INTRODUCTION TO NEURAL NETWORKS

Giles Strong

6ªs Jornadas de Engenharia Física, IST - 04/03/20

giles.strong@outlook.com

twitter.com/Giles_C_Strong

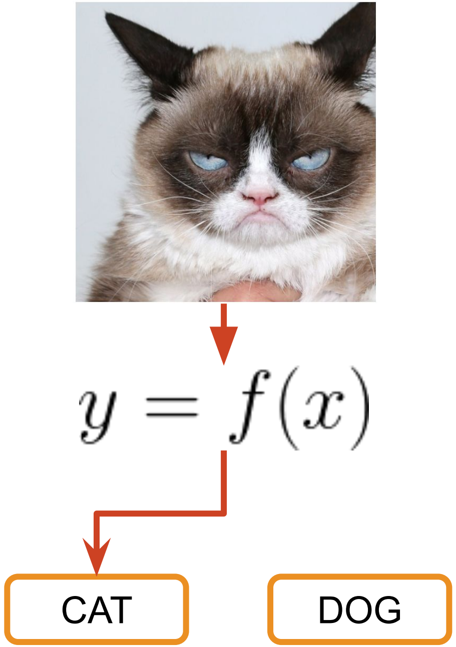
Amva4newphysics.wordpress.com

github.com/GilesStrong

MOTIVATION FOR NEURAL NETWORKS

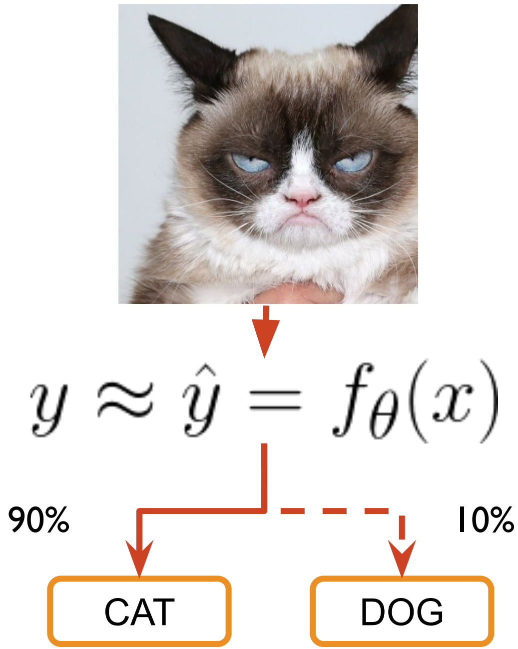
FUNCTIONAL APPROXIMATION

- Have data *x* in some observation space and want to map data to some target space *y* : $y = f(x)$, e.g.
  - pixel values → cat or dog (image classification)
  - particle momenta → signal or background (particle collision classification)
  - environment space → reward (reinforcement learning)
  - text → sentiment (sentiment classification)



FUNCTIONAL APPROXIMATION

- If domain theory doesn't offer a (computationally cheap) analytical map, can instead approximate it with a sufficiently flexible parameterisation:
  - $y \approx \hat{y} = f_\theta(x)$

