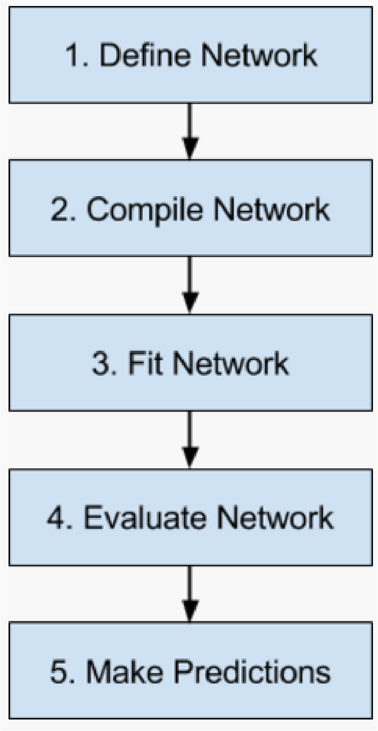
Keras: A high level API with best practice defaults

Keras workflow



1. Define Network

2. Compile Network

3. Fit Network

4. Evaluate Network

5. Make Predictions

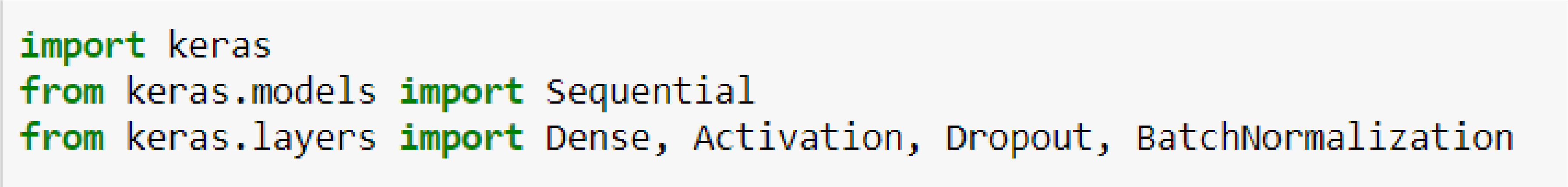
- Keras is a high-level NN API for TensorFlow and Theano.
- Is now shipped with TF (or can be imported as python library)
- Can be mixed with TF code
- Offers many predefined layers
- Each layer has a default "best practice choices of parameters"
- Allows for easy and fast prototyping (define only key parameters)
- Supports fcNN, CNNs, RNNs ...
- Supports arbitrary connectivity schemes and NN architectures

Keras: import keras and the required layers

Keras provides two ways to define a model:

1) Sequential API: simple, good for linear stack of layers

2) Functional API: flexible, required for complicated architectures



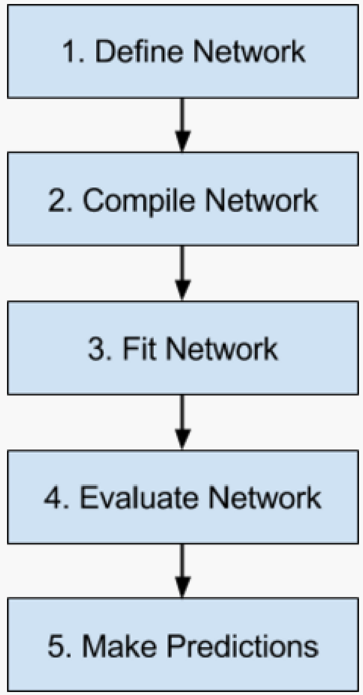
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, BatchNormalization
```

For documentation see:

https://keras.io/

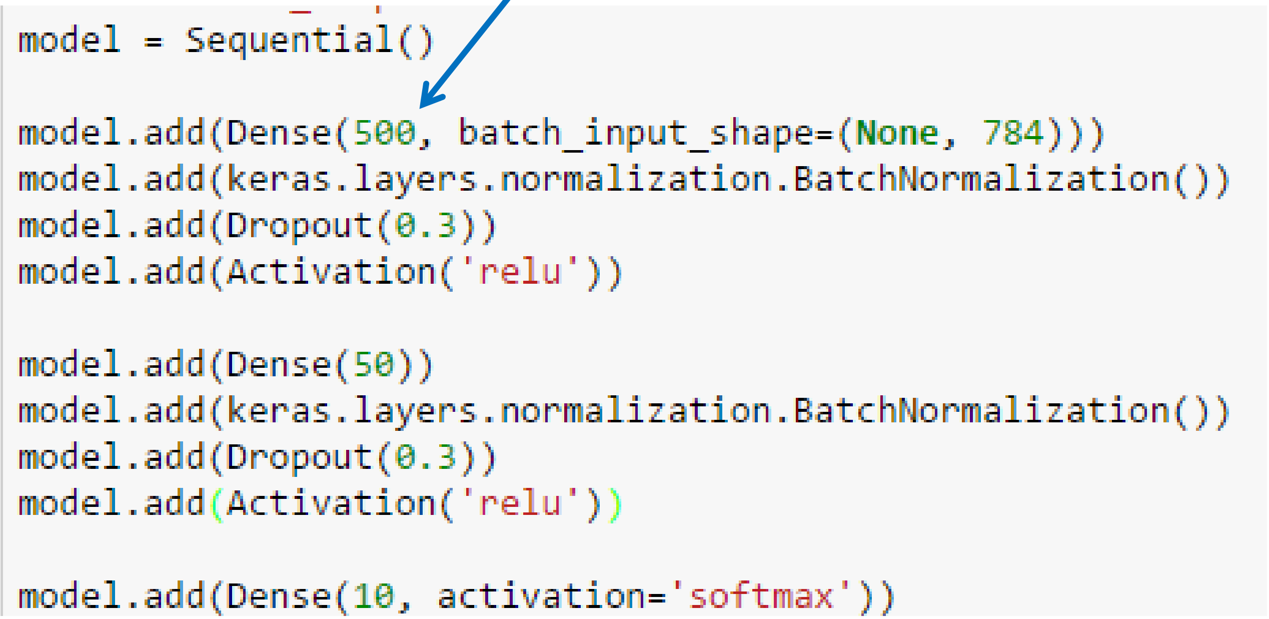
Remark: Keras can be used as API for theano and tf and which differ in the shape of expectedtensors – i.e. #channels is in tf at last position and in theano not.

Keras: A high level API with best practice defaults



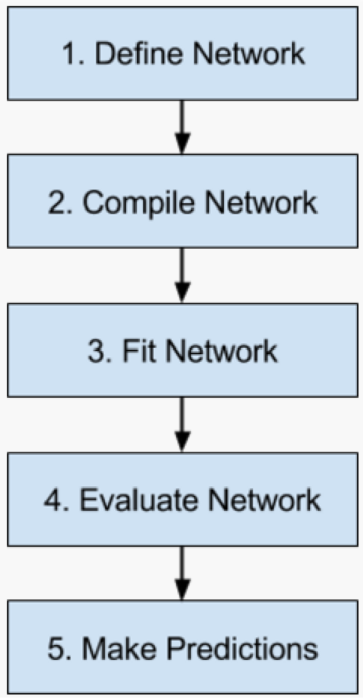
1. Define Network
2. Compile Network
3. Fit Network
4. Evaluate Network
5. Make Predictions

Number of neurons in (first) hidden dense layers (will be input to next layer)



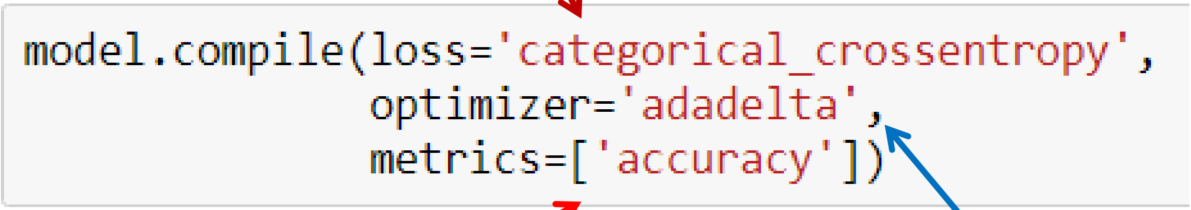
```
model = Sequential()

