Data Science: Deep Learning in Python (part 1)

SECTION 1: WELCOME

- Introduction and outline (video 1)

- This course → Fundamentals of Neural Networks

  objective → Build a strong foundation for more advanced ideas such as CNN's or RNN's.

- what is DL useful for

  ↳ Like all ML → Making predictions

  DL → Does it better ↗ Beat an AlphaGo champ!

  ↳ e.g.
  - Google's AlphaGo
  - Self-driving cars
  - Google search
  - Stock prediction
  - Recognizing faces in an image
  - Predict outcome of next presidential election.

Under active development

- when it comes to ML algorithms → DL will give you the best performance.

Learning by example -> Course project

- Engineer at e-commerce company
- Want to predict user actions based on data you've collected from your site.

Techinques Learned —Apply→

↳ outline:

- Softmax -> Going from binary to multi-class classification.
  -> we will learn to discriminate between K number of things.
- How to train a NN -> Backpropagation
  ↳ Extension of Logistic Regression
- XOR / Donut problems -> NN are capable of automatically discriminate features. -> No more feature engineering.
- Building a NN -> 1) Numpy -> "By hand"
  2) Tensorflow
     ↳ "plug 'n play"
  3) Tensorflow playground ->
     Visualize what a NN is doing.

- More projects -> Facial expression Recognition

- Where does this Course fit into your DL studies? (Video 2)

- DL -> Huge topic -> Important to have this discussion.
  - Good idea to understand previous ideas and how DL depends on these ideas.

- Before -> Logistic Regression (Neuron) LoR
  -> Linear Regression LR
  - Grand father of ML
  - Foundations

Calculus & Probability

Ideas that are the foundations of NN

- Have some data
- Fit a model to the data
- The loss -> Negative log-likelihood
- Minimize loss via gradient descent (GD)

- After -> Apply these ideas to more complex things
  - we may learn about more advanced libraries such as Keras, PyTorch, TensorFlow and Theano.

-> These libraries make writing complex NN a lot easier.

-> Training takes a really LONG time.

GPU's improve speed by orders of magnitude

Used by these modern DL libraries

Applied

-> Specialized architectures (CNN's, RNN's, Recursive & Memory NN)

-> Specialized problems (CV, NLP, RL)

↳ useful in a wide variety of applications

- This course (DOESN'T)
  - ↳ • Abstraction approach? -> (NO)
    - ↳ Give me the API so I don't have to worry about the details.

OK for narrow problems

- Problem arises -> Need to do something different -> Now stuck

!

Smallest change -> Huge obstacle
↳ You won't understand what you are doing.

- Due to this, it is important to understand DL on more than a superficial level.

Any random coder can copy some KERAS Lines. -> If it is this easy -> why should they hire you?

This Course -> (DoES)

- We cover the above details
- A bit more complex than LoR / LR
  -> Not so specialized to require a Library.
  -> Back propagation foundation -> How a NN learns
  ↳ The basis of CNN's, RNN's and pretty much all NN's.

-> You will learn how DL really works.

Who this course is NOT FOR

- Not for people without the neccessary background knowledge.
- Not a "for dummies" course
- Those looking to learn about advanced architectures. -> ~~CNN's~~

Learn about NN first!

- where to get the code (video 3)
- Github url:

  https://github.com/lazyprogrammer/machine_learning_examples

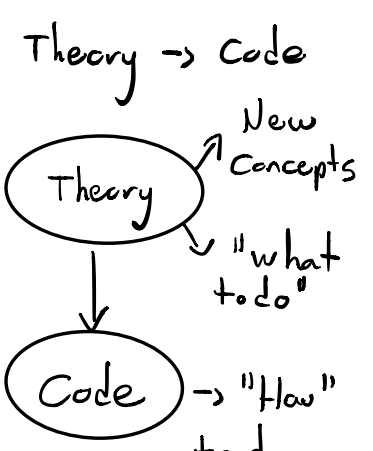
  - class folder: ann_class
  - Project folder: ann_logistic_extra
  - Extra repositories -> Pay attention at the lectures
- In the terminal: (Do NOT copy)

```
git clone <url>
git pull -> Do it often to ensure you have the latest version.
```

- Do NOT fork -> you won't get updates and it may confuse other students.

