

[Start Here](#)[Blog](#)[Books](#)[About](#)[Contact](#)

Search...



Need help with Deep Learning? [Take the FREE Mini-Course](#)

Binary Classification Tutorial with the Keras Deep Learning Library

by **Jason Brownlee** on June 7, 2016 in **Deep Learning**



Keras is a Python library for deep learning that wraps the efficient numerical libraries TensorFlow and Theano.

Keras allows you to quickly and simply design and train neural network and deep learning models.

In this post you will discover how to effectively use the Keras library in your machine learning project by working through a binary classification project step-by-step.

After completing this tutorial, you will know:

- How to load training data and make it available to Keras.
- How to design and train a neural network for tabular data.
- How to evaluate the performance of a neural network model in Keras on unseen data.
- How to perform data preparation to improve model performance.
- How to tune the topology and configuration of the neural network.

Get Your Start in Machine Learning

Let's get started.

- **Update Oct/2016:** Updated examples for Keras 1.1.0 and scikit-learn v0.18.
- **Update Mar/2017:** Updated example for Keras 2.0.2, TensorFlow 1.0.1 and Theano 0.9.0.



Binary Classification Worked Example with the Keras Deep Learning Library
Photo by [Mattia Merlo](#), some rights reserved.

1. Description of the Dataset

The dataset we will use in this tutorial is the [Sonar dataset](#).

This is a dataset that describes sonar chirp returns bouncing off different services. The 60 input variables are the strength of the returns at different angles. It is a binary classification problem that requires a model to differentiate rocks from metal cylinders.

You can learn more about this dataset on the [UCI Machine Learning repository](#). You can [download the dataset](#) for free and place it in your working directory with the filename sonar.csv.

It is a well-understood dataset. All of the vari

Get Your Start in Machine Learning

f 0

to 1. The output variable is a string “M” for mine and “R” for rock, which will need to be converted to integers 1 and 0.

A benefit of using this dataset is that it is a standard benchmark problem. This means that we have some idea of the expected skill of a good model. Using cross-validation, a neural network [should be able to achieve performance](#) around 84% with an upper bound on accuracy for custom models at around 88%.

Need help with Deep Learning in Python?

Take my free 2-week email course and discover MLPs, CNNs and LSTMs (with sample code).

Click to sign-up now and also get a free PDF Ebook version of the course.

Start Your FREE Mini-Course Now!

2. Baseline Neural Network Model Performance

Let's create a baseline model and result for this problem.

We will start off by importing all of the classes and functions we will need.

```
1 import numpy
2 import pandas
3 from keras.models import Sequential
4 from keras.layers import Dense
5 from keras.wrappers.scikit_learn import KerasClassifier
6 from sklearn.model_selection import cross_val_score
7 from sklearn.preprocessing import LabelEncoder
8 from sklearn.model_selection import StratifiedKFold
9 from sklearn.preprocessing import StandardScaler
10 from sklearn.pipeline import Pipeline
```

Next, we can initialize the random number generator to ensure that we always get the same results when executing this code. This will help if we are debugging.

```
1 # fix random seed for reproducibility
2 seed = 7
3 numpy.random.seed(seed)
```

Now we can load the dataset using [pandas](#) :

Get Your Start in Machine Learning

id 1

output variable (Y). We use pandas to load the data because it easily handles strings (the output variable), whereas attempting to load the data directly using NumPy would be more difficult.

```
1 # load dataset
2 dataframe = pandas.read_csv("sonar.csv", header=None)
3 dataset = dataframe.values
4 # split into input (X) and output (Y) variables
5 X = dataset[:,0:60].astype(float)
6 Y = dataset[:,60]
```

The output variable is string values. We must convert them into integer values 0 and 1.

We can do this using the LabelEncoder class from scikit-learn. This class will model the encoding required using the entire dataset via the fit() function, then apply the encoding to create a new output variable using the transform() function.

```
1 # encode class values as integers
2 encoder = LabelEncoder()
3 encoder.fit(Y)
4 encoded_Y = encoder.transform(Y)
```

We are now ready to create our neural network model using Keras.