model.add(Dense(500, batch_input_shape=(None, 784)))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(50))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))

model.add(Dense(10, activation='softmax'))
```

Keras: A high level API with best practice defaults



1. Define Network
2. Compile Network
3. Fit Network
4. Evaluate Network
5. Make Predictions

loss function that will be minimized

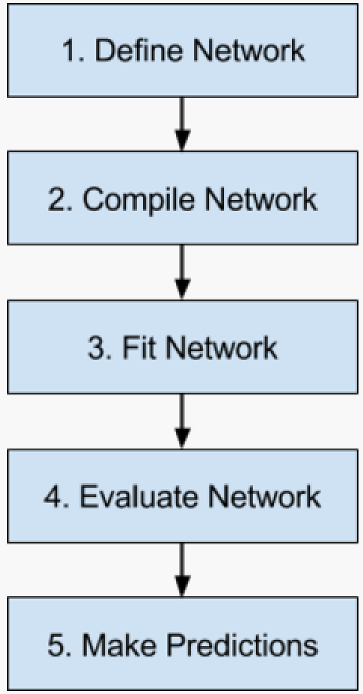


```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

easiest optimizer is SGD (stochastic gradient descent)

Which metrics besides ‘loss’ do we want to collect during training

Keras: A high level API with best practice defaults



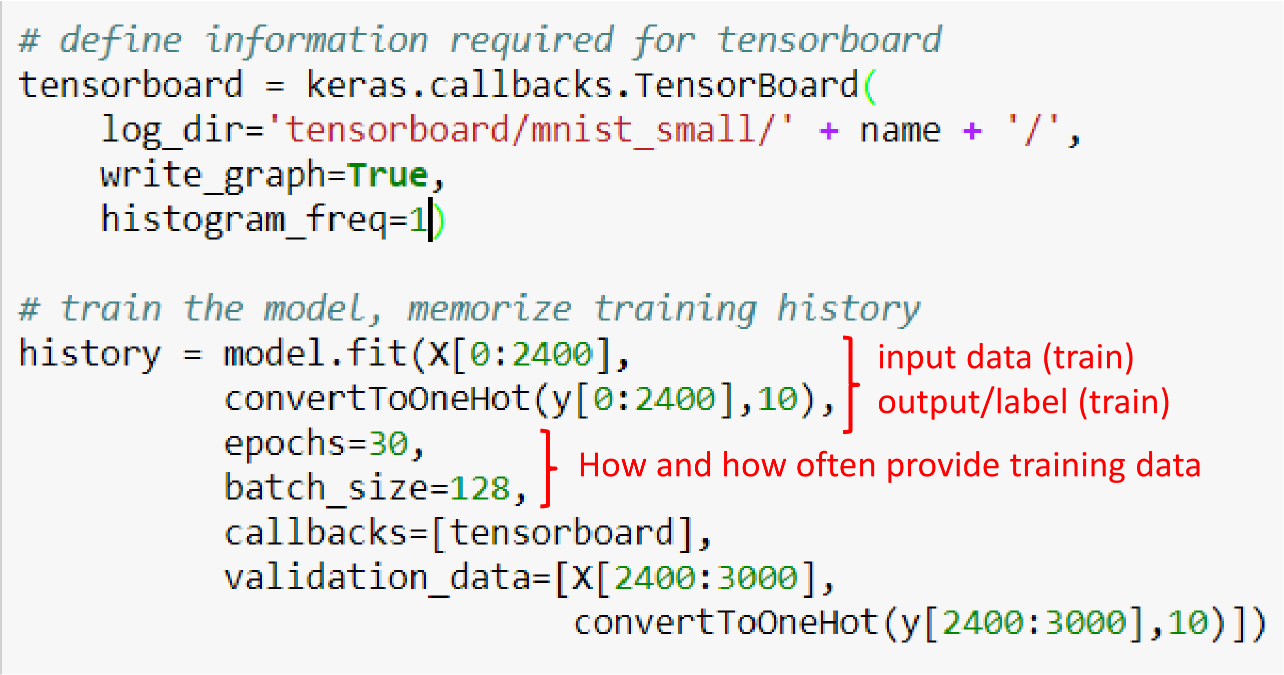
1. Define Network

2. Compile Network

3. Fit Network

4. Evaluate Network

5. Make Predictions



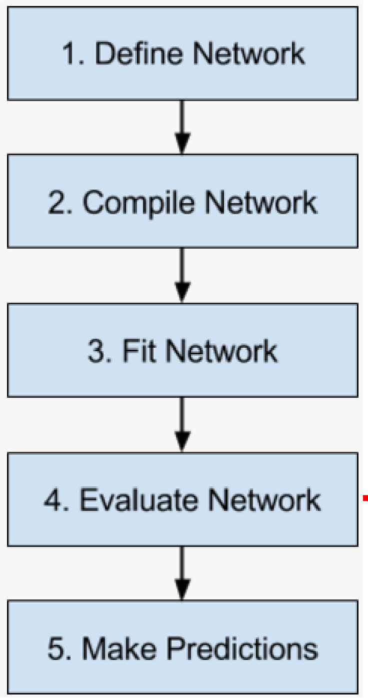
```
