How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras

by **Jason Brownlee** on August 9, 2016 in **Deep Learning**

Hyperparameter optimization is a big part of deep learning.

The reason is that neural networks are notoriously difficult to configure and there are a lot of parameters that need to be set. On top of that, individual models can be very slow to train.

In this post you will discover how you can use the grid search capability from the scikit-learn python machine learning library to tune the hyperparameters of Keras deep learning models.

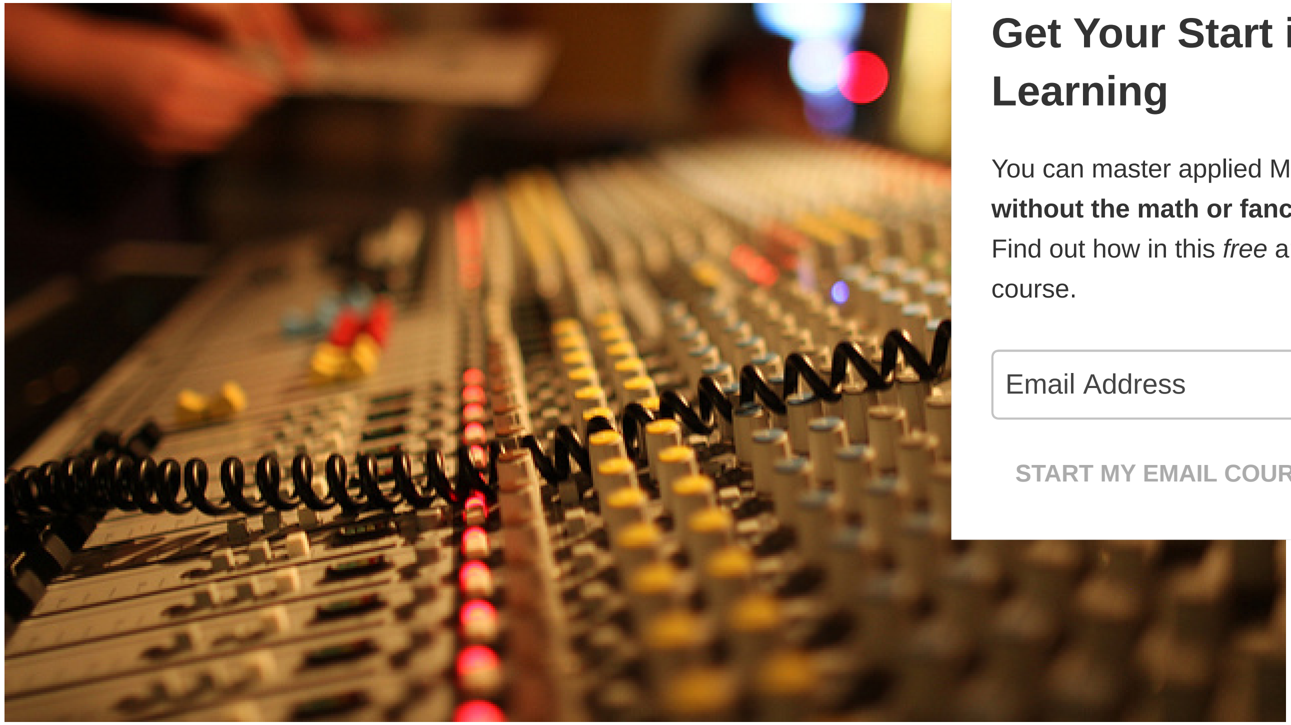
[redacted]

After reading this post you will know:

- How to wrap Keras models for use in scikit-learn and how to use grid search.
- How to grid search common neural network parameters such as learning rate, dropout rate, epochs and number of neurons.
- How to define your own hyperparameter tuning experiments on your own projects.

Let's get started.

- **Update Nov/2016**: Fixed minor issue in displaying grid search results in code examples.
- **Update Oct/2016**: Updated examples for Keras 1.1.0, TensorFlow 0.10.0 and scikit-learn v0.18.
- **Update Mar/2017**: Updated example for Keras 2.0.2, TensorFlow 1.0.1 and Theano 0.9.0.
- **Update Sept/2017**: Updated example to use Keras 2 "epochs" instead of Keras 1 "nb_epochs".



How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras
Photo by 3V Photo, some rights reserved.

Overview

In this post, I want to show you both how you can use the scikit-learn grid search capability and give you a suite of examples that you can copy-and-paste into your own project as a starting point.

Below is a list of the topics we are going to cover:

1. How to use Keras models in scikit-learn.
2. How to use grid search in scikit-learn.
3. How to tune batch size and training epochs.
4. How to tune optimization algorithms.
5. How to tune learning rate and momentum.
6. How to tune network weight initialization.
7. How to tune activation functions.
8. How to tune dropout regularization.
9. How to tune the number of neurons in the hidden layer.

How to Use Keras Models in scikit-learn

Keras models can be used in scikit-learn by wrapping them with the **KerasClassifier** or **KerasRegre**

To use these wrappers you must define a function that creates and returns your Keras sequential mo ent when constructing the **KerasClassifier** class.

For example:

```
def create_model():
    ...
    return model

model = KerasClassifier(build_fn=create_model)
```

The constructor for the **KerasClassifier** class can take default arguments that are passed on to the calls to **model.fit()**, such as the number of epochs and the batch size.

[redacted]

For example:

```
def create_model():
    ...
    return model

model = KerasClassifier(build_fn=create_model, epochs=10)
```

The constructor for the **KerasClassifier** class can also take new arguments that can be passed to your custom **create_model()** function. These new arguments must also be defined in the signature of your **create_model()** function with default parameters.

For example:

```
def create_model(dropout_rate=0.0):
    ...
    return model

model = KerasClassifier(build_fn=create_model, dropout_rate=0.2)
```

You can learn more about the scikit-learn wrapper in Keras API documentation.

How to Use Grid Search in scikit-learn

Grid search is a model hyperparameter optimization technique.

In scikit-learn this technique is provided in the **GridSearchCV** class.

When constructing this class you must provide a dictionary of hyperparameters to evaluate in the **pa** parameter name and an array of values to try.

By default, accuracy is the score that is optimized, but other scores can be specified in the **score** argument of the **GridSearchCV** constructor.

By default, the grid search will only use one thread. By setting the **n_jobs** argument in the **GridSearchCV** constructor to -1, the process will use all cores on your machine. Depending on your Keras backend, this may interfere with the main neural network training process.

The **GridSearchCV** process will then construct and evaluate one model for each combination of parameters. Cross validation is used to evaluate each individual model and the default of 3-fold cross validation is used, although this can be overridden by /

[redacted]

constructor.

Below is an example of defining a simple grid search:

```
param_grid = dict(epochs=[10,20,30])
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X, Y)
```

Once completed, you can access the outcome of the grid search in the result object returned from **grid.fit()**. The **best_score_** member provides access to the best score observed during the optimization procedure and the **best_params_** describes the combination of parameters that achieved the best results.

You can learn more about the GridSearchCV class in the scikit-learn API documentation.

Problem Description

Now that we know how to use Keras models with scikit-learn and how to use grid search in scikit-lea

All examples will be demonstrated on a small standard machine learning dataset called the Pima Inc is a small dataset with all numerical attributes that is easy to work with.

1. Download the dataset and place it in your currently working directly with the name **pima-indian**

As we proceed through the examples in this post, we will aggregate the best parameters. This is not can interact, but it is good for demonstration purposes.

Note on Parallelizing Grid Search

All examples are configured to use parallelism (**n_jobs=-1**).

If you get an error like the one below:

```
INFO (theano.gof.compilelock): Waiting for existing lock by process '55614' (I am process '55613')
INFO (theano.gof.compilelock): To manually release the lock, delete ...
```

Kill the process and change the code to not perform the grid search in parallel, set **n_jobs=1**.

[redacted]

How to Tune Batch Size and Number of Epochs

In this first simple example, we look at tuning the batch size and number of epochs used when fitting

The batch size in iterative gradient descent is the number of patterns shown to the network before th n the training of the network, defining how many patterns to read at a time and keep in memory.

The number of epochs is the number of times that the entire training dataset is shown to the network ne batch size, such as LSTM recurrent neural networks and Convolutional Neural Networks.

Here we will evaluate a suite of different mini batch sizes from 10 to 100 in steps of 20.

The full code listing is provided below.

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, verbose=0)
# define the grid search parameters
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

Running this example produces the following output.

```
Best: 0.686198 using {'epochs': 100, 'batch_size': 20}
0.348958 (0.024774) with: {'epochs': 10, 'batch_size': 10}
0.348958 (0.024774) with: {'epochs': 50, 'batch_size': 10}
0.466146 (0.149269) with: {'epochs': 100, 'batch_size': 10}
0.647135 (0.021236) with: {'epochs': 10, 'batch_size': 20}
0.660156 (0.014616) with: {'epochs': 50, 'batch_size': 20}
0.686198 (0.024774) with: {'epochs': 100, 'batch_size': 20}
0.489583 (0.075566) with: {'epochs': 10, 'batch_size': 40}
0.652344 (0.019918) with: {'epochs': 50, 'batch_size': 40}
0.654948 (0.027866) with: {'epochs': 100, 'batch_size': 40}
0.518229 (0.032264) with: {'epochs': 10, 'batch_size': 60}
0.605469 (0.052213) with: {'epochs': 50, 'batch_size': 60}
0.665365 (0.004872) with: {'epochs': 100, 'batch_size': 60}
0.537760 (0.143537) with: {'epochs': 10, 'batch_size': 80}
0.591146 (0.094954) with: {'epochs': 50, 'batch_size': 80}
0.658854 (0.054904) with: {'epochs': 100, 'batch_size': 80}
```

[redacted]

```
0.402344 (0.107735) with: {'epochs': 10, 'batch_size': 100}
0.652344 (0.033299) with: {'epochs': 50, 'batch_size': 100}
0.542969 (0.157934) with: {'epochs': 100, 'batch_size': 100}
```

We can see that the batch size of 20 and 100 epochs achieved the best result of about 68% accuracy.

How to Tune the Training Optimization Algorithm

Keras offers a suite of different state-of-the-art optimization algorithms.

In this example, we tune the optimization algorithm used to train the network, each with default parameters.

This is an odd example, because often you will choose one approach a priori and instead focus on tuning its parameters on your problem (e.g. see the next example).

Here we will evaluate the suite of optimization algorithms supported by the Keras API.

The full code listing is provided below.

```
# Use scikit-learn to grid search the batch size and epochs
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model(optimizer='adam'):
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
optimizer = ['SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax', 'Nadam']
param_grid = dict(optimizer=optimizer)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

Running this example produces the following output.

```
Best: 0.704427 using {'optimizer': 'Adam'}
0.348958 (0.024774) with: {'optimizer': 'SGD'}
0.348958 (0.024774) with: {'optimizer': 'RMSprop'}
0.471354 (0.156586) with: {'optimizer': 'Adagrad'}
0.669271 (0.029635) with: {'optimizer': 'Adadelta'}
0.704427 (0.031466) with: {'optimizer': 'Adam'}
0.682292 (0.016367) with: {'optimizer': 'Adamax'}
0.703125 (0.003189) with: {'optimizer': 'Nadam'}
```

The results suggest that the ADAM optimization algorithm is the best with a score of about 70% accu

How to Tune Learning Rate and Momentum

It is common to pre-select an optimization algorithm to train your network and tune its parameters.

By far the most common optimization algorithm is plain old Stochastic Gradient Descent (SGD) because it is so well understood. In this example, we will look at optimizing the SGD learning rate and momentum parameters.

Learning rate controls how much to update the weight at the end of each batch and the momentum controls how much to let the previous update influence the current weight update.

We will try a suite of small standard learning rates and a momentum values from 0.2 to 0.8 in steps of 0.2, as well as 0.9 (because it can be a popular value in practice).

Generally, it is a good idea to also include the number of epochs in an optimization like this as there [redacted] per batch (learning rate), the number of updates per epoch (batch size) and the number of epochs.

The full code listing is provided below.

```
# Use scikit-learn to grid search the learning rate and momentum
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.optimizers import SGD
# Function to create model, required for KerasClassifier
def create_model(learn_rate=0.01, momentum=0):
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    optimizer = SGD(lr=learn_rate, momentum=momentum)
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
learn_rate = [0.001, 0.01, 0.1, 0.2, 0.3]
momentum = [0.0, 0.2, 0.4, 0.6, 0.8, 0.9]
param_grid = dict(learn_rate=learn_rate, momentum=momentum)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

Running this example produces the following output.

