# Tensorflow:

<https://www.youtube.com/watch?v=PsE73jk55cE>

Tensorflow is changed quite often by Google, so stay tuned. Using TF 2.0 is essential these days as Google changes namings, Capitation, etc.

It’s an advantage using Google COlab for free GPU. (dual-core, 12gb RAM and GPU).

GPU installing is a pain and not easy with all drivers, selection of right hardware, etc. 🡪 prefer to work on Cloud.

Why Tensorflow -> support by Google Cloud and in Python.

Alternatives:

* MXNet: Apache foundations deep learning API
* Theano: Python from the academics that create deep learning
* Keras: also by google higher level framework that allows the use of TensorFlow, MXNetand Theano interchangeably. Torch is used by Google DeepMind, the Facebook AI Research IBG, etc. 🡪 very advanced and powerful, but not mainstream.
* PaddlePaddle – Baidu’s deep learning API
* Deeplearning4J – Java based support major platforms
* Computational Network Toolkit (CNTK) – Microsoft. Supports Windows/ Linux, commandline only.
* H2O – Java based, support all major platforms. Limited support for computer vision. No GPU support

Tensorflow is low-level mathematics API similar to Numpy – however Tensorflow is built for ML. TF allows you to define compute graphs with Python.

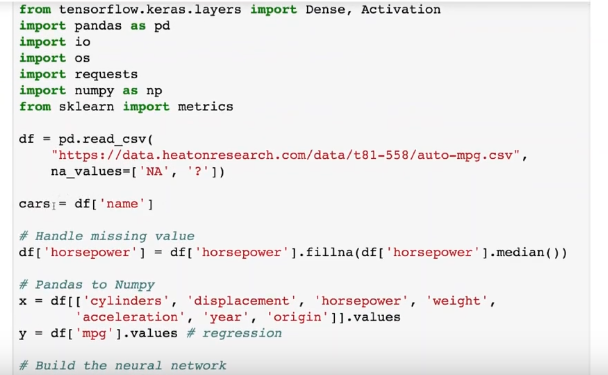
Tensorboard command line utility can be used to view these graphs.

# KERAS - Regression

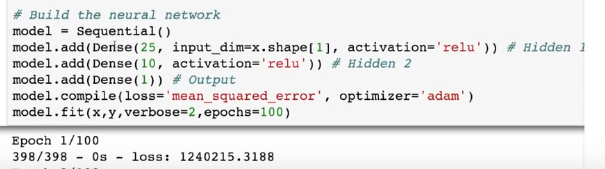
<https://www.youtube.com/watch?v=PsE73jk55cE>

Classical Data – MPG data set.

**Data prep**



Define x and y 🡪 x = predictors, y = label



Model = Sequential() … Model we create. Layers that go in sequence.

Model.add(Dense(25, input\_dim= x.shape[1], activation=’relu’)) … a dense layer is a simple layer where every neuron in the first layer in connected to the next layer. It’s very common especially for tabular data.

Input-shape[1] - how many columns we have

Input-shape[0] - how many row we have

Activation=’relu’ – rectified linear unit

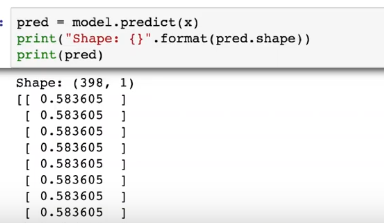
Model.add(Dense(10, input\_dim= x.shape[1], activation=’relu’)) … here we do not have to tell again the dimensions.

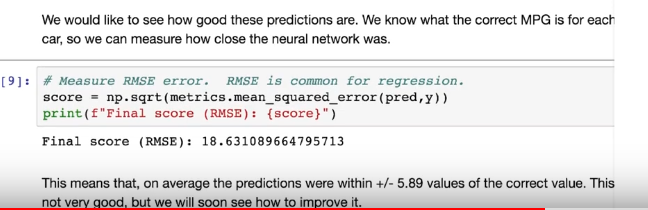
Model.add(Dense(1)) … Output. Always for 1 for regressions!!

Model.compile(loss = ‘mean\_squared\_error’, optimized= ‘adam’) …. Also important in regression to have mean squared error in your loss. Optimizer usually will use “Adam”, other one are possible as well.