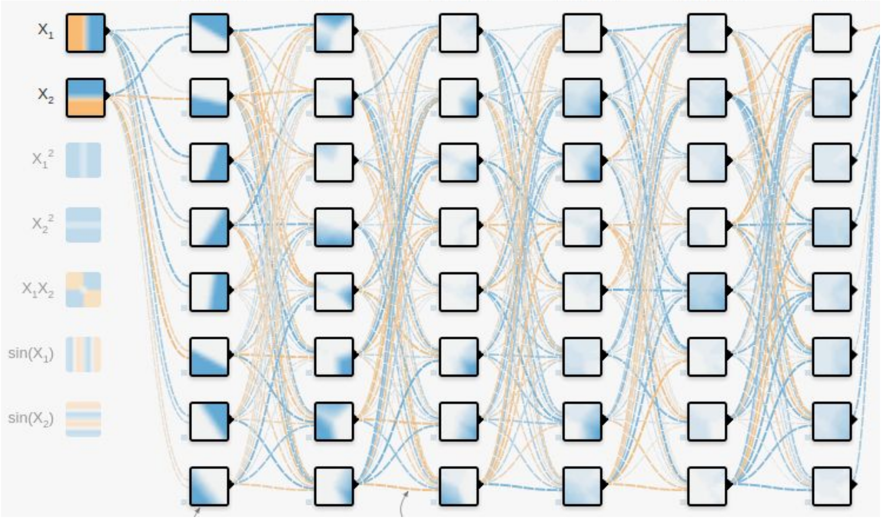


LOSS FUNCTIONS

- Aim is to adjust parameters θ in order to make approximated *y* match true *y* as closely as possible:
  $$\theta = \arg\min_{\theta} \mathcal{L}\left(y, \hat{y}_{\theta}\right)$$
- Where L is a function which quantifies the error (difference between prediction and target): the *loss function*
- Exact form of loss function varies from problem to problem
- However, making f_{θ} sufficiently flexible requires many continuously valued parameters: finding the optimal parameters by brute force is intractable

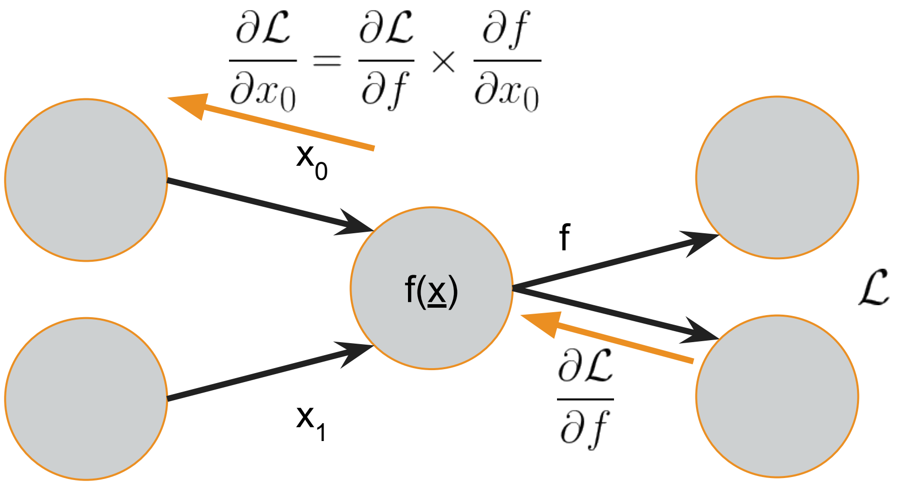
PARAMETER OPTIMISATION

- A neural network is special form of f_θ in which the prediction is continuously differentiable with respect to the parameters
  - Interactive demo



PARAMETER OPTIMISATION

- By also using a continuously differentiable loss function, the exact effect of each parameter on the performance of the network can be analytically computed:
  - $\nabla_{\theta_k} \mathcal{L}$
- This is done via *backpropagation*
- See e.g. Stanford CS231n for an in-depth explanation



PARAMETER OPTIMISATION

- The optimal parameters can now be converged to via iterative updates in the direction of steepest slope of the loss function:

$$\theta_{k+1} = \theta_k - \gamma \nabla_{\theta_k} \mathcal{L}$$

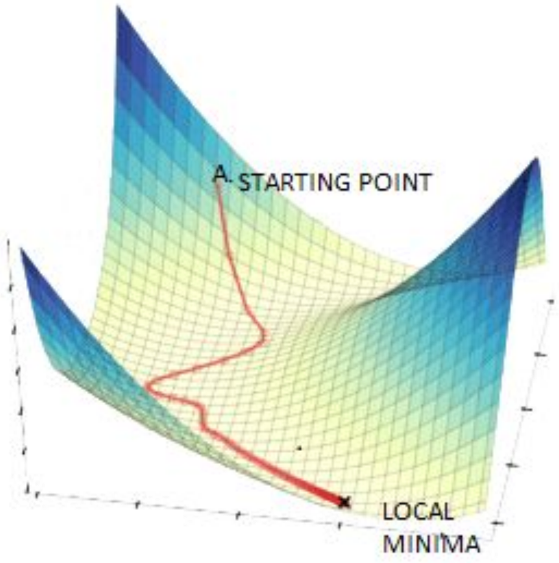


Image source

PARAMETRIC FORM

- Parameters (mostly) take the form of multiplicative matrices (weights) and additive offsets (biases) and are said to be arranged in *layers*, each of which apply a linear transformation to the previous layer:
  - $w \cdot x + b$
- By interspacing these linear transformations with non-linear *activation functions* more complex transformations can be achieved:
  - $\mathcal{A}\left(w \cdot x + b\right)$
- Stacking linear layers and activation functions produces e.g. a 2-layer network:
  - $\mathcal{A}\left(w_1 \cdot \mathcal{A}\left(w_0 \cdot x + b_0\right) + b_1\right)$

PARAMETRIC FORM

- Explicitly, the linear transformation is:

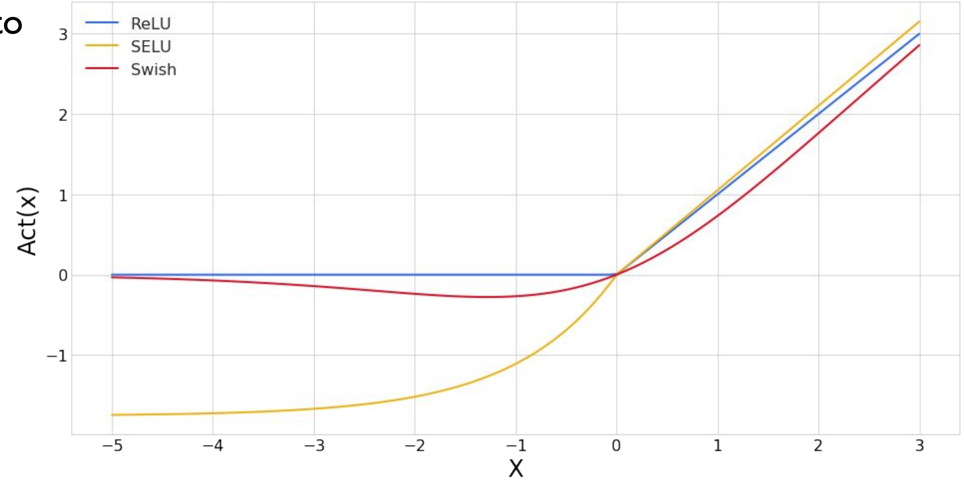
$$\begin{pmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,N} \\ w_{1,0} & & & \\ & & & \\ \dots & & & \\ w_{M,0} & & & w_{M,N} \end{pmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ \\ \dots \\ \\ x_N \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \\ \dots \\ \\ b_M \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ \\ \dots \\ \\ h \end{bmatrix}$$

- And the activation is then applied to each element of the resulting vector via broadcasting:
  - $\mathcal{A}\left(\bar{h}\right)$

