Science & Technology Facilities Council
ISIS Neutron and Muon Source

Machine Learning for Beginners and Intermediates

Lecture 1

Jos Cooper

Overview

- Introduction
- The Neuron
- Terminology
  - Weights
  - Labels
  - Activation
  - Loss
- Back Propogation
- Build your own perceptron

Outline of the School

- For every lecture/tutorial there will be a 2 hour session
- Every day will have a catch up session at the end of the second session to ask questions or finish tutorial material if you need
- All material will be available online as well as instructions on how to create the environments which you will be working in
- All tutorial sessions will be using cloud notebooks which can give tutors remote access to help debugging

Acknowledgements

- Thanks to the SciML (STFC) group
  - https://www.scd.stfc.ac.uk/Pages/Scientific-Machine-Learning.aspx
- Jülich Supercomputing Centre (JSC)
  - https://www.fz-juelich.de/ias/jsc/EN/Home/home_node.html

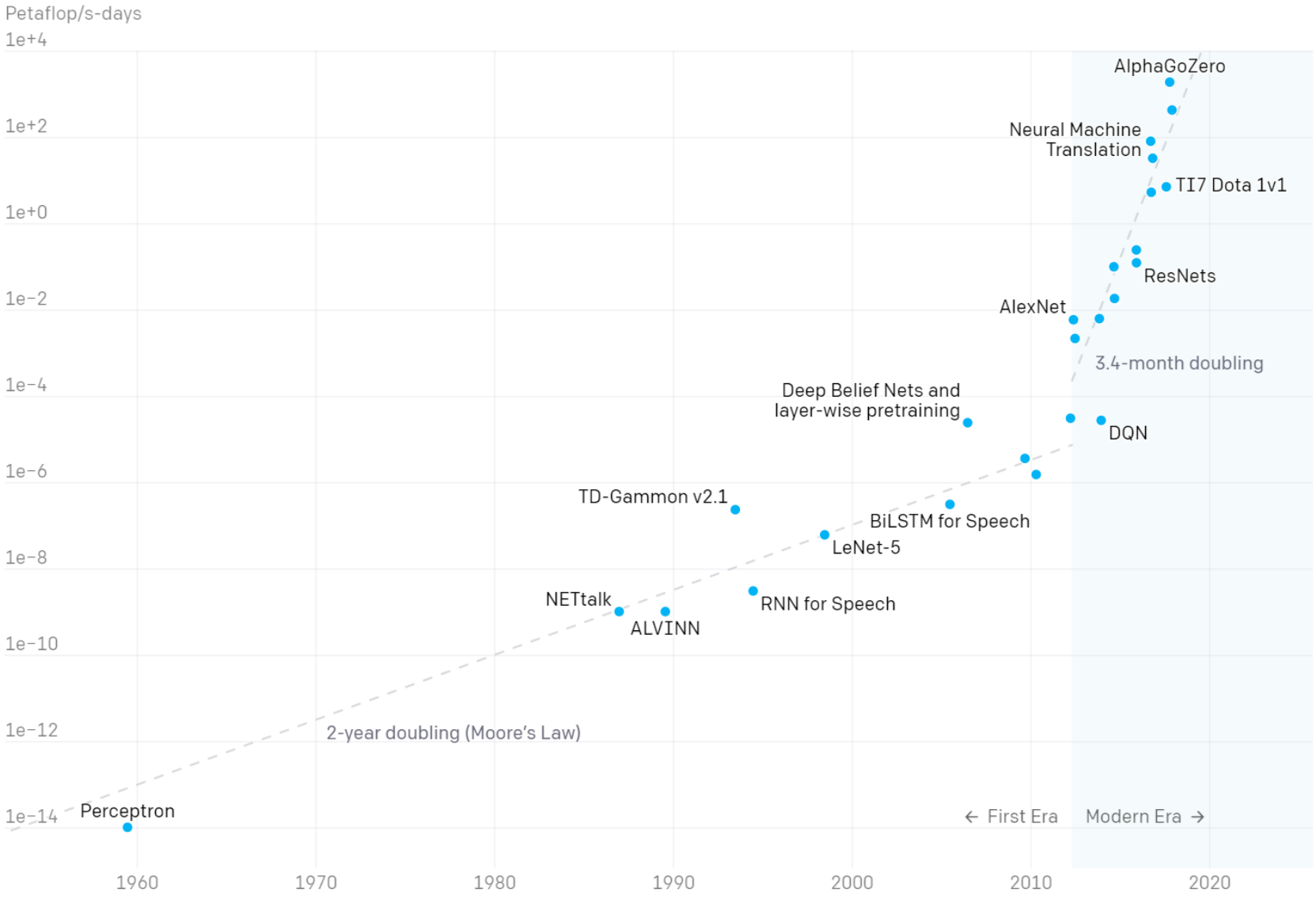
Housekeeping

- Has everyone signed up to everything?
- How tutorials will work
  - Breakout rooms
  - Help others if you can (slack or breakout)
  - Ask host for help otherwise
- Large group, so please stay muted
  - Ask questions in chat to any host
- Meetings mostly recorded
  - Feel free to keep camera off

A short story

- I started in ML ~4 years ago when a student of mine showed me what was possible
- Designed this course to be perfect for me back then
  - Accessible
  - Full of code examples to try
  - Showcase a lot of the different possible techniques

The rise of machine learning



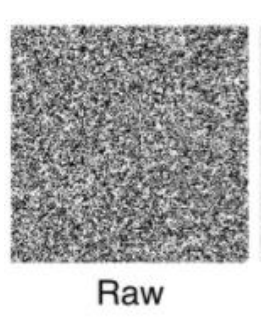
https://openai.com/blog/ai-and-compute/

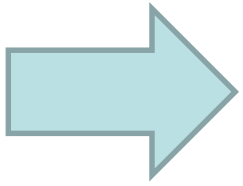
Some numbers

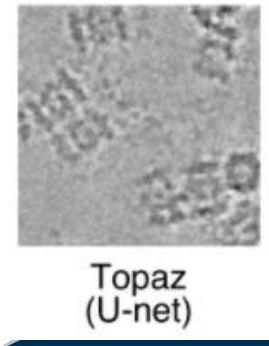
- Beating Moore's law by a factor of 7!
- Current largest model ~1 Trillion parameters (https://arxiv.org/pdf/2101.03961.pdf)
- Teaching an AI to play go from scratch cost ~$35M
- Now exists >100Gb datasets for most applications
  - e.g. https://commoncrawl.org/ https://www.tensorflow.org/datasets/catalog/overview

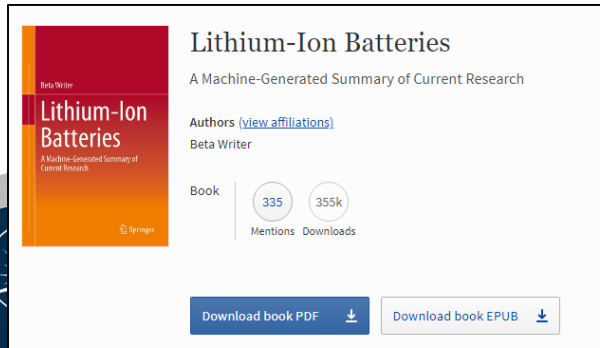
And in science too

- SLAC accelerator optimized by ML
  - https://ml.slac.stanford.edu/science/accelerator
- ML outperforms oncologists at cancer diagnosis
  - https://www.nature.com/articles/d41586-020-00847-2
- ML now preferred denoising technique for cryo EM (S/N ~0.1!)
  - https://www.nature.com/articles/s41467-020-18952-1