We are going to use scikit-learn to evaluate the model using stratified k-fold cross validation. This is a resampling technique that will provide an estimate of the performance of the model. It does this by splitting the data into k-parts, training the model on all parts except one which is held out as a test set to evaluate the performance of the model. This process is repeated k-times and the average score across all constructed models is used as a robust estimate of performance. It is stratified, meaning that it will look at the output values and attempt to balance the number of instances that belong to each class in the k-splits of the data.

To use Keras models with scikit-learn, we must use the KerasClassifier wrapper. This class takes a function that creates and returns our neural network model. It also takes arguments that it will pass along to the call to fit() such as the number of epochs and the batch size.

Let's start off by defining the function that creates our baseline model. Our model will have a single fully connected hidden layer with the same number of neurons as input variables. This is a good default starting point when creating neural networks.

The weights are initialized using a small Gaussian random number. The Rectifier activation function is used. The output layer contains a single neuron in order to make predictions. It uses the sigmoid activation function in order to produce a probability output in the range of 0 to 1 that can easily and automatically be converted to crisp class values.

Finally, we are using the logarithmic loss fun

Get Your Start in Machine Learning

preferred loss function for binary classification problems. The model also uses the efficient Adam optimization algorithm for gradient descent and accuracy metrics will be collected when the model is trained.

```
1 # baseline model
2 def create_baseline():
3     # create model
4     model = Sequential()
5     model.add(Dense(60, input_dim=60, kernel_initializer='normal', activation='relu'))
6     model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
7     # Compile model
8     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
9     return model
```

Now it is time to evaluate this model using stratified cross validation in the scikit-learn framework.

We pass the number of training epochs to the KerasClassifier, again using reasonable default values. Verbose output is also turned off given that the model will be created 10 times for the 10-fold cross validation being performed.

```
1 # evaluate model with standardized dataset
2 estimator = KerasClassifier(build_fn=create_baseline, nb_epoch=100, batch_size=5, verbose=0)
3 kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
4 results = cross_val_score(estimator, X, encoded_Y, cv=kfold)
5 print("Results: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Running this code produces the following output showing the mean and standard deviation of the estimated accuracy of the model on unseen data.

```
1 Baseline: 81.68% (7.26%)
```

This is an excellent score without doing any hard work.

3. Re-Run The Baseline Model With Data Preparation

It is a good practice to prepare your data before modeling.

Neural network models are especially suitable to having consistent input values, both in scale and distribution.

An effective data preparation scheme for tabular data when building neural network models is standardization. This is where the data is rescaled such that the mean value for each attribute is 0 and the standard deviation is 1. This preserves Gaussian and Gaussian-like distributions whilst normalizing the central tendencies for each attribute.

We can use scikit-learn to perform the stand

Get Your Start in Machine Learning

[StandardScaler](#) class.

Rather than performing the standardization on the entire dataset, it is good practice to train the standardization procedure on the training data within the pass of a cross-validation run and to use the trained standardization to prepare the “unseen” test fold. This makes standardization a step in model preparation in the cross-validation process and it prevents the algorithm having knowledge of “unseen” data during evaluation, knowledge that might be passed from the data preparation scheme like a crisper distribution.

We can achieve this in scikit-learn using a [Pipeline](#). The pipeline is a wrapper that executes one or more models within a pass of the cross-validation procedure. Here, we can define a pipeline with the StandardScaler followed by our neural network model.

```
1 # evaluate baseline model with standardized dataset
2 numpy.random.seed(seed)
3 estimators = []
4 estimators.append(('standardize', StandardScaler()))
5 estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, epochs=100, batch_s
6 pipeline = Pipeline(estimators)
7 kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
8 results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
9 print("Standardized: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Running this example provides the results below. We do see a small but very nice lift in the mean accuracy.

```
1 Standardized: 84.56% (5.74%)
```

4. Tuning Layers and Number of Neurons in The Model

There are many things to tune on a neural network, such as the weight initialization, activation functions, optimization procedure and so on.

One aspect that may have an outsized effect is the structure of the network itself called the network topology. In this section, we take a look at two experiments on the structure of the network: making it smaller and making it larger.

These are good experiments to perform when tuning a neural network on your problem.

4.1. Evaluate a Smaller Network

I suspect that there is a lot of redundancy in the input variables for this problem.