COMPONENTS OF NEURAL NETWORKS

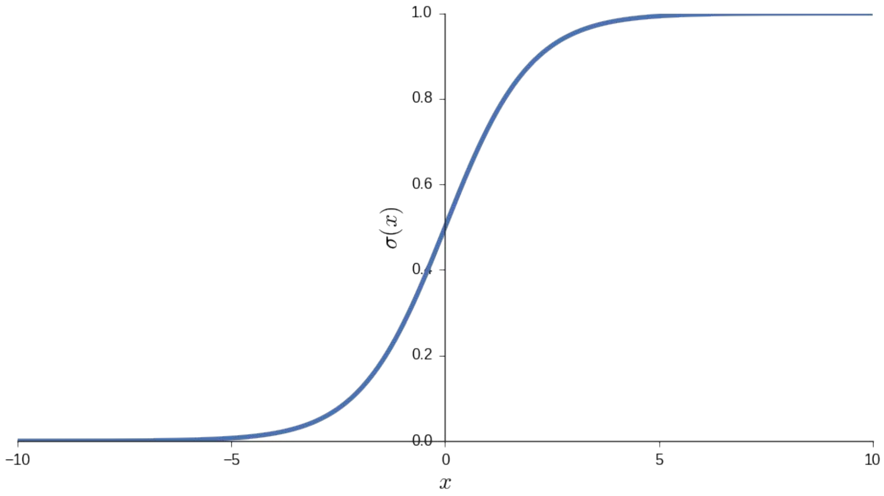
ACTIVATION FUNCTIONS

- Activation functions allow the network to perform non-linear transformations
  - Can learn to model complex functions more easily
- Any function can be used, provided it is continuously differentiable
  - Need to propagate gradients
- Modern standard is ReLU
  - But new ones are being developed
  - E.g. SELU and Swish



ACTIVATION FUNCTIONS

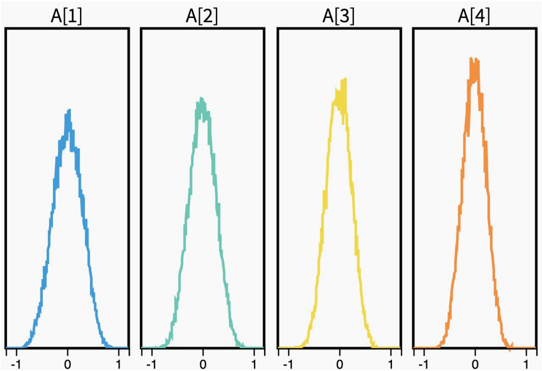
- The final activation function is the *output activation function*
  - It determines the range of outputs of the network
- The form used is therefore determined by the problem being solved, e.g.:
  - Binary Classification: Sigmoid
    - Range [0,1]
  - Multi-class classification: Softmax
    - Range [0,1] and output sum = 1
  - Multi-label classification: Sigmoid
  - Regression: linear
    - Range [-∞,∞]



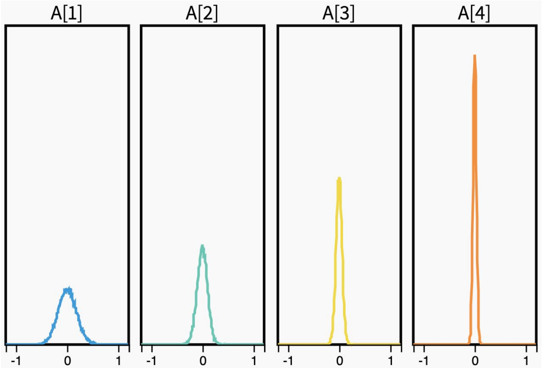
Sigmoid $\sigma(x) = \frac{1}{1 + e^{-x}}$

PARAMETER INITIALISATION

- The parameters must begin from some random starting values
- If these starting values are of the wrong scale, the network can be difficult to train
  - Interactive demo: https://www.deeplearning.ai/ai-notes/initialization/
- Several *initialisation schemes* exist, depending on the activation function used:
  - Sigmoid, linear: Glorot/Xavier
  - ReLU: Kaiming/He
  - LSUV initialisation runs a test loop using training data to rescale starting weights appropriately



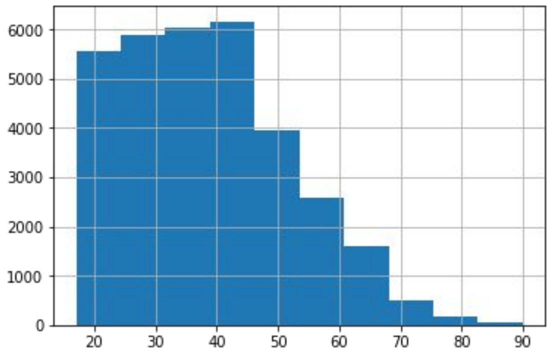
Activations for Glorot init



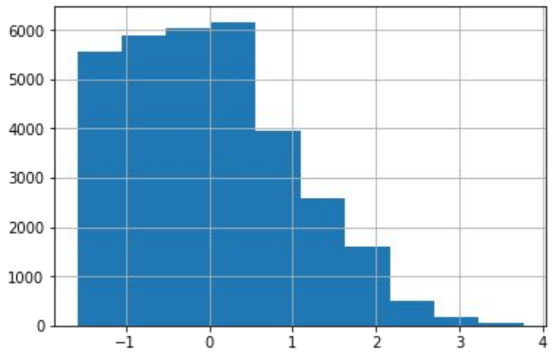
Activations for Uniform init

PREPROCESSING

- These initialisation schemes generally aim to output a unit-Gaussian distribution when the input in unit-Gaussian
- This means that for optimal training, the input data should have a mean of zero and a standard deviation of one
- Transforming the input data is called *preprocessing*
- In this case we should subtract the mean of each feature for the training data and divide by their standard deviations