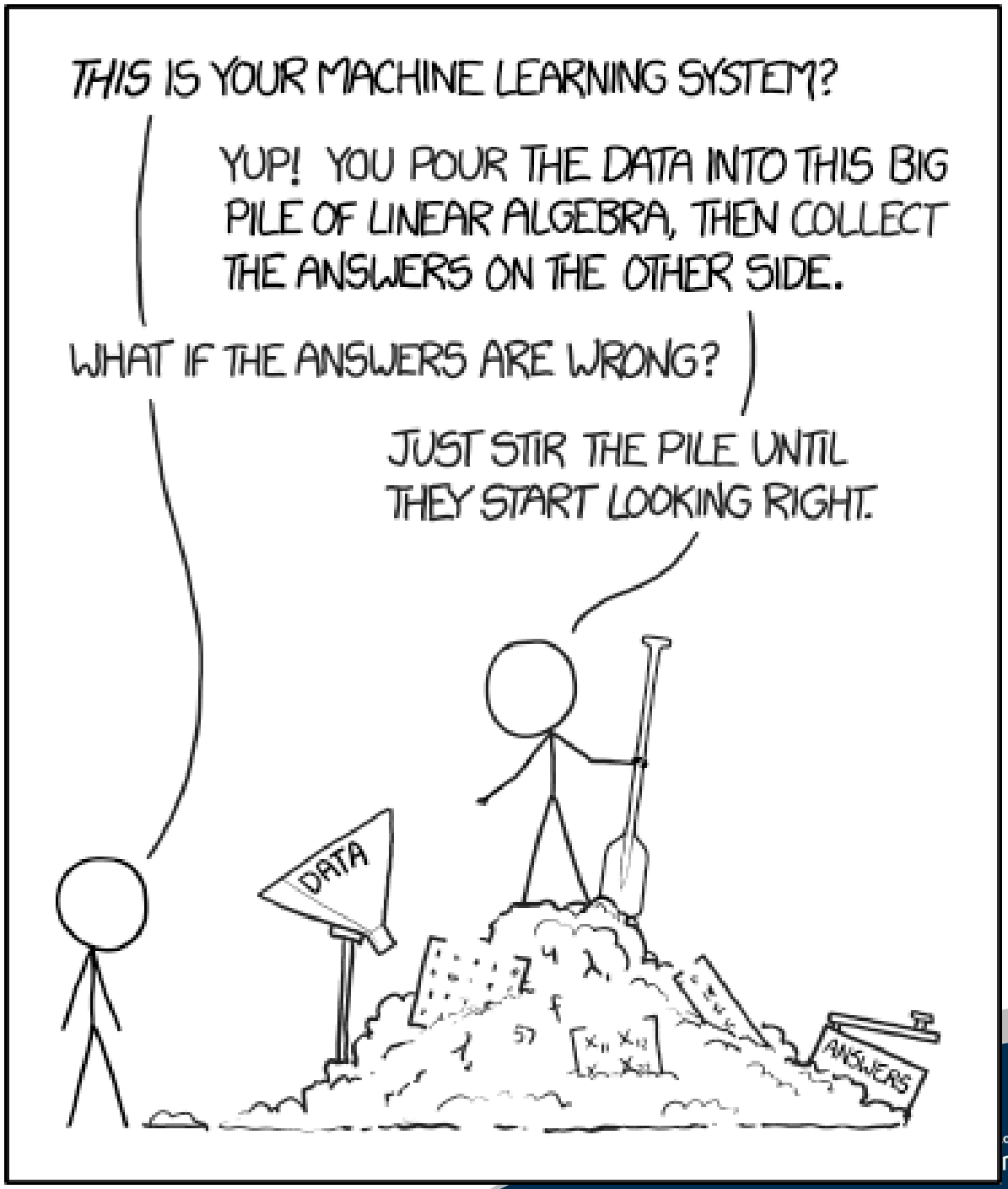






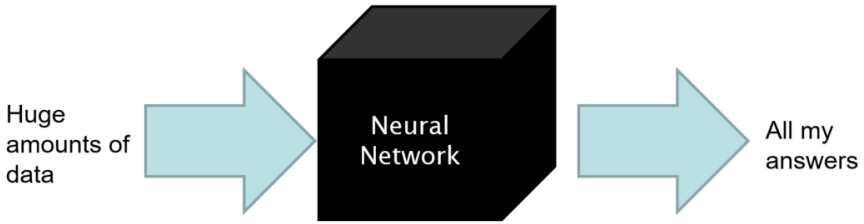
Aims of today

- Become familiar with the terminology
- Understand the basics of a neuron and neural networks
  - Including an overview of the maths
- Introduce Tensorflow, Keras and Pytorch
  - The things which make everything easy
- Build your first networks!

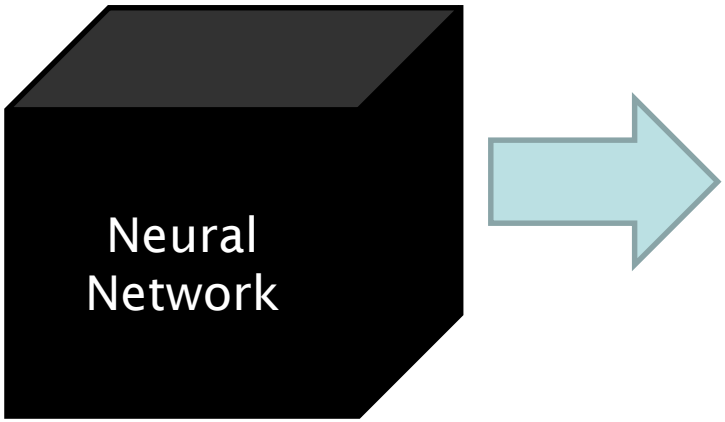

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

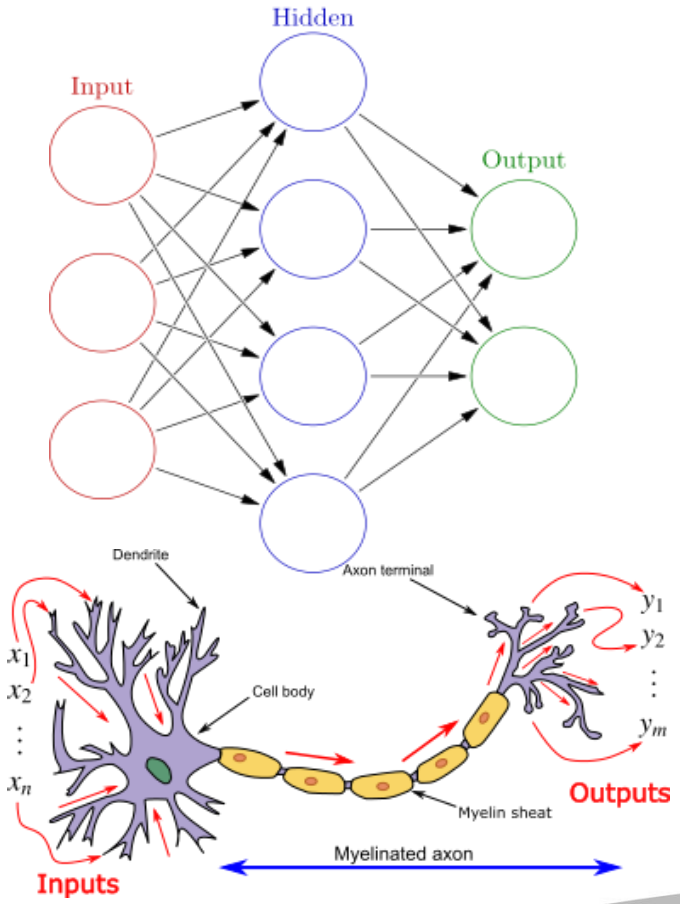
Debunk some myths…

- Machine learning can't solve all your problems
  - Quite often neural networks are **not** appropriate
  - Especially if you can't define the problem well
- You need terabytes of data
  - As you will see data can be very small
  - But it has to be **good** data (and labelled usually)
- ML is only for experts
  - We hope to show you it can be highly accessible!



