Hyper-parameter tuning

**GBDT Hyper Parameter Tuning**

<table>
<thead>
<tr>
<th>Hyper Parameter</th>
<th>Tuning Approach</th>
<th>Range</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Trees</td>
<td>Fixed value</td>
<td>100-1000</td>
<td>Depending on dataset</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>Fixed =&gt; Fine Tune</td>
<td>[2 - 10] / # of Trees</td>
<td>Depending on # of trees</td>
</tr>
<tr>
<td>Row Sampling</td>
<td>Grid Search</td>
<td>[.5, .75, 1.0]</td>
<td></td>
</tr>
<tr>
<td>Column Sampling</td>
<td>Grid Search</td>
<td>[.4, .6, .8, 1.0]</td>
<td></td>
</tr>
<tr>
<td>Min Leaf Weight</td>
<td>Fixed =&gt; Fine Tune</td>
<td>.3(3% of rare events)</td>
<td>Rule of thumb</td>
</tr>
<tr>
<td>Max Tree Depth</td>
<td>Grid Search</td>
<td>[4, 6, 8, 10]</td>
<td></td>
</tr>
<tr>
<td>Min Split Gain</td>
<td>Fixed</td>
<td>0</td>
<td>Keep it 0</td>
</tr>
</tbody>
</table>

Best GBDT implementation today: [https://github.com/tschen/xgboost](https://github.com/tschen/xgboost) by Tianqi Chen (U of Washington)

### Tuning XGBoost

- **eta (learning rate) + num_round (number of trees)**
  - Examine objective metric in training/validation to quickly find good configuration
  - Target around 100 trees
- **max_depth (start with 6)** -- This is different from R GBM
- **min_child_weight (start with 1/sqrt(event rate))**
- **colsample_bytree (.3-.5)**
- **subsampling (leave at 1.0)**
- **gamma (usually is it OK to leave at 0.0)**

### Xgboost

- Eta
- Gamma
- Max_depth
- Min_child_weight
- Subsample
- Colsample_bytree
- Lambda
- alpha

- 0.01, 0.015, 0.025, 0.05, 0.1
- 0.05-0.1, 0.3, 0.5, 0.7, 0.9, 1.0
- 3, 5, 7, 9, 12, 15, 17, 25
- 1, 3, 5, 7
- 0.6, 0.7, 0.8, 0.9, 1.0
- 0.6, 0.7, 0.8, 0.9, 1.0
- 0.01-0.1, 1.0, RS*
- 0, 0.1, 0.5, 1.0 RS*

[https://github.com/aarshayj.analytics_vidhya/blob/master/Articles/Parameter_Tuning_XGBoost_with_Example/XGBoost%20models.ipynb](https://github.com/aarshayj.analytics_vidhya/blob/master/Articles/Parameter_Tuning_XGBoost_with_Example/XGBoost%20models.ipynb)
Hyper-parameter tuning

```python
# XGBoost on Otto dataset, Tune n_estimators
from pandas import read_csv
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot

# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
y = dataset[:,8]
# grid search
model = XGBClassifier()
n_estimators = range(10, 400, 20)
param_grid = dict(n_estimators=n_estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=7)
grid_search = GridSearchCV(model, param_grid, scoring="neg_log_loss", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X, y)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
# plot
pyplot.errorbar(n_estimators, means, yerr=stds)
pyplot.title("XGBoost n_estimators vs Log Loss")
pyplot.xlabel('n_estimators')
pyplot.ylabel('Log Loss')
pyplot.savefig('n_estimators.eps')
```
if True:
    test0=-1.
    for i in (0.01, 0.05, 1.0):
        for j in (100, 500, 1000):
            for k in (3, 5, 10):
                for l in (1.0, 1.0):
                    for m in (1.0, 1.0):
                        for n in (0, 2, 5):
                            clf = XGBClassifier( max_depth=k, learning_rate=i, n_estimators=j, subsample=l, colsample_bytree=m, gamma=n)
                            clf.fit(X,y)
                            test=np.mean(cross_val_score(clf, X, y, cv=3, scoring='roc_auc'))
                            if test0 < test:
                                print(i,j,k,l,m,n,test)
                            test0=test
                            pickle.dump(clf, open("pima.pickle.dat", "wb"))

# some time later...
# load model from file
loaded_model = pickle.load(open("pima.pickle.dat", "rb"))
# make predictions for test data
y_pred = loaded_model.predict(X)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y, predictions)
print("Accuracy: %2f%%" % (accuracy * 100.0))
Gaussian Processes regression: basic introductory example

The figures illustrate the interpolating property of the Gaussian Process model as well as its probabilistic nature in the form of a pointwise 95% confidence interval.

https://machinelearningmastery.com/what-is-bayesian-optimization/
import numpy as np
from numpy import loadtxt
from xgboost import XGBClassifier
from bayes_opt import BayesianOptimization
from sklearn.model_selection import cross_val_score

pbounds = {
    'learning_rate': (0.01, 1.0),
    'n_estimators': (100, 1000),
    'max_depth': (3, 10),
    'subsample': (1.0, 1.0),  # Change for big datasets
    'colsample': (1.0, 1.0),  # Change for datasets with lots of features
    'gamma': (0, 5)}

def xgboost_hyper_param(learning_rate, n_estimators, max_depth, subsample, colsample, gamma):
    dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',

    X = dataset[:,:8]
    y = dataset[:,8]
    max_depth = int(max_depth)
    n_estimators = int(n_estimators)
    clf = XGBClassifier( max_depth=max_depth, learning_rate=learning_rate, n_estimators=n_estimators,
                         subsample=subsample, colsample=colsample, gamma=gamma)
    return np.mean(cross_val_score(clf, X, y, cv=3, scoring='roc_auc'))

optimizer = BayesianOptimization( f=xgboost_hyper_param, pbounds=pbounds, random_state=1)
optimizer.maximize(init_points=3, n_iter=24, acq='ei', xi=0.01)
Stacking Ensemble