Original feature



Preprocessed feature

LOSS FUNCTIONS

- The loss function defines the problem to be solved by quantifying what is a 'good' and 'bad' prediction
- Several standard ones exist, e.g.:
  - Binary Classification: binary cross-entropy
  - Multi-class classification: categorical cross-entropy
  - Multi-label classification: binary cross-entropy
  - Regression: Mean squared-error or mean absolute-error
- Loss functions can also be *weighted* to account for data-points differently
  - E.g. to encourage unbiased responses

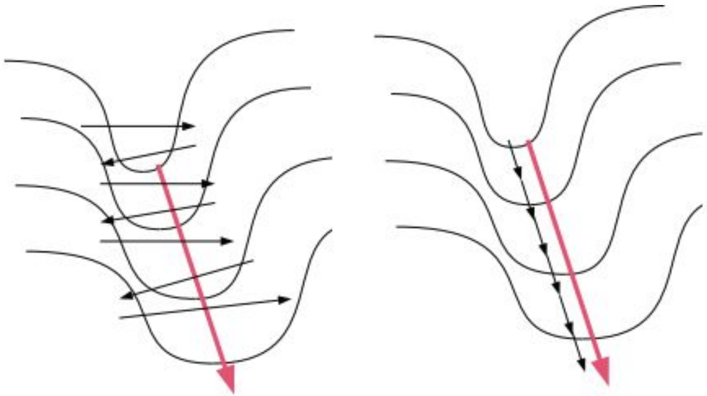
Average over data points

Difference between prediction and truth

$$MSE = \boxed{\frac{1}{N}\sum_{n=1}^{N}}\boxed{(\hat{y_n} - y_n)}^2$$

OPTIMISERS

- The default optimiser updates in the direction of steepest descent of the loss function by a fixed amount
- But potentially O(100,000) parameters = O(100,000) dimensional space
- Very likely to end up in "narrow valleys", where steepest descent results in oscillation between valley walls



How GD moves Ideal moves

OPTIMISERS

- Various improvements to *vanilla* gradient descent:
  - Accumulate *momentum*
    - Cancels out in wall-to-wall direction
    - Accumulates in valley floor direction
  - Scale learning rate (step-size) per parameter based on past gradients
    - Smaller steps on steeper slopes
    - Larger steps on shallower slopes
  - Interactive demo
  - Overview
- Adam is generally a good starting point