Lets open the black box…





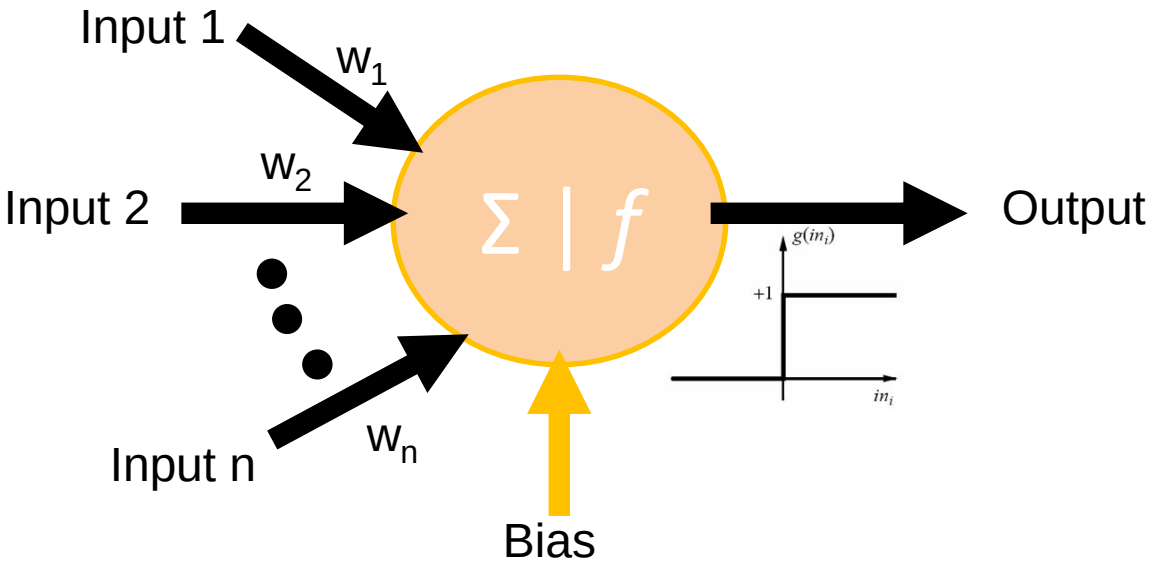
- Neural → based on neurons
- First coined in 1959

THE NEURON

Neurons

- Neural network is a network of neurons!
- What is a neuron in this context?
- Takes some input and uses some (hidden) activation function to decide when to “fire” e.g. 0 or 1
- Realised in 1957 by Frank Rosenblatt as “The perceptron”

The perceptron



- Used as a binary classifier
  - Separation of points into two categories, e.g. those above a certain line, and those below
- Train it to find that line

Some terminology

- We know the equation of a straight line: $y = m * x + c$
- However, in ML terms this becomes:

$$y = w * x + b$$

- y - *label*
- *w - weight*
- *x – feature (input)*
- *b - bias*

More terminology

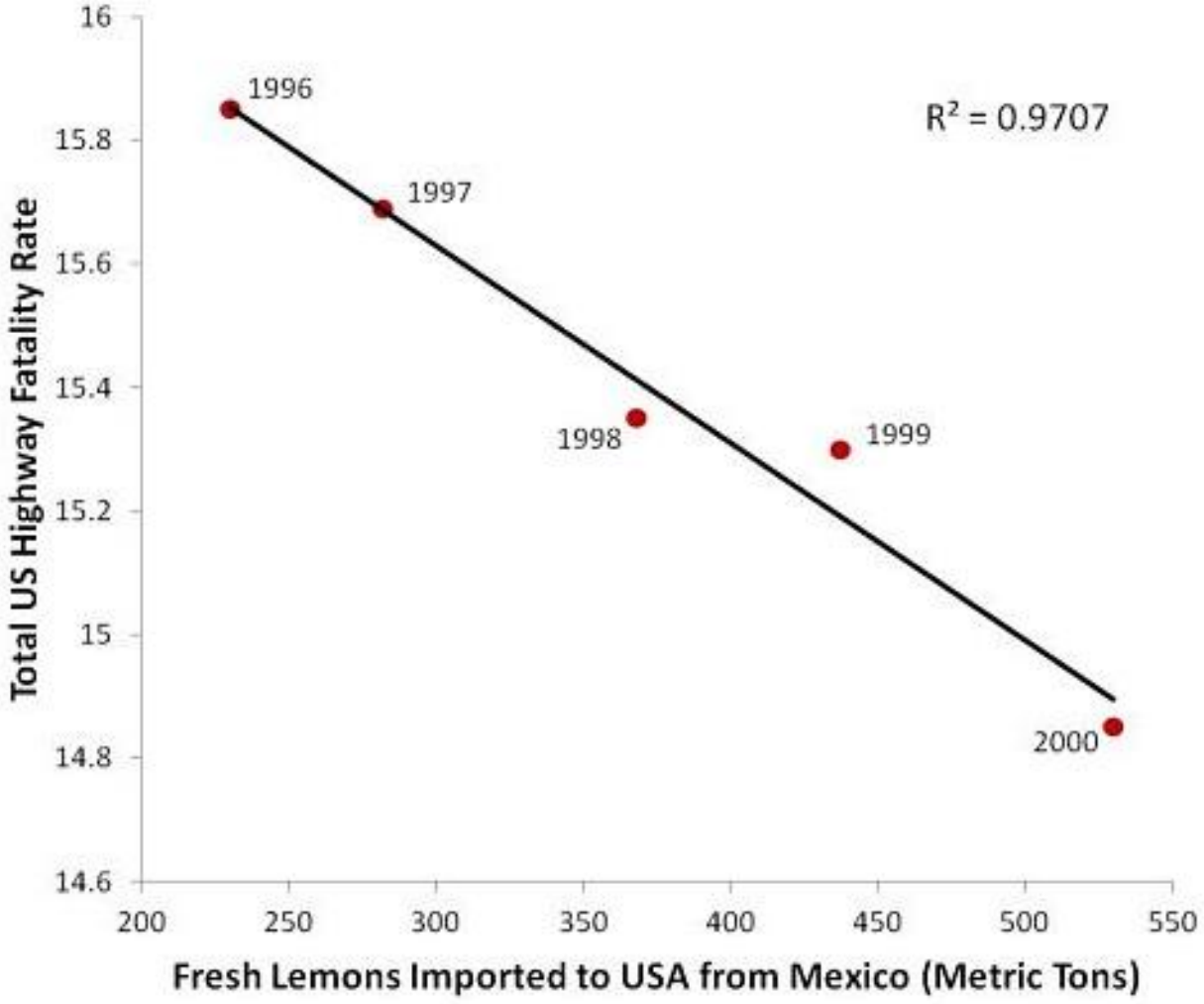
- Generally x can be a vector
  - We may know more than one thing about our data which will determine the label

$$y = w_1 x_1 + w_2 x_2 + \cdots + b$$

$$\rightarrow y = \boldsymbol{w}.\boldsymbol{x} + b$$

  - We need a different weight for each feature because some might be more important than others
- Data can be either labelled, or unlabelled, e.g. we may have (y,x) or just (?,x)

Weights

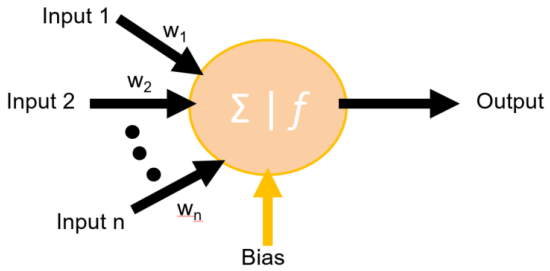

16
15.8
15.6
15.4
15.2
15
14.8
14.6
Total US Highway Fatality Rate
1996
1997
1998
1999
2000
R² = 0.9707
200
250
300
350
400
450
500
550
Fresh Lemons Imported to USA from Mexico (Metric Tons)