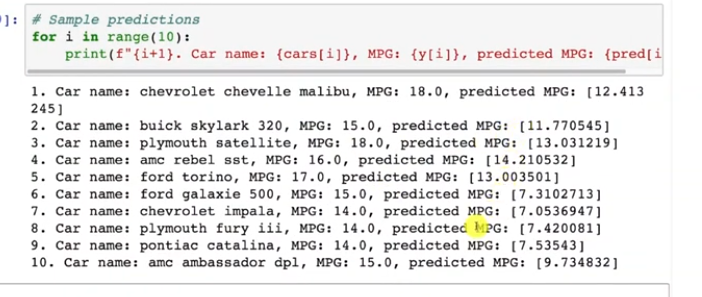
Model.fil(x,y, verbose=2, epochs=100) … epochs is how long NN are trained for. Verbose means print out data as we go… during run we see that the loss shrinks and model gets better

Output:





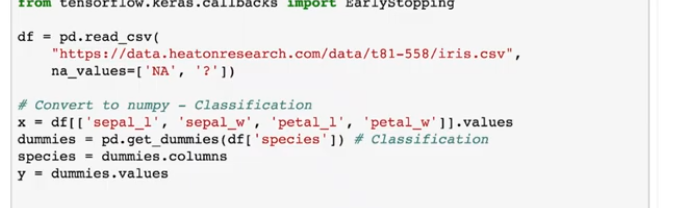
You can also print out various cars with real MPG and the predicted avalues



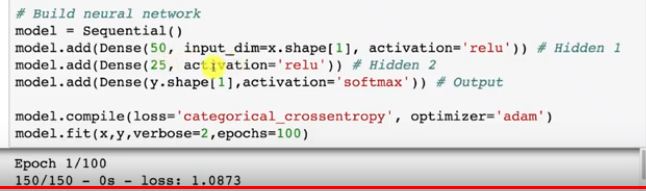
# KERAS – Classification

Iris data set

**Data-prep:**



It is very simple.



Major difference:

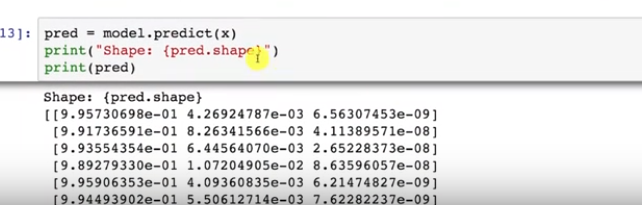
* activation is „softmax” instead of “relu” (if not binary)
* Loss is “categorical\_crossentropy” instead of “mean\_squared\_error”.

If binary category (true vs. false), another loss function would have been used.

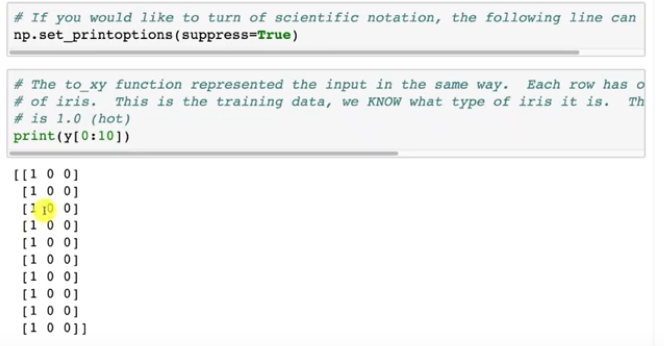
Pred = model.predict(x)

Print(“Shape: {pred.shape}”)

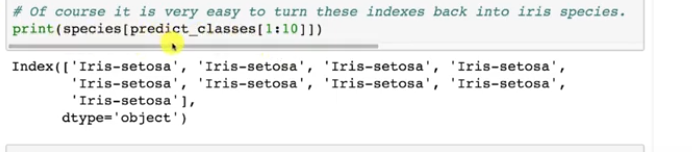
Print(pred)



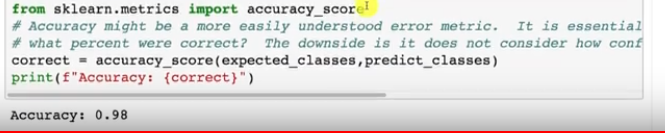
If you want to avoid scientific notation



Or if you want the name instead of the class:



Accuracy:



Feed-in ad-hoc values for predictions:

