Introduction to Keras

Keras Sequential

```python
from keras.models import Sequential
from keras.layers import Dense, Activation
```

```
Using TensorFlow backend.
```

```python
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax')])

# or
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))

# or
model = Sequential([
    Dense(32, input_shape=(784,), activation='relu'),
    Dense(10, activation='softmax')])
```

```python
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_165 (Dense)            (None, 32)                25120
_________________________________________________________________
activation_113 (Activation)  (None, 32)                0
_________________________________________________________________
dense_166 (Dense)            (None, 10)                330
_________________________________________________________________
activation_114 (Activation)  (None, 10)                0
=================================================================
Total params: 25,450.0
Trainable params: 25,450.0
Non-trainable params: 0.0
_________________________________________________________________
```

Setting Optimizer

compile

```
compile(self, optimizer, loss, metrics=None, sample_weight_mode=None)
```

Configures the learning process.

Arguments

- **optimizer**: str (name of optimizer) or optimizer object. See optimizers.
- **loss**: str (name of objective function) or objective function. See objectives.
- **metrics**: list of metrics to be evaluated by the model during training and testing. Typically you will use `metrics=['accuracy']`. See metrics.

```
model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

Training the model

```
fit(self, x, y, batch_size=32, epochs=10, verbose=1, callbacks=None, validation_split=0.0, validatio
```

Trains the model for a fixed number of epochs.

Arguments

- **x**: input data, as a Numpy array or list of Numpy arrays (if the model has multiple inputs).
- **y**: labels, as a Numpy array.
- **batch_size**: integer. Number of samples per gradient update.
- **epochs**: integer, the number of epochs to train the model.
- **verbose**: 0 for no logging to stdout, 1 for progress bar logging, 2 for one log line per epoch.
- **callbacks**: list of `keras.callbacks.Callback` instances. List of callbacks to apply during training. See callbacks.
- **validation_split**: float (0. < x < 1). Fraction of the data to use as held-out validation data.
- **validation_data**: tuple (x_val, y_val) or tuple (x_val, y_val, val_sample_weights) to be used as held-out validation data. Will override validation_split.
- **shuffle**: boolean or str (for 'batch'). Whether to shuffle the samples at each epoch. 'batch' is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks.

Preparing MNIST data

```
from keras.datasets import mnist
import keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

num_classes = 10
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
60000 train samples
10000 test samples
```

Fit Model

```
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1)
```

```
Epoch 1/10
60000/60000 [==============================] - 3s - loss: 0.4897 - acc: 0.8680
Epoch 2/10
60000/60000 [==============================] - 2s - loss: 0.2425 - acc: 0.9325
Epoch 3/10
60000/60000 [==============================] - 2s - loss: 0.1993 - acc: 0.9442
Epoch 4/10
60000/60000 [==============================] - 1s - loss: 0.1728 - acc: 0.9514
Epoch 5/10
60000/60000 [==============================] - 1s - loss: 0.1519 - acc: 0.9570
Epoch 6/10
60000/60000 [==============================] - 2s - loss: 0.1378 - acc: 0.9612
Epoch 7/10
60000/60000 [==============================] - 2s - loss: 0.1263 - acc: 0.9636
Epoch 8/10
60000/60000 [==============================] - 2s - loss: 0.1155 - acc: 0.9669
Epoch 9/10
60000/60000 [==============================] - 1s - loss: 0.1071 - acc: 0.9694
Epoch 10/10
60000/60000 [==============================] - 1s - loss: 0.0997 - acc: 0.9715
```

Fit with Validation

```
model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1,
          validation_split=.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 2s - loss: 0.5146 - acc: 0.8616 - val_loss: 0.2425 - val_acc: 0.9322
Epoch 2/10
54000/54000 [==============================] - 1s - loss: 0.2618 - acc: 0.9266 - val_loss: 0.1934 - val_acc: 0.9442
Epoch 3/10
54000/54000 [==============================] - 1s - loss: 0.2161 - acc: 0.9397 - val_loss: 0.1717 - val_acc: 0.9537
Epoch 4/10
54000/54000 [==============================] - 1s - loss: 0.1879 - acc: 0.9470 - val_loss: 0.1519 - val_acc: 0.9570
Epoch 5/10
54000/54000 [==============================] - 1s - loss: 0.1676 - acc: 0.9528 - val_loss: 0.1440 - val_acc: 0.9603
Epoch 6/10
54000/54000 [==============================] - 1s - loss: 0.1506 - acc: 0.9566 - val_loss: 0.1296 - val_acc: 0.9638
Epoch 7/10
54000/54000 [==============================] - 1s - loss: 0.1378 - acc: 0.9603 - val_loss: 0.1281 - val_acc: 0.9627
Epoch 8/10
54000/54000 [==============================] - 1s - loss: 0.1268 - acc: 0.9626 - val_loss: 0.1177 - val_acc: 0.9660
Epoch 9/10
54000/54000 [==============================] - 1s - loss: 0.1175 - acc: 0.9659 - val_loss: 0.1159 - val_acc: 0.9657
Epoch 10/10
54000/54000 [==============================] - 1s - loss: 0.1096 - acc: 0.9677 - val_loss: 0.1131 - val_acc: 0.9662
```

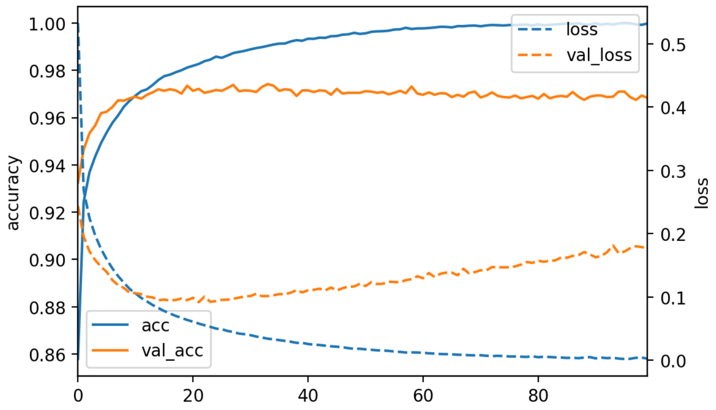
Evaluating on Test Set