# define information required for tensorboard
tensorboard = keras.callbacks.TensorBoard(
    log_dir='tensorboard/mnist_small/' + name + '/',
    write_graph=True,
    histogram_freq=1)

# train the model, memorize training history
history = model.fit(X[0:2400],
          convertToOneHot(y[0:2400],10),
          epochs=30,
          batch_size=128,
          callbacks=[tensorboard],
          validation_data=[X[2400:3000],
                           convertToOneHot(y[2400:3000],10)])
```

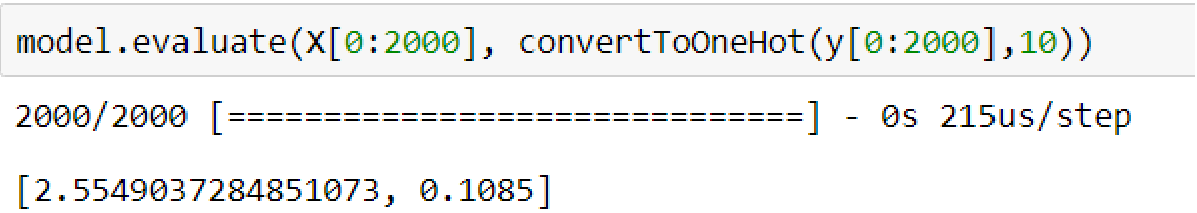
input data (train)

output/label (train)

How and how often provide training data

Keras: A high level API with best practice defaults

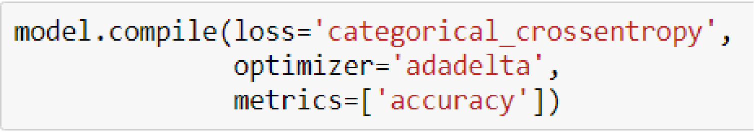




```
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))

2000/2000 [==============================] - 0s 215us/step

[2.5549037284851073, 0.1085]
```