- How to succeed in this course? (video 4)

Guideline #0 -> write code!



- You should be able to go from theory to code on your own.
- Good to start from scratch
  - Whenever you write the same code.
- Another practice opportunity -> Modify your own code for your own data sets.
  - Changes are minimal regardless of the application.

- watch the lectures in order! 1, 2 85
- All that we code comes from previous lectures.

Guideline #1 -> • Make generous use of the QA forum.
• Note some questions might require "research" work.

Great insight comes with the appropriate guidance.

Guideline #2 -> Meet the prerequisites
-> Some review will be provided

Guideline #3 -> Code! -> Implement everything in code.

SECTION II
REVIEW

• Review Section Introduction (Video 5)

Review Section -> Quick review of the prerequisites needed to understand this course.

The "Neuron" -> Major focus -> Logistic Regression
Review
A NN -> Network of Neurons

-> ML -> Nothing but a geometry problem.

Mechanics of LoR { ① -> How to make predictions
② -> How to train a LoR Model.

① -> Must be able to make predictions before training.

What does Machine Learning do? (Video 6)

- (ML) -> Nothing but a geometry problem.

  Nothing but a "bunch" of dots in a grid -> It is what we do w/ this dots that makes ML what it is.

- Most basic ML problem -> we have an input in the x-axis and we would like to predict what's in the y-axis.

  (LR) problem. -> GOAL
  =
  Find the line that best fits the data

  Line of best fit.

Writing the model

$\hat{y} = wx + b$, x = height, $\hat{y}$ = predicted height

If we have more than 1 input: → Real world problems!

$$\hat{y} = w_2 x_2 + w_1 x_1 + b = \boxed{w^T x + b}$$

Convention: $x = 0$ → Denotes the size of w & x

Classification vs. Regression

- Regression → Tries to predict a continous value. → LR provides the groundwork necessary to progress to DL.
- Classification → Tries to predict a category

  e.g → Input = (Height, weight) → Prediction

  - 2 categories → Binary classification ⇐ whether or not the patient has high blood pressure (1) or not (0).

    ⇓

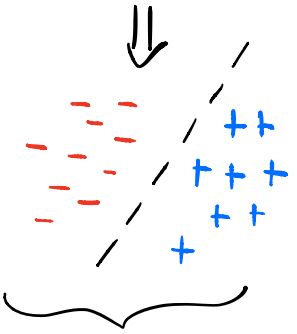
    Tries to predict the label out of two different categories.
  - LoR → Linear model for classification → uses a line or a plane for the model. → Provides a (boundary) ← Separates (+) & (−) examples.

All data is the same

↳ The example above had x_1 = height
x_2 = weight
and we were trying to predict $\hat{y}$ = High blood pressure

- what if we switched the features x_1 = weight
x_2 = height
- what if we have a completely different problem? → (# hours studied, Midterm grade) ⇒ Pass Final?

No PROBLEM ⇒ The semantics of the problem has chaged, however, the mechanics of the problem hasn't.



Problem still is → "we have this blue dots & red dots and we need to find the line that best separates the dots"

ALL DATA IS THE SAME ⇐

No matter the nature of the problem.

Summary

↳ ML is nothing but a geometry problem.

- You can plug in any data set from your own area of interest
- Code doesn't change
- Algorithms don't change.

} ML works no matter where your data comes from.

Neuron Predictions (video 7)

LoR → Most important prerequisite of the course.

↳ A NN is just a network of Logistic units.

In LoR, we separate our categories with a line or a plane of the form:

$$w^T x + b = 0$$

Note that this equation will return a number but we need to predict a category.