Weights

- Weights are usually allowed to vary between 0 and 1
- However this means that if our input features are very different values we may struggle to make the network learn e.g. the value of feature x_1 should be 1000 times smaller than x_2

→ we usually need to normalise the input features' values

The perceptron



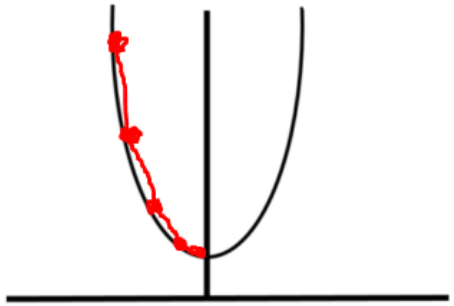
- Modifying weights and bias will allow us to approach the “true” line
- Input labelled data and adjust weights if the outcome is wrong
- How much to adjust? → Learning rate

→ $w_i = w_i$ + error*learning_rate*x_i

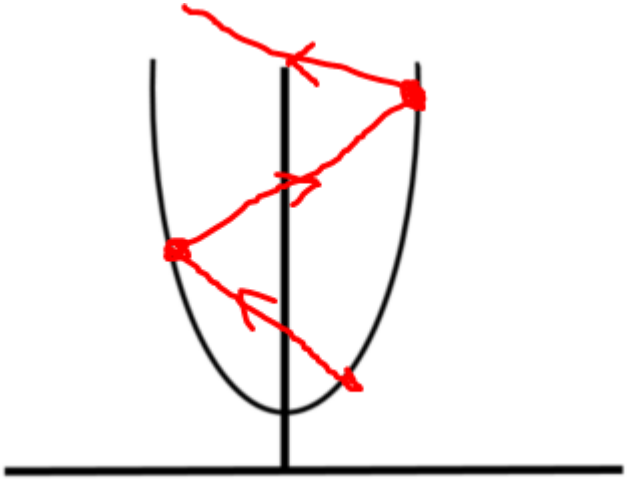
→ Training

Learning rate

- Determines the size of the step we take



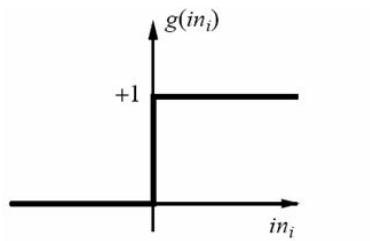
- Too small will take a long time to train
- Too large…

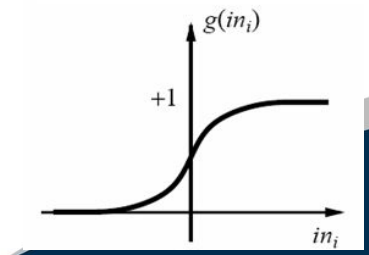


Limitations of the perceptron

- https://www.youtube.com/watch?v=cNxadbrN_aI&ab_channel=Pseudo1ntellectual
- Single layer perceptron can't solve non-linear problems
- No good way of training a network of perceptons except by randomly changing weights

→Change the activation function





Activation Function

- Defines the output of a node (based on its input)
  - For the perceptron it is 1 or 0 depending on whether the inputs exceed the threshold value (much like a neuron)
- Can take **many** forms
- Another simple example is the identity activation
  → $y=x$
- Step → continuous function allows calculation of the gradient!
- This allows training through backpropagation

Activation Function

- We will investigate linear activation soon but should also point out a few features of others
  - Linear, sigmoid, tanh, ReLU…
- To be a "universal approximator" we need non-linearity
- Sigmoid and Tanh were used extensively initially
- But…
- Suffer from saturation
  - Vanishing gradient problem
  - Difficult to know which direction the parameters should move to reduce the error
  - Limits network depth

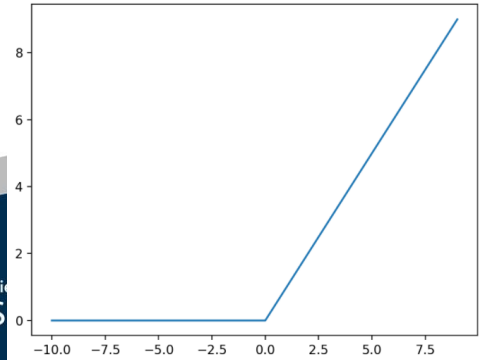
ReLU

- **Re**ctified **L**inear **U**nit
- A milestone in neural network development
- A lot faster to calculate output than others

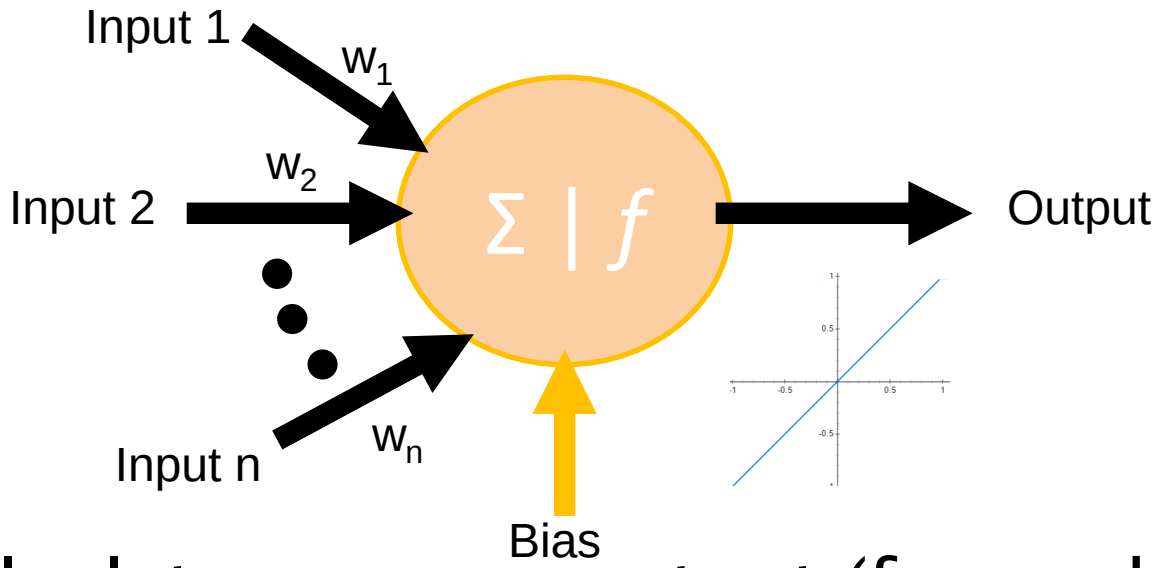
$$output = \max(0, x)$$

$$gradient = \begin{matrix} 0\ if \leq 0 \\ slope\ otherwise \end{matrix}$$

- Capable of outputting a true zero (tanh +sigmoid can't)
- Generally networks are easier to optimize if nearly linear
- Though does have issues leading to "improvements"
  - Leaky ReLU, GELU…
- But still the default for everything now



Backpropogation algorithm



1. Calculate error on output (forward pass)
2. Calculate gradient of weights w.r.t. loss
3. Update weights (backwards pass)
4. GOTO 1.