[redacted]

```
Best: 0.680990 using {'learn_rate': 0.01, 'momentum': 0.0}
0.348958 (0.024774) with: {'learn_rate': 0.001, 'momentum': 0.0}
0.348958 (0.024774) with: {'learn_rate': 0.001, 'momentum': 0.2}
0.467448 (0.151098) with: {'learn_rate': 0.001, 'momentum': 0.4}
0.662760 (0.012075) with: {'learn_rate': 0.001, 'momentum': 0.6}
0.669271 (0.030647) with: {'learn_rate': 0.001, 'momentum': 0.8}
0.666667 (0.035564) with: {'learn_rate': 0.001, 'momentum': 0.9}
0.680990 (0.024360) with: {'learn_rate': 0.01, 'momentum': 0.0}
0.677083 (0.026557) with: {'learn_rate': 0.01, 'momentum': 0.2}
0.427083 (0.134575) with: {'learn_rate': 0.01, 'momentum': 0.4}
0.427083 (0.134575) with: {'learn_rate': 0.01, 'momentum': 0.6}
0.544271 (0.146518) with: {'learn_rate': 0.01, 'momentum': 0.8}
0.651042 (0.024774) with: {'learn_rate': 0.01, 'momentum': 0.9}
0.651042 (0.024774) with: {'learn_rate': 0.1, 'momentum': 0.0}
0.651042 (0.024774) with: {'learn_rate': 0.1, 'momentum': 0.2}
0.572917 (0.134575) with: {'learn_rate': 0.1, 'momentum': 0.4}
0.572917 (0.134575) with: {'learn_rate': 0.1, 'momentum': 0.6}
0.651042 (0.024774) with: {'learn_rate': 0.1, 'momentum': 0.8}
0.651042 (0.024774) with: {'learn_rate': 0.1, 'momentum': 0.9}
0.533854 (0.149269) with: {'learn_rate': 0.2, 'momentum': 0.0}
0.427083 (0.134575) with: {'learn_rate': 0.2, 'momentum': 0.2}
0.427083 (0.134575) with: {'learn_rate': 0.2, 'momentum': 0.4}
0.651042 (0.024774) with: {'learn_rate': 0.2, 'momentum': 0.6}
0.651042 (0.024774) with: {'learn_rate': 0.2, 'momentum': 0.8}
0.651042 (0.024774) with: {'learn_rate': 0.2, 'momentum': 0.9}
0.455729 (0.146518) with: {'learn_rate': 0.3, 'momentum': 0.0}
0.455729 (0.146518) with: {'learn_rate': 0.3, 'momentum': 0.2}
0.455729 (0.146518) with: {'learn_rate': 0.3, 'momentum': 0.4}
0.348958 (0.024774) with: {'learn_rate': 0.3, 'momentum': 0.6}
0.348958 (0.024774) with: {'learn_rate': 0.3, 'momentum': 0.8}
0.348958 (0.024774) with: {'learn_rate': 0.3, 'momentum': 0.9}
```

We can see that relatively SGD is not very good on this problem, nevertheless best results were ach um of 0.0 with an accuracy of about 68%.

How to Tune Network Weight Initialization

Neural network weight initialization used to be simple: use small random values.

Now there is a suite of different techniques to choose from. Keras provides a laundry list.

In this example, we will look at tuning the selection of network weight initialization by evaluating all o[redacted]

We will use the same weight initialization method on each layer. Ideally, it may be better to use different weight initialization schemes according to the activation function used on each layer. In the example below we use rectifier for the hidden layer. We use sigmoid for the output layer because the predictions are binary.

The full code listing is provided below.

```
# Use scikit-learn to grid search the weight initialization
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model(init_mode='uniform'):
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer=init_mode, activation='relu'))
    model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
init_mode = ['uniform', 'lecun_uniform', 'normal', 'zero', 'glorot_normal', 'glorot_uniform', 'he_normal', 'he_uniform']
param_grid = dict(init_mode=init_mode)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

[redacted]

Running this example produces the following output.

```
Best: 0.720052 using {'init_mode': 'uniform'}
0.720052 (0.024360) with: {'init_mode': 'uniform'}
0.348958 (0.024774) with: {'init_mode': 'lecun_uniform'}
0.712240 (0.012075) with: {'init_mode': 'normal'}
0.651042 (0.024774) with: {'init_mode': 'zero'}
0.700521 (0.010253) with: {'init_mode': 'glorot_normal'}
0.674479 (0.011201) with: {'init_mode': 'glorot_uniform'}
0.661458 (0.028940) with: {'init_mode': 'he_normal'}
0.678385 (0.004872) with: {'init_mode': 'he_uniform'}
```

We can see that the best results were achieved with a uniform weight initialization scheme achieving a performance of about 72%.

How to Tune the Neuron Activation Function

The activation function controls the non-linearity of individual neurons and when to fire.

Generally, the rectifier activation function is the most popular, but it used to be the sigmoid and the ta e suitable for different problems.

In this example, we will evaluate the suite of different activation functions available in Keras. We will e require a sigmoid activation function in the output for the binary classification problem.

Generally, it is a good idea to prepare data to the range of the different transfer functions, which we v

The full code listing is provided below.

```
# Use scikit-learn to grid search the activation function
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model(activation='relu'):
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer='uniform', activation=activatio
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
activation = ['softmax', 'softplus', 'softsign', 'relu', 'tanh', 'sigmoid', 'hard_sigmoid', 'linear']
param_grid = dict(activation=activation)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

[redacted]

Running this example produces the following output.

```
Best: 0.722656 using {'activation': 'linear'}
0.649740 (0.009744) with: {'activation': 'softmax'}
0.720052 (0.032106) with: {'activation': 'softplus'}
0.688802 (0.019225) with: {'activation': 'softsign'}
0.720052 (0.018136) with: {'activation': 'relu'}
0.691406 (0.019401) with: {'activation': 'tanh'}
0.680990 (0.009207) with: {'activation': 'sigmoid'}
0.691406 (0.014616) with: {'activation': 'hard_sigmoid'}
0.722656 (0.003189) with: {'activation': 'linear'}
```

Surprisingly (to me at least), the ‘linear’ activation function achieved the best results with an accuracy of about 72%.

How to Tune Dropout Regularization

In this example, we will look at tuning the dropout rate for regularization in an effort to limit overfitting

To get good results, dropout is best combined with a weight constraint such as the max norm constra[redacted]

For more on using dropout in deep learning models with Keras see the post:

- Dropout Regularization in Deep Learning Models With Keras

This involves fitting both the dropout percentage and the weight constraint. We will try dropout percentages between 0.0 and 0.9 (1.0 does not make sense) and maxnorm weight constraint values between 0 and 5.

The full code listing is provided below.

```
# Use scikit-learn to grid search the dropout rate
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
# Function to create model, required for KerasClassifier
def create_model(dropout_rate=0.0, weight_constraint=0):
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, kernel_initializer='uniform', activation='linear',                    )
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
weight_constraint = [1, 2, 3, 4, 5]
dropout_rate = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
param_grid = dict(dropout_rate=dropout_rate, weight_constraint=weight_constraint)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
```

[redacted]

```
grid_result = grid.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))
```

Running this example produces the following output.

```
Best: 0.723958 using {'dropout_rate': 0.2, 'weight_constraint': 4}
0.696615 (0.031948) with: {'dropout_rate': 0.0, 'weight_constraint': 1}
0.696615 (0.031948) with: {'dropout_rate': 0.0, 'weight_constraint': 2}
0.691406 (0.026107) with: {'dropout_rate': 0.0, 'weight_constraint': 3}
0.708333 (0.009744) with: {'dropout_rate': 0.0, 'weight_constraint': 4}
0.708333 (0.009744) with: {'dropout_rate': 0.0, 'weight_constraint': 5}
0.710937 (0.008438) with: {'dropout_rate': 0.1, 'weight_constraint': 1}
0.709635 (0.007366) with: {'dropout_rate': 0.1, 'weight_constraint': 2}
0.709635 (0.007366) with: {'dropout_rate': 0.1, 'weight_constraint': 3}
0.695312 (0.012758) with: {'dropout_rate': 0.1, 'weight_constraint': 4}
0.695312 (0.012758) with: {'dropout_rate': 0.1, 'weight_constraint': 5}
0.701823 (0.017566) with: {'dropout_rate': 0.2, 'weight_constraint': 1}
0.710938 (0.009568) with: {'dropout_rate': 0.2, 'weight_constraint': 2}
0.710938 (0.009568) with: {'dropout_rate': 0.2, 'weight_constraint': 3}
0.723958 (0.027126) with: {'dropout_rate': 0.2, 'weight_constraint': 4}
0.718750 (0.030425) with: {'dropout_rate': 0.2, 'weight_constraint': 5}
0.721354 (0.032734) with: {'dropout_rate': 0.3, 'weight_constraint': 1}
0.707031 (0.036782) with: {'dropout_rate': 0.3, 'weight_constraint': 2}
0.707031 (0.036782) with: {'dropout_rate': 0.3, 'weight_constraint': 3}
0.694010 (0.019225) with: {'dropout_rate': 0.3, 'weight_constraint': 4}
0.709635 (0.006639) with: {'dropout_rate': 0.3, 'weight_constraint': 5}
0.704427 (0.008027) with: {'dropout_rate': 0.4, 'weight_constraint': 1}
0.717448 (0.031304) with: {'dropout_rate': 0.4, 'weight_constraint': 2}
0.718750 (0.030425) with: {'dropout_rate': 0.4, 'weight_constraint': 3}
0.718750 (0.030425) with: {'dropout_rate': 0.4, 'weight_constraint': 4}
0.722656 (0.029232) with: {'dropout_rate': 0.4, 'weight_constraint': 5}
0.720052 (0.028940) with: {'dropout_rate': 0.5, 'weight_constraint': 1}
0.703125 (0.009568) with: {'dropout_rate': 0.5, 'weight_constraint': 2}
0.716146 (0.029635) with: {'dropout_rate': 0.5, 'weight_constraint': 3}
0.709635 (0.008027) with: {'dropout_rate': 0.5, 'weight_constraint': 4}
0.703125 (0.011500) with: {'dropout_rate': 0.5, 'weight_constraint': 5}
0.707031 (0.017758) with: {'dropout_rate': 0.6, 'weight_constraint': 1}
0.701823 (0.018688) with: {'dropout_rate': 0.6, 'weight_constraint': 2}
0.701823 (0.018688) with: {'dropout_rate': 0.6, 'weight_constraint': 3}
0.690104 (0.027498) with: {'dropout_rate': 0.6, 'weight_constraint': 4}
0.695313 (0.022326) with: {'dropout_rate': 0.6, 'weight_constraint': 5}
```

[redacted]

```
0.697917 (0.014382) with: {'dropout_rate': 0.7, 'weight_constraint': 1}
0.697917 (0.014382) with: {'dropout_rate': 0.7, 'weight_constraint': 2}
0.687500 (0.008438) with: {'dropout_rate': 0.7, 'weight_constraint': 3}
0.704427 (0.011201) with: {'dropout_rate': 0.7, 'weight_constraint': 4}
0.696615 (0.016367) with: {'dropout_rate': 0.7, 'weight_constraint': 5}
0.680990 (0.025780) with: {'dropout_rate': 0.8, 'weight_constraint': 1}
0.699219 (0.019401) with: {'dropout_rate': 0.8, 'weight_constraint': 2}
0.701823 (0.015733) with: {'dropout_rate': 0.8, 'weight_constraint': 3}
0.684896 (0.023510) with: {'dropout_rate': 0.8, 'weight_constraint': 4}
0.696615 (0.017566) with: {'dropout_rate': 0.8, 'weight_constraint': 5}
0.653646 (0.034104) with: {'dropout_rate': 0.9, 'weight_constraint': 1}
0.677083 (0.012075) with: {'dropout_rate': 0.9, 'weight_constraint': 2}
0.679688 (0.013902) with: {'dropout_rate': 0.9, 'weight_constraint': 3}
0.669271 (0.017566) with: {'dropout_rate': 0.9, 'weight_constraint': 4}
0.669271 (0.012075) with: {'dropout_rate': 0.9, 'weight_constraint': 5}
```

We can see that the dropout rate of 0.2% and the maxnorm weight constraint of 4 resulted in the best accuracy of about 72%.

How to Tune the Number of Neurons in the Hidden Layer

The number of neurons in a layer is an important parameter to tune. Generally the number of neuror of the network, at least at that point in the topology.

Also, generally, a large enough single layer network can approximate any other neural network, at le

In this example, we will look at tuning the number of neurons in a single hidden layer. We will try valu

A larger network requires more training and at least the batch size and number of epochs should ide

The full code listing is provided below.

```
# Use scikit-learn to grid search the number of neurons
import numpy
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
# Function to create model, required for KerasClassifier
def create_model(neurons=1):
    # create model
```

[redacted]

```
    model = Sequential()
    model.add(Dense(neurons, input_dim=8, kernel_initializer='uniform', activation='linear', kernel_constraint=maxnorm(4)))
    model.add(Dropout(0.2))
    model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load 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 = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, verbose=0)
# define the grid search parameters
neurons = [1, 5, 10, 15, 20, 25, 30]
param_grid = dict(neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

Running this example produces the following output.