The data describes the same signal from dif

Get Your Start in Machine Learning

more relevant than others. We can force a type of feature extraction by the network by restricting the representational space in the first hidden layer.

In this experiment, we take our baseline model with 60 neurons in the hidden layer and reduce it by half to 30. This will put pressure on the network during training to pick out the most important structure in the input data to model.

We will also standardize the data as in the previous experiment with data preparation and try to take advantage of the small lift in performance.

```

1 # smaller model
2 def create_smaller():
3     # create model
4     model = Sequential()
5     model.add(Dense(30, input_dim=60, kernel_initializer='normal', activation='relu'))
6     model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
7     # Compile model
8     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
9     return model
10 estimators = []
11 estimators.append(('standardize', StandardScaler()))
12 estimators.append(('mlp', KerasClassifier(build_fn=create_smaller, epochs=100, batch_s
13 pipeline = Pipeline(estimators)
14 kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
15 results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
16 print("Smaller: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))

```

Running this example provides the following result. We can see that we have a very slight boost in the mean estimated accuracy and an important reduction in the standard deviation (average spread) of the accuracy scores for the model.

This is a great result because we are doing slightly better with a network half the size, which in turn takes half the time to train.

```

1 Smaller: 86.04% (4.00%)

```

4.2. Evaluate a Larger Network

A neural network topology with more layers offers more opportunity for the network to extract key features and recombine them in useful nonlinear ways.

We can evaluate whether adding more layers to the network improves the performance easily by making another small tweak to the function used to create our model. Here, we add one new layer (one line) to the network that introduces another hidden layer with 30 neurons after the first hidden layer.

Our network now has the topology:

Get Your Start in Machine Learning

```
1 60 inputs -> [60 -> 30] -> 1 output
```

The idea here is that the network is given the opportunity to model all input variables before being bottlenecked and forced to halve the representational capacity, much like we did in the experiment above with the smaller network.

Instead of squeezing the representation of the inputs themselves, we have an additional hidden layer to aid in the process.

```
1 # larger model
2 def create_larger():
3     # create model
4     model = Sequential()
5     model.add(Dense(60, input_dim=60, kernel_initializer='normal', activation='relu'))
6     model.add(Dense(30, kernel_initializer='normal', activation='relu'))
7     model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
8     # Compile model
9     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
10    return model
11 estimators = []
12 estimators.append(('standardize', StandardScaler()))
13 estimators.append(('mlp', KerasClassifier(build_fn=create_larger, epochs=100, batch_size=10)))
14 pipeline = Pipeline(estimators)
15 kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
16 results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
17 print("Larger: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Running this example produces the results below. We can see that we do not get a lift in the model performance. This may be statistical noise or a sign that further training is needed.