Weight updating rule

- Assume simple loss function (SSE) and using a linear activation function (adaptive linear)

$$Loss = \sum_i (y - y_{pred})^2$$

- At each step we move in the opposite direction to the gradient
- For a single weight we calculate the gradient

$$\frac{\partial Loss}{\partial w_i} = \sum_i \frac{\partial}{\partial w_i}((y - y_{pred})^2)$$

$$\frac{\partial Loss}{\partial w_i} = \sum_i 2\,(y - y_{pred})\frac{\partial}{\partial w_i}((y - y_{pred}))$$

$$y_{pred} = \sum_i \boldsymbol{x} \cdot \boldsymbol{w} \;\; \& \;\; \frac{\partial y}{\partial w_i} = 0$$

$$\frac{\partial Loss}{\partial w_i} = \sum_i 2\,(y - y_{pred})(\frac{\partial \boldsymbol{w} \cdot \boldsymbol{x}}{\partial w_i})$$

$$\frac{\partial Loss}{\partial w_i} = \sum_i 2\,(y - y_{pred}) \cdot \boldsymbol{x}$$

Weight updating rule

$$\frac{\partial Loss}{\partial w_i} = \sum_i 2\left(y - y_{pred}\right) \cdot \boldsymbol{x}$$

- We now know how to easily calculate our gradient
- How should we change the weights?

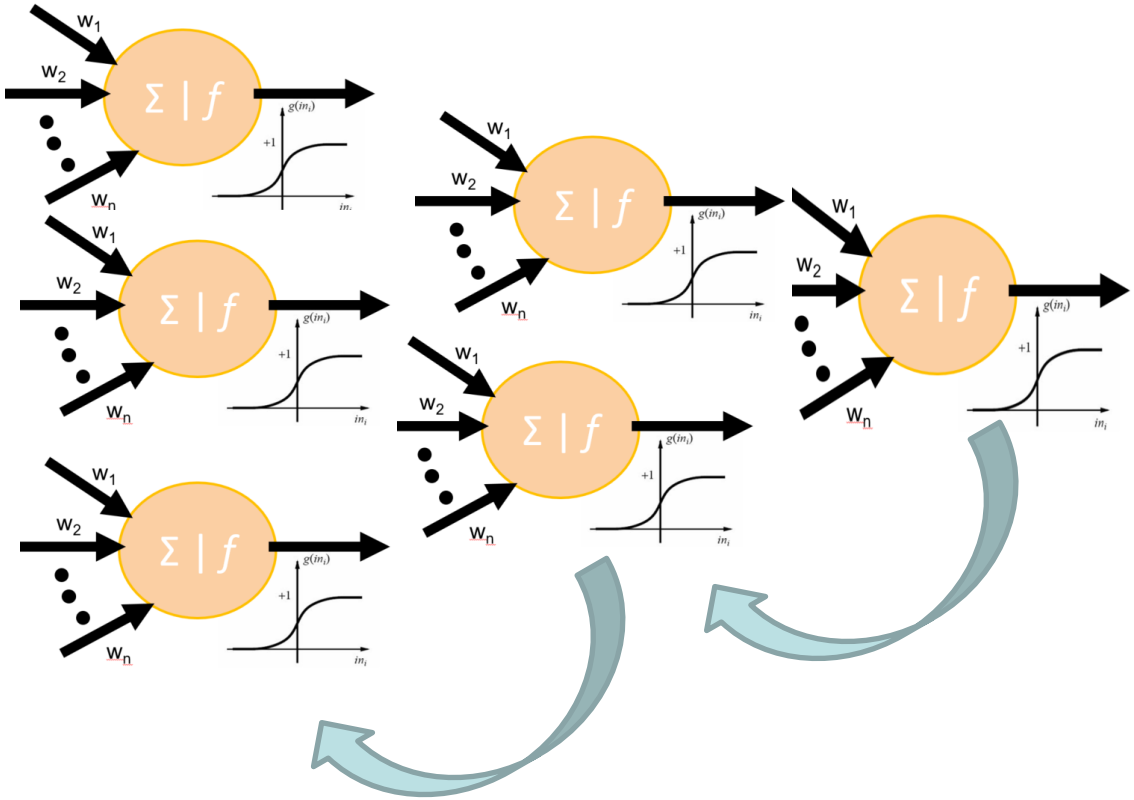
$$\boldsymbol{w} \rightarrow \boldsymbol{w} + Learning\_rate * (-) gradient$$

$$\boldsymbol{w} \rightarrow \boldsymbol{w} + Learning\_rate * \sum_i 2\left(y_{pred} - y\right) \cdot \boldsymbol{x}$$

- Now we can do this, lets go over the whole process

NEURAL NETWORKS

Trainable networks!

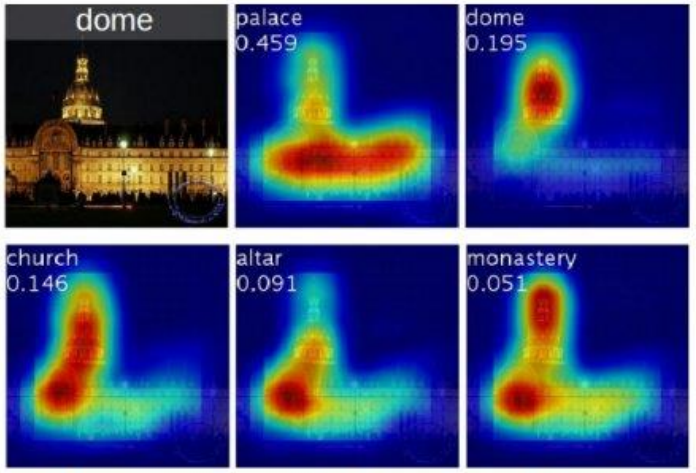


- Through calculation of the gradient we can now back project the correction needed to get a smaller error
- Hidden Layers!

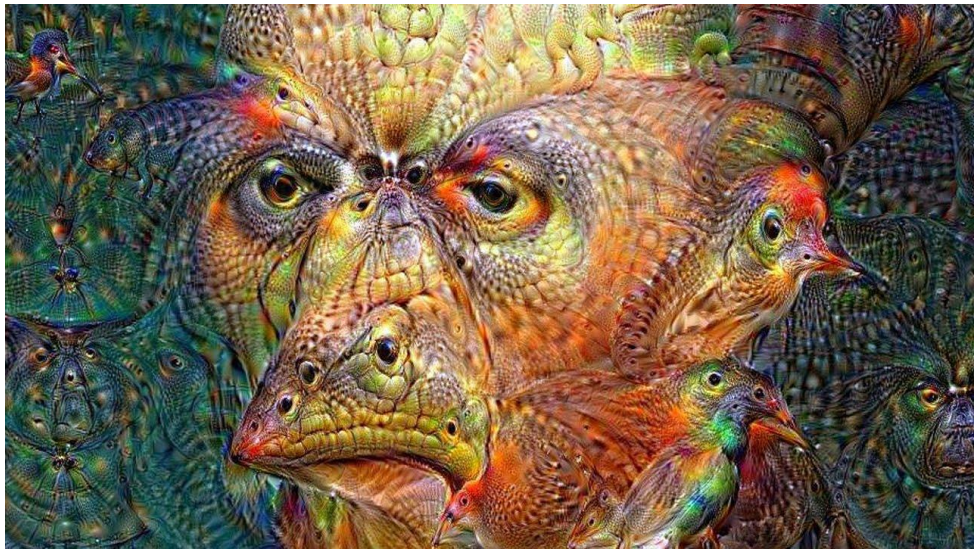
Hidden layers

- Layers of neurons which are neither input nor output layers
- They can “learn” new features (combinations of input features which you might not have thought of)
- Dense layers
  - Every neuron is connected to every other neuron in the layers before and after
- Convolutional layers
  - The neurons apply a filter to the inputs (more on this in lecture 3)

Hidden layers



Class activation maps of top 5 predictions



- New features
- New representations
- A limited way of understanding NN decisions

Hidden layer demo

- https://developers.google.com/machine-learning/crash-course/feature-crosses/playground-exercises
  - Non-linear data requires non-linear features
- https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises
  - Or we can add non-linearity and hidden layers!