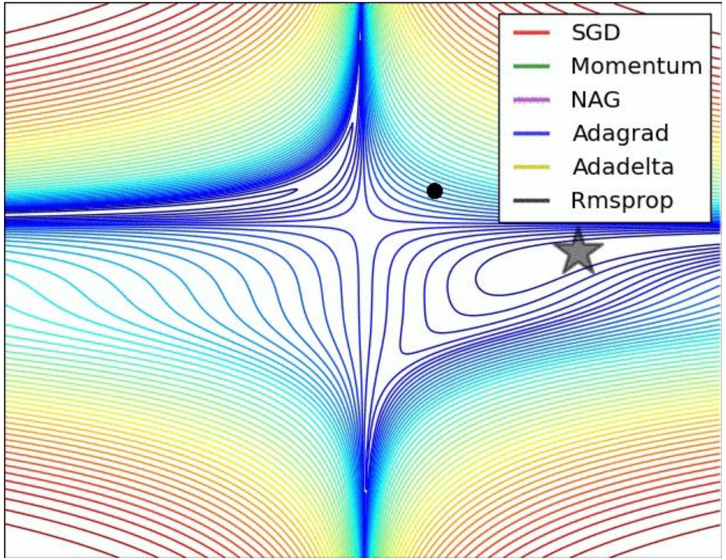


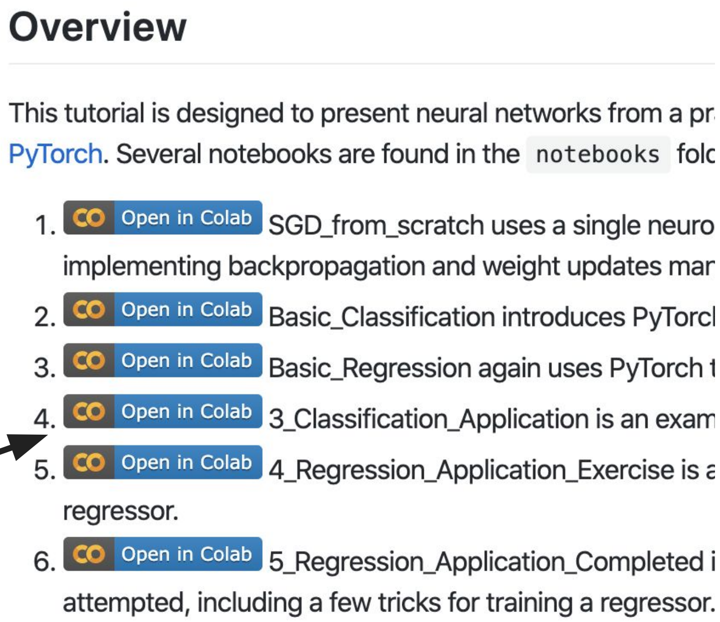
Image source

TUTORIAL

PYTORCH TUTORIAL

- https://github.com/GilesStrong/PyTorch_Tutorial
- Jupyter notebooks runnable on Google Colab