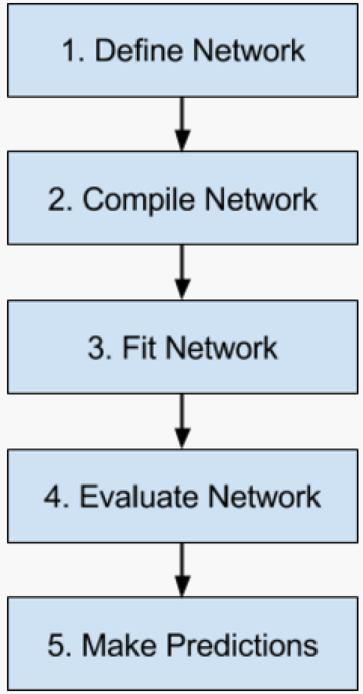
loss

metrics: accuracy



```
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])
```

Keras: A high level API with best practice defaults



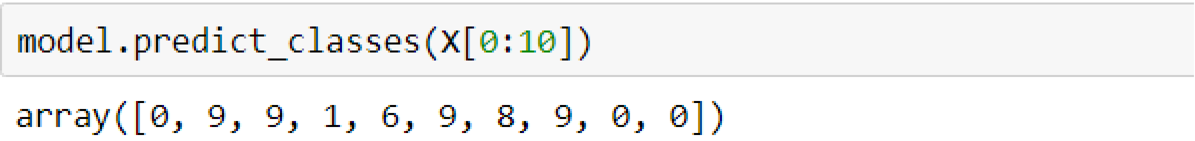
1. Define Network

2. Compile Network

3. Fit Network

4. Evaluate Network

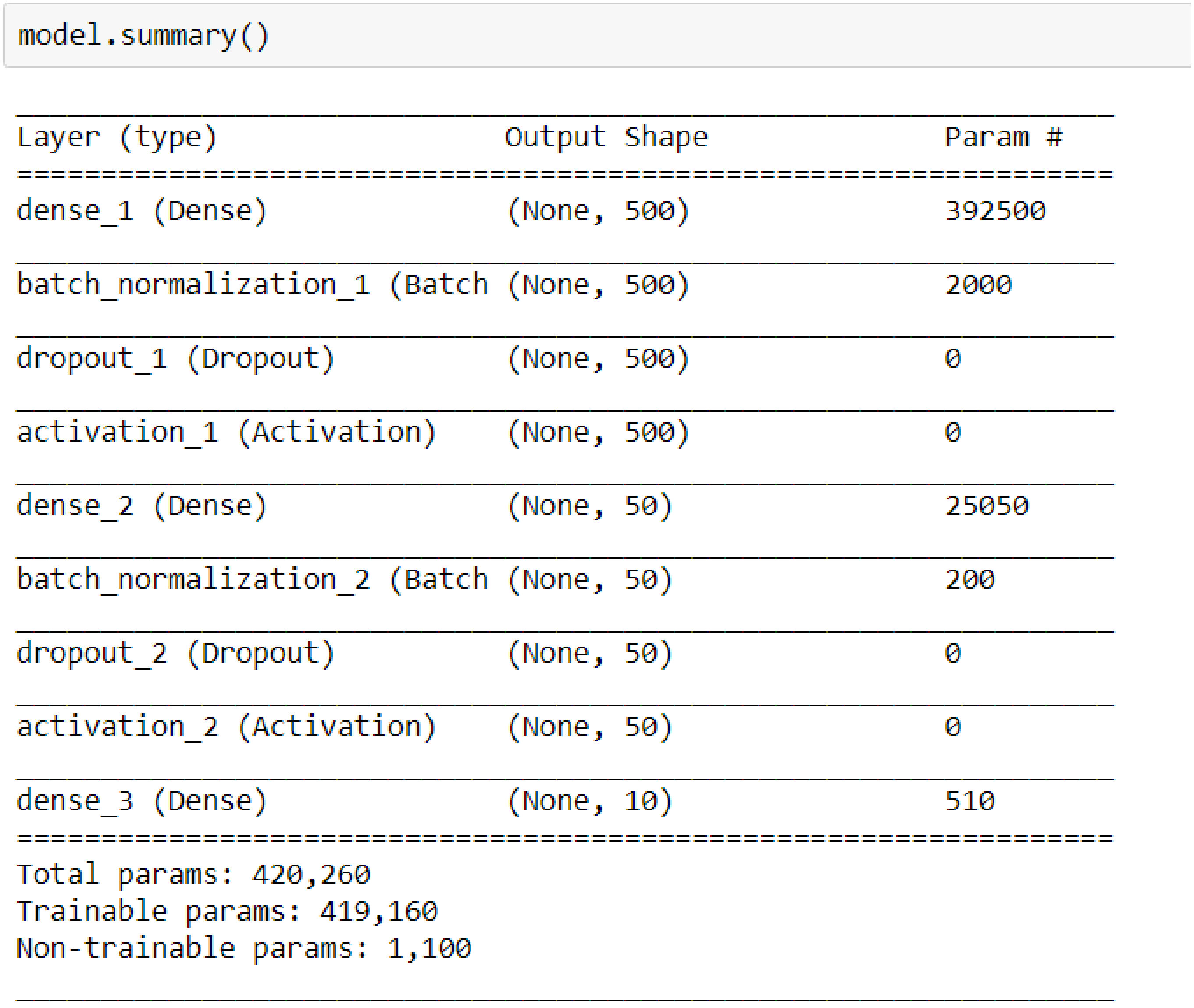
5. Make Predictions



```
model.predict_classes(X[0:10])
```

```
array([0, 9, 9, 1, 6, 9, 8, 9, 0, 0])
```