```
1 Larger: 83.14% (4.52%)
```

With further tuning of aspects like the optimization algorithm and the number of training epochs, it is expected that further improvements are possible. What is the best score that you can achieve on this dataset?

Summary

In this post, you discovered the Keras Deep Learning library in Python.

You learned how you can work through a binary classification problem step-by-step with Keras, specifically:

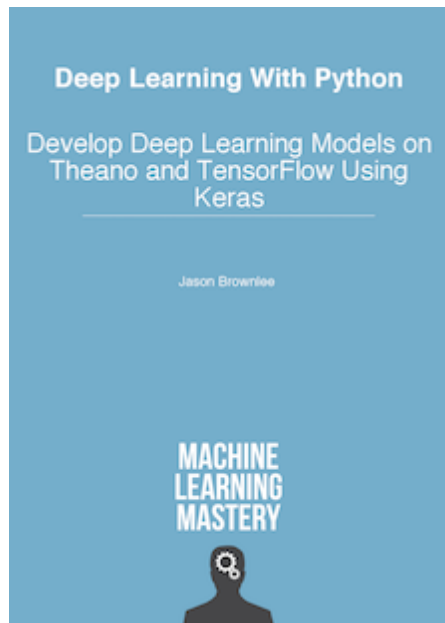
- How to load and prepare data for use in Keras.
- How to create a baseline neural network model.
- How to evaluate a Keras model using scikit-learn and stratified k-fold cross validation.
- How data preparation schemes can lift performance

Get Your Start in Machine Learning

- How experiments adjusting the network topology can lift model performance.

Do you have any questions about Deep Learning with Keras or about this post? Ask your questions in the comments and I will do my best to answer.

Frustrated With Your Progress In Deep Learning?



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. Just Results.

[Click to learn more.](#)



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](#) →

< Automate Machine Learning Workflows with Pipelines in Python and scikit-learn

Save

Get Your Start in Machine Learning

arn >

70 Responses to *Binary Classification Tutorial with the Keras Deep Learning Library*



Matt June 15, 2016 at 12:21 pm #

REPLY ↩

Excellent post with straightforward examples. Thanks for posting Jason!



Jason Brownlee June 15, 2016 at 1:41 pm #

REPLY ↩

You're very welcome Matt.



Shanky SHarma July 11, 2016 at 4:11 pm #

REPLY ↩

Hi Jason,

How can we use a test dataset here, I am new to machine Learning and so far I have only come across k-fold methods for accuracy measurements, but I'd like to predict on a test set, can you share an example of that.

Thank you.

Get Your Start in Machine Learning



Jason Brownlee July 12, 2016 at 5:24 am #

REPLY ↩

Hi Shanky,

There is an example of evaluating a neural network on a manual verification dataset while the model is being fit here:

<http://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/>

You can use the `model.evaluate()` function to evaluate your fit model on new data, there is an example at the end of this deep learning tutorial:

<http://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>

You can learn more about test options for evaluating machine learning algorithms here:

<http://machinelearningmastery.com/how-to-choose-the-right-test-options-when-evaluating-machine-learning-algorithms/>



Paul July 12, 2016 at 9:08 am #

REPLY ↩

Hi Jason,

After following this tutorial successfully I started playing with the model to learn more.

Eventually I got to the point where I added `model.predict` inside the baseline.

However when I print back the predicted Ys they are scaled. Is there a way to use standard scalar and then get your prediction back to binary?

Thanks



Jason Brownlee August 15, 2016 at 11:19 am #

REPLY ↩

Hi Paul, I would advise you to scale your data before hand and keep the coefficients used to scale, then reuse them later to reverse the scaling of predictions.



Cedric August 8, 2016 at 3:06 am #

REPLY ↩

Hi Jason,

great post! Very helpful introduction to binary classification in Keras

Get Your Start in Machine Learning

I was wondering, how would one print the progress of the model training the way Keras usually does in this example particularly?



Jason Brownlee August 8, 2016 at 5:49 am #

REPLY ↩

Thanks Cedric.

You can print progress with an epoch by setting `verbose=1` in the call to `model.fit()`. You can just see progress across epochs by setting `verbose=2` and turn off output with `verbose=0`.

Progress is turned off here because we are using k-fold cross validation which results in so many more models being created and in turn very noisy output.



Aakash Nain August 10, 2016 at 3:51 am #

REPLY ↩

Hello Jason,

Excellent tutorial. Consider a situation now. Suppose the data set loaded by you is the training set and the test set is given to you separately. I created the model as you described but now I want to predict the outcomes for test data and check the prediction score for the test data. How can I do that ?