How do we go from a continous value to a category? → ENCODING

Making Predictions

↳ • we encode our categories as 0 and 1. ⇝ using a sigmoid function

• Sigmoid

↳ $\sigma(z) = \frac{1}{1+e^{-z}}$

where: $0 \leq \sigma(z) \leq 1$

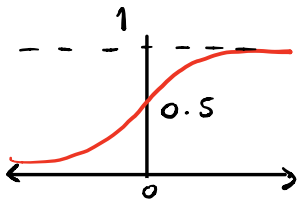
Now, consider:

$$P(y=1 \mid x) = \sigma(w^T x + b),$$

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

Note that if $a \to +\infty \Rightarrow \sigma(a) = 1$

$a \to -\infty \Rightarrow \sigma(a) = 0$



⇒ we typically use $a = 0.5$ to make predictions:

If $w^T x + b < 0 \Rightarrow$ output < 0.5

$w^T x + b > 0 \Rightarrow$ output > 0.5

Recall we can only predict 0 or 1

use the round() function in Python ⇝ Thus:

$P(y=1|x) = 1$ if $w^T x + b > 0$

$P(y=1|x) = 0$ if $w^T x + b < 0$

Python

Dotting vs. Looping

Loop method: (Accumulator) → slow

```
result = 0
for i in range(D):
    result += a[i] * b[i]
```

$$a^T b = \sum_{i=1}^{D} a_i b_i$$

Numpy method: → optimal

```
a.dot(b)
```

- Calculating Multiple predictions
  - usually, we are given a set of samples X of shape N x D
    - where: N: Number of samples
      D: Number of features
  - Another Convention → Lowercase letter for 1 - sample (A Vector)
    → uppercase letter for multiple samples (A matrix)

- One way to calculate multiple predictions:

```
predictions = []
for i in range(N):          -> Not Ideal (For-loop)
    p = np.round( sigmoid( w.dot(X[i]) + b ) )
    predictions.append(p)
```

- Better:

$$\text{predictions} = \text{Sigmoid}\left(X.\underbrace{\text{dot}(w) + b}_{\text{No for-loop}}\right)$$

$\downarrow$

Note: $X \in \mathbb{R}^{N \times D}$

$w \in \mathbb{R}^{D \times 1}$

$\Rightarrow Xw \in \mathbb{R}^{N \times 1}$ (X.dot(w) + b)

and $\underbrace{b \in \mathbb{R}}$

we add this scalar using Python Broadcasting.

The scalar is reshaped to size $N \times 1$ and added element-wise to all N elements of Xw.

- Summary (1L)
  - How to make predictions with a LoR model.
  - How to choose w? -> Training the model!

Neuron Training (video 8)

Review - Learning

- The "learning" part of Machine Learning simply means finding the weights of the model.

To DL analogies w/ the brain are only analogies:

↳ Common analogy -> The strength of the connections b/w synapses in your brain change when you form new memories.

A hypothesis ONLY! -> • We Do NOT know how memories are formed.

New research suggests memories could be stored in RNA. -> could be even transferred b/w organisms.

- The cost/loss function -> Central component in Learning

↳ Perspective -> ML is nothing but a probability problem.

-> we are solving a maximum likelihood problem

↳ e.g. we'd like to determine p(heads) from a series of flips:

↓

- Flip it 1000 times and keep track of the results → Set up your likelihood function & minimize it wrt p(heads)

⇓

closed form solution ↳ Likelihood = $p(\text{heads})\, p(\text{heads}) \cdots p(\text{tails})\, p(\text{tails})$

$$p(\text{heads}) = \frac{\#\,\text{heads}}{\text{total \# of coin tosses}}$$

- **Binary Cross-entropy**

↳ Negative log-likelihood

$$J = -\left\{ \sum_{n=1}^{N} t_n \log(y_n) + (1 - t_n) \log(1 - y_n) \right\}$$

↳ More conventions:

- N = Number of samples

  n → Index (other index variables i, j, k...)

* Be careful w/ the ambiguity of naming targets and predictions.

↳ Note we can also write our loss function as:

$$J = -\left\{ \sum_{n=1}^{N} y_n \log\left[P(y_n = 1 \mid x_n)\right] + (1 - y_n) \log\left[1 - P(y_n = 1 \mid x_n)\right] \right\}$$

↳ Much more cumbersome, however, much more convenient to discuss how to get predictions.