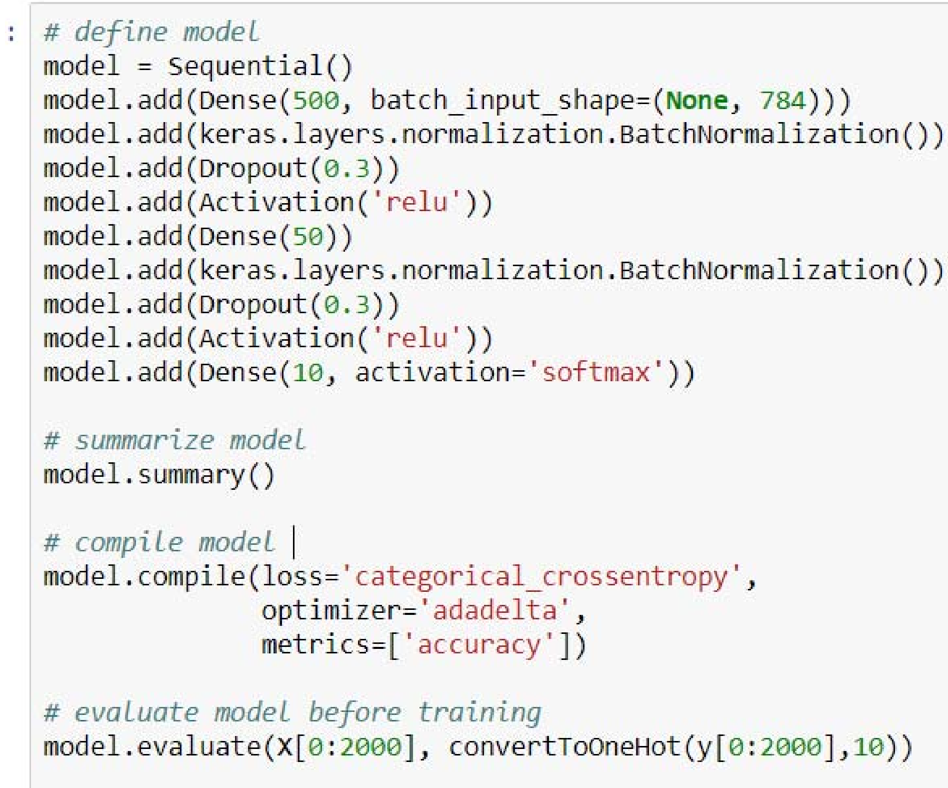
Keras: gives nice summary of model architecture