Some tools for ML

- Where can we run our models?
- Desktop
  - Buy your own GPU
- Local cloud compute
  - Need correct packages and GPU's can be tricky
- Supercomputer
  - If you have one
  - Packages *can* be an issue
- Google colab

Google colab

- Free online python interpreter with ML emphasis
- GPU and TPU access
- A lot of relevant packages pre-installed
- Great for running jupyter notebooks*
- Can even pip/conda install packages
- https://colab.research.google.com/

Jupyter Notebooks

- Interactive python interpreter
- Great for testing ideas, plotting, and exploring data
- Less suitable for production code
  - But can be used as a UI
- https://jupyter.org/

EXERCISE 1

Science & Technology Facilities Council
ISIS Neutron and Muon Source

Machine Learning for Beginners and Intermediates

Lecture 2

Jos Cooper

Science & Technology Facilities Council
ISIS Neutron and Muon Source

Overview

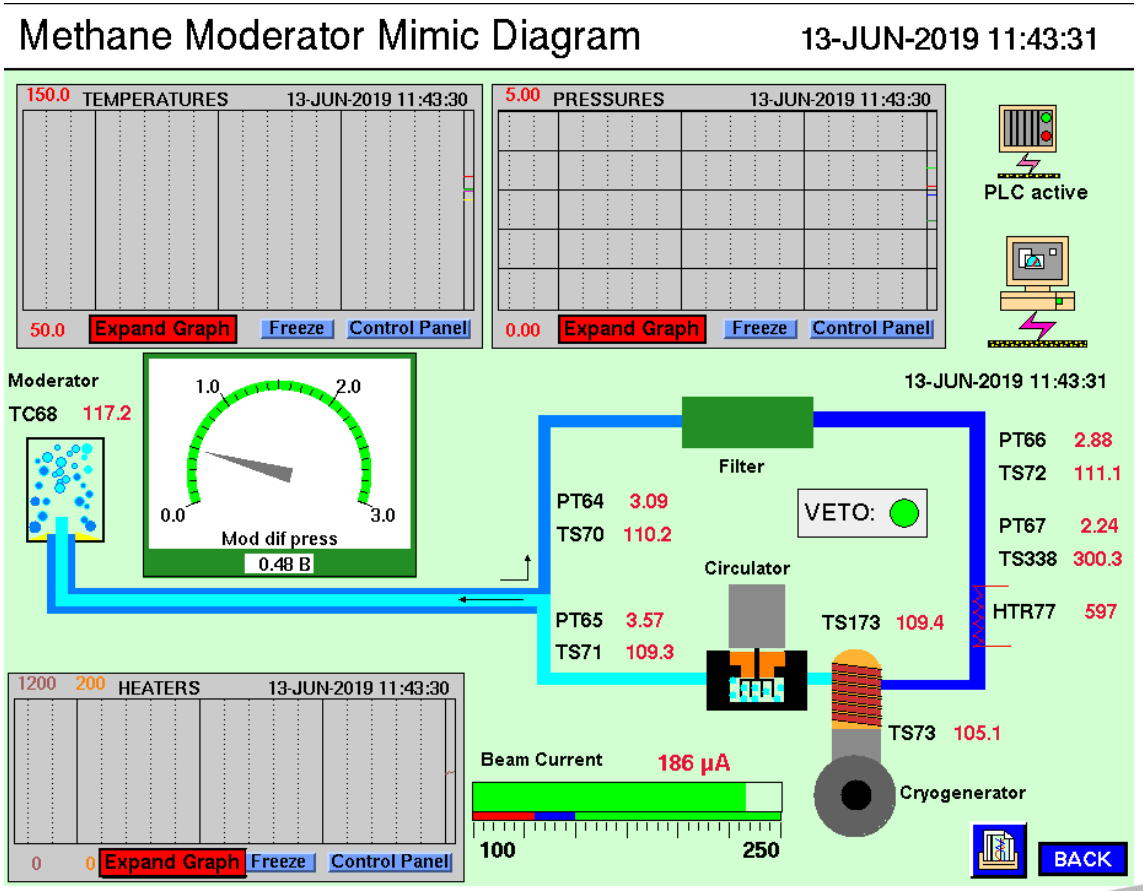
- (A lot more terminology)
- Data
- Over fitting
- Loss Functions
- Training
- Optimizers
- Exercise 1
- API's
- Exercise 2

DATA

Data

- The data is the most important part of machine learning
- 80+% of the work is in the data
- You can't ever do better than your training data
- Downloadable datasets are great!
  - But they aren't what you actually want to implement
- How would you label 10 000 datasets?

Example

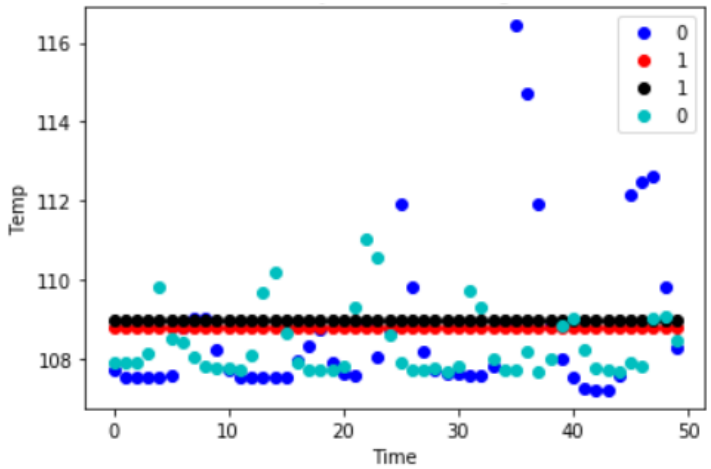

Methane Moderator Mimic Diagram
13-JUN-2019 11:43:31
150.0 TEMPERATURES 13-JUN-2019 11:43:30
50.0 Expand Graph Freeze Control Panel
5.00 PRESSURES 13-JUN-2019 11:43:30
0.00 Expand Graph Freeze Control Panel
PLC active
Moderator
TC68 117.2
1.0 2.0 0.0 3.0
Mod dif press
0.48 B
13-JUN-2019 11:43:31
Filter
PT66 2.88
TS72 111.1
PT64 3.09
TS70 110.2
VETO:
PT67 2.24
TS338 300.3
Circulator
PT65 3.57
TS71 109.3
TS173 109.4
HTR77 597
1200 200 HEATERS 13-JUN-2019 11:43:30
0 0 Expand Graph Freeze Control Panel
TS73 105.1
Beam Current 186 µA
100 250
Cryogenerator
BACK

Example

- Multiple sensors measuring different things
  - Temperature, pressure, ON/OFF, beam current…
- Values recorded at different time intervals
- Some sensors “less stable” than others

Example

1. Decide what to use (most important features, or all possible?)
2. Interpolate all measurements
3. Normalise all values
4. Split time series into “chunks” as an input
5. Label chunks (working/faulty)



Validation data

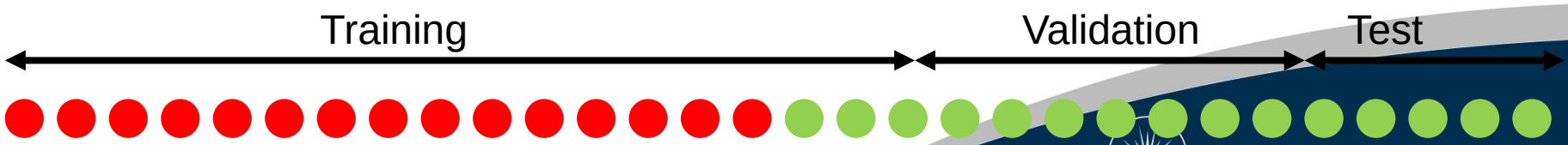
- We take our initial dataset and split it up
- We train on our "training data" then see if we are overfitting on our "validation data"
- Plotting the evolution of the test and validation losses show if we are over fitting

