slides: <http://bit.ly/lukas-qcom-course>

git clone <https://github.com/lukas/ml-class>

instruction: <https://gist.github.com/lukas/dd8b25a680fce822548db0d223b5028e#file-gistfile1-txt>

examples in the class: <https://app.wandb.ai/ml-class/> (installation instruction is also here. Read carefully)

from: <https://github.com/lukas> and <https://github.com/vanpelt>

AWS instructions: go/lukas-ML, or <https://github.qualcomm.com/cloud-integration-team/Deep-Learning-Class/blob/master/documentation/EC2_INTERACTIVE.md>

slides: bit.ly/qcom-ml-class, or <https://drive.google.com/file/d/1u_Yu2DTVpW0WQ1d2A_zgI60Z-mggJ9K5/view>

Tips:

* go to "Stack overflow" web site for questions
* "Decision trees" is probably the most robust model to use (no weird data can cause problems)
* "boosted tree" is one of the most common algo in production.
* "adam" is one of the most robust method as optimizer ("perceptron-linear.py"):

model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

* Whenver use neural network, always normalize data, see "perceptron-normalize.py"
* Neural network - last layer use "softmax", previous layers "relu" is good enough - other func won't make much difference, but "relu" is faster
* Need to install "sudo pip install scikit-image" for CNN example to process "dog.jpg"
* For "mlp" - multi-layer perceptron, use dropout technique if data is over fitting (training accuracy greater than validation accuracy)

Knowledge points:

* How to selection subset of features (in "pipeline.py"):

*from sklearn.feature\_selection import Select Best*

*from sklearn.feature\_selection import chi2*

*p=Pipeline(steps=[('counts',CountVectorizer()),*

*('select', SelectKBest(chi2, k=1000))*

*('multinomialnb',MultinomialNB())])*

* Datasets from Keras:

# for digits

**from**keras.datasets**import**mnist

(X\_train,y\_train),(X\_test,y\_test)=mnist.load\_data()

#for fashion images

**from**keras.datasets**import**mnist

(X\_train,y\_train),(X\_test,y\_test)=mnist.load\_data()

* Use "model.summary()" to print out the table for # of parameters. E.g. below:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

flatten\_1 (Flatten) (None, 784) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 100) 78500

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 10) 1010

=================================================================

Total params: 79,510

Trainable params: 79,510

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_