```
Best: 0.714844 using {'neurons': 5}
0.700521 (0.011201) with: {'neurons': 1}
0.714844 (0.011049) with: {'neurons': 5}
0.712240 (0.017566) with: {'neurons': 10}
0.705729 (0.003683) with: {'neurons': 15}
0.696615 (0.020752) with: {'neurons': 20}
0.713542 (0.025976) with: {'neurons': 25}
0.705729 (0.008027) with: {'neurons': 30}
```

We can see that the best results were achieved with a network with 5 neurons in the hidden layer with an accuracy of about 71%.

Tips for Hyperparameter Optimization

This section lists some handy tips to consider when tuning hyperparameters of your neural network. [redacted]

- **k-fold Cross Validation**. You can see that the results from the examples in this post show some variance. A default cross-validation of 3 was used, but perhaps k=5 or k=10 would be more stable. Carefully choose your cross validation configuration to ensure your results are stable.
- **Review the Whole Grid**. Do not just focus on the best result, review the whole grid of results and look for trends to support configuration decisions.
- **Parallelize**. Use all your cores if you can, neural networks are slow to train and we often want to try a lot of different parameters. Consider spinning up a lot of AWS instances.
- **Use a Sample of Your Dataset**. Because networks are slow to train, try training them on a smaller sample of your training dataset, just to get an idea of general directions of parameters rather than optimal configurations.
- **Start with Coarse Grids**. Start with coarse-grained grids and zoom into finer grained grids once you can narrow the scope.
- **Do not Transfer Results**. Results are generally problem specific. Try to avoid favorite configurations on each new problem that you see. It is unlikely that optimal results you discover on one problem will transfer to your next project. Instead look for broader trends like number of layers or relationships between parameters.
- **Reproducibility is a Problem**. Although we set the seed for the random number generator in N ere is more to reproducibility when grid searching wrapped Keras models than is presented in this p

Summary

In this post, you discovered how you can tune the hyperparameters of your deep learning networks i

Specifically, you learned:

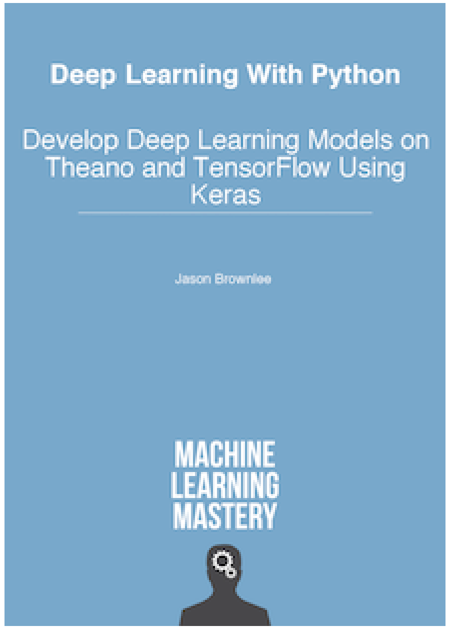
- How to wrap Keras models for use in scikit-learn and how to use grid search.
- How to grid search a suite of different standard neural network parameters for Keras models.
- How to design your own hyperparameter optimization experiments.

Do you have any experience tuning hyperparameters of large neural networks? Please share your stories below.

Do you have any questions about hyperparameter optimization of neural networks or about this post? Ask your questions in the comments and I will do my best to answer.

[redacted]

Frustrated With Your Progress In Deep Learning?


Deep Learning With Python
Develop Deep Learning Models on Theano and TensorFlow Using Keras
Jason Brownlee
MACHINE LEARNING MASTERY

What If You Could Develop A Network in Minutes

…with just a few lines of Python

Discover how in my new Ebook: Deep Learning With Python

It covers **self-study tutorials** and **end-to-end projects** on topics like:
Multilayer Perceptrons, *Convolutional Nets* and *Recurrent Neural Nets*, and more…

Finally Bring Deep Learning To Your Own Projects

Skip the Academics. Jus

Click to learn mo

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning.
Learn more.

View all posts by Jason Brownlee →

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

‹ How To Work Through a Binary Classification Project in Weka Step-By-Step How to Work Through [redacted] ›

237 Responses to *How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras*

Yanbo August 9, 2016 at 9:10 am # REPLY

As always excellent post,. I've been doing some hyper-parameter optimization by hand, but I'll definitely give Grid Search a try.

Is it possible to set up a different threshold for sigmoid output in Keras? Rather then using 0.5 I was thinking of trying 0.7 or 0.8

Jason Brownlee August 15, 2016 at 11:10 am #

Thanks Yanbo.

I don't think so, but you could implement your own activation function and do anything you wish.

Shudhan September 5, 2016 at 6:20 pm #

My question is related to this thread. How to get the probablities as the output? I dont w
that no activation function is needed in the output layer. Similiar implementation will get me the probabilities ?? or the output will exceed 0 and 1??

Jason Brownlee September 6, 2016 at 9:41 am # REPLY

Hi Shudhan, you can use a sigmoid activation and treat the outputs like probabilities (they will be in the range of 0-1).

[redacted]

Swapna November 2, 2017 at 11:51 pm #

REPLY

excellent post

Jason Brownlee November 3, 2017 at 5:18 am #

REPLY

Thanks Swapna.

eclipsedu August 18, 2016 at 5:55 pm #

Sound awesome!Will this grid search method use the full cpu(which can be 8/16 cores) ?

Jason Brownlee August 19, 2016 at 5:23 am #

It can if you set n_jobs=-1

Reza August 18, 2016 at 6:00 pm #

Hi,
Great post,
Can I use this tips on CNNs in keras as well?
Thanks!

Jason Brownlee August 19, 2016 at 5:24 am #

They can be a start, but remember it is a good idea to use a repeating structure in a large [redacted]
and pool size.

Prashant August 22, 2016 at 4:55 pm #

REPLY

Hi Jason, First of all great post! I applied this by dividing the data into train and test and used train dataset for grid fit. Plan was to capture best parameters in train and apply them on test to see accuracy. But it seems grid.fit and model.fit applied with same parameters on same dataset (in this case train) give different accuracy results. Any idea why this happens. I can share the code if it helps.

Jason Brownlee August 23, 2016 at 6:00 am #

You will see small variation in the performance of a neural net with the same parameters fro
of the technique and how very hard it is to fix the random number seed successfully in python/numpy

You will also see small variation due to the data used to train the method.

Generally, you could use all of your data to grid search to try to reduce the second type of variation (
significance tests to compare populations of results to see if differences are significant to sort out the

I hope that helps.

vinay August 22, 2016 at 9:05 pm #

hi, I think this will best tutorial i ever found on web….Thanks for sharing….is it possible to use these tips on LSTM, Bilstm cnnlstm

Jason Brownlee August 23, 2016 at 5:57 am #

REPLY

Thanks Vinay, I'm glad it's useful.

Absolutely, you could use these tactics on other algorithm types.

[redacted]

shudhan September 2, 2016 at 3:26 pm #

REPLY

Best place to learn the tuning.. my question – is it good to follow the order you mentioned to tune the parameters? I know the most significant parameters should be tuned first

Jason Brownlee September 3, 2016 at 6:56 am #

REPLY

Thanks. The order is a good start. It is best to focus on areas where you think you will get tl structure of the network (layers and neurons).

Satheesh September 27, 2016 at 12:24 am #

when I am using the categorical_entropy loss function and running the grid search with n_jobs r class”, but the same thing is working fine with binary_entropyloss. Can you tell me if I am making any mi

```
def create_model(optimizer=’adam’):
# create model
model.add(Dense(30, input_dim=59, init=’normal’, activation=’relu’))
model.add(Dense(15, init=’normal’, activation=’sigmoid’))
model.add(Dense(3, init=’normal’, activation=’sigmoid’))
# Compile model
model.compile(loss=’categorical_crossentropy’, optimizer=optimizer, metrics=[‘accuracy’])
return model

# Create Keras Classifier
print “———————— Running Grid Search on Keras Classifier for epochs and batch ——————–”
clf = model = KerasClassifier(build_fn = create_model, verbose=0)
param_grid = {“batch_size”:range(10, 30, 10), “nb_epoch”:range(50, 150, 50)}
```

```
optimizer = [‘SGD’, ‘RMSprop’, ‘Adagrad’, ‘Adadelta’, ‘Adam’, ‘Adamax’, ‘Nadam’]
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=4)
grid_result = grid.fit(x_train, y_train)
print(“Best: %f using %s” % (grid_result.best_score_, grid_result.best_params_))
```

Jason Brownlee September 27, 2016 at 7:44 am # REPLY

Strange Satheesh, I have not seen that before.

Let me know if you figure it out.

Kai September 18, 2017 at 10:01 pm #

I came cross and solved the problem several days ago. Please use “epochs” instead of “cannot pickle object class” means the neuron network cannot be built because of some errors. (

Jason Brownlee September 19, 2017 at 7:40 am #

Glad to hear it.

I updated the example to use “epochs” to work with Keras 2.

L Fenu November 9, 2016 at 7:47 pm # REPLY

excellent post, thanks. It’s been very helpful to get me started on hyperparameterisation.

One thing I haven’t been able to do yet is to grid search over parameters which are not proper to the NN but to the trainign set. For example, I can fine-tune the input_dim parameter by creating a function generator which takes care of creating the function that v

[redacted]

# fp_subset is a subset of columns of my whole training set.

create_basic_ANN_model = kt.ANN_model_gen(# defined elsewhere
input_dim=len(fp_subset), output_dim=1, layers_num=2, layers_sizes=[len(fp_subset)/5, len(fp_subset)/10,],
loss='mean_squared_error', optimizer='adadelta', metrics=['mean_squared_error', 'mean_absolute_error']
)

model = KerasRegressor(build_fn=create_basic_ANN_model, verbose=1)
# define the grid search parameters
batch_size = [10, 100]
epochs = [5, 10]

param_grid = dict(batch_size=batch_size, nb_epoch=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=7)

grid_results = grid.fit(trX, trY)

this works but only as a for loop over the different fp_subset, which I must define manually.
I could easily pick the best out of every run but it wuld be great if I could fold them all inside a big grid de

However, until now haven't been able to figure out a way to get that in my head.
If the wrapper function is useful to anyone, I can post a generalised version here.

Jason Brownlee November 10, 2016 at 7:42 am #

Good question.

You might just need to us a loop around the whole lot for different projections/views of your training data.

L Fenu November 11, 2016 at 1:05 am #

REPLY

Thanks. I ended up coding my own for loop, saving the results of each grid in a dict, sorting the hash by the perofrmance metrics, and picking the best model.

Now, the next question is: How do I save the model's architecture and weights to a .json .hdf5 fil[redacted] do I extract the best model out of the gridsearch results?

Jason Brownlee November 11, 2016 at 10:04 am #

REPLY ↩

Well done.

No need. Once you know the parameters, you can use them to train a new standalone model on all of your training data and start making predictions.

Fenu Luca November 15, 2016 at 3:23 am #

I may have found a way. How about this?

```
best_model = grid_result.best_estimator_.model
best_model_file_path = 'your_pick_here'
model2json = best_model.to_json()
with open( best_model_file_path+'.json', 'w') as json_file:
json_file.write(model2json)
best_model.save_weights(best_model_file_path+'.h5′)
```

volador November 14, 2016 at 6:21 pm #

Hi Jason, I think this is very best deep learning tutorial on the web. Thanks for your work. I have a question is :how to use the heuristic algorithm to optimize Hyperparameters for Deep Learning Models in Python With Keras, these algorithms like: Genetic algorithm, Particle swarm optimization, and Cuckoo algorithm etc. If the idea could be experimented, could you give an example

Jason Brownlee November 15, 2016 at 7:50 am #

[redacted]

Thanks for your support volador.

You could search the hyperparameter space using a stochastic optimization algorithm like a genetic algorithm and use the mean performance as the cost function orf fitness function. I don't have a worked example, but it would be relatively easy to setup.

Jan de Lange November 15, 2016 at 6:50 am #

REPLY ↰

Hi Jason, very helpful intro into gridsearch for Keras. I have used your guidance in my code, but rather than using the default 'accuracy' to be optimized, my model requires a specific evaluation function to be optimized. You hint at this possibility in the introduction, but there is no example of it. I have followed the SciKit-learn documentation, but I fail to come up with the correct syntax.

I have posted my question at StackOverflow, but since it is quite specific, it requires understanding of Sc

Perhaps you can have a look? I think it would nicely extend your tutorial.

http://stackoverflow.com/questions/40572743/scikit-learn-grid-search-own-scoring-object-syntax

Thanks, Jan

Jason Brownlee November 15, 2016 at 8:02 am #

Sorry Jan, I have not used a custom scoring function before.

Here are a list of built-in scoring functions:
http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

Here is help on defining your own scoring function:
http://scikit-learn.org/stable/modules/model_evaluation.html#defining-your-scoring-strategy-from-metric-functions

Let me know how you go.

Jan de Lange November 16, 2016 at 7:31 am #

Yup, same sources as I referenced in my post at Stackoverflow.

[redacted]

Jason Brownlee November 16, 2016 at 9:35 am #

REPLY

Excellent. Good luck Jan.

Anthony Ohazulike December 6, 2016 at 12:46 am #

REPLY

Good tutorial again Jason…keep on the good job!

Jason Brownlee December 6, 2016 at 8:26 am #

Thanks Anthony.

nrcjea001 December 13, 2016 at 10:48 pm #

Hi Jason

First off, thank you for the tutorial. It's very helpful.

I was also hoping you would assist on how to adapt the keras grid search to stateful lstms as discussed in

http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

I've coded the following:

```
# create model
model = KerasRegressor(build_fn=create_model, nb_epoch=1, batch_size=bats,
verbose=2, shuffle=False)
```

[redacted]

```
# define the grid search parameters
h1n = [5, 10] # number of hidden neurons
param_grid = dict(h1n=h1n)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=5)

for i in range(100):
grid.fit(trainX, trainY)
grid.reset_states()
```

Is grid.reset_states() corrrect? or would you suggest creating function callback for reset states.

Thanks,

Jason Brownlee December 14, 2016 at 8:27 am #

Great question.

With stateful LSTMs we must control the resetting of states after each epoch. The sklearn framewor that way to me off the cuff.

I think you may have to grid search stateful LSTM params manually with a ton of for loops. Sorry.

If you discover something different, let me know. i.e. there may be a way in the back door to the skle own custom epoch handing.

REPLY

Thomas Maier December 21, 2016 at 2:53 am #

Hi Jason

Thanks a lot for this and all the other great tutorials!

I tried to combine this gridsearch/keras approach with a pipeline. It works if I tune nb_epoch or batch_size, but I get an error if I try to tune the optimizer or something else in the keras building function (I did not forget to include the variable as an argument):

[redacted]

```
def keras_model(optimizer = ‘adam’):
model = Sequential()
model.add(Dense(80, input_dim=79, init= ‘normal’))
model.add(Activation(‘relu’))
model.add(Dense(1, init=’normal’))
model.add(Activation(‘linear’))
model.compile(optimizer=optimizer, loss=’mse’)
return model

kRegressor = KerasRegressor(build_fn=keras_model, nb_epoch=500, batch_size=10, verbose=0)

estimators = []
estimators.append((‘imputer’, preprocessing.Imputer(strategy=’mean’)))
estimators.append((‘scaler’, preprocessing.StandardScaler()))
estimators.append((‘kerasR’, kRegressor))
pipeline = Pipeline(estimators)

param_grid = dict(kerasR__optimizer = [‘adam’,’rmsprop’])

grid = GridSearchCV(pipeline, param_grid, cv=5, scoring=’neg_mean_squared_error’)
```

Do you know this problem?

Thanks, Thomas

Jason Brownlee December 21, 2016 at 8:44 am #

Thanks Thomas. I’ve not seen this issue.

I think we’re starting to push the poor Keras sklearn wrapper to the limit.

Maybe the next step is to build out a few functions to do manual grid searching across network configs.

Jimi December 21, 2016 at 3:26 pm #

[redacted]

Great resource!

Any thoughts on how to get the “history” objects out of grid search? It could be beneficial to plot the loss and accuracy to see when a model starts to flatten out.

Jason Brownlee December 22, 2016 at 6:30 am #

REPLY ↩

Not sure off the cuff Jimi, perhaps repeat the run standalone for the top performing configuration.

DeepLearning January 4, 2017 at 6:08 am #

Thanks for the post. Can we optimize the number of hidden layers as well on top of number of r
Thanks

Jason Brownlee January 4, 2017 at 9:00 am #

Yes, it just may be very time consuming depending on the size of the dataset and the numb

Try it on some small datasets from the UCI ML Repo.

DeepLearning January 4, 2017 at 12:02 pm #

REPLY ↩

Thanks. Would you mind looking at below code?

def create_model(neurons=1, neurons2=1):
# create model
model = Sequential()
model.add(Dense(neurons1, input_dim=8))

[redacted]

model.add(Dense(neurons2))
model.add(Dense(1, init='uniform', activation='sigmoid'))
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
return model
# define the grid search parameters
neurons1 = [1, 3, 5, 7]
neurons2=[0,1,2]
param_grid = dict(neurons1=neurons1, neurons2=neurons2)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X, Y)

This code runs without error (I excluded certain X, y parts for brevity) but when I run “grid.fit(X, Y) it gives AssertionError

I’d appreciate if you can show me where I am wrong.

DeepLearning January 4, 2017 at 12:26 pm #

Update” It worked when I deleted 0 from neurons2. Thanks

Jason Brownlee January 5, 2017 at 9:16 am #

Excellent, glad to hear it.

Jason Brownlee January 5, 2017 at 9:16 am #

REPLY

A Dense() with a value of 0 neurons might blow up. Try removing the 0 from your neurons2 array.

A good debug strategy is to cut code back to the minimum, make it work, then and add complexity. Here. Try searching a grid of 1 and 1 neurons, make it all work, then expand the grid you search.

Let me know how you go.

[redacted]

DeepLearning January 9, 2017 at 11:04 am # REPLY

I keep getting error messages and I tried a big for loops that scan for all possible combinations of layer numbers, neuron numbers, other optimization stuff within defined limits. It is very time consuming code, but I could not figure it out how to adjust layer structure and other optimization parameters in the same code using GridSearch. If you would provide a code for that in your blog one day, that would be much appreciated. Thanks.

Jason Brownlee January 10, 2017 at 8:55 am # REPLY

I'll try to find the time.

Rajneesh January 11, 2017 at 10:48 am #

Hi Jason,
Many thanks for this awesome tutorial !

Jason Brownlee January 12, 2017 at 9:24 am #

I'm glad you found it useful Rajneesh.

Andy January 22, 2017 at 1:02 pm # REPLY

Hi Jason,

Great tutorial! I'm running into a slight issue. I tried running this on my own variation of the code and got

TypeError: get_params() got an unexpected keyword argument 'deep'

[redacted]

I copied and pasted your code using the given data set and got the same error. The code is showing an error on the grid_result = grid.fit(X, Y) line. I looked through the other comments and didn't see anyone with the same issue. Do you know where this could be coming from?

Thanks for your help!

YechiBechi January 23, 2017 at 2:18 am #

REPLY

same issue here,

great tutorial, life saver.

Jason Brownlee January 23, 2017 at 8:35 am #

Hi Andy, sorry to hear that.

Is this happening with a specific example or with all of them?

Are you able to check your version of Python/sklearn/keras/tf/theano?

UPDATE:

I can confirm the first example still works fine with Python 2.7, sklearn 0.18.1, Keras 1.2.0 and Tens

Andy January 25, 2017 at 7:12 am #

REPLY

The only differences are I am running Python 3.5 and Keras 1.2.1. The example I ran previously was the grid search for the number of neurons in a layer. But I just ran the first example and got the same error.

Do you think the issue is due to the next version of Python? If so, what should my next steps be?

Thanks for your help and quick response!

[redacted]

Jannes January 27, 2017 at 5:51 am #

REPLY ↩

It’s a bug in Keras 1.2.1. You can either downgrade to 1.2.0 or get the code from their github (where they already fixed it).

Jason Brownlee January 27, 2017 at 12:22 pm #

Yes, I have a write up of the problem and available fixes here:
http://stackoverflow.com/questions/41796618/python-keras-cross-val-score-error/41841066#41841066

Andy January 28, 2017 at 4:21 pm #

Thank you so much for your help!

kono February 8, 2017 at 3:14 am #

Jason,

Can you use early_stopping to decide n_epoch?

Jason Brownlee February 8, 2017 at 9:36 am #

REPLY ↩

Yes, that is a good method to find a generalized model.

[redacted]

Jayant February 23, 2017 at 4:33 am #

REPLY

Hi Jason,

Really great article. I am a big fan of your blog and your books. Can you please explain your following statement?

"A default cross-validation of 3 was used, but perhaps k=5 or k=10 would be more stable. Carefully choose your cross validation configuration to ensure your results are stable."

I didn't see anywhere cross-validation being used.

Jason Brownlee February 23, 2017 at 8:56 am #

Hi Jayant,

Grid search uses k-fold cross-validation to evaluate the performance of each combination of parame

Jing February 28, 2017 at 2:09 am #

Hi Jason,
thanks for this awesome tutorial !
I have two questions: 1. In "model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accu
GridSearchCV also has scoring parameter, if I set "scoring='f1'",which one is used for evaluate the resul
parameters ,e.g. 'accuracy'and 'f1' evaluating the results of grid search ?

Jason Brownlee February 28, 2017 at 8:13 am #

REPLY

Hi Jing,

You can set the “scoring” argument for GridSearchCV with a string of the performance measure to [redacted] learn about this argument here:
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

You can see a full list of supported scoring measures here:
http://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

As far as I know you can only grid search using a single measure.

Jing February 28, 2017 at 12:50 pm # REPLY

Thank you so much for your help!

Jing February 28, 2017 at 1:54 pm #

I find no matter what evaluate parameters used in GridSearchCV “scoring”,”metrics” in
program gives “ValueError: The model is not configured to compute accuracy.You should pass ‘r
So, if I set:
model.compile(loss=’binary_crossentropy’, optimizer=’adam’, metrics=[‘accuracy’])
grid = GridSearchCV(estimator=model, param_grid=param_grid, scoring=’recall’)
the grid_result.best_score_ =0.72.My question is: 0.72 is accuracy or recall ? Thank you!

Jason Brownlee March 1, 2017 at 8:31 am # REPLY

Hi Jing,

When using GridSearchCV with Keras, I would suggest not specifying any metrics when compiling your Keras model.

I would suggest only setting the “scoring” argument on the GridSearchCV. I would expect the metric reported by GridSearchCV to be the one that you specified.

I hope that helps.

[redacted]

Dan March 8, 2017 at 4:13 am #

REPLY

Great Blogpost. Love it. You are awesome Jason. I got one question to GridsearchCV. As far as i understand the crossvalidation already takes place in there. That's why we do not need any kfold anymore.
But with this technique we would have no validation set correct? e.g. with a default value of 3 we would have 2 training sets and one test set.

That means in kfold as well as in GridsearchCV there is no requirement for creating a validation set anymore?

Thanks

Jason Brownlee March 8, 2017 at 9:44 am #

Hi Dan,

Yes, GridSearchCV performs cross validation and you must specify the number of folds. You can ho

parameters found by the search if you like. This is optional.

Dan March 9, 2017 at 3:25 am #

Thank you for the quick response Jason. Especially considering the huge amount of qu

Jason Brownlee March 9, 2017 at 9:55 am #

REPLY

I'm here to help, if I can Dan.

[redacted]

Johan Steunenberg March 22, 2017 at 8:25 pm # REPLY ↰

What I'm missing in the tutorial is the info, how to get the best params in the model with KERAS. Do I pickup the best parameters and call 'create_model' again with those parameters or can I call the GridSearchCV's 'predict' function? (I will try out for myself but for completeness it would be good to have it in the tutorial as well.)

Jason Brownlee March 23, 2017 at 8:49 am # REPLY ↰

I see, but we don't know the best parameters, we must search for them.

Maycown Miranda April 5, 2017 at 2:09 am #

Hi, Jason. I am getting

```
/usr/local/lib/python2.7/dist-packages/keras/wrappers/scikit_learn.py in check_params(self=, params={'b
80 legal_params += inspect.getargspec(fn)[0]
81 legal_params = set(legal_params)
82
83 for params_name in params:
84 if params_name not in legal_params:
—> 85 raise ValueError('{} is not a legal parameter'.format(params_name))
params_name = 'epochs'
86
87 def get_params(self, _):
88 """Gets parameters for this estimator.
89
```

ValueError: epochs is not a legal parameter

Jason Brownlee April 9, 2017 at 2:32 pm #

[redacted]

It sounds like you need to upgrade to Keras v2.0 or higher.

Chandra Sutrisno Tjhong November 28, 2017 at 10:46 am #

REPLY

I experienced the same problem.I upgraded my keras and the same problem still occurs.

Usman May 3, 2017 at 7:56 am #

REPLY

Nice tutorial. I would like to optimize the number of hidden layers in the model. Can you please guide in this regard. thanks

Jason Brownlee May 4, 2017 at 7:59 am #

Thanks Usman.

Consider exploring specific patterns, e.g. small-big-small, etc.

Carl May 5, 2017 at 12:58 pm #

Do you know any way this could be possible using a network with multiple inputs?

http://imgur.com/a/JJ7f1

DanielP May 9, 2017 at 4:26 pm #

REPLY

Hi Jason, great to see posts like this – amazing job!

Just noticed, when you tune the optimisation algorithm SGD performs at 34% accuracy. As no paramete[redacted] it takes the default configuration, lr=0.01, momentum=0.0.

Later on, as you look for better configurations for SGD, best result (68%) is found when {'learn_rate': 0.01, 'momentum': 0.0}.

It seems to me that these two experiments use exactly the same network configuration (including the same SGD parameters), yet their resulting accuracies differ significantly. Do you have any intuition as to why this may be happening?

Jason Brownlee May 10, 2017 at 8:43 am #

REPLY ↩

Hi Daniel, yes great point.

Neural networks are stochastic and give different results when evaluated on the same data.

Ideally, each configuration would be evaluated using the average of multiple (30+) repeats.

This post might help:
http://machinelearningmastery.com/randomness-in-machine-learning/

Pradanuari May 14, 2017 at 3:13 am #

Hi Jason!
absolutely love your tutorial! But would you mind to give tutorial for how to tune the number of hidden lay

Thanks

Jason Brownlee May 14, 2017 at 7:32 am #

REPLY ↩

I have an example here:
http://machinelearningmastery.com/exploratory-configuration-multilayer-perceptron-network-time-series-forecasting/

Pradanuari May 14, 2017 at 11:32 pm #

REPLY

Thank you so much Jason!

Jason Brownlee May 15, 2017 at 5:53 am #

REPLY

I'm glad it helped Pradanuari.

Ibrahim El-Fayoumi May 17, 2017 at 12:53 pm #

Hello Jason
I tried to use your idea in a similar problem but I am getting error : AttributeError: 'NoneType' object has
it looks like the model does not define loss function?

This is the error I get:

```
b\site-packages\keras-2.0.4-py3.5.egg\keras\wrappers\scikit_learn.py in fit(self=, x=memmap([[[ 0., 0., 0
…, 0., 0., …, 0., 0., 0.]]], dtype=float32), y=array([[ 0., 0., 0., …, 0., 0., 0.],
…0.],
[ 0., 0., 0., …, 0., 1., 0.]]), **kwargs={})
135 self.model = self.build_fn(
136 **self.filter_sk_params(self.build_fn.__call__))
137 else:
138 self.model = self.build_fn(**self.filter_sk_params(self.build_fn))
139
–> 140 loss_name = self.model.loss
loss_name = undefined
self.model.loss = undefined
141 if hasattr(loss_name, '__name__'):
142 loss_name = loss_name.__name__
```

143 if loss_name == 'categorical_crossentropy' and len(y.shape) != 2:
144 y = to_categorical(y)

AttributeError: 'NoneType' object has no attribute 'loss'

Process finished with exit code 1

Regards
Ibrahim

[redacted]

Jason Brownlee May 18, 2017 at 8:26 am #

REPLY

Does the example in the blog post work on your system?

Ibrahim El-Fayoumi May 18, 2017 at 12:18 pm #

Ok, I think your code needs to be placed after
if __name__ == '__main__':

to work with multiprocess…

But thanks for the post is great…

Jason Brownlee May 19, 2017 at 8:12 am #

REPLY

Not on Linux and OS X when I tested it, but thanks for the tip.

Gautam August 25, 2017 at 11:33 pm #

[redacted]

n_jobs=-1 doesnt work on Windows.

@Ibrahim: Can you please explain, what part of the code needs to be behind if __name__ == '__main__':)

Edward May 21, 2017 at 3:17 am #

REPLY ↰

Hello Jason!
I do the first step – try to tune Batch Size and Number of Epochs and get
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
Best: 0.707031 using {'epochs': 100, 'batch_size': 40}
After that I do the same and get
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
Best: 0.688802 using {'epochs': 100, 'batch_size': 20}
And so on
The problem is in the grid_result.best_score_

I expect that in the second step (for ample tuning optimizer) I will get grid_result.best_score_ better than
grid_result.best_params_ from the first step). But it is not true
Tune all Hyperparameters is a very long time

How to fix it?

Jason Brownlee May 21, 2017 at 6:01 am #

Consider tuning different parameters, like network structure or number of input features.

Edward May 21, 2017 at 7:18 pm #

REPLY ↰

Thanks a lot Jason!

[redacted]

pattijane May 21, 2017 at 7:44 am #

REPLY

Hello,

I'd like to have your opinion about a problem:

I have two loss function plots, with SGD and Adamax as optimizer with same learning rate.
Loss function of SGD looks like the red one, whereas Adamax's looks like blue one.
(http://cs231n.github.io/assets/nn3/learningrates.jpeg)

I have better scores with Adamax on validation data. I'm confused about how to proceed, should I choose Adamax and play with learning rates a little more, or go on with SGD and somehow try to improve performance?

Thanks!

Jason Brownlee May 22, 2017 at 7:49 am #

Explore both, but focus on the validation score of interest (e.g. accuracy, RMSE, etc.) over

For example, you can get very low loss and get worse accuracy.

pattijane May 22, 2017 at 6:35 pm #

Thanks for your response! I experimented with different learning rates and found out a reasonable one, (good for both Adamax and SGD) and now I try to fix learning rate and optimizer and focus on other hyperparameters such as batch-size and number of neurons. Or would be better if I set those first?

Jason Brownlee May 23, 2017 at 7:49 am #

REPLY

[redacted]

Number of neurons will have a big effect along with learning rate.

Batch size will have a smaller effect and could be optimized last.

Lotem May 23, 2017 at 1:47 am #

REPLY

Thanks for this post!

One question – why not use grid search on all the parameters together, rather than preforming several grid searches and finding each parameter separately? surly the results are not the same…

Jason Brownlee May 23, 2017 at 7:54 am #

Great question,

In practice, the datasets are large and it can take a long time and require a lot of RAM.

StatsSorceress May 25, 2017 at 6:52 am #

Hi Jason,

Excellent post!

It seems to me that if you use the entire training set during your cross-validation, then your cross-validation error is going to give you an optimistically biased estimate of your validation error. I think this is because when you train the final model on the entire dataset, the validation set you create to estimate test performance comes out of the training set.

My question is: assuming we have a lot of data, should we use perhaps only 50% of the training data for cross-validation for the hyperparameters, and then use the remaining 50% for fitting the final model (and a portion of that remaining 50% would be used for the validation set)? That way we wouldn't be using the same data twice. I am assuming in this case that we would also have a separate test set.

Jason Brownlee June 2, 2017 at 11:38 am #

REPLY

Yes, it is a good idea to hold back a test set when tuning.

Yang May 27, 2017 at 5:35 am #

REPLY

Thanks for your valuable post. I learned a lot from it.
When I wrote my code for grid search, I encountered a question:

I use fit_generator instead of fit in keras.
Is it possible to use grid search with fit_generator ?

I have some Merge layers in my deep learning model.
Hence, the input of the neural network is not a single matrix.
For example:
Suppose we have 1,000 samples
Input = [Input1,Input2]
Input1 is a 1,000 *3 matrix
Input2 is a 1,000*3*50*50 matrix (image)

When I use the fit in your post, there is a bug….because the input1 and input2 don't have the same dime
work with grid search ?

Thanks in advance!

Yang May 27, 2017 at 6:46 am #

REPLY

Please ignore my previous reply.
I find an answer here: https://github.com/fchollet/keras/issues/6451
Right now, the GridsearchCV using the scikit wrapper for network with multiple inputs is not available.

Kate liu May 28, 2017 at 4:31 pm #

REPLY

Hi Jason, thank you for your good tutorial of the grid research with Keras. I followed your example with my own dataset. It could be run. But when I using the autoencoder structure, instead of the sequential structure, to gird the parameters with my own data. It could not be run. I don't know the reason. Could you help me? Are there any differences between the gird of sequential structure and the grid of model structure?

The follows are my codes:

```
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
import numpy as np
from keras.optimizers import SGD, Adam, RMSprop, Adagrad
from keras.regularizers import l1,l2
from keras.models import Model
import pandas as pd
from keras.models import load_model

np.random.seed(2017)

def create_model(optimizer='rmsprop'):

# encoder layers
encoding_dim =140
input_img = Input(shape=(6,))
encoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(input_img)
encoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(encoded)
encoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(encoded)
encoder_output = Dense(encoding_dim, activation='relu',W_regularizer=l1(0.01))(encoded)

# decoder layers
decoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(encoder_output)
decoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(decoded)
```

[redacted]

```
decoded = Dense(300, activation='relu',W_regularizer=l1(0.01))(decoded)
decoded = Dense(6, activation='relu',W_regularizer=l1(0.01))(decoded)

# construct the autoencoder model
autoencoder = Model(input_img, decoded)

# construct the encoder model for plotting
encoder = Model(input_img, encoder_output)

# Compile model
autoencoder.compile(optimizer='RMSprop', loss='mean_squared_error',metrics=['accuracy'])

return autoencoder
```

Jason Brownlee June 2, 2017 at 12:09 pm #

I'm surprised, I would not think the network architecture would make a difference.

Sorry, I have no good suggestions other than try to debug the cause of the fault.

Kate liu May 28, 2017 at 4:36 pm #

the command of autoencoder.compile is modified as the follows:

```
# Compile model
autoencoder.compile(optimizer=optimizer, loss='mean_squared_error',metrics=['accuracy'])
```

Rahul May 30, 2017 at 12:07 am #

REPLY

Can we do this for functional API as well ?

[redacted]

Jason Brownlee June 2, 2017 at 12:28 pm #

Perhaps, I have not done this.

Ian Worthington May 30, 2017 at 10:36 pm #

REPLY ↰

Thanks for a great tutorial Jason, appreciated.

njobs=-1 didn't work very well on my Windows 10 machine: took a very long time and never finished.

https://stackoverflow.com/questions/28005307/gridsearchcv-no-reporting-on-high-verbosity seems to suggest this is (or at least was in 2015) a known problem under Windows so I changed to n_jobs=1, which also allowed me to see throughput using verb

Jason Brownlee June 2, 2017 at 12:37 pm #

Thanks for the tip.

Ian Worthington May 31, 2017 at 1:56 am #

Jason —

Given all the parameters it is possible to adjust, is there any recommendation for which should be fixed f e change when others are changed?

Jason Brownlee June 2, 2017 at 12:39 pm #

REPLY ↰

Great question, see this paper:
https://arxiv.org/abs/1206.5533

[redacted]

Ian Worthington June 3, 2017 at 2:39 am # REPLY

Thanks Jason, I'll check it out.

Mario June 9, 2017 at 12:10 am # REPLY

Hi and thank you for the resource.

Am I right in my understanding that this only works on one machine?

Any hints / pointers on how to run this on a cluster? I have found https://goo.gl/Q9Xy7B as a potential av

Any comment at all? Information on the subject is scarce.

Jason Brownlee June 9, 2017 at 6:26 am #

Yes, this example is for a single machine. Sorry, I do not have examples for running on a cl

Shaun June 16, 2017 at 11:54 pm #

Hi Jason,

I'm a little bit confused about the definition of the "score" or "accuracy". How are they made? I believe that they are not simply comparing the results with target, otherwise it will be the overfitting model being the best (like the more neurons the better).

But on the other hand, they are just using those combinations of parameters to train the model, so what is the difference between I manually set the parameters and see my result good or not, with risk of overfitting and the grid search that creates an accuracy score to determine which one is the best?

Best regards,

[redacted]

Jason Brownlee June 17, 2017 at 7:30 am #

REPLY

The grid search will provide an estimate of the skill of the model with a set of parameters.

Any one configuration in the grid search can be set and evaluated manually.

Neural networks are stochastic and will give different predictions/skill when trained on the same data.

Ideally, if you have the time/compute the grid search should use repeated k-fold cross validation to provide robust estimates of model skill. More here: http://machinelearningmastery.com/evaluate-skill-deep-learning-models/

Does that help?

Shaun June 20, 2017 at 2:30 am #

I'm new to the NN, a little bit puzzled. So say, if I have to many neurons that leads to ov or test set), can grid search detect it by the score?

My guess is yes, because there is a validation set in the GridsearchCV. Is that correct?

Jason Brownlee June 20, 2017 at 6:39 am #

A larget network can overfit.

The idea is to find a config that does well on the train and validation sets. We require a robust test harness. With enough resources, I'd recommend repeated k-fold cross validation within the grid search.

Huyen June 19, 2017 at 4:21 pm #

REPLY

One more very useful tutorial, thank Jason.

One question about GridSearch in my case. I have tried to tune parameters of my neural network for reg[redacted] GridSearch totally long, like forever even though I have limited to the number. I saw in your code:

grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)

Normally, n_jobs=1, can I increase that number to improve the performances?

Jason Brownlee June 20, 2017 at 6:36 am # REPLY

We often cannot grid search with neural nets because it takes so long!

Consider running on a large computer in the cloud over the weekend.

Bobo June 21, 2017 at 4:57 am #

Hi Jason,

Any idea how to use GridSearchCV if you don’t want cross validation?

Jason Brownlee June 21, 2017 at 8:19 am #

GridSearch supports k-fold cross-validation by default. That is what the “CV” is in the name
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Bobo June 24, 2017 at 3:07 am # REPLY

So sklearn has no GridSearch without cross validation?
In any case I found kind of a hack here to get rid of cv:
https://stackoverflow.com/questions/44636370/scikit-learn-gridsearchcv-without-cross-validation

[redacted]

Jason Brownlee June 24, 2017 at 8:04 am #

REPLY ↩

You can configure the k in CV to 1 to it does train/test. Then configure it to repeat.

makis June 28, 2017 at 11:54 pm #

REPLY ↩

Hello. Thank you for the nice tutorial.

I am trying to combine pipeline and gridsearch.

Inside my keras model i use kernel_initializer=init_mode.
Then I am trying to assign values to the init_mode dictionary in order to perform the gridsearch.

I get the following error: ValueError: init_mode is not a legal parameter

My code is here: https://www.dropbox.com/s/57n777j9w8bxf4t/keras_grid.py?dl=0

Any tip? Thank you

Abhijith Darshan Ravindra July 11, 2017 at 6:31 am #

Hi Dr. Brownlee,

When I run this in Spyder IDE nothing happens after grid.fit.

It just appears to do nothing.

Any suggestions as to why?

Jason Brownlee July 11, 2017 at 10:34 am #

[redacted]

Consider running from the command line.

The grid search may take a long time.

DY July 14, 2017 at 6:11 am #

REPLY

Hello Dr Brownlee,

I saved your example codes into .py file and run it. Nothing happens after grid.fit. However, if I run line by line from your example codes it works. Do you know why?

Jason Brownlee July 14, 2017 at 8:36 am #

It may take a long time. Consider reducing the scope of the search to see if you car

Tryfon September 18, 2017 at 11:46 pm #

I had the same issue with you (using spyder and python 3.6) but after changing the parame stuck although spyder showed it was running in the backgound (I checked the CPU usage and was running).

Don't ask the reason why is that. My guess would be that it has to do with your system and the fact t GPU).

Jason Brownlee September 19, 2017 at 7:47 am #

REPLY

Consider running the example from the command line instead.

Kamal Thapa July 27, 2017 at 3:46 pm # REPLY

How can I do Hyper-parameter optimization for MLPRegressor in scikit learn?

Jason Brownlee July 28, 2017 at 8:28 am # REPLY

Yes.

Josep August 3, 2017 at 2:31 am #

Hi Jason,
I'm unable to apply the grid search to a seq to seq LSTM network (Keras Regressor model in the scikit A
to r^2 (or any scoring function for regression problems) the model.fit expect a 2 dim input vector, not the
Otherwise, if I left the default scoring algorithm named "_passthrough_scorer"(I don't know what it does
best_score doesn't match with the real best parametrization. I'm really confused…I'll had to write the gri

Josep August 3, 2017 at 2:42 am #

I've solved it, I share it if someone have the same issue…,If you set the gridsearch scoring
Keras model.

Josep August 3, 2017 at 2:49 am # REPLY

Sorry for bothering, but the results of the approach I've said are incorrect. I don't know what to do.

REPLY

Jason Brownlee August 3, 2017 at 6:54 am #

Hi Josep,

Consider writing your own for loop to iterate over params and run a Cross Validation for the params within the loop.

This is how I do it now for large/complex models.

kotb August 8, 2017 at 7:10 pm #

REPLY

Can i use this grid search without using keras model

Jason Brownlee August 9, 2017 at 6:27 am #

For sure!

Aman Garg August 19, 2017 at 3:35 am #

Hello Jason,

Thanks for such a nice tutorial.

Instead of getting a output as 'Best: 0.720052 using {'init_mode': 'uniform'}' , it would be really nice if you could show us how to visualize this result with matplotlib so that it gets more easier.

Jason Brownlee August 19, 2017 at 6:23 am #

REPLY

Great suggestion, thanks.

Michael August 20, 2017 at 4:42 am #

REPLY ↩

Hi, Jason. Thanks, again, for all of the blog posts and example code. I'm trying to tune my binary classification Keras neural network. My dataset includes about 50,000 entries with 52 (numeric) variables. Using Grid Search, I've tested all sorts of combinations of layer size, number of epochs, batch size, optimizers, activations, learning rates, dropout rates, and L2 regularization parameters. My grid search shows every combination performs the same. For example, here is a snippet from my latest results:

Best: 0.876381 using {'act': 'relu', 'opt': 'Adam'}
0.876381 (0.003878) with: {'act': 'relu', 'opt': 'Adam'}
0.876381 (0.003878) with: {'act': 'relu', 'opt': 'SGD'}
0.876381 (0.003878) with: {'act': 'relu', 'opt': 'Adagrad'}
0.876381 (0.003878) with: {'act': 'relu', 'opt': 'Adadelta'}
0.876361 (0.003880) with: {'act': 'tanh', 'opt': 'Adam'}
0.876381 (0.003878) with: {'act': 'tanh', 'opt': 'SGD'}

But I also get 0.876381 whether I have 1000 nodes or 1 node, and for every other combo I've tested. I'v
my input data with no impact.

Do you have any thoughts on why I'm having trouble finding different combinations of parameters that a

Thank you for your help! You rock!

Jason Brownlee August 20, 2017 at 6:09 am #

Very odd results. Double check your train/test data.

Also, see this post for a long list of ideas to try:
http://machinelearningmastery.com/improve-deep-learning-performance/

Michael August 20, 2017 at 9:20 am #

REPLY ↩

Thanks

[redacted]

Shubham Kumar September 3, 2017 at 11:54 am #

REPLY

Hey Jason.
I was using grid search to tune hyperparameters for a CNN-LSTM classification problem.
I used the code template on your blog about sequence classification.
MY original data has 38932 instances, but for tuning I am using only 1000 to save time.
But even then, I am not sure how to best search for those parameters and save time.

Is it a bad idea to search for hyper parameters in a small subset (almost 1/40th of training in my case).
Will the result vary largely when I use actual data size?
Also, I passed in several parameters for the grid search. Left it overnight and it still hadn't made enough
How can I speed up this process?

Jason Brownlee September 3, 2017 at 3:44 pm #

The result will be biased, but perhaps might give you an idea of the direction in which to pro

I often run a lot of sanity check grid searches on small samples to get ideas on which direction to pu

More data will result in less biased estimates of model skill, often proportionately to some point of di

Shubham Kumar September 4, 2017 at 3:10 am #

REPLY

Great !
I did read that one of the sanity checks is to check whether the model overfits on a small sample! If yes, then we are good to go…
I am slightly new to building proper models and find this part exciting but a little intimidating at the same time !
I am going to use only a few hyper parameters at a time, and keep the rest constant and check what happens !

[redacted]

Love your posts ! They are amazingly helpful .
Does the Python LSTM book have code snippets in Python 3 as well?
Coz it becomes a little difficult to search for the right modules and attributes otherwise :/

Jason Brownlee September 4, 2017 at 4:39 am #

REPLY

THanks.

Yes, the code in my LSTM book was tested with Python 2.7 and Python 3.5.

Kaushal Shetty September 8, 2017 at 12:24 am #

Hi Jason, Is this a valid approach to decide the number of layers?

```
def neural_train(layer1 = 1,layer2 = 1,layer3 = 1,layers = 1):

input_tensor = Input(shape=(2001,))
x = Dense(units = layer1,activation='relu')(input_tensor)
if layers == 2:
x = Dense(layer2,activation = 'relu')(x)
if layers ==3 :
x = Dense(layer2,activation = 'relu')(x)
x = Dense(layer3,activation = 'relu')(x)

output_tensor = Dense(10,activation='softmax')(x)
model = Model(input_tensor,output_tensor)
model.compile(optimizer = 'rmsprop',loss='categorical_crossentropy',metrics = ['accuracy'])
return model

layer1 = [1024,512]
layer2 = [256,100]
layer3 = [60,40]
epochs = [10,11]
```

[redacted]

layers = [2,3]
param_grid = dict(epochs = epochs,layer1 = layer1,layer2 = layer2,layer3 = layer3,layers=layers)
model = KerasClassifier(build_fn = neural_train)
gsv_model = GridSearchCV(model,param_grid=param_grid)
gsv_model.fit(x_train,y_train)

Jason Brownlee September 9, 2017 at 11:45 am #

REPLY

Maybe, you must have a test harness that you can trust, then explore different configurations of your model.

I have more on robustly evaluating neural nets here:
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

ari September 9, 2017 at 1:29 am #

Very helpful post Jason. Thanks for this. Are there any advantages for using gridsearch over so knowledge is one faster than the other?

Jason Brownlee September 9, 2017 at 11:58 am #

Depends on your data and model. Use the took that you prefer.

Shubham Kumar September 10, 2017 at 4:38 am #

REPLY

{‘split0_test_score’: array([0.6641791, 0.6641791, 0.6641791, 0.6641791]), ‘split1_test_score’: array([0.65413534, 0.65413534, 0.65413534, 0.65413534]), ‘split2_test_score’: array([0.69924811, 0.69924811, 0.69924811, 0.69924811]), ‘mean_test_score’: array([0.6725, 0.6725, 0.6725, 0.6725]), ‘std_test_score’: array([0.01931902, 0.01931902, 0.01931902, 0.01931902]), ‘rank_test_score’: array([

0.67669174, 0.67669174, 0.67669174]), ‘split1_train_score’: array([0.68164794, 0.68164794, 0.68164[redacted]
0.65917602, 0.65917602, 0.65917602, 0.65917602]), ‘mean_train_score’: array([0.67250523, 0.67250523, 0.67250523, 0.67250523]), ‘std_train_score’: array([0.00963991, 0.00963991, 0.00963991, 0.00963991]), ‘mean_fit_time’: array([36.72573058, 37.0244147 , 38.12670692, 40.71116368]), ‘std_fit_time’: array([0.4829061 , 0.35207924, 0.13746276, 2.71443639]), ‘mean_score_time’: array([1.49508754, 1.76741695, 2.14029002, 2.67426189]), ‘std_score_time’: array([0.04907801, 0.11919153, 0.07953362, 0.13931651]), ‘param_dropout’: masked_array(data = [0.2 0.5 0.6 0.7],
mask = [False False False False],
fill_value = ?)
, ‘params’: ({‘dropout’: 0.2}, {‘dropout’: 0.5}, {‘dropout’: 0.6}, {‘dropout’: 0.7})}

Hey. I was hypertuning a model on 4 different choices of hyper parameters. However, in the grid_results_ dictionary, the rank_test_score key has array with all same values. I find that confusing. Shouldn’t it have 4 different values in each place?
Something like [1,3,2,4] ?
What could be the explanation for this?

Shubham Kumar September 10, 2017 at 4:50 am #

It must have something to do with all mean_test_scores being the same ,

Jason Brownlee September 11, 2017 at 12:03 pm #

If you are testing 4 different values for one parameter, then you must build 4 models/comple

Does that help?

Shubham Kumar September 13, 2017 at 5:20 am #

REPLY

I am sorry. That’s confusing. 4 models or complete 4 runs means ?

Things are different if we are gridsearching/randomsearching just for one hyperparameter?

Does it have something to do with the actual code used to write TensorFlow /keras ?

[redacted]

Jason Brownlee September 13, 2017 at 12:36 pm # REPLY

If you have one parameter and you want to test 4 values, each value needs one run. Ideally, we would run many times for each parameter value and take the average skill score given the stochastic nature of ML algorithms.

For a random search, you run for as long as you like.

Does that help?

Shubham Kumar September 13, 2017 at 11:17 pm #

What I understand is that when we have more than 1 (say 2) hyper-parameters in a grid, then fo epochs as I have specified, with as many training-cross-validation sets as specified (the CV in GridSearc training-cross-validation set, we get the avg accuracy over all the cross-validation sets for every combina

So when you say 1 run only in the case of a single hyperparameter, that means only 1 training-crossvali any averaging involved.

Is that what I have to do? Change the training-crossValidation set to just 1?

Jason Brownlee September 15, 2017 at 12:06 pm #

Yes, 1 run is one CV pass (k folds).

Rishi September 18, 2017 at 5:18 am # REPLY

Jason,
would you please post an example of inheriting from KerasClassifier (or KerasRegressor) to create your

the most part:

class MLP_Regressor(KerasRegressor):

def __init__(self, **sk_params):
super().__init__(build_fn=None, **sk_params)

def __call__(self, optimizer='adam', loss='mean_squared_error', **kwargs):
# more code goes here (that was previously in 'build_fn'

I can include this in a pipeline and it runs perfectly:
MLP Pipeline(memory=None,
steps=[('MLP',)])

Only thing is: The Keras documentation includes the 'build_fn' keyword argument:

keras.wrappers.scikit_learn.KerasClassifier(build_fn=None, **sk_params)

While the actual KerasClassifier class definition shows the following in its __init__ method:

def __init__(self, model, optimizer='adam', loss='categorical_crossentropy', **kwargs):
super(KerasClassifier, self).__init__(model, optimizer, loss, **kwargs)

I'm not sure if my __init__ in MLP_Regressor has been setup correctly (to avoid hidden bugs in the futur

Would greatly appreciate it! (I've searched, but couldn't find a single example of KerasClassifier inheritar

Jason Brownlee September 18, 2017 at 5:49 am #

Thanks for the suggestion, I have not done this but perhaps in the future.

Rishi November 21, 2017 at 12:54 pm #

REPLY ↩

Jason, managed to get the inherited class working perfectly now:

class MLP_Classifier(KerasClassifier):

[redacted]

```
def __init__(self, build_fn=None, **sk_params):
self.sk_params = sk_params
super().__init__(build_fn=None, **sk_params)

def __call__(self, callbacks=None, layer_sizes=None,activations=None,input_dim=0,init='normal',optimizer='adam', metrics='accuracy',
loss='binary_crossentropy', use_dropout_input=False, use_dropout_hidden=False):
""""
Constructs, compiles and return a Keras model
Implements the "build_fn" function

Returns a "Sequential" model
""""
# Code to build a model (that would typically go in "build_fn") goes here.
return model
```

Jason Brownlee November 22, 2017 at 10:47 am #

Well done!

Tmn September 20, 2017 at 2:45 am #

Hi Jason,

I can not thank you enough. I am sure that there are many people like me who have learnt a lost from yo following your tutorial for more than 3 year now. Before I was using R however, recently I moved to python for Deep learning. And I find your tutorial as usual, exceptional. I think Andrew Ng and CS231n (andrej karpathy), theoretical course and your programming course on deep learning is one of the best in the world. You rock! Thanks a lot.

I do have a question 🙂 as well.
The grid search parameter tuning works perfectly with CPU. I agree with your suggestion not to tune everything at once. Now I moved to GPU implementation. I was able to execute the code if I chose options n_job=1. However, if I do multi-threading n_job=-1. I am getting "CUDA_ERROR_OUT_OF_MEMORY". I have GeForce GTX 1080. Did you happen to encounter simila

Once, again thank you.

[redacted]

Jason Brownlee September 20, 2017 at 6:00 am #

REPLY ↰

Thanks for all of your support!

Yes, I have the same and I would recommend using a “single thread” and let the GPU do its thing for a given single run.

In general, I’d recommend contrasting different approaches to grid searching (cpu/gpu) and use the approach that is overall faster for your specific tests.

Tmn September 20, 2017 at 11:33 pm #

Hi Jason,
Thank you for the response. The parameter search using CPU (n_job=-1) is (2.961489-4.97775
142.151023) second.

One more thing, after grid search I have value for parameters {batch_size, activation, neurons, l
wonder why reusing these model parameter does not provide the same results, now accuracy is
same parameter the accuracy remains the same (52%). I could not achieve the accuracy as sho
doing 5-fold CV I do not expect the accuracy to be the same since it is stochastic process but it
you also happen to encounter the same thing ?

Also the best parameter values changes in each executions with an accuracy SD±5%.

Thanks

P.S:
Below code is something I am doing to limit GPU memory usage and run multiple grid search. However, we should know the memory usage in advance (cs231n.github.io/convolutional-networks/#case). Let me know if it makes sense.

Also, we can use n-job. I tried with n_job = 2 however the GPU memory is allocated based on fraction. I am searching how to allocated memory based on MB. I will do more research on this “CUDA_ERROR_OUT_OF_MEMORY” and update you.

[redacted]

import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.3
set_session(tf.Session(config=config))

Thanks!

Jason Brownlee September 21, 2017 at 5:42 am #

REPLY

The results for the standalone model should fit into the distribution of the grid search results – if you repeated each grid search result many times, e.g. 10-30. See this post on evaluating model skill of neural networks:
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

Nice, sorry, I cannot give you good advice on grid searching with the GPU, it is not somethin
serially or across AWS instances.

TMN October 6, 2017 at 2:12 am #

Hi Jason,

Could you please help on how to do features normalization while doing the grid search a
automatically here, GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1
= scaler.transform(X_train), this will introduce bias in cross-validation. Also, if possible, c
learn wrapper with Keras for advance options, is their any limitation on wrapper ?
Thanks

Without normalization:
Best: 0.535211 using {'learn_rate': 0.01, 'dropout_rate': 25, 'batch_size': 40, 'neurons': 200, 'init_mode': 'lecun_uniform', 'optimizer': 'SGD', 'activation': 'relu', 'epochs': 1000}

With normalization:
Best: 0.695775 using {'optimizer': 'SGD', 'batch_size': 132, 'init_mode': 'lecun_uniform',

'neurons': 200, 'activation': 'relu'}

[redacted]

Jason Brownlee October 6, 2017 at 5:37 am #

Perhaps you can normalize your data prior to the grid search?

TMN October 6, 2017 at 10:59 am #

I normalize my data prior to grid search using X = scaler.transform(X_train) but dont you think it would introduce bias in the performance. Normally, I expect to normalize train set and use that normalization factor May be I did not understand you properly, how do you do normalization prior to grid sea

Thanks

Jason Brownlee October 6, 2017 at 11:07 am #

Yes, it's a struggle or trade-off.

Perhaps you can see if a Pipeline will work in the grid search, it may, but I expect it will e

Perhaps the bias is minor and you can ignore it.

Perhaps you can implement your own grid search loop to only use training data to calculate data scaling coefficients.

TMN October 6, 2017 at 6:44 pm #

I started looking at the pipeline (http://scikit-learn.org/stable/modules/pipeline.html) on how they have been using it for SVM, lets see. I would expect the pipeline to work for Keras as well, as this is a classical problem in machine learning. Why do you expect error here? I wanted to take the full advantage from automatic grid search. Well, the final option will b

The bias is really significant in 5-repeated 10-fold CV. Thanks

[redacted]

Without normalization:
Best: 0.535211 using {'learn_rate': 0.01, 'dropout_rate': 25, 'batch_size': 40, 'neurons': 200, 'init_mode': 'lecun_uniform', 'optimizer': 'SGD', 'activation': 'relu', 'epochs': 1000}

With normalization:
Best: 0.695775 using {'optimizer': 'SGD', 'batch_size': 132, 'init_mode': 'lecun_uniform', 'epochs': 1000, 'learn_rate': 0.01, 'dropout_rate': 25, 'neurons': 200, 'activation': 'relu'}

Jason Brownlee October 7, 2017 at 5:51 am #

If it works, that is great. I have seen cases where when grid search + keras get

I have a tutorial on Pipeline here that might help:
https://machinelearningmastery.com/automate-machine-learning-workflows-pipelines-py

HWU September 22, 2017 at 6:52 am #

This is such a great, thorough tutorial. Thanks for keeping your tutorials up to date! It's so nice t
work because they've been tested on recent versions of required packages.

REPLY

Jason Brownlee September 23, 2017 at 5:32 am #

Thanks!

REPLY

Marjan September 29, 2017 at 1:08 pm #

Thank you for your great tutorial. I tried to use it for my model with multiple inputs. but It didn`t work. I fo[redacted]
multiple inputs. it gives me an error for grid.fit([input1,input2],y)
Do you have any suggestion to handle it?
Thanks,

Jason Brownlee September 30, 2017 at 7:34 am #

REPLY ↩

Sorry I do not. Perhaps run the grid search manually (e.g. your own for loop)?

Buz Fifer October 5, 2017 at 7:06 am #

When I run your code to tune the dropout_rate, I get the following error:
ValueError: dropout_rate is not a legal parameter

In fact, I get this error for all labels except epochs and batch_size. Both of these were recognized and ra
anywhere, even in API docs. Any suggestions?

Jason Brownlee October 5, 2017 at 5:16 pm #

What do you mean by valid labels exactly?

Buz Fifer October 6, 2017 at 3:02 am #

REPLY ↩

Sorry, I should have included the code in the first place. I have added comments in the code to show exactly what I tried for each parameter.

# ———— Define Keras Classifier Wrapper
model1 = KerasClassifier(build_fn=kerasModel1, epochs=5, batch_size=10, verbose=0)

# ———— define the grid search parameters
mybatchs = [10, 20, 128]
myepochs = [5, 10, 20, 50, 60, 80, 100]
mylearn = [0.001, 0.002, 0.0025, 0.003]
myopts = ['Adam', 'Nadam', 'RMSprop']
myinits = ['uniform', 'normal', 'lecun_uniform', 'lecun_normal', 'glorot_uniform', 'glorot_normal']
mydrop = [0.10, 0.20, 0.30, 0.35, 0.40, 0.50, 0.60, 0.70, 0.80]

# ————– Not Recognized
#param_grid = dict(optimizer=myopts)
#param_grid = dict(learn_rate=mylearn)
#param_grid = dict(learning_rate=mylearn)
#param_grid = dict(init=myinits)
#param_grid = dict(init_mode=myinits)
#param_grid = dict(dropout_rate=mydrop)

# ———— Recognized
#param_grid = dict(epochs=myepochs) # —– OK
#param_grid = dict(batch_size=mybatchs) # —– OK

I removed comment # and ran each one separately. For example, running the first param_grid v
parameter. They all got the same rejection notice except for epochs and batch_size.
I hope that helps.

Buz Fifer October 6, 2017 at 3:09 am #

Just to be clearer, each parameter had it's own name in the error message as follows:

Error – optimizer is not a valid parameter
Error – learn_rate is not a valid parameter
Error – learning_rate is not a valid parameter
Error – init is not a valid parameter
Error – init_mode is not a valid parameter
Error – dropout_rate is not a valid parameter

Jason Brownlee October 6, 2017 at 5:39 am #

REPLY

That is odd, I don’t have any good ideas, other than continue to debug and try different variations to see if you can expose the cause of the issue.

Double check all of your python libraries are up to date.

ritika October 6, 2017 at 11:49 pm #

REPLY

Hi Jason, Very nice tutorial..very well explained

Jason Brownlee October 7, 2017 at 5:55 am #

Thanks.

TC October 17, 2017 at 10:27 am #

Hi Jason thanks for the great post.

Let’s say I’m using 5 fold CV on a relatively small dataset (not necessarily for a deep learning model). In
might be quite high, and just by chance, a point on the grid that is in reality far from optimal, might be selected as the “best”.

So are there any approaches to smooth out the response surface of the grid search, to deal with “spikes” in performance due to variance?

Jason Brownlee October 17, 2017 at 4:05 pm #

REPLY

Wonderful question.

Yes, we can approach this problem by increasing the number of repeats (not folds) of each param [redacted]

TC October 20, 2017 at 8:40 am #

REPLY

Hi Jason, by “number of repeats” do you mean to just repeat the process many times, with perhaps a different random seed?

Jason Brownlee October 21, 2017 at 5:23 am #

REPLY

Exactly.

Lea October 20, 2017 at 10:09 pm #

Thank you for this great tutorial! I tried to adapt the code for a CNN, but I am running constantly

That is the code:

```
def create_model(nb_filters=3, nb_conv=2, pool=20):
model = Sequential()
model.add(Convolution1D(nb_filters, nb_conv, activation='relu',
input_shape=(X.shape[1], X.shape[2]), padding="same"))
model.add(MaxPooling1D(pool))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
return model
```

```
model = KerasClassifier(build_fn=create_model(), verbose=0)
```

```
nb_conv = [2, 4, 6, 8, 10]
pool= [10, 20, 30, 50]
```

[redacted]

param_grid = dict(nb_conv=nb_conv, epochs=pool)
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X, y)

And the error I am getting is “nb_conv is not a legal parameter”. Unfortunately, I do not understand why.

Jason Brownlee October 21, 2017 at 5:37 am #

REPLY ↩

The API has changed:
https://keras.io/layers/convolutional/

went October 22, 2017 at 2:55 am #

Hi Jason,

Great post and Thank you.

What do you think is the best sequence when tuning all those Hyperparameters? I think difference sequ

Jason Brownlee October 22, 2017 at 5:32 am #

This post has some ideas (at the end):
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Also see the referenced paper.

Bgie October 23, 2017 at 6:28 am #

REPLY ↩

Hi Jason,

[redacted]

What a great blog, I very much appreciate you sharing some of your expertise!

I want to grid search the hyperparams from my CNN, but I'm using data augmentation with ImageDataGenerator. So I'm not calling model.fit but model.fit_generator for the actual training.
This does not seem to be supported through the grid search..
Am I forced to write my own KerasClassifier implementation?

Would you advise to just fall back to using (nested) for loops instead, or would I be missing some 'magic' from the existing scikit gridsearch?

Jason Brownlee October 23, 2017 at 4:10 pm # REPLY

I would recommend writing your own for loops to grid search instead.

Shubham Kumar October 26, 2017 at 3:58 am #

Hey Jason!

Needed help with model improvement!
Can you help me in understanding how to realize whether your model is suffering from bad local minima, vanishing/exploding gradient problem?

Jason Brownlee October 26, 2017 at 5:34 am #

If you have exploding or vanishing gradients, then you will have NaN outputs.

This post will give you ideas on how to lift skill:
http://machinelearningmastery.com/improve-deep-learning-performance/

This post will give you advice on how to effectively evaluate your model:
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

Shubham Kumar November 6, 2017 at 7:05 am #

REPLY

NaN outputs as in my predictions ?
Or the weights ?
If exploding gradient then weight will be very large (probably NaN) hence output would also be NaN.
But how will this logic be used for vanishing gradients. I this case the weights basically stop changing r8?

Shubham Kumar November 6, 2017 at 7:07 am #

REPLY

Should I use some kind of code that checks by how much the weights at each layer are changing , and if after a certain threshold they haven't changed by a certain amount, I'll declare vanishing gradient !

Jason Brownlee November 7, 2017 at 9:44 am #

Try gradient clipping on the optimization algorithm.

Mustafa Murat ARAT October 28, 2017 at 12:45 am #

I have a question for you, Jason and for general audience. I tried to find optimal number of neur function which contains my deep learning model. It is fast enough for the values I define and I get a resu e, it is extremely slow and never reached to an end. How long does it take on your computer?

Jason Brownlee October 28, 2017 at 5:14 am #

REPLY

You could try to test fewer parameters or try to search on a smaller dataset?

Mustafa Murat ARAT October 29, 2017 at 4:12 am #

REPLY

Hey Jason,

Thank you for your quick reply. I try grid search for number of neurons on Iris data set for the purpose of learning. I scale the data first and then transform and encode the dependent variable. However, first of all, even though I use small data set or fewer parameters, it is slow; second of all, when I get the results, it is all zero. This is very basic example and I am pretty much sure that my code is correct but I guess I am missing out something.