Don’t select a model, combine them

```python
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import VotingClassifier

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)

# create the sub models
estimators = []
model1 = LogisticRegression()
estimators.append(('logistic', model1))
model2 = DecisionTreeClassifier()
estimators.append(('cart', model2))
model3 = SVC()
estimators.append(('svm', model3))

# create the ensemble model
ensemble = VotingClassifier(estimators)
results = model_selection.cross_val_score(ensemble, X, Y, cv=kfold)
print(results.mean())
```
Advantages of LightGBM (2017)
1. **Faster training speed and higher efficiency:** Light GBM use histogram based algorithm i.e it buckets continuous feature values into discrete bins which fasten the training procedure.
2. **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.
3. **Better accuracy than any other boosting algorithm:** It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.
4. **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.
5. **Parallel learning supported.**

Catboost improves over LightGBM by handling categorical features better.

https://github.com/Microsoft/LightGBM/blob/master/docs/Experiments.rst#comparison-experiment
```python
>>> aa = np.zeros((3,3))
>>> aa[0,0] = 1
>>> aa[0,1] = 2
>>> aa[0,2] = 3
>>> aa[1,0] = 4
>>> aa[1,1] = 5
>>> aa[1,2] = 6
>>> aa[2,0] = 7
>>> aa[2,1] = 8
>>> aa[2,2] = 9
>>> aa
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]])
>>> a = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]])
>>> a
    0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
>>> a.drop(0, axis=1)
    1  2
0  2  3
1  5  6
2  8  9
```

```python
>>> import pandas as pd
>>> from pandas import DataFrame
>>> import numpy as np
df_train = pd.read_csv('C:/testAI/regression/regression.train', header=None, sep='	')
df_test = pd.read_csv('C:/testAI/regression/regression.test', header=None, sep='	')
y_train = df_train[0]
y_test = df_test[0]
X_train = df_train.drop(0, axis=1)
X_test = df_test.drop(0, axis=1)
```
# coding: utf-8
import lightgbm as lgb
import pandas as pd
if lgb.compat.MATPLOTLIB_INSTALLED:
    import matplotlib.pyplot as plt
else:
    raise ImportError('You need to install matplotlib for plot_example.py."
print('Loading data..."
#
    df_train = pd.read_csv('C:/testAI/regression/regression.train', header=None, sep="\t")
    df_test = pd.read_csv('C:/testAI/regression/regression.test', header=None, sep="\t")
    y_train = df_train[0]
    y_test = df_test[0]
    X_train = df_train.drop(0, axis=1)
    X_test = df_test.drop(0, axis=1)

# create dataset for lightgbm
lgb_train = lgb.Dataset(X_train, y_train)
    lgb_test = lgb.Dataset(X_test, y_test, reference=lgb_train)
# specify your configurations as a dict
    params = {'num_leaves': 5, 'metric': ['l1', 'l2'], 'verbose': 0}
evals_result = {}  # to record eval results for plotting
print('Starting training..."
# train
    gbm = lgb.train(params, lgb_train, num_boost_round=100, valid_sets=[lgb_train, lgb_test],
    feature_name=['f' + str(i + 1) for i in range(X_train.shape[-1])], categorical_feature=[21],
    evals_result=evals_result, verbose_eval=10)
print('Plotting metrics recorded during training..."
ax = lgb.plot_metric(evals_result, metric='l1')
plt.show()

print('Plotting feature importances..."
ax = lgb.plot_importance(gbm, max_num_features=10)
plt.show()
print('Plotting split value histogram..."
ax = lgb.plot_split_value_histogram(gbm, feature='f26', bins='auto')
plt.show()
print('Plotting 54th tree...") one tree use categorical feature to split
ax = lgb.plot_tree(gbm, tree_index=53, figsize=(15, 15), show_info=['split_gain'])
plt.show()
print('Plotting 54th tree with graphviz..."
graph = lgb.create_tree_digraph(gbm, tree_index=53, name="Tree54")
gap unharden(3.006)
Explaining Feature Importance

- Default Scikit-learn’s feature importance
- Permutation feature importance
- Drop Column feature importance
- Observation level feature importance

https://towardsdatascience.com/explaining-feature-importance-by-example-of-a-random-forest-d9166011959e
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import lightgbm as lgb
from numpy import loadtxt
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
X = dataset[:,0:8]
y = dataset[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
lgb_train = lgb.Dataset(X_train, y_train)
lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)
# specify your configurations as a dict
params = {
    'boosting_type': 'gbdt', 'objective': 'binary', 'metric': {'l2', 'l1'}, 'num_leaves': 31, 'learning_rate': 0.05,
    'feature_fraction': 0.9, 'bagging_fraction': 0.8, 'bagging_freq': 5, 'verbose': 0
}
print('Starting training...')
# train
gbm = lgb.train(params, lgb_train, num_boost_round=20, valid_sets=lgb_eval, early_stopping_rounds=5)
print('Saving model...')
# save model to file
gbm.save_model('model.txt')
print('Starting predicting...')
# predict
ypred = gbm.predict(X_test, num_iteration=gbm.best_iteration)
y_pred = [round(value) for value in ypred]
if True:
    accuracy = accuracy_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)
    print("Accuracy: ", accuracy)
    print("Confusion matrix:")
    print(cm)
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.collections import LineCollection
from sklearn import manifold
from sklearn.metrics import euclidean_distances
from sklearn.decomposition import PCA

n_samples = 20
seed = np.random.RandomState(seed=3)
X_true = seed.randint(0, 20, 2 * n_samples).astype(np.float)
X_true = X_true.reshape((n_samples, 2))

# Center the data
X_true -= X_true.mean()
similarities = euclidean_distances(X_true)

# Add noise to the similarities
noise = np.random.rand(n_samples, n_samples)
noise = noise + noise.T
noise[np.arange(noise.shape[0]), np.arange(noise.shape[0])] = 0
similarities += noise

mds = manifold.MDS(n_components=2, max_iter=3000, eps=1e-9, random_state=seed,
                   dissimilarity="precomputed", n_jobs=1)

pos = mds.fit(similarities).embedding_

nmds = manifold.MDS(n_components=2, metric=False, max_iter=3000, eps=1e-12,
                    dissimilarity="precomputed", random_state=seed,
                    n_jobs=1, n_init=1)
npos = nmds.fit_transform(similarities)

# Rescale the data
pos *= np.sqrt((X_true ** 2).sum()) / np.sqrt((pos ** 2).sum())
npos *= np.sqrt((X_true ** 2).sum()) / np.sqrt((npos ** 2).sum())

clf = PCA(n_components=2)
X_true = clf.fit_transform(X_true)
pos = clf.fit_transform(pos)
npos = clf.fit_transform(npos)

fig = plt.figure(1)

ax = plt.axes([0., 0., 1., 1.])
s = 100
plt.scatter(X_true[:, 0], X_true[:, 1], color="navy", s=s, lw=0, label="True Position")
plt.scatter(pos[:, 0], pos[:, 1], color="turquoise", s=s, lw=0, label="MDS")
plt.scatter(npos[:, 0], npos[:, 1], color="darkorange", s=s, lw=0, label="NMDS")
plt.legend(scatterpoints=1, loc="best", shadow=False)

similarities = similarities.max() / similarities * 100
similarities[np.isinf(similarities)] = 0

# Plot the edges

start_idx, end_idx = np.where(pos)

segments = [(X_true[i, :], X_true[j, :]) for i in range(len(pos)) for j in range(len(pos))]
values = np.abs(similarities)
lc = LineCollection(segments, zorder=0, cmap=plt.cm.Blues, norm=plt.Normalize(0,
                                                                         values.max()))
lc.set_array(similarities.flatten())
lc.set_linewidths(np.full(len(segments), 0.5))
ax.add_collection(lc)
plt.show()
t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.

```python
>>> import numpy as np
>>> from sklearn.manifold import TSNE
>>> X = np.array([[0, 0, 0], [0, 1, 1], [1, 0, 1], [1, 1, 1]])
>>> X_embedded = TSNE(n_components=2).fit_transform(X)
>>> X_embedded.shape
(4, 2)
```

PCA (principal component analysis) is a linear dimension reduction technique that seeks to maximize variance and preserves large pairwise distances.

https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
https://lvdmaaten.github.io/tsne/
https://github.com/aviolante/sas-python-work/blob/master/tSneExampleBlogPost.ipynb
t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data.
Keras: The Python Deep Learning library

Use Keras if you need a deep learning library that:
• Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
• Supports both convolutional networks and recurrent networks, as well as combinations of the two.
• Runs seamlessly on CPU and GPU.

Anaconda, Tensorflow, Keras Installation on Windows
https://www.youtube.com/watch?v=CcKf-iZ8umk&t=145s

Anaconda, Tensorflow, Keras Installation on Linux
https://www.youtube.com/watch?v=lyYaZWxgods

Python for Data Science Cheat Sheet: Keras
https://keras.io
Deep Learning packages

- TensorFlow : Google
- Keras : Google
- PyTorch : Facebook
- Chainer : Preferred Networks in partnership with IBM, Intel, Microsoft, and Nvidia
- Caffe : UC Berkeley
- CNTK : Microsoft
Reference

https://github.com/fchollet/keras-resources

https://www.youtube.com/watch?v=2L2u303FA8
Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model. Keras was initially developed for researchers, with the aim of enabling fast experimentation.

Keras has the following key features:

- It allows the same code to run seamlessly on CPU or GPU.
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
- It has built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- It supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, and so on. This means Keras is appropriate for building essentially any deep-learning model, from a generative adversarial network to a neural Turing machine.
Keras: high-level NN API

Keras is a model-level library, providing high-level building blocks for developing DL models. It doesn’t handle low-level operations such as tensor manipulation and differentiation.

Thanks to symbolic differentiation, you’ll never have to implement the Backpropagation algorithm by hand.

Currently, the three existing backend implementations are the TensorFlow backend, the Theano backend, and the Microsoft Cognitive Toolkit (CNTK) backend.

High-level NN API
Runs on top of TF, Theano, CNTK
Simple: stacking layers, connecting graphs
Open source
High performance
250,000 developers
Linux/MacOs/Windows, automatic differentiation, Cuda
Keras

It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Comparison of deep-learning software

Tensorflow 1.4: Keras

Keras has well over 250,000 users.

Keras is used at Google, Netflix, Uber, CERN, Yelp, Square, and hundreds of startups working on a wide range of problems.
PyTorch is an open-source machine learning library for Python, based on Torch, used for applications such as natural language processing. It is primarily developed by Facebook's artificial-intelligence research group, and Uber's "Pyro" software for probabilistic programming is built on it.
Install Keras

1. Install the Python scientific suite—NumPy and SciPy—and make sure you have a Basic Linear Algebra Subprogram (BLAS) library installed so your models run fast on CPU.

2. Install two extras packages that come in handy when using Keras: HDF5 (for saving large neural-network files) and Graphviz (for visualizing neural-network architectures).

3. Make sure your GPU can run deep-learning code, by installing CUDA drivers and cuDNN.


5. Install Keras.

2019, Installing TensorFlow, Keras, & Python 3.7 in Windows
https://www.youtube.com/watch?v=59duINoc8GM
https://www.youtube.com/watch?v=z0qhKP2liHs
We can now dive into practical Keras examples.

```python
from keras.models import Sequential

# only for linear stacks of layers, which is the most common network architecture by far
models.Sequential()
layers.Input(shape=(784,))
layers.Dense(32, activation='relu')
layers.Dense(10, activation='softmax')
```
Keras

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

'y = f(x) = f^4( f^{3}(f^{2}(f^{1}(x))))$

Output, y

Input, x

Width

Depth

1. Define Network

2. Compile Network

3. Fit Network

4. Evaluate Network

5. Make Predictions

ReLU, linear, sigmoid, softmax

uniform, normal

(f * g)(t) $\triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau) \, d\tau$. 

sigmoid
tanhReLUsoftplus
Yes you can run keras models on GPU. Few things you will have to check first.
1. your system has GPU (Nvidia. As AMD doesn't work yet)
2. You have installed the GPU version of tensorflow
3. You have installed CUDA
4. Verify that tensorflow is running with GPU

```python
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
```
OR

```python
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

Output will be something like this:
```python
[ name: "/cpu:0"device_type: "CPU", name: "/gpu:0"device_type: "GPU"
]
```

Once all this is done your model will run on GPU:

To check if keras(>=2.1.1) is using GPU:
```python
from keras import backend as K
K.tensorflow_backend._get_available_gpus()
```

You need to add the following block after importing keras. I am working on a machine which have 56 core CPU, and a GPU.
```python
import keras
import tensorflow as tf
config = tf.ConfigProto( device_count = {'GPU': 1 , 'CPU': 56} )
sess = tf.Session(config=config)
k keras.backend.set_session(sess)
```

https://keras.io/utils/#multi_gpu_model
https://stackoverflow.com/questions/45662253/can-i-run-keras-model-on-gpu
GPU (2/3)

If you’re going to buy a GPU, which one should you choose? The first thing to note is that it must be an NVIDIA GPU. NVIDIA is the only graphics computing company that has invested heavily in deep learning so far, and modern deep-learning frameworks can only run on NVIDIA cards.

**Keras 2.0.9**

```python
from keras.utils.training_utils import multi_gpu_model
model = cnn_model()
model = multi_gpu_model(model, gpus=4)
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(train_x, train_y, validation_data=(test_x, test_y), nb_epoch=20, batch_size=32, verbose=1)
```

https://keras.io/utils/#multi_gpu_model
http://blog.naver.com/PostView.nhn?blogId=ossiriand&logNo=221010346484&parentCategoryNo=&categoryNo=48&viewDate=&isShowPopularPosts=true&from=search
https://www.youtube.com/watch?v=wQ8BIBpya2k&t=766s
https://www.youtube.com/watch?v=j-3vuBynnOE
https://www.youtube.com/watch?v=XNKeayZW4dY
import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model
import numpy as np
num_samples = 1000
height = 224
width = 224
num_classes = 1000
# Instantiate the base model (or "template" model).
# We recommend doing this with under a CPU device scope,
# so that the model's weights are hosted on CPU memory.
# Otherwise they may end up hosted on a GPU, which would
# complicate weight sharing.
with tf.device('/cpu:0'):
    model = Xception(weights=None, input_shape=(height, width, 3), classes=num_classes)
    # Replicates the model on 8 GPUs.
    # This assumes that your machine has 8 available GPUs.
    parallel_model = multi_gpu_model(model, gpus=8)
    parallel_model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
    # Generate dummy data.
    x = np.random.random((num_samples, height, width, 3))
    y = np.random.random((num_samples, num_classes))
    # This 'fit' call will be distributed on 8 GPUs.
    # Since the batch size is 256, each GPU will process 32 samples.
    parallel_model.fit(x, y, epochs=20, batch_size=256)
    # Save model via the template model (which shares the same weights):
    model.save('my_model.h5')

https://keras.io/utils/#multi_gpu_model
파일로부터 데이터를 읽는 방법, 예제:
파일속의 내용을 모두 스트링으로 읽어낸 다음, 숫자로 변환합니다.

```python
aa=[]
bb=[]
colors []
afile=open("fort.11","r")
ii=0
for line in afile:
    if len(line.split()) ==4:
        continue
    if len(line.split()) ==2:
        ii=ii+1
        bb.append(float(line.split()[1]))
        aa.append(int(line.split()[0]))
        colors.append(float(line.split()[0])/230.)
afile.close()
aa=np.array(aa)
bb=np.array(bb)
colors=np.array(colors)
```

```
list_of_lines = ['First line', 'Second line', 'Third line']
append_multiple_lines('target00.txt', list_of_lines)
```

```python
def append_new_line(file_name, text_to_append):
    with open(file_name, "a+") as file_object:
        file_object.seek(0)
        data = file_object.read(100)
        if len(data) > 0:
            file_object.write("n")
        file_object.write(text_to_append)

def append_multiple_lines(file_name, lines_to_append):
    with open(file_name, "a+") as file_object:
        appendEOL = False
        file_object.seek(0)
        data = file_object.read(100)
        if len(data) > 0:
            appendEOL = True
        for line in lines_to_append:
            if appendEOL == True:
                file_object.write("n")
            else:
                appendEOL = True
            file_object.write(line)
```

`http://incredible.egloos.com/7506439`
`http://incredible.egloos.com/7485948`
Practice does not make perfect. Only perfect practice makes perfect.

Vince Lombardi
Optimizers in Keras

• SGD
• SGD with momentum
• Adam
• AdaGrad
• RMSprop
• AdaDelta
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import numpy as np

trX=np.linspace(-1, 1, 101)
trY=2*trX+np.random.randn(*trX.shape)*0.33

model=Sequential()
model.add(Dense(output_dim=1, input_dim=1, init='normal', activation='linear'))
model.compile(optimizer=SGD(lr=0.01), loss='mean_squared_error', metrics=['accuracy'])
model.fit(trX,trY, epochs=100, verbose=1)
# Create your first MLP in Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
numpy.random.seed(7)  # Fix random seed for reproducibility
# Load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter="",)
# split into input (X) and output (Y) variables
X = dataset[:, 0:8]
Y = dataset[:, 8]
# Create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
# Evaluate the model
scores = model.evaluate(X, Y)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
Multi-Class Classification

<table>
<thead>
<tr>
<th>Iris-setosa,</th>
<th>Iris-versicolor,</th>
<th>Iris-virginica</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,</td>
<td>0,</td>
<td>0</td>
</tr>
<tr>
<td>0,</td>
<td>1,</td>
<td>0</td>
</tr>
<tr>
<td>0,</td>
<td>0,</td>
<td>1</td>
</tr>
</tbody>
</table>

```python
# define baseline model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(8, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```python
# load dataset
dataframe = pandas.read_csv("iris.csv", header=None)
dataset = dataframe.values
X = dataset[:, 0:4].astype(float)
Y = dataset[:, 4]
```
```python
>>> from sklearn import preprocessing
>>> le = preprocessing.LabelEncoder()
>>> le.fit([1, 2, 2, 6])
LabelEncoder()
>>> le.classes_
array([1, 2, 6])
>>> le.transform([1, 1, 2, 6])
array([0, 0, 1, 2]...)
>>> le.inverse_transform([0, 0, 1, 2])
array([1, 1, 2, 6])

>>> le = preprocessing.LabelEncoder()
>>> le.fit(['paris', 'paris', 'tokyo', 'amsterdam'])
LabelEncoder()
>>> list(le.classes_)
['amsterdam', 'paris', 'tokyo']
>>> le.transform(['tokyo', 'tokyo', 'paris'])
array([2, 2, 1]...)
>>> list(le.inverse_transform([2, 2, 1]))
['tokyo', 'tokyo', 'paris']
```
# load dataset

dataframe = pandas.read_csv("housing.csv", delim_whitespace=True, header=None)
dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:, 0:13]
Y = dataset[:, 13]

def wider_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=13, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    # compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```python
import time
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import random
import tensorflow as tf

tf.set_random_seed(12)
np.random.seed(34)
random.seed(56)
random.seed(time.time())
start_time=time.clock()
dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter="",)
X = dataset[:,0:8]
Y = dataset[:,8]
X, Y = shuffle(X,Y,random_state=0)
x_train, x_test, y_train, y_test= train_test_split(X,Y, test_size=0.2)
model = Sequential()
model.add(Dense(12, input_dim=8, init='normal', activation='relu'))
for i in range(4):
    model.add(Dense(8, init='normal', activation='relu'))
model.add(Dense(1, init='normal', activation='sigmoid'))
model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, validation_split=0.10, epochs=100, batch_size=5, verbose=2)
scores = model.evaluate(x_test, y_test)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
predictions=model.predict(x_test)
rounded=[round(x[0]) for x in predictions]
print(rounded)
if True:
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    model.save_weights("model.h5")
    print("Saved model to disk")
print((time.clock()-start_time)/60./60.,'hours')
```

Pima-indian/training

Binary classification
from keras.models import model_from_json
from sklearn.utils import shuffle
import time
import numpy as np
import random
import tensorflow as tf
tf.set_random_seed(12)
np.random.seed(34)
random.seed(56)
random.seed(time.time())
start_time=time.clock()

dataset = np.loadtxt("pima-indians-diabetes.csv", delimiter="",\nX = dataset[:,0:8]
Y = dataset[:,8]
X, Y = shuffle(X,Y,random_state=0)

# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
# evaluate loaded model on test data
loaded_model.compile(loss="mean_squared_error", optimizer="adam")
predicted = loaded_model.predict(X)
print((time.clock()-start_time),"sec")

Pima-indian/prediction
Binary classification
df = pd.read_csv('iris.csv', names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species'])
data_set = df.values
X = data_set[:, 0:4].astype(float)
obj_y = data_set[:, 4]

encoder = LabelEncoder()
encoder.fit(obj_y)
Y_encoded = encoder.transform(obj_y)
Y = np_utils.to_categorical(Y_encoded)

X, Y = shuffle(X, Y, random_state=0)
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2)

model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
for i in range(3):
    model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.summary()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train, validation_split=0.10, epochs=100, batch_size=5, verbose=2)
scores = model.evaluate(x_test, y_test)
print('
Test: Loss: {:.4f}'.format(scores[0]))
print('
Test: Accuracy: {:.4f}'.format(scores[1]))
y_pred = model.predict(x_test)
y_pred = np.argmax(y_pred, axis=1)
y_test = np.argmax(y_test, axis=1)
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
if True:
    # serialize model to JSON
    model_json = model.to_json()
    with open('model.json', 'w') as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights('model.h5')
    print('Saved model to disk')
print((time.clock() - start_time) / 60. / 60., 'hours')
df = pd.read_csv('iris.csv', names=["sepal_length", "sepal_width", "petal_length", "petal_width", "species"])
data_set = df.values
X = data_set[:, 0:4].astype(float)
obj_y = data_set[:, 4]

coder = LabelEncoder()
coder.fit(obj_y)
Y_encodered = encoder.transform(obj_y)
Y = np_utils.to_categorical(Y_encodered)

X, Y = shuffle(X,Y,random_state=0)
x_train, x_test, y_train, y_test= train_test_split(X,Y, test_size=0.2)

# load json and create model
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
# evaluate loaded model on test data
loaded_model.compile(loss='mean_squared_error', optimizer='adam')
predicted = loaded_model.predict(X)
print((time.clock()-start_time),'sec')
Categorical variables

There are many ways to encode categorical variables for modeling, although the three most common are as follows:

1. Integer Encoding: Where each unique label is mapped to an integer.
2. One Hot Encoding: Where each label is mapped to a binary vector.
3. Learned Embedding: Where a distributed representation of the categories is learned.
df = pd.read_csv('housing.csv', delim_whitespace=True, header=None)
data_set = df.values
X = data_set[:, 0:13]
Y = data_set[:, 13]
X_train, X_validation, Y_train, Y_validation = train_test_split(X, Y, test_size=0.2)
model = Sequential()
model.add(Dense(30, input_dim=13, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(6, activation='relu'))
model.add(Dense(1))
model.summary()
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(X_train, Y_train, validation_split=0.10, epochs=200, batch_size=10, verbose=2)
Y_prediction = model.predict(X_validation).flatten()
for i in range(10):
    real_price = Y_validation[i]
predicted_price = Y_prediction[i]
    print('Real Price: {:.3f}, Predicted Price: {:.3f}'.format(real_price, predicted_price))
if True:
    # serialize model to JSON
    model_json = model.to_json()
    with open("model.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model.h5")
    print("Saved model to disk")
print((time.clock()-start_time)/60./60.,'hours')
df = pd.read_csv('housing.csv', delim_whitespace=True, header=None)
data_set = df.values
X = data_set[:, 0:13]
Y = data_set[:, 13]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)

if True:
    # load json and create model
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights("model.h5")
    print("Loaded model from disk")

    loaded_model.compile(loss='mean_squared_error', optimizer='adam')
    Y_prediction = loaded_model.predict(X_test).flatten()
    for i in range(10):
        real_price = Y_test[i]
        predicted_price = Y_prediction[i]
        print('Real Price: {:.3f}, Predicted Price: {:.3f}'.format(real_price, predicted_price))

print((time.clock()-start_time)/60./60.,'hours')
Saving/Loading Keras models

- Use `.save` method to save the model
- Use `load_model` function to load saved model
- Saved file contains –
  - Architecture of the model
  - Weights and biases
  - State of the optimizer
- Saving weights
- Loading all the weights and loading weights layer wise

```python
from keras.models import load_model

model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
del model  # deletes the existing model

# returns a compiled model identical to the previous one
model = load_model('my_model.h5')

model.save_weights('my_model_weights.h5')
model.load_weights('my_model_weights.h5', by_name=True)
```
General layers

- Core layers
- Convolution layers
- Pooling layers
- Locally-connected layers
- Recurrent layers
- Merge layers
- Advanced activation layers
- Normalization layers
- Noise layers
- Layer wrappers
- Dropout
- Embedding

https://keras.io/layers/about-keras-layers/
Implementing a neural network in Keras

- **Five major steps**
  - Preparing the input and specify the input dimension (size)
  - Define the model architecture and build the computational graph
  - Specify the optimizer and configure the learning process
  - Specify the Inputs, Outputs of the computational graph (model) and the Loss function
  - Train and test the model on the dataset

**Note:** Gradient calculations are taken care by Auto-Differentiation and parameter updates are done automatically in the backend.
Models

Keras models – Sequential model API
• Sequential model
• Linear stack of layers
• Useful for building simple models
• Simple classification network
• Encoder – Decoder models

Keras models – Functional API
• Functional Model
• Multi – input and Multi – output models
• Complex models which forks into 2 or more branches
• Models with shared (Weights) layers

https://keras.io/models/about-keras-models/
# example of plotting learning curves
from sklearn.datasets import make_classification
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from matplotlib import pyplot

# create the dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)
# determine the number of input features
n_features = X.shape[1]
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(1, activation='sigmoid'))
# compile the model
sgd = SGD(learning_rate=0.001, momentum=0.8)
model.compile(optimizer=sgd, loss='binary_crossentropy')
# fit the model
history = model.fit(X, y, epochs=100, batch_size=32, verbose=0, validation_split=0.3)
# plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Cross Entropy')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()

https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/
from keras.layers import Input, Embedding, LSTM, Dense
from keras.models import Model
import numpy as np
np.random.seed(0)
main_input = Input(shape=(100,), dtype='int32', name='main_input')
x = Embedding(output_dim=512, input_dim=10000, input_length=100)(main_input)
lstm_out = LSTM(32)(x)
auxiliary_output = Dense(1, activation='sigmoid', name='aux_output')(lstm_out)
auxiliary_input = Input(shape=(5,), name='aux_input')
x = keras.layers.concatenate([lstm_out, auxiliary_input])
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
x = Dense(64, activation='relu')(x)
main_output = Dense(1, activation='sigmoid', name='main_output')(x)
model = Model(inputs=[main_input, auxiliary_input], outputs=[main_output, auxiliary_output])

from keras.models import Model
from keras.layers import Input, Dense

a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b)

model = Model(inputs=[a1, a2], outputs=[b1, b2, b3])

https://keras.io/getting-started/functional-api-guide/
Visualization of a DNN in Keras

from keras.models import Sequential
from keras.layers import Dense
from keras.utils.vis_utils import plot_model

model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(2, input_dim=1, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
print(model.summary())

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>dense_1 (Dense)</td>
<td>(None, 2)</td>
<td>4</td>
</tr>
<tr>
<td>dense_2 (Dense)</td>
<td>(None, 1)</td>
<td>3</td>
</tr>
</tbody>
</table>

Total params: 7
Trainable params: 7
Non-trainable params: 0
MNIST

MNIST contains 70,000 images of handwritten digits: 60,000 for training and 10,000 for testing. The images are grayscale, $28 \times 28$ pixels, and centered to reduce preprocessing and get started quicker.

```python
>>> from keras.datasets import mnist
# Load pre-shuffled MNIST data into train and test sets
>>> (X_train, y_train), (X_test, y_test) = mnist.load_data()
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.pkl.gz
>>> print(X_train.shape)
(60000, 28, 28)
>>> print(X_test.shape)
(10000, 28, 28)
>>> print(y_train.shape)
(60000,)
>>> print(y_test.shape)
(10000,)
```
K-fold cross-validation

```python
>>> from sklearn import datasets, linear_model
>>> from sklearn.model_selection import cross_val_score
>>> diabetes = datasets.load_diabetes()
>>> X = diabetes.data[:150]
>>> y = diabetes.target[:150]
>>> lasso = linear_model.Lasso()
>>> print(cross_val_score(lasso, X, y, cv=3))
[0.33150734 0.08022311 0.03531764]
```

```python
for n in [3, 5]:
    kfold = KFold(n_splits=n, shuffle=True, random_state=0)
    scores = cross_val_score(logreg, iris.data, iris.target, cv=kfold)
    print('n_splits={}, cross validation score: {}'.format(n, scores))
```

```python
>>> from sklearn import datasets, linear_model
>>> from sklearn.model_selection import cross_val_score
diabetes = datasets.load_diabetes()
X = diabetes.data[:150]
y = diabetes.target[:150]
lasso = linear_model.Lasso()
print(cross_val_score(lasso, X, y, cv=3))
[0.33150734 0.08022311 0.03531764]
```

```python
np.mean(cross_val_score(elf, X_train, y_train, cv=3, scoring='roc_auc'))
```

K-fold cross-validation

```python
# scikit-learn k-fold cross-validation
from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test]))

train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]

from sklearn.datasets import load_iris
iris=load_iris()
from sklearn.model_selection import cross_val_score
logreg=LogisticRegression()
score=cross_val_score(logreg, iris.data, iris.target, cv=5)
print(score)

from sklearn.model_selection import KFold
kfold=KFold(n_splits=5, shuffle=True, random_state=1)
score=cross_val_score(logreg, iris.data, iris.target, cv=kfold)
```

https://machinelearningmastery.com/k-fold-cross-validation/
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator, FormatStrFormatter, AutoMinorLocator

ii=[] ; aa=[] ; bb=[] ; kount=0
afile=open("g1","r")
for line in afile:
    if len(line.split()) >= 11:
        if line.split()[6] == 'loss:':
            ii.append(kount)
            aa.append(float(line.split()[7]))
            bb.append(float(line.split()[10]))
        kount=kount+1
afile.close()
ii=np.array(ii)
aa=np.array(aa)
bb=np.array(bb)
plt.figure(figsize=(6,6))
ax=plt.axes()
ax.set_xlabel('Epoch',fontsize=20)
ax.set_ylabel('Loss function',fontsize=20)
majorLocator= MultipleLocator(500)
minorLocator= AutoMinorLocator()
majorFormatter= FormatStrFormatter('%d')
minorFormatter= FormatStrFormatter('%d')
ax.xaxis.set_major_locator(majorLocator)
ax.xaxis.set_major_formatter(majorFormatter)
ax.xaxis.set_minor_locator(minorLocator)
majorLocator= MultipleLocator(20)
minorLocator= AutoMinorLocator()
majorFormatter= FormatStrFormatter('%d')
minorFormatter= FormatStrFormatter('%d')
ax.yaxis.set_major_locator(majorLocator)
ax.yaxis.set_major_formatter(majorFormatter)
ax.yaxis.set_minor_locator(minorLocator)
ax.tick_params(which='major', length=2, color='black')
ax.tick_params(which='minor', length=4, color='brown')
ax.set_facecolor("ivory")               # ax.set_facecolor("beige")
plt.grid(True)
plt.plot(ii,aa,'o', linewidth=12, alpha=0.2, c='red')
plt.plot(ii,bb,'--', linewidth=1, alpha=0.9, c='blue')
plt.xlim(0, 3000)
plt.ylim(20, 160)
plt.legend(('train','validation'),loc="upper right", prop={'size': 16})
plt.savefig(str1,dpi=150)
plt.show()
# partition the data into training and testing splits using 75% of the data for training and the remaining 25% for testing

(\text{trainX}, \text{testX}, \text{trainY}, \text{testY}) = \text{train\_test\_split}(\text{data}, \text{labels}, \text{test\_size}=0.25, \text{random\_state}=42)
from sklearn.datasets import load_boston
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostRegressor
from sklearn.metrics import mean_squared_error, make_scorer, r2_score
import matplotlib.pyplot as plt

boston = load_boston()
x, y = boston.data, boston.target
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.15)
abreg = AdaBoostRegressor()
params = {'n_estimators': [50, 100], 'learning_rate': [0.01, 0.05, 0.1, 0.5], 'loss': ['linear', 'square', 'exponential']}
score = make_scorer(mean_squared_error)
gridsearch = GridSearchCV(abreg, params, cv=5, return_train_score=True)
gridsearch.fit(xtrain, ytrain)
print(gridsearch.best_params_)
best_estim = gridsearch.best_estimator_
best_estim.fit(xtrain, ytrain)
ytr_pred = best_estim.predict(xtrain)
mse = mean_squared_error(ytr_pred, ytrain)
r2 = r2_score(ytr_pred, ytrain)
print("MSE: %.2f" % mse)
print("R2: %.2f" % r2)
ypred = best_estim.predict(xtest)
mse = mean_squared_error(ytest, ypred)
r2 = r2_score(ytest, ypred)
print("MSE: %.2f" % mse)
print("R2: %.2f" % r2)
x_ax = range(len(ytest))
plt.scatter(x_ax, ytest, s=5, color="blue", label="original")
plt.plot(x_ax, ypred, lw=0.8, color="red", label="predicted")
plt.legend()
plt.show()
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
from keras.datasets import mnist
from keras.utils import np_utils
batch_size=128 ; nb_classes=10 ; nb_epoch=100
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train=X_train.reshape(6000,784)
X_test=X_test.reshape(1000,784)
X_train=X_train.astype('float32')/255.
X_test=X_test.astype('float32')/255.
Y_Train=np_utils.to_categorical(y_train, nb_classes)
Y_Test=np_utils.to_categorical(y_test, nb_classes)
model=Sequential()
model.add(Dense(output_dim=10, input_dim=(784,), init='normal', activation='softmax'))
model.compile(optimizer=SGD(lr=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
history=model.fit(X_train, Y_Train, nb_epoch=100, batch_size=128, verbose=1)
evaluation=model.evaluate(X_test,Y_Test,verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f',%(evaluation[0], evaluation[1]))
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
from keras.optimizers import RMSprop
from keras.datasets import mnist
from keras.utils import np_utils