Click the badges to load in Colab

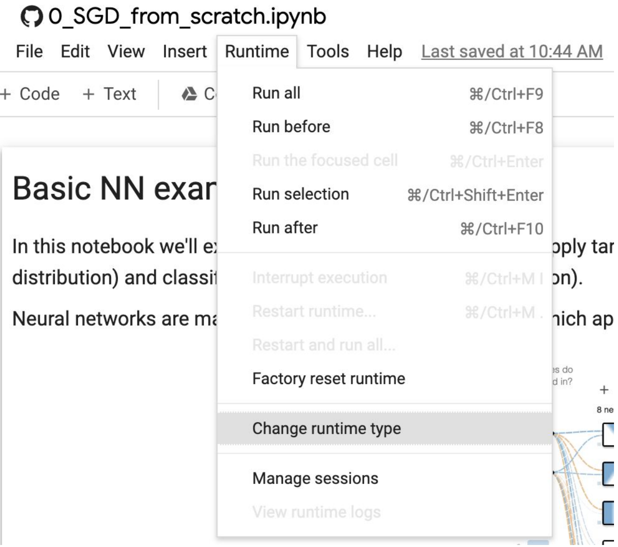


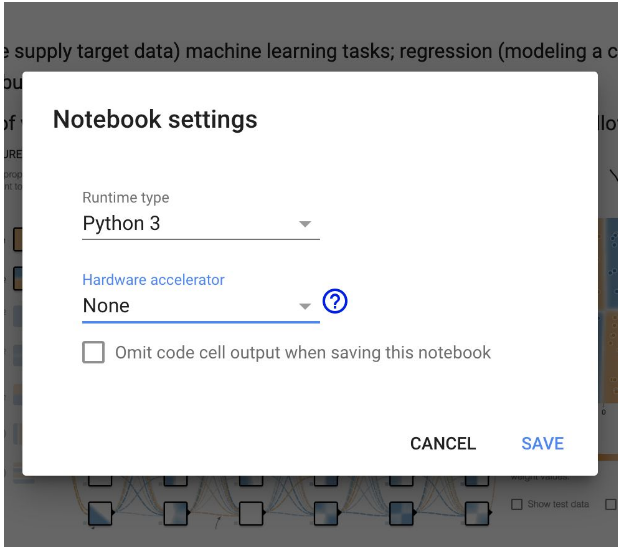
Overview

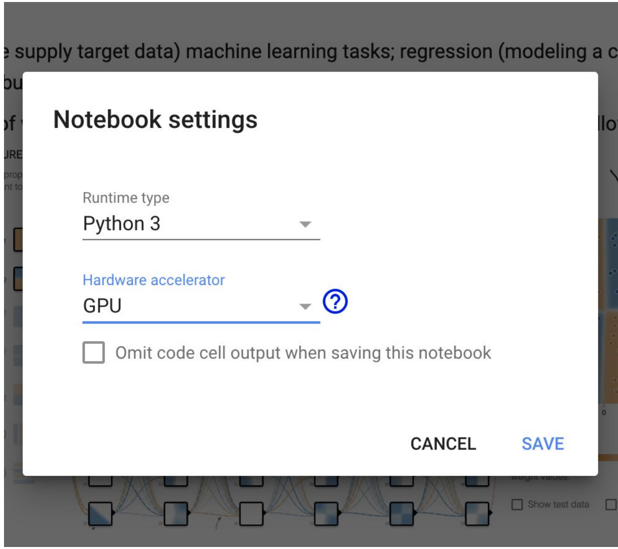
This tutorial is designed to present neural networks from a pr
PyTorch. Several notebooks are found in the `notebooks` folc

1. CO Open in Colab SGD_from_scratch uses a single neuro implementing backpropagation and weight updates mar
2. CO Open in Colab Basic_Classification introduces PyTorcl
3. CO Open in Colab Basic_Regression again uses PyTorch t
4. CO Open in Colab 3_Classification_Application is an exam
5. CO Open in Colab 4_Regression_Application_Exercise is a regressor.
6. CO Open in Colab 5_Regression_Application_Completed i attempted, including a few tricks for training a regressor.

GOOGLE COLAB RUNTIME







WORKING IN COLAB

- Colab is similar to Jupyter notebooks
- Run cells by clicking in them and pressing:
  - Control + Enter to run the cell and keep it selected
  - Shift + Enter to run the cell and select the next
- “Runtime” → “Restart Runtime” used to restart and clear memory
- “Runtime” → “Manage Sessions” used to shut down old sessions

FURTHER LEARNING RESOURCES

LIBRARIES

- Most ML development done in Python 3
- Two main libraries: PyTorch & TensorFlow
- Both relatively low-level = need good understanding of NNs to use directly; but wrapper libraries exist to provide high-level APIs, e.g.
  - Keras - no longer developed standalone, but now included in TensorFlow 2.x
  - Fast.AI - PyTorch wrapper with best practices for image, text, & tabular data but doesn’t support weighted data
  - LUMIN - My own library (in beta) - PyTorch wrapper with best practices for weighted tabular data, plus utilities for HEP, statistics, and interpretation

THEORY & PRACTICE: COURSES

- Fast.AI - free, practical courses; videos + library; top-down experiment first, theory later teaching style:
  - Machine learning - Fundamentals for data science + Python programming
  - Deep learning I - Best practices for image, text, & tabular data
  - Deep learning II - Building DNNs from scratch
- Stanford course - YouTube lecture series on theory of NNs
- Yandex MLHEP course - annual week-long intensive introduction to ML for HEP

THEORY & PRACTICE: EXPERIENCE

- Kaggle - data science challenge platform; wide range of challenges, get to see how others approach problems
- Paper reimplementation - helps get more familiar with library, and comfortable changing parts of it, e.g. SELU activation, categorical embedding, learning-rate annealing, and weight averaging

INTERESTING & USEFUL PAPERS

- A disciplined approach to neural network hyper-parameters: Part 1 -- learning rate, batch size, momentum, and weight decay - Smith 2018
- SGDR: Stochastic Gradient Descent with Warm Restarts - Loshchilov & Hutter, 2016
- Entity Embeddings of Categorical Variables - Guo & Berkhahn, 2016
- Regularization for Deep Learning: A Taxonomy - Kukačka, Golkov, & Cremers, 2017