Jason Brownlee August 15, 2016 at 11:18 am #

REPLY ↩

You can use `model.predict()` to make predictions and then compare the results to the known outcomes.

This post provides an example of what you want:

<http://machinelearningmastery.com/5-step-life-cycle-neural-network-models-keras/>

Get Your Start in Machine Learning



Sally October 26, 2016 at 4:14 am #

REPLY ↩

Dear Jason,

Thanks for this excellent tutorial , may I ask you regarding this network model; to which deep learning models does it belong? is it Deep Belief Network, CNN, stacked auto-encoder or other?

Thanks in advance



Jason Brownlee October 26, 2016 at 8:32 am #

REPLY ↩

It is a deep neural network.

Note that the DBN and autoencoders are generally no longer mainstream for classification problems like this example.

CNN are state of the art and used with image data.

I hope this helps.



Sally November 7, 2016 at 11:33 pm #

REPLY ↩

Thanks Jason for you reply, I have another question regarding this example. How can I know the reduced features after making the network smaller as in section 4.1. you have obliged the network to reduce the features in the hidden layer from 60 to 30. how can I know which features are chosen after this step? also can I know the weight that each feature got in participation in the classification process?



Jason Brownlee November 8, 2016 at 9:53 am #

REPLY ↩

Hi Sally,

The features are weighted, but the weighting is complex, because of the multiple layers. It would not be accurate to take just the input weights and use that to determine feature importance or which features are required.

The hidden layer neurons are not the same as the input features, I hope that is clear. Perhaps I misunderstand your question and you can elaborate what you mean?

Get Your Start in Machine Learning

Sally November 8, 2016 at 9:51 pm #

REPLY ↩

Hi Jason,

My case is as follows: I have something similar to your example. I have a deep Neural network with 11 features. I used a hidden layer to reduce the 11 features to 7 and then fed it to a binary classifier to classify the values to A class or B class. The first thing I need to know is that which 7 features of the 11 were chosen? can I have a way in the code to list them? the second thing I need to know is the average value for each feature in the case of classifying the record as class A or B. In more details; when feature 1 have an average value of 0.5 , feature 2 have average value of 0.2, feature 3 value of 0.3 ,,, etc. then the record is classified as class A. I need something like that; how can I have such value ?



Jason Brownlee November 9, 2016 at 9:51 am #

REPLY ↩

Hi Sally,

The number of nodes in a hidden layer is not a subset of the input features. They are an entirely new nonlinear recombination of input data. You cannot list out which features the nodes in a hidden layer relate to, because they are new features that relate to all input features. Does that make sense?



Sally November 9, 2016 at 6:20 pm #

REPLY ↩

Oh Yup!! I thought it is a kind of features selection that is done via the hidden layers!! so that if I need to make a feature selection I have to do it before creating the model. The second question that I did not get answer for it, is how can I measure the contribution of each feature at the prediction? in another words; how can I get the ”_features_importance_” . I tried to do it in the code but it is not applied to the “pipeline” model in line 16. Where can I use the function of “features_importance” to view each feature contribution in the prediction



Jason Brownlee November 10, 2016 at 7:39 am #

REPLY ↩

Hi Sally, you may be able to calculate feature importance using a neural net, I don't know. You may have to research this question yourself sorry.

Get Your Start in Machine Learning

Sally November 10, 2016 at 8:40 am #

REPLY ↩

I search it but unfortunately I did not get it 😞 .. Thanks for your cooperation



Sunil Manikani November 16, 2016 at 10:47 pm #

REPLY ↩

Hi Jason,
Excellent tutorial indeed!!!

While using PyDev in eclipse I ran into trouble with following imports ...

```
from keras.models import Sequential  
from keras.layers import Dense
```

I downloaded latest keras-master from git and did
sudo python setup.py install because my latest PIP install of keras gave me import errors. [Had to remove it.]

Hope it helps someone. My two cents, contributing to your excellent post.

Thanks a ton! Once again.

Warm regards,
Sunil M



Jason Brownlee November 17, 2016 at 9:53 am #

REPLY ↩

Thanks for the note Sunil.

I'm not an IDE user myself, command line all the way.



Sally January 5, 2017 at 12:47 pm #

REPLY ↩

Dear Jason,

I have another question regarding this example. As you know; deep learning performs well with large data-sets and mostly overfits with small data-sets. The dataset in this example have only 208 record, and the deep model achieved pretty good results. How can this meet the idea of deep learning with large datasets?

Get Your Start in Machine Learning



Jason Brownlee January 6, 2017 at 9:04 am #

REPLY ↩

Hi Sally,

Don't read too much into it. It is a demonstration of an MLP on a small binary classification problem.

MLPs scale. If the problem was sufficiently complex and we had 1000x more data, the model performance would continue to improve.



Sally January 7, 2017 at 8:12 am #

REPLY ↩

Thanks Jason for the reply, but could you please explain me how you find out that the data is 1000x ?? you have 208 record with 60 input value for each? did you multiply them to get this number? so that we can have the determine that a data is complex or not? another this could you help me by published articles that approve that MLP scale if the problem was complex?? Sorry for all these question but I am working on some thing relevant on my project and I need to prove and cite it



Jason Brownlee January 7, 2017 at 8:42 am #

REPLY ↩

Sorry, no, I meant if we had one thousand times the amount of data.



Sidharth Kumar February 3, 2017 at 12:01 am #

REPLY ↩

In multiple category classification like MNIST we have 10 outputs for everyone of 0 to 9. Why in binary classification we have only 1 output? We should have 2 outputs for each 0 and 1. Can you explain.



Jason Brownlee February 3, 2017 at 10:01 am #

REPLY ↩

Great question Sidharth.

We can use two output neurons for binary classification.

Alternatively, because there are only two

Get Your Start in Machine Learning

neuron with an activation function that outputs a binary response, like sigmoid or tanh.

They are generally equivalent, although the simpler approach is preferred as there are fewer weights to train.

Finally, you can one output neuron for a multi-class classification if you like and design a custom activation function or interpret a linear output value into the classes. This approach often does not capture sufficient complexity in the problem – e.g. like the network wanting to suggest an input may have potential membership in more than one class (a confusing input pattern) and it assumes an ordinal relationship between classes which is often invalid.



Pablo March 18, 2017 at 3:02 am #

REPLY ↩

I dont get it, how and where you do that.

Do you use 1 output node and if the sigmoid output is ≈ 0.5 is considered class B ??

Is that correct? Where in the code do you do that?



SYKim February 6, 2017 at 10:47 pm #

REPLY ↩

Hi Jason. Thanks. Your tutorials are really helpful!

I made a small network(2-2-1) which fits XOR function.

I found that without `numpy.random.seed(seed)` accuracy results can vary much.

Sometimes it learns quickly but in most cases its accuracy just remain near 0.25, 0.50, 0.75 etc...

So I needed to try several times to find some proper seed value which leads to high accuracy.

Is it common to try several times with the same model until it succeeds?

Also there was a case where it's trapped in the local optimum but after a long time it gets out of it and accuracy reach 1.0

What if there's a very big network and it takes 2~3 weeks to train it?

Do people just start training and start it again if there is not much improvement for some time?

Do people run the same model with different initialization values on different machines?

Is there any method to know if its accuracy will go up after a week?

Thank you!

Get Your Start in Machine Learning



Jason Brownlee February 7, 2017 at 10:16 am #

REPLY ↩

Great questions, see this post on randomness and machine learning:
<http://machinelearningmastery.com/randomness-in-machine-learning/>

I hope that helps as a start.



Mark February 8, 2017 at 4:26 pm #

REPLY ↩

Hi Jason. Thanks for the tutorial.

I want to implement autoencoder to do image similarity measurement. Cloud you please provide some tips/directions/suggestions to me how to figure this out ? Thanks



Jason Brownlee February 9, 2017 at 7:21 am #

REPLY ↩

Sorry, I do not have an example of using autoencoders.



Chan February 9, 2017 at 10:13 pm #

REPLY ↩

Hi Brownlee:

How would I save and load the model of KerasRegressor.