batch_size=128; nb_classes=10 ; nb_epoch=100
(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')/255.
X_test=X_test.astype('float32')/255.
Y_train=np_utils.to_categorical(y_train,  nb_classes=nb_classes)
Y_test=np_utils.to_categorical(y_test,  nb_classes=nb_classes)

model=Sequential()
model.add(Dense(output_dim=625, input_dim=784, init='normal'))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(output_dim=625, input_dim=625, init='normal'))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(output_dim=10, input_dim=625, init='normal'))
model.add(Activation('softmax'))
model.compile(optimizer=RMSprop(lr=0.001, rho=0.9), loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

history=model.fit(X_train,Y_train,  nb_epoch=100,  batch_size=128,  verbose=1)
evaluation=model.evaluate(X_test, Y_test,  verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f % (evaluation[0], evaluation[1]))
Gaussian blur can be used to obtain a smooth grayscale digital image of a halftone print.

https://en.wikipedia.org/wiki/Convolution
convolution

Left: Halftone dots. Right: How the human eye would see this sort of arrangement from a sufficient distance.
import numpy as np
np.random.seed(123)  # for reproducibility
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, pooling
from keras.utils import np_utils
from keras.datasets import mnist

(X_train, y_train), (X_test, y_test) = mnist.load_data()
print(X_train.shape)
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)
print(X_train.shape)
X_train = X_train.astype('float32') /255.
X_test = X_test.astype('float32') /255.
Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Conv2D(32, 3, 3, activation='relu', input_shape=(28, 28, 1)))
model.add(Conv2D(32, 3, 3, activation='relu'))
model.add(pooling.MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_train, Y_train, batch_size=32, nb_epoch=10, verbose=1)
score = model.evaluate(X_test, Y_test, verbose=0)
print(model.metrics_names)
print(score)

Use CNNs For:
- Image data
- Classification prediction problems
- Regression prediction problems
- More generally, CNNs work well with data that has a spatial relationship.

Keras built-in VGG-16 net module

```python
from keras.applications.vgg16 import VGG16
import cv2

from keras.applications.inception_v3 import InceptionV3
```

Convolution layers

1D
2D
3D

Pooling layers

- Max pool
  
  \[ \text{keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')} \]

- Average pool
  
  \[ \text{keras.layers.pooling.AveragePooling2D(pool_size=(2, 2), strides=None, padding='valid')} \]

- Up sampling
  
  \[ \text{keras.layers.convolutional.UpSampling2D(size=(2, 2))} \]
# example of a CNN model with an identity or projection residual module
from keras.models import Model
from keras.layers import Input
from keras.layers import Activation
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import add
from keras.utils import plot_model

# function for creating an identity or projection residual module
def residual_module(layer_in, n_filters):
    merge_input = layer_in
    # check if the number of filters needs to be increase, assumes channels last format
    if layer_in.shape[-1] != n_filters:
        merge_input = Conv2D(n_filters, (1,1), padding='same', activation='relu', kernel_initializer='he_normal')(layer_in)
    # conv1
    conv1 = Conv2D(n_filters, (3,3), padding='same', activation='relu', kernel_initializer='he_normal')(layer_in)
    # conv2
    conv2 = Conv2D(n_filters, (3,3), padding='same', activation='linear', kernel_initializer='he_normal')(conv1)
    # add filters, assumes filters/channels last
    layer_out = add([conv2, merge_input])
    # activation function
    layer_out = Activation('relu')(layer_out)
    return layer_out

# define model input
visible = Input(shape=(256, 256, 3))
# add vgg module
layer = residual_module(visible, 64)
# create model
model = Model(inputs=visible, outputs=layer)
# summarize model
model.summary()
# plot model architecture
plot_model(model, show_shapes=True, to_file='residual_module.png')

Batch normalization is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network.

Specifically, we can see that classification accuracy on the train and test datasets leaps above 80% within the first 20 epochs, as opposed to 30-to-40 epochs in the model without batch normalization.

Batch normalization

We have only generated 100 samples, which is small for a neural network, providing the opportunity to overfit the training dataset and have higher error on the test dataset: a good case for using regularization. Further, the samples have noise, giving the model an opportunity to learn aspects of the samples that don’t generalize.

https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/
RNN, LSTM

[10, 20, 30, 40, 50, 60, 70, 80, 90]

<table>
<thead>
<tr>
<th>X,</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>10, 20, 30</td>
<td>40</td>
</tr>
<tr>
<td>20, 30, 40</td>
<td>50</td>
</tr>
<tr>
<td>30, 40, 50</td>
<td>60</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Use RNNs For:
- Text data
- Speech data
- Classification prediction problems
- Regression prediction problems
- Generative models

Don't Use RNNs For:
- Tabular data
- Image data

**LSTM Models for Time Series Forecasting**

Long Short-Term Memory networks (LSTMs) can be applied to time series forecasting. There are many types of LSTM models that can be used for each specific type of time series forecasting problem.

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
RNN, LSTM

- RNNs are used on sequential data – Text, Audio, Genomes etc.
- Recurrent networks are of three types
  - Vanilla RNN
  - LSTM
  - GRU
- They are feedforward networks with internal feedback
- The output at time “t” is dependent on current input and previous values

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
Popular Deep learning Architectures

- Popular Convolution networks
  - Alex net
  - VGG
  - Res-Net
  - DenseNet

- Generative models
  - Autoencoders
  - Generative adversarial networks