Test data

- In practise we often go a step further and split the dataset into three to include a small "test dataset" which we reserve for when the network has completed *all* training which we can validate our final trained network on
- **Don't** train on validation or test data

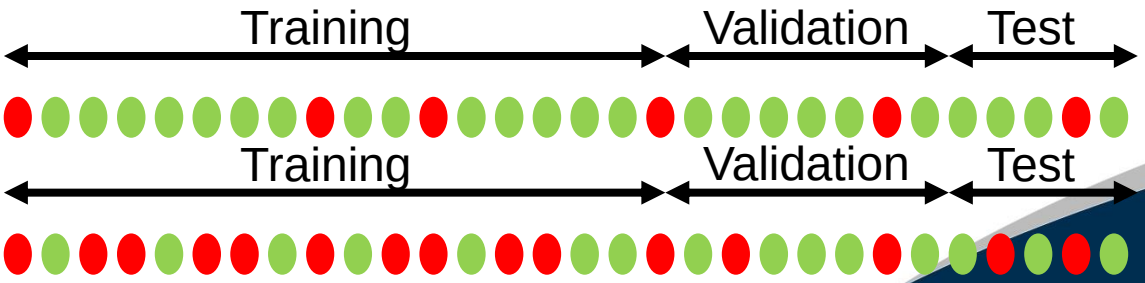
The importance of randomisation

- Consider curating a dataset of pictures of cats and dogs
  - You search for cat pictures then download them
  - Then search for dog pictures and add them to your dataset
- **Must** randomise dataset before doing anything else



Dataset content

- Data should be representative of the data we want to predict on
- If at all possible we want a *balanced* dataset
  - Loss can be minimised very quickly by choosing most common category (local minimum but hard to escape)



Aside on data formats

- HDF5 file format is great for ML
  - Allows for partial file I/O
  - Stores sparse data very efficiently
  - Allows for on the fly (de)compression
- HDF5 loads nicely into a pandas dataframe
  - Like numpy array but can hold disparate types of data
  - Labels rather than indexes

Data considerations

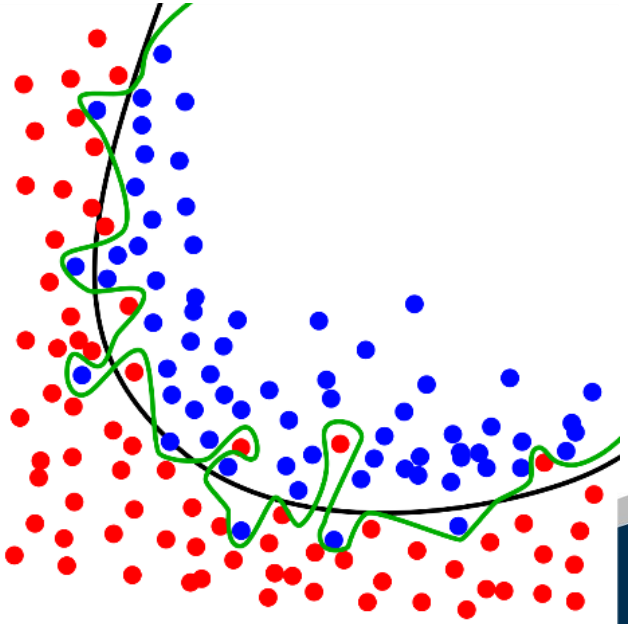
- A rough rule of thumb is that you don't want to have many more parameters than you do training data
- When randomising, be sure to keep labels attached correctly
  - Though randomising labels can be used to test if you are doing better than random guessing!

Data Rules

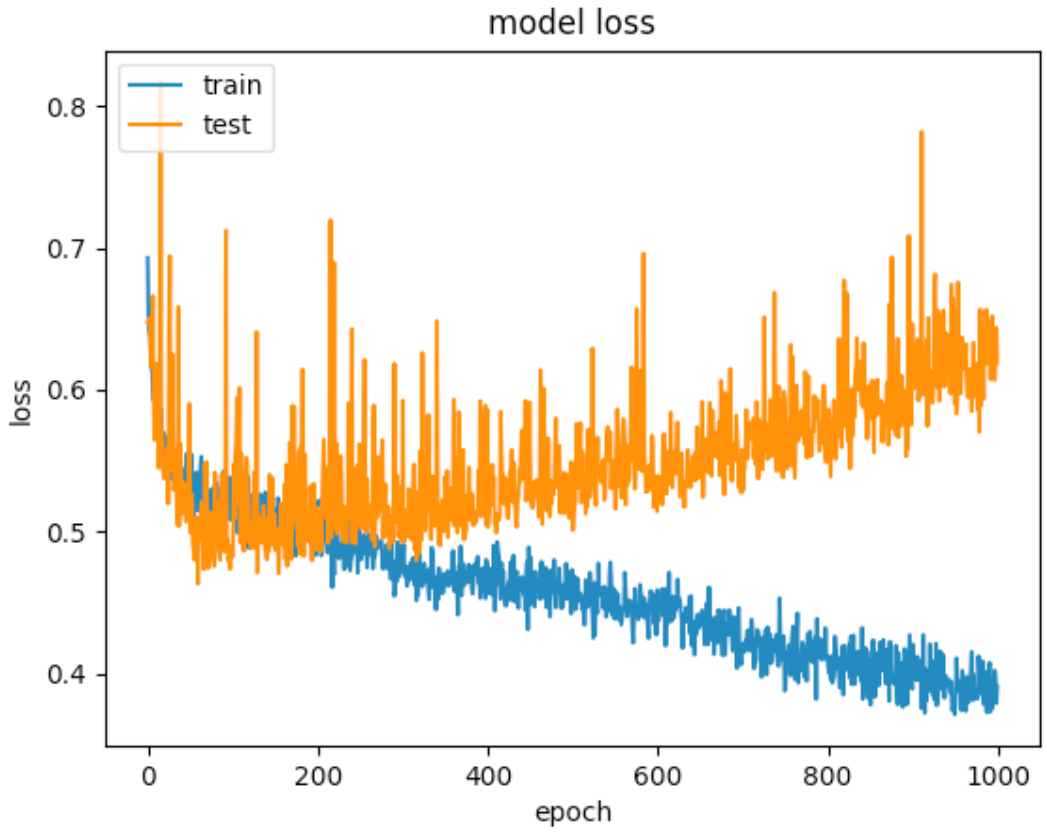
1. Use appropriate data
2. Use balanced data
3. Randomise the whole dataset (if possible)
4. Split the data into training/validation/test (~80/10/10)

Overfitting

- We have accidentally “learned” too many of the little features of the training dataset
  - Stops us generalising to new data



Overfitting



Overfitting

1. We might have too many model parameters for the amount of data we are feeding it
2. We may have tried to fit for too long
3. Data may not be balanced

Loss

- Final prediction error called Loss in machine learning
- Loss functions can take many (infinite) forms
- Can be user defined but a lot are bundled

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class

Regression loss functions

- General default:
  - Mean Square Error (MSE) or absolute (MAE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y - y_{pred})^2$$

- Penalise very wrong predictions more harshly than slightly wrong
  - Though this may skew your results
- No loss function solves all problems ☺

Categorical loss

- Need to turn “dog” or “cat” into a number
- NN’s don’t predict a class, they give a numerical confidence/probability for every class

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- We use something like a softmax activation as final layer in categorisation

Categorical loss functions

- Need to compare numerical values

1. Predict a “probability” for each class (softmax)
2. Compare each category
3. Calculate loss

Categorical cross entropy!

True	Pred
0	0.15
0	0.02
1	0.83

(https://en.wikipedia.org/wiki/Cross_entropy
https://en.wikipedia.org/wiki/Softmax_function)

So we know the basics…

- ***Regression*** we want to predict a continuous value e.g. house price, lattice coupling strength, a moderator temperature at some time in the future

- ***Classification*** we want to predict into one of several categories e.g. is this a picture of a dog or cat, how many peaks are in this data, is this control system behaving?