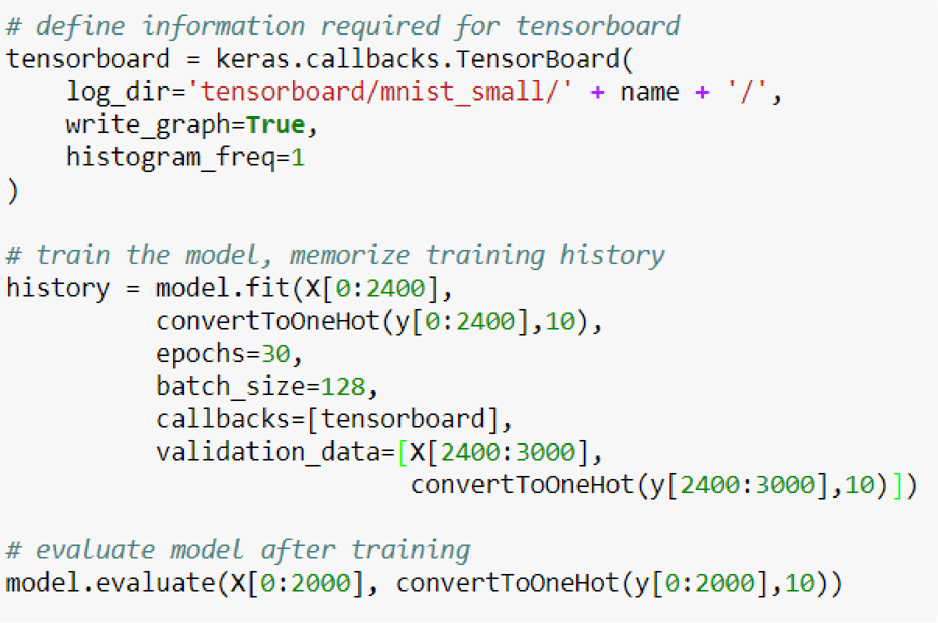
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 500)               392500

_________________________________________________________________
batch_normalization_1 (Batch (None, 500)               2000

_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0

_________________________________________________________________
activation_1 (Activation)    (None, 500)               0

_________________________________________________________________
dense_2 (Dense)              (None, 50)                25050

_________________________________________________________________
batch_normalization_2 (Batch (None, 50)                200

_________________________________________________________________
dropout_2 (Dropout)          (None, 50)                0

_________________________________________________________________
activation_2 (Activation)    (None, 50)                0

_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
=================================================================
Total params: 420,260
Trainable params: 419,160
Non-trainable params: 1,100

_________________________________________________________________
```

Keras: put the code together



```
# define model
model = Sequential()
model.add(Dense(500, batch_input_shape=(None, 784)))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(keras.layers.normalization.BatchNormalization())
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(Dense(10, activation='softmax'))

# summarize model
model.summary()

# compile model
model.compile(loss='categorical_crossentropy',
              optimizer='adadelta',
              metrics=['accuracy'])

# evaluate model before training
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))
```



```
# define information required for tensorboard
tensorboard = keras.callbacks.TensorBoard(
    log_dir='tensorboard/mnist_small/' + name + '/',
    write_graph=True,
    histogram_freq=1
)

# train the model, memorize training history
history = model.fit(X[0:2400],
          convertToOneHot(y[0:2400],10),
          epochs=30,
          batch_size=128,
          callbacks=[tensorboard],
          validation_data=[X[2400:3000],
                           convertToOneHot(y[2400:3000],10)])

# evaluate model after training
model.evaluate(X[0:2000], convertToOneHot(y[0:2000],10))
```