```
estimator = KerasRegressor(...)
```

I use estimator.model.save(), it works,
but it should call estimator.fit(X, Y) first, or it would throw "no model" error.

Besides, I have no idea about how to load the model to estimator.

It is easier to use normal model of Keras to save/load model, while using Keras wrapper of scikit_learn to save/load model is more difficult for me.

Would you please tell me how to do this.
Thanks a lot.



Jason Brownlee February 10, 2017 at 10:13 pm #

Get Your Start in Machine Learning

REPLY ↩



Hi Chan, you could try pickle?

I find it easier to use KerasClassifier to explore models and tuning, and then using native Keras with save/load for larger models and finalizing the model.



Emerson February 10, 2017 at 4:23 pm #

REPLY ↩

Awesome tutorial, one of the first I've been able to follow the entire way through.

I would love to see a tiny code snippet that uses this model to make an actual prediction. I figured it would be as easy as using `estimator.predict(X[0])`, but I'm getting errors about the shape of my data being incorrect (None, 60) vs (60, 1).



Jason Brownlee February 11, 2017 at 4:54 am #

REPLY ↩

I'm glad to hear it Emerson.

Yes, you can make a prediction with:

```
1 yhat = model.predict(X)
```

You may need to reshape your data into a 2D array:

```
1 data = data.reshape(60, 1)
```



Carlos Castellanos July 3, 2017 at 7:24 am #

REPLY ↩

Hi Jason, such an amazing post, congrats! I have some doubts regarding Emerson's question and your answer.

I want to separate cross-validation and prediction in different stages basically because they are executed in different moments, for that I will receive to receive a non-standardized input vector X with a single sample to predict. I was able to save the model using callbacks so it can be reused to predict but I'm a bit lost on how to standardize the input vector without loading the entire dataset before predicting, I was trying to pickle the pipeline state but nothing good came from that road, is this possible? do you have any example on how to do it? Thanks!

Get Your Start in Machine Learning

Jason Brownlee July 6, 2017 at 9:56 am #

REPLY ↩



To standardize all you need is the mean and standard deviation of the training data for each variable.



Chris Cummins February 24, 2017 at 4:47 am #

REPLY ↩

Fantastic tutorial Jason, thank you. Here's my Jupyter notebook of it: <https://github.com/ChrisCummins/phd/blob/master/learn/keras/Sonar.ipynb>



Jason Brownlee February 24, 2017 at 10:12 am #

REPLY ↩

Nice work Chris.



Dmitri Levitin March 24, 2017 at 7:46 am #

REPLY ↩

I have a difficult question. I have google weekly search trends data for NASDAQ companies, over 2 year span, and I'm trying to classify if the stock goes up or down after the earnings based on the search trends, which leads to 104 weeks or features. I ran this data and received no signal Results: 48.55% (4.48%).

However, in my non machine learning experiments i see signal. If i take the diffs (week n – week $n+1$), creating an array of 103 diffs. I then average out all the stocks that went up and average out all the stocks that went down. When i predict a new stock for the same 2 year time period, I compare in a voting like manner week n of new stock to week n of stocks labeled up, and labeled down. Whoever has more votes wins. In this simple method i do see signal.

Thoughts?



Jason Brownlee March 24, 2017 at 8:02 am #

REPLY ↩

Short term movements on the stock market are a random walk. The best you can do is a persistence forecast as far as I know.

Get Your Start in Machine Learning

Dmitri Levitin March 24, 2017 at 8:15 am #

REPLY ↩



But I'm not comparing movements of the stock, but its tendency to have an upward day or downward day after earnings, as the labeled data, and the google weekly search trends over the 2 year span becoming essentially the inputs for the neural network. So then it becomes a classification problem.

As described above in the 2nd paragraph i see signal, based on taking the average of the weeks that go up after earnings vs ones that go down, and comparing the new week to those 2 averages. I'm just not sure how to interpret that into a neural network.

BTW, awesome tutorial, i will follow all of your tutorials.



Dmitri Levitin March 24, 2017 at 8:19 am #

REPLY ↩

I meant to say i take the average of each week for all the labeled companies that go up after earnings creating an array of averages, and same for the companies that go down after earnings. I then compare the weeks of the new stock, over the same time period to each of the prior arrays. An i do see signal, but how to make that work with neural networks.



Dmitri Levitin March 24, 2017 at 10:31 am #

REPLY ↩

Another question. Using this methodology but with a different set of data I'm getting accuracy improvement with each epoch run. But in the end i get Results: 52.64% (15.74%). Any idea why? I thought results were related to the average accuracy.

Epoch 1/10

0s – loss: 1.1388 – acc: 0.5130

Epoch 2/10

0s – loss: 0.6415 – acc: 0.6269

Epoch 3/10

0s – loss: 0.4489 – acc: 0.7565

Epoch 4/10

0s – loss: 0.3568 – acc: 0.8446

Epoch 5/10

0s – loss: 0.3007 – acc: 0.8808

Epoch 6/10

0s – loss: 0.2611 – acc: 0.9326

Epoch 7/10

Get Your Start in Machine Learning

0s – loss: 0.2260 – acc: 0.9430

Epoch 8/10

0s – loss: 0.1987 – acc: 0.9689

Epoch 9/10

0s – loss: 0.1771 – acc: 0.9741

Epoch 10/10

0s – loss: 0.1556 – acc: 0.9741

Results: 52.64% (15.74%)



Jason Brownlee March 25, 2017 at 7:30 am #

REPLY ↩

Perhaps the model is overfitting the training data?

Consider slowing down learning with some regularization methods like dropout.



Michael April 21, 2017 at 6:05 am #

REPLY ↩

Hi Jason,

Can this type of classifier (which described in this tutorial) can be used for ordinal classification (with binary classification)?

Thanks,



Jason Brownlee April 21, 2017 at 8:43 am #

REPLY ↩

I would use the network as is or phrase the problem as a regression problem and round results.

Get Your Start in Machine Learning



Ahmad May 18, 2017 at 8:12 pm #

REPLY ↩

Hello Jason,

How can I save the pipelined model?

I mean in the past it was easy when we only implemented a model and we fit it ...
but now how can I save this in order to load it and make predictions later on?



Jason Brownlee May 19, 2017 at 8:17 am #

REPLY ↩

I believe you cannot save the pipelined model.

You must use the Keras API alone to save models to disk. At least as far as I know.



Rob June 20, 2017 at 8:03 am #

REPLY ↩

Thanks a lot for this great post! I am trying to learn more about machine learning and your blog has been a huge help.

Any idea why I would be getting very different results if I train the model without k-fold cross validation? e.g. If I run