```
score = model.evaluate(X_test, y_test, verbose=0)
print("Test loss: {:.3f}".format(score[0]))
print("Test Accuracy: {:.3f}".format(score[1]))
```

```
Test loss: 0.120
Test Accuracy: 0.966
```

Loggers and Callbacks

```
history_callback = model.fit(X_train, y_train, batch_size=128,
                             epochs=100, verbose=1, validation_split=.1)
pd.DataFrame(history_callback.history).plot()
```



Wrappers for sklearn

```python
from keras.wrappers.scikit_learn import KerasClassifier, KerasRegressor
from sklearn.model_selection import GridSearchCV

def make_model(optimizer="adam", hidden_size=32):
    model = Sequential([
        Dense(hidden_size, input_shape=(784,)),
        Activation('relu'),
        Dense(10),
        Activation('softmax'),
    ])
    model.compile(optimizer=optimizer,loss="categorical_crossentropy",
                  metrics=['accuracy'])
    return model

clf = KerasClassifier(make_model)
param_grid = {'epochs': [1, 5, 10],  # epochs is fit parameter, not in make_model!
              'hidden_size': [32, 64, 256]}
grid = GridSearchCV(clf, param_grid=param_grid, cv=5)
grid.fit(X_train, y_train)
```

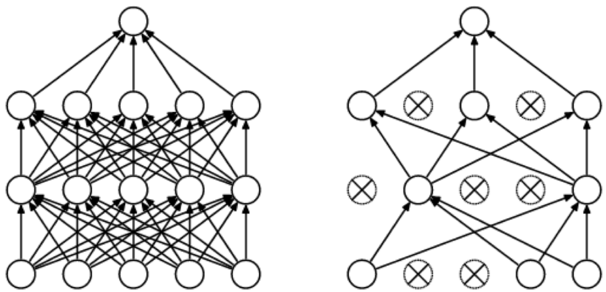
```python
res = pd.DataFrame(grid.cv_results_)
res.pivot_table(index=["param_epochs", "param_hidden_size"],
                values=['mean_train_score', "mean_test_score"])
```

param_epochs	param_hidden_size	mean_test_score	mean_train_score
1	32	0.930017	0.935350
	64	0.941433	0.948358
	256	0.959117	0.966929
5	32	0.956417	0.969746
	64	0.967317	0.983113
	256	0.973900	0.992196
10	32	0.960100	0.979671
	64	0.968617	0.992025
	256	0.975050	0.996396

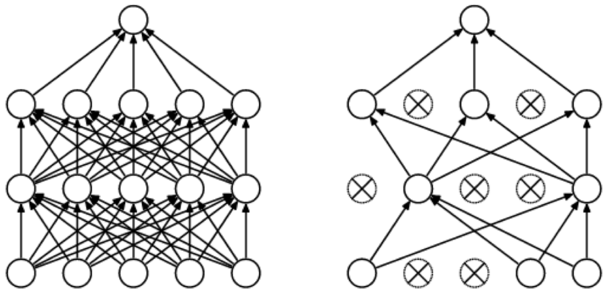
Drop-out

;

Drop-out Regularization


(a) Standard Neural Net (b) After applying dropout.

Drop-out Regularization



(a) Standard Neural Net (b) After applying dropout.

- https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf
- Rate often as high as .5, i.e. 50% of units set to zero!
- Predictions: use all weights, down-weight by rate

Ensemble Interpretation

- Every possible configuration represents different network.
- With p=.5 we jointly learn $\binom{n}{n/2}$ networks
- Networks share weights
- For last layer dropout: prediction is approximate geometric mean of predictions of sub-networks.

Implementing Drop-Out

```
from keras.layers import Dropout

model_dropout = Sequential([
    Dense(1024, input_shape=(784,), activation='relu'),
    Dropout(.5),
    Dense(1024, activation='relu'),
    Dropout(.5),
    Dense(10, activation='softmax'),
])
model_dropout.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
history_dropout = model_dropout.fit(X_train, y_train, batch_size=128,
                            epochs=20, verbose=1, validation_split=.1)
```


;

When to use drop-out

- Avoids overfitting
- Allows using much deeper and larger models
- Slows down training somewhat
- Wasn’t able to produce better results on MNIST (I don’t have a GPU) but should be possible