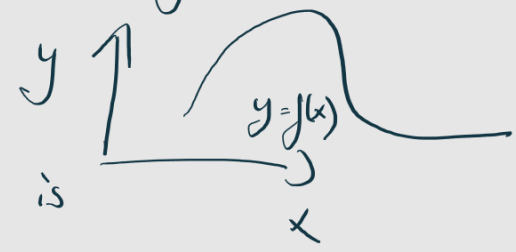
Training loop

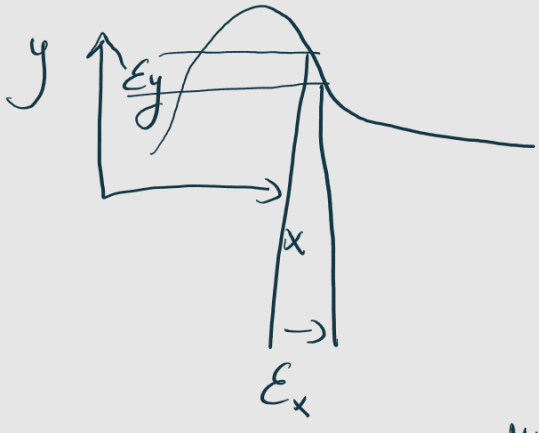
1) Draw a batch of training samples
2) Forward pass => obtain predictions

3) Compute the loss of the model on the batch
4) Update the weights of the model in a way that slightly reduces the loss of the batch

$f(x) = y$

because the fuction is continuous





$f(x + \varepsilon_x) = y + a \cdot \varepsilon_x$

The slope a = derivative

$a < 0 \Rightarrow x \uparrow, f(x) \downarrow$

$a > 0 \Rightarrow x \uparrow, f(x) \uparrow$

magnitude of a $\Rightarrow$ how much it increases/decrease the fuction

$\Rightarrow$ how abrupt is the slope

```
y_pred = dot(W, x)
loss_value = loss(y_pred, y_true)
```

$\Rightarrow$ how do we update W to make the loss value smaller?

loss_value = f(W)

we can compute partial derivatives with respect to some values

grad(loss_value, W0)[i, j] $\Rightarrow$ indicates the direction and magnitude of change when modifying W[i, j]

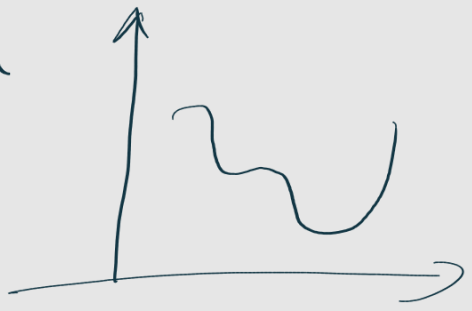
$\Rightarrow$ you move W in the opposite direction of the gradient

$W_1 = W_0 - step * grad(f(W_0), W_0)$

1) Draw a batch of training examples
2) Obtain predictions
3) compute the loss
4) Back propagation (compute the gradient of the loss with regards to the models parameters)
5) Move the parameters a little in the opposite direction from the gradient

True SDG → one sample
mini-batch → with a mini batch
Every step on all data => batch gradient descent

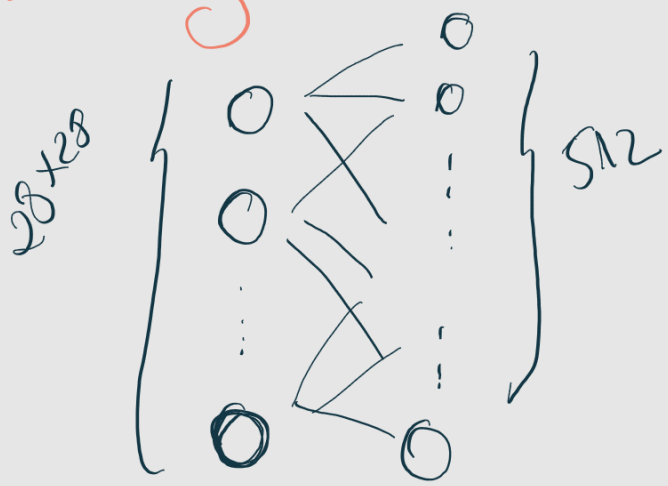
Momentum → convergence speed
→ Local Minima



The chain Rule

$$grad(y, x) = grad(y, x_3) * grad(x_3, x_2) * grad(x_2, x_1) * grad(x_1, x)$$

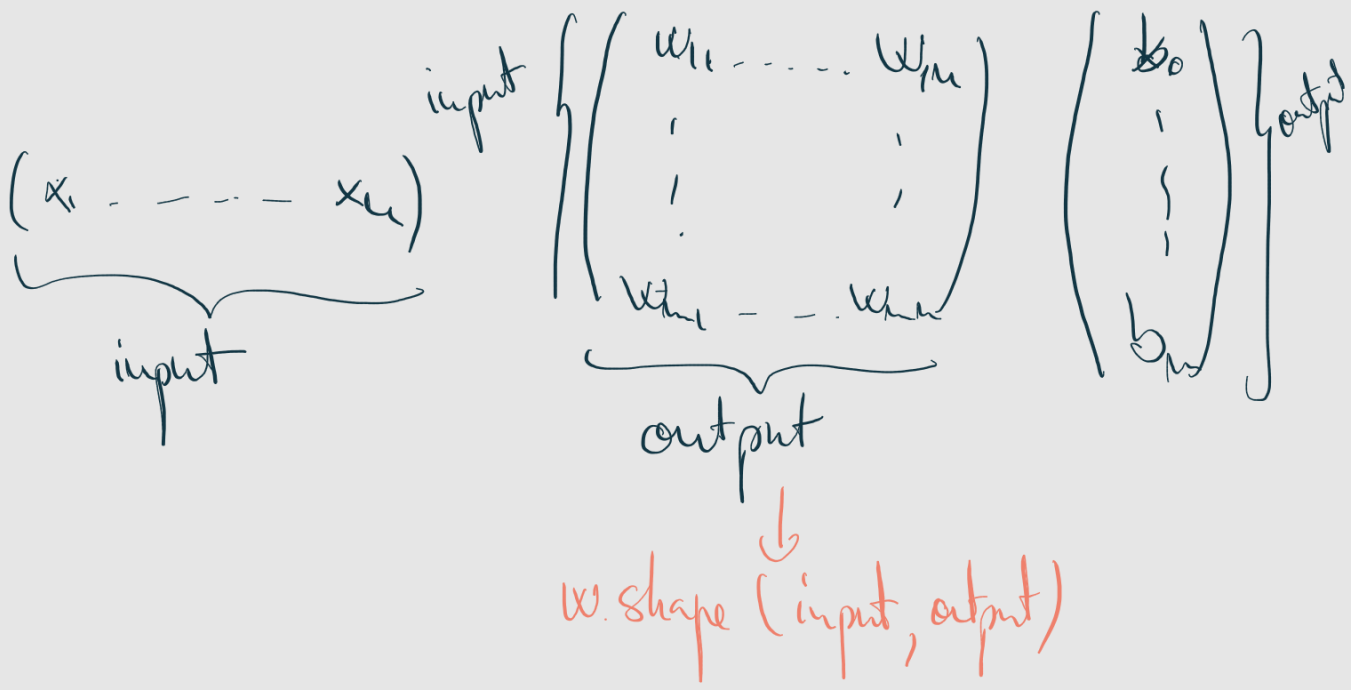
Reimplementing MNIST



Naive Dense
↳ 1 layer

output = activation (dot(W, input) + b)

W b



W.shape (input, output)