```
Best: 0.000000 using {‘neurons’: 3}
0.000000 (0.000000) with: {‘neurons’: 3}
0.000000 (0.000000) with: {‘neurons’: 5}
```

THE CODE:

```
from pandas import read_csv
import numpy
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from sklearn.model_selection import GridSearchCV

dataframe=read_csv(“iris.csv”, header=None)
dataset=dataframe.values
X=dataset[:,0:4].astype(float)
Y=dataset[:,4]

seed=7
numpy.random.seed(seed)

#encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
```

[redacted]

```
encoded_Y = encoder.transform(Y)
#one-hot encoding
dummy_y = np_utils.to_categorical(encoded_Y)

scaler = StandardScaler()
X = scaler.fit_transform(X)

def create_model(n_neurons):
model = Sequential()
model.add(Dense(n_neurons, input_dim=X.shape[1], activation='relu')) # hidden layer
model.add(Dense(3, activation='softmax')) # output layer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
return model

model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=10, initial_epoch=0, v
# define the grid search parameters
neurons=[3, 5]

#this does 3-fold classification. One can change k.
param_grid = dict(n_neurons=neurons)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(X, dummy_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))
```

Jason Brownlee October 29, 2017 at 5:59 am #

REPLY ↩

Sorry, I cannot debug your code/problem for you.

[redacted]

Mustafa Murat ARAT October 30, 2017 at 8:29 am #

I totally understand you. Thank you so much, though. I figured out my mistake. Iris dataset is very well balanced so I need to shuffle the data because GridSearchCV is using 3-Fold Cross Validation.

Jason Brownlee October 30, 2017 at 3:49 pm #

Glad to hear it.

jenny November 8, 2017 at 4:12 am #

Thanks for sharing such a wonderful tutorial. Learnt many new things.

How can i save all the models that the grid search is generation with identifiers for each model?

I am an R user. This how I do it in R to save models with passing parameter values to its names.

xgb.object <- paste0('/path/xgb_disc20_new_',

sample.sizes[i], '_', s,'_',nrounds[j],'_',max.depth[k],'_',eta[l], '.RData')

write.table(cbind(sample.sizes[i], s,nrounds[j],max.depth[k],eta[l],tpr, tnr, acc, roc.area,

concordance), paste0('/path/xgb_disc20_new_', min.sample.size,'_', max.sample.size,

'.csv'), append=TRUE, sep=",",row.names=FALSE,col.names=FALSE)

How can this be achieved in python for keras(neural network) and other models in other libraries?

Jason Brownlee November 8, 2017 at 9:29 am #

REPLY ↩

I would recommend using grid search to find the parameters for a well performing model then train [redacted] you can then save.

jenny November 8, 2017 at 9:34 pm #

REPLY

thank you jason for your quick reply . I will try that way.

Wassim November 16, 2017 at 11:26 pm #

REPLY

Hi Jason,
Thank you for the great tutorial. I just have an issue when using exactly your code: when I try to paralleli
error “AttributeError: Can’t get attribute ‘create_model’ on ” while it works well without parallelization. Any
Thank you,
Wassim

Jason Brownlee November 17, 2017 at 9:25 am #

I’m not sure, perhaps you cannot parallelize the grid search with Keras models.

Sangwon Chae November 28, 2017 at 9:51 pm #

REPLY

Hi Jason,

The example code calculates the best score for accuracy to obtain the hyperparameter.

In my problem, I want to find RMSE rather than accuracy because it is regression problem (numerical prediction).

However, ‘grid_result.cv_resluts_’ only provides ‘fit_time’ and ‘score’, so it can not calculate RMSE.

[redacted]

What should I do?

Thank you.

Jason Brownlee November 29, 2017 at 8:23 am #

REPLY

You can change the configuration to calculate MSE (e.g. scoring='neg_mean_squared_error') and then take the square root.

Learn more here:
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Estelle December 5, 2017 at 7:37 am #

Hi Jason,

Thank you for this post.

Is there anything that prevents me to use Grid Search with train_on_batch() instead of fit()?

Thank you for letting me know.

All the best,

Estelle

Jason Brownlee December 5, 2017 at 10:26 am #

REPLY

I think the wrapper is quite limited and does not offer this facility via sklearn.

Estelle December 6, 2017 at 8:15 am #

[redacted]

Thanks for your quick answer.

All the best,

Estelle

Jason Brownlee December 6, 2017 at 9:08 am # REPLY

No problem.

Peter December 8, 2017 at 1:56 pm #

Thanks very much for the tutorial. It is extremely helpful for my work. I came across a problem w
want to run the same grid search on different datasets. Everything works fine on the first dataset. But wh
program got stuck there. I run the grid search with n_jobs=-1 and put keras.backend.clear_session() bet
data twice in your examples. Could you please kindly help me with this issue?

Jason Brownlee December 8, 2017 at 2:30 pm #

I'm sorry to hear that, perhaps change n_jobs to 1?

Peter December 8, 2017 at 2:40 pm # REPLY

Thanks for the quick reply. It works when n_jobs=1, but I do need parallel threads for speed.

Jason Brownlee December 9, 2017 at 5:34 am #

The neural network will be using all the cores, so running multiple threads may not [redacted]

Peter December 11, 2017 at 9:53 am #

I got it to work by just fitting one dataset in the python script and looping the python script over multiple datasets in a bash script. I am still not clear why second fitting fails in python, but this is a not-so-beautiful workaround.

Jason Brownlee December 11, 2017 at 4:52 pm #

Glad to hear that you made some progress.

Daniel Pamplona December 13, 2017 at 1:37 am #

Hi Jason

Thank you so much for sharing your knowledge.
I am trying to optimize the number of hidden layers.
I can´t figure it out how to do it with keras (actually I am wondering how to set up the function create_mo
Could you please help me?
Thank you

Jason Brownlee December 13, 2017 at 5:42 am #

REPLY

Perhaps the number of layers could be a parameter to your function.

Sean December 15, 2017 at 1:43 am #

REPLY

Hi Jason,

Thanks for this insightful and useful tutorial as always

No doubt your blog posts are arguably the best in the field of data sciences

Best wishes

Jason Brownlee December 15, 2017 at 5:36 am #

REPLY

Thanks Sean.

Sean December 16, 2017 at 12:06 am #

Hello Jason,
I decided to try the code on a textual data of about 3000 tweets having binary classification (Y) and the t
size and number of epochs

but got the following error:

```
Name: 0, Length: 2185, dtype: object, names=['dense_1_input'], shapes=[(None, 8)], ch
    148                             raise ValueError(
    149                                 'Error when checking ' + exception_prefix +
    150                                 ': expected ' + names[i] +
    151                                 ' to have shape ' + str(shapes[i]) +
    152                                 ' but got array with shape ' +
--> 153                                 str(array.shape))
        array.shape = (2185, 1)
    154     return arrays
    155
    156
    157 def _standardize_sample_or_class_weights(x_weight, output_names, weight_type):

ValueError: Error when checking input: expected dense_1_input to have shape (None, 8)
```

Here's the modified code below:

[redacted]

```
# Use scikit-learn to grid search the batch size and epochs
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(12, input_dim=8, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
# load dataset
dataset = pd.read_csv('training.txt', delimiter = '\t', quoting = 3, header = None)
# split into input (X) and output (Y) variables
X = dataset.iloc[:,0]
Y = dataset.iloc[:,1]
# create model
model = KerasClassifier(build_fn=create_model, verbose=0)
# define the grid search parameters
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.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))
```

Thanks

Jason Brownlee December 16, 2017 at 5:29 am # REPLY

Sorry to hear that, it’s not clear to me. Perhaps post to stackoverflow to get help debugging your code?

Olivier Blais December 16, 2017 at 3:24 am # REPLY

Hi Jason, first thanks for your articles! Super useful!

I tried to execute the gripsearch but cam up with parallelism issues. I have a Windows OS and I get this error when I try to run the script on multiple cpus:

ImportError: [joblib] Attempting to do parallel computing without protecting your import on a system that does not support forking. To use parallel computing in a script, you must protect your main loop using “if __name__ == ‘__main__'”. Please see the joblib docu

Do you know how I should address that?

Thanks in advance

Jason Brownlee December 16, 2017 at 5:35 am #

Perhaps try setting the number of jobs to 1?

Olivier Blais December 28, 2017 at 5:30 am #

Hi Jason! Yes this works but it is very slow as this is not parallel. Do you understand why it cannot run in parallel and how to fix that?

Thanks again !
Olivier

Jason Brownlee December 28, 2017 at 2:10 pm # REPLY

The backend is parallelized and the two levels of parallelization are in conflict.

Shabnam December 18, 2017 at 2:21 pm # REPLY

Thanks a lot for such a wonderful post. Overall, there are a lot of parameters that need to be tuned. I was thinking to use RandomizedSearchCV instead of GridSearchCV. Still, it will be time consuming for a lot of simulations. Do you have any suggestion for fast parameter tuning? For example, can we say that specific parameters have more effect on scores, so lets try to Grid/RandomizedSearchCV them first?

Jason Brownlee December 18, 2017 at 3:32 pm #

Yes, there are some great tips at the end of this post:
https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batc

Henry December 20, 2017 at 10:51 pm #

Dear Jason,

Fantastic post, thank you for this wonderful tutorial.

I was wondering if it would be more appropriate to tune all the hyperparameters at one go instead of breaking it up into various parts as shown above – you may be doing it for the sake of visibility of how each component is tuned but would it be better to tune everything together since there might be "interactions between the hyperparameters" which would not be captured if they were tuned separately?

Jason Brownlee December 21, 2017 at 5:25 am # REPLY

[redacted]

If you have the resources, then sure.

Hao January 3, 2018 at 2:26 am #

REPLY

Hi Jason,

Many thanks for a series of excellent posts!

I have an extremely imbalanced data set to study, of which #negative : #positive is about 100:1. When I built the first model, I performed 10-fold validation and in each validation round, I use oversampling to add positive samples on training data, but not on testing data. Now I question is: if I want to perform hyperparameter search, how do I tell GridSearchCV() to do oversampling for each round of cross-validation?

Many thanks

Jason Brownlee January 3, 2018 at 5:40 am #

Good question, you might need to use a Pipeline and have data prep happen within it.

Leave a Reply

[redacted]

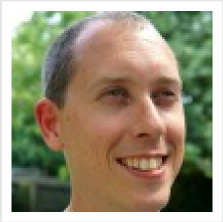
Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery



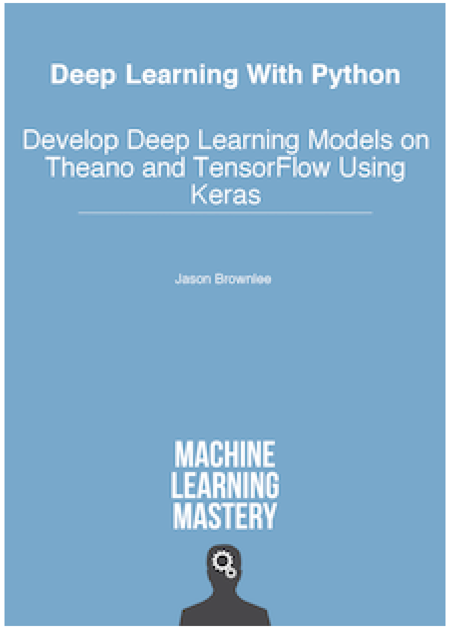
Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

Finally Get Started With Deep Learning

Sick of the fancy math and need for super computers?
Looking for step-by-step tutorials?
Want end-to-end projects?

[redacted]


Deep Learning With Python
Develop Deep Learning Models on Theano and TensorFlow Using Keras
Jason Brownlee
MACHINE LEARNING MASTERY

POPULAR

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

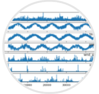
JULY 21, 2016

Your First Machine Learning Project in Python Step-By-Step

JUNE 10, 2016

Develop Your First Neural Network in Python With Keras Step-By-Step

MAY 24, 2016



Multivariate Time Series Forecasting with LSTMs in Keras

AUGUST 14, 2017

How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda

MARCH 13, 2017

[redacted]

Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras
JULY 26, 2016

Time Series Forecasting with the Long Short-Term Memory Network in Python
APRIL 7, 2017

Regression Tutorial with the Keras Deep Learning Library in Python
JUNE 9, 2016

Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras
AUGUST 9, 2016

Privacy | Contact | About