↓

Training →
- No closed form solution to determine the MLP.
- ↳ Must use numerical optimization methods like GD.

other methods:
- L-BFGS
- Conjugate Gradient

Not used as often in DL

{ using GD - we update our weights as follows:

$$w \leftarrow w - \eta \nabla_w J$$

$$b \leftarrow b - \eta \frac{\partial J}{\partial b}$$

↳ How to determine $\nabla_w J$ & $\frac{\partial J}{\partial b}$

$$\nabla_w J = X^T (Y - T) = \sum_{x_n = 1}^{N} x_n (y_n - t_n)$$

$$\frac{\partial J}{\partial b} = \sum_{n=1}^{N} (y_n - t_n)$$

↳ Note that: $w \in \mathbb{R}^D$ $\quad Y \in \mathbb{R}^N$

$x_n \in \mathbb{R}^D$ $\quad y_n, t_n \in \mathbb{R}$

$X \in \mathbb{R}^{N \times D}$

For which: $\nabla_w J \in \mathbb{R}^D$ $\quad \frac{\partial J}{\partial b} \in \mathbb{R}$

↳ $X^T (Y - T)$

$(D \times N)$ $(N \times 1)$

$(D \times 1)$

↳ In code:

$\nabla_w J = X^T (Y - T)$ -> gradient_w = X.T.dot(Y-T)

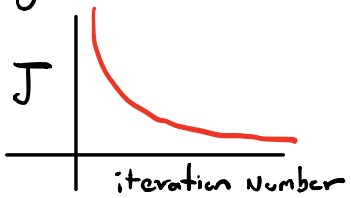
$\frac{\partial J}{\partial b} = \sum_{n=1}^{N} (y_n - t_n)$ -> gradient_b = np.sum(X-T, axis=0)

↓

Now, we update our weights as follows:

```
for epoch in range(num_epochs)
    w = w - eta * X.T.dot(Y-T)
    b = b - eta * np.sum(Y-T, axis=0)
```

we do the above update procedure until we observe a Convergence in our cost J:



Maximize or minimize?

- Maximizing the log-likelihood is equivalent to minimizing the negative log-likelihood
  - Semantically -> we refer to the objective as a cost or a loss -> Makes sense to minimize.

Regularization -> No longer MLP -> MAP (Maximum a posteriori)

↳

$$J_{L1} = J + \lambda_1 |w|$$
$$J_{L2} = J + \lambda_2 |w|^2$$

} Learning process does not change

L_1 (Encourages sparsity)
L_2 (Encourages small weights)

one final note

↳ In J -> If target is 1, then $y = 1$ yields minimum loss -> $J = \log(1) = 0$

⇓

HOWEVER -> For $y = 1$ -> $w \to \infty$

DON'T WANT -> we use regularization to prevent this.

- Deep Learning Readiness Test (Video 9)

1. F
2. Prediction output of sigmoid function
3. C.E Loss function
4. weight update equation by GD
5. Gradient of J -> calculates "error" w.r.t to w

- Review section Summary (Video 10)
  - * Course pattern: Prediction than training
  - * ML -> Geometry Problem (ALL Data is the same)

SECTION III

Preliminaries: From Neurons to Neural Networks

- Neural Networks with No Math (Video 11)
  - NN w/ no math -> Lets obtain a high level intuition of NN
    - no math: still required!

↳ (NN) → Another way to do Supervised learning.

↳ All models (LoR, KNN, NB, SVM, DT) have the same two functions.

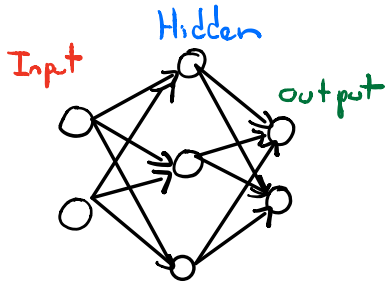
(x, y) → train() → Learn model parameters from the data.

X_test → predict() → Make accurate predictions using the params learned during training

⇓

$\hat{y}$ → our predictions

NN Architecture ↳ NN are just networks of LoR (neurons) units. → Architecture is more complex.