A basic neural network

1. Assemble input features (array of values)
2. Define hidden layers and their activation functions
3. Define output and its activation function
4. Define a loss function and optimizer
5. Train the network

What is training doing?

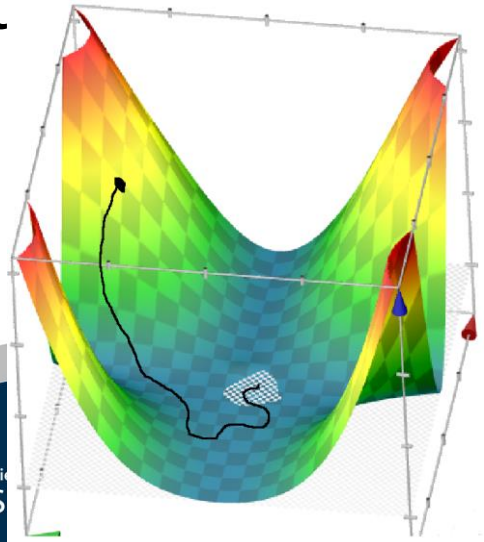
- We initiate our network at some random point on the multi-dimensional loss surface

→Want to find the minimum

- Every batch we update weights based on the learning rate and value of gradient

→Step down the loss surface

→Gradient descent



Gradient Descent

$$w_i = w_i - const.* \frac{\partial(loss)}{\partial w_i}$$

- Constant is determined by learning rate and *stuff*

*Stuff depends on optimizer

Available optimizers

- SGD
- RMSprop
- Adam
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

Optimizers

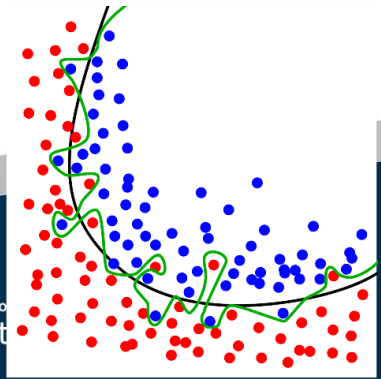
- FAR too much to go into here except to point out:

1. Fixed learning rate not optimal – want to decay it somehow as you get close
2. Incorporating “momentum” into your descent smooths the route
3. May even want different learning rates for different parameters
4. In practise ADAM is very popular

https://arxiv.org/pdf/1412.6980.pdf

Training a network (supervised learning)

- Simplest form:
- Initiate network with random weights
- Input pairs of features with labels
- Use backprop to change weights so that the input produces the label
- Remembering
  - Try not to over fit
  - Check on validation

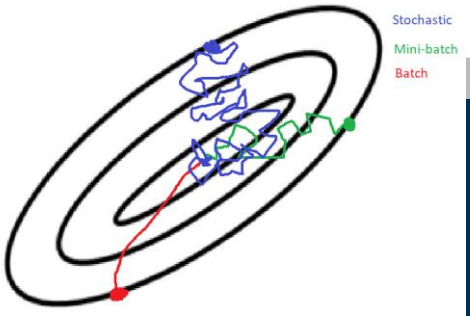


Epoch

- Every time we cycle through the *entire* dataset we say we have progressed by one epoch
- Training is often performed over many (even hundreds of) epochs
- So that we don't jump around too much we often only update the network weights after we have processed several data points – effectively averaging
- Number of inputs we process before updating the model weights is known as the batch size
  - Even if it is one!

Batches

- Batch size is how much of the dataset we feed the network at once
- Batches of one → Stochastic
  - Noisy but easier to handle data volumes
- Batches of entire dataset →Batch
  - Least noise, not always possible to load all your data at once
- Batches of parts of dataset → Mini-batch
  - Middle ground and the most often used (in fact quite often defaulted to 32)



Hyper Parameters

- Batch size is an example of a hyper parameter
- These are parameters which control the learning process
  - Not neuron/node weights since they change during training
- Include:
  - Number of hidden layers and nodes, activation function, learning rate, number of epochs, batch size… many more

EXERCISE 1

HTTPS://COLAB.RESEARCH.GOOGLE.COM/NOTEBOOKS/
HTTPS://GITHUB.COM/JFKCOOPER/LENS_ML_SCHOOL_2021/

MACHINE LEARNING API'S

Perceptron → NN's

- Programming backprop by hand isn't so hard, but would get inefficient for large networks
- As we have just seen, there are tools to make this easier
  - Automatic differentiation
  - Automatic node connecting
  - "Drop in" activation functions and losses
- API's

Tensorflow

- Developed by Google in 2015
- More interfaces than just python
  - C, C++, Java, Go, Swift, JavaScript…
- Compiles to any device from same code
  - CPU, GPU, TPU
- Very well documented and supported

PyTorch

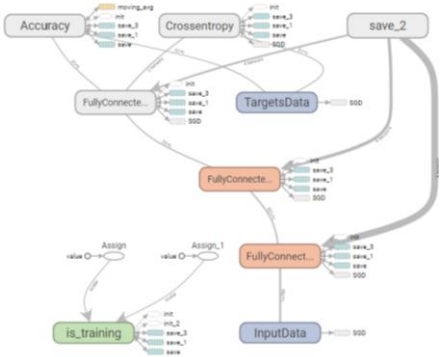
- Developed by Facebook in 2016
- Primarily python
- Slight changes in code to change between CPU/GPU
- Also well documented and supported

Others!

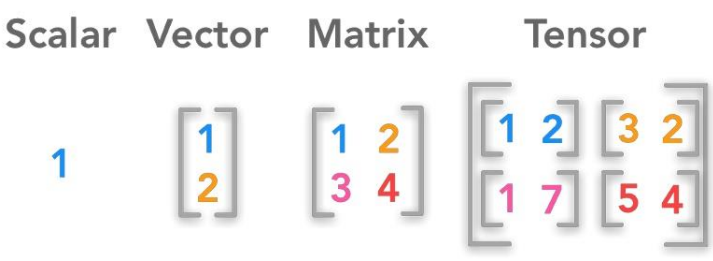
- OpenCV (computer vision – C++)
- Apache Spark (large scale data – Scala)
- Apache MXNet (deep learning – Many)
- Sklearn (SciPy extension for ML – Python)
- ….

Commonalities

- Based on computational graphs which you compile
  - The additional compilation step is how they port easily to GPU's
  - Look the same as python but NOT



- Use tensors for data
  - Generalisation of a matrix (kind of) but a subtle difference



Tensors

- Ideal for multi-dimensional data flows
- Come with well defined rules about their transformations
- Linear algebra is generally very fast on GPU's
- Example: colour image is just a rank-3 tensor (R, G, and B channels for *nxm* image is *nxmx3* tensor

Which to choose

- Tensorflow and Pytorch used to be different (static vs dynamic graph definition) but now converging
- Not so much different in current versions in terms of functionality
  - Tensorboard *is* good though
- Personal preference…

Keras

- Sits on top of tensorflow (or CNTK or theano) and provides “bigger building blocks”
- Like a simplified wrapper for TF
- Easy to get quick results
- Easy to debug
- Not so suitable for complex problems

Some quick tips from experience

- To enable GPU usage on local machine you need CUDA libraries (NVIDIA)
  - AMD seems a lot harder to run with
  - Check your GPU is supported (last ~3/4 years)
  - Don't get most recent version of either CUDNN or tensorflow
- Finally, if you get 100% accuracy on your network you have almost certainly got something wrong

EXERCISE 2

HTTPS://COLAB.RESEARCH.GOOGLE.COM/NOTEBOOKS/
HTTPS://GITHUB.COM/JFKCOOPER/LENS_ML_SCHOOL_2021/