NN are ↳ Graphs → Nodes + edges

- one or more hidden layers
- Every node is connected to every node in the next layer.
- Input layer → Features are fed into the input layer → Biology analogy

⇓

Rescaled by a weight

⇓

Hidden layer(s)

- Sensory nodes such as mechanoreceptors.
- Visual receptors in your eyes.

- weights → Analogous to a synapse in your brain. ↳ Stronger weights imply stronger weights from one node to another.

  Strong signal → Sent from neuron to neuron w/o much attenuation.

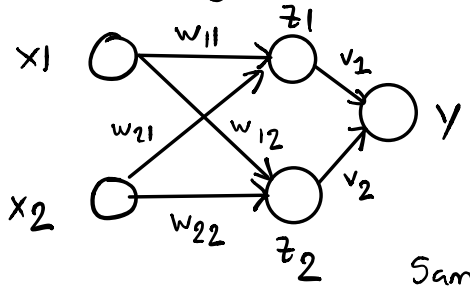
- Signals → Travel from the input node, to the hidden layer and into the output.

  The output aims for a target.

NN → Networks of Neurons ⇒ Layers of LoR units

Learning (Backpropagation)

↳ Consider the following NN:



→ IN NN the learning part is called the backpropagation

→ The error gets "propagated" backwards.

↳ • V depends on error at Y

• W depends on errors at Z

Same pattern if more layers!

A lot of modern NN techniques are based on back propagation (CNN's, RNN's ----) → weights get updated based on this propagated error

Introduction to the E-Commerce Course Project (video 12)

Course Project Intro

- This project has business value
- It is important to take in real data that could be in the form of:
  - xLs spreadsheet
  - SQL table
  - Log Files

  → we should be able to format them into a suitable for consumption by DL algorithms.
- focus → Suppose you are a Data Scientist in an e-commerce store.
- Want to predict user actions in our website.
  - Direct monetary impact: ||
  - Predict bounce → Prompt: We could suggest the user to take other action other than leaving the site

⇓

- Discover areas of site which are weak. ↔ Could be improved

  e.g. users begin checkout but not complete it → why?

- Ex. → Mobile & user friendliness → Improve friendliness of mobile app.

- We can make data driven decisions → use science to improve user experience.

The e-commerce

Data

- Format: csv file
- Header:
  - Is mobile (0/1)
  - N_products_viewed → (int ≥ 0)

    Viewed by the user during a session where the action was taken.
  - Visit_duration → (real ≥ 0)
  - Is_returning_visitor (0/1)

- Time of day (0/1/2/3)

"Buckets" → • split into 4 categories

* In ML → Geometrical distances matter

- Assumptions?

users in the same bucket will behave similarly.

This is what we will predict → • user_actions → our labels

(bounce / add_to_cart / begin_checkout / finish_checkout)

Classification problem

→ Logistic class: Binary classification (Ignore last 2 classes)

→ NN class: Multi-class classification

Considers all labels

- Data Pre-processing

↳ • we cannot feed categories in our LoR/NN model → Numerical vectors required

↳ Solution → one-hot encoding:

cat 1	Cat 2	N
1	0	0
0	1	⋮
0	⋮	0
⋮	⋮	1
0	0	

↳ Dealing w/ binary categories:

- Is_mobile & is_returning_visitor are also categories?

Yes, but we don't need 2 columns "off" and "on"

Just absorb the "off" into the bias term.

↳ Dealing with numerical columns

- Normalization ($\mu = 0$, $\sigma = 0$) $z = \frac{x - \mu}{\sigma}$

⇓

- Simple way of treating numbers

Small range -> Sigmoid $\sigma(a)$ has a more pronanced effect.

↳ e.g. Large values:

$\sigma(1) \rightarrow$ too

NOT USEFUL -> No effect!

Integration with the Course

↳ Each SL ML model has 2 tasks:

① Prediction -> Input -> Predict output

② Training -> How to make the output accurate? -> Need to alter the weights!

Corresponding Section In the Course

THEORY + IMPLEMENTATION

↳ ① How will we make predictions on our e-commerce data?

② How will we train our model on the e-commerce data?