```
model = create_baseline()
model.fit(X, encoded_Y, epochs=100, batch_size=5, validation_split=0.3)
```

It outputs a val_acc of around 0.38. But if I run your code using k-fold I am getting an accuracy of around 75%

Full code snippet is here <https://gist.github.com/robiamcd/e94b4d393346b2d62f9ca2fcec1cfd>

Any idea why this might be happening?

Get Your Start in Machine Learning



Jason Brownlee June 21, 2017 at 8:07 am #

REPLY ↩

Hi Rob, yes neural networks are stochastic. See this post:

<http://machinelearningmastery.com/randomness-in-machine-learning/>

See here for how to get a more robust estimate of neural network model skill:

<http://machinelearningmastery.com/evaluate-skill-deep-learning-models/>



Rob June 22, 2017 at 7:15 am #

REPLY ↩

I ran it many times and I was consistently getting around 75% accuracy with k-fold and 35% without it. Turns out I wasn't shuffling the array when I wasn't using k-fold so the validation target set was almost all 1s and the training set was mostly 0s. I added `numpy.random.shuffle(dataset)` and it's all good now.



Jason Brownlee June 23, 2017 at 6:37 am #

REPLY ↩

I'm glad to hear you got to the bottom of it Rob!



Michael June 21, 2017 at 11:33 pm #

REPLY ↩

Hi Jason,

In this post you mentioned the ability of hidden layers with less neurons than the number of neurons in the previous layers to extract key features.

Is it possible to visualize or get list of these selected key features in Keras? (For exmaple, for networks with high number of features)?

Thanks,
Michael



Jason Brownlee June 22, 2017 at 6:06 am #

REPLY ↩

You may, I am not aware if an example sorry.

Get Your Start in Machine Learning

joseph June 25, 2017 at 6:45 pm #

REPLY ↩

Hi I would love to see object location / segmentation network for identifying object locations and labeling them.



Jason Brownlee June 26, 2017 at 6:07 am #

REPLY ↩

Thanks for the suggestion joseph.



Parth July 19, 2017 at 1:58 am #

REPLY ↩

Hi Jason, how do we know which structure is best for a neural network? Any resources you could point me to?(I don't mind going through the math)



Jason Brownlee July 19, 2017 at 8:27 am #

REPLY ↩

Nope. There is no good theory for this.

Copy other designs, use trial and error. Design robust experiments to test many structures.



Fan Feng August 5, 2017 at 7:43 pm #

REPLY ↩

Thanks for your sharing.



Jason Brownlee August 6, 2017 at 7:37 am #

REPLY ↩

I'm glad it helped!



Alex Mikhalev August 28, 2017 at 4:37 am #

REPLY ↩

Thank you for sharing, but it needs

Get Your Start in Machine Learning

see <http://www.cloudypoint.com/Tutorials/discussion/python-solved-can-i-send-callbacks-to-a-kerasclassifier/>



Jason Brownlee August 28, 2017 at 6:51 am #

REPLY ↩

Glad to hear it.



Valentin September 19, 2017 at 7:15 pm #

REPLY ↩

Hi Jason! Thanks so much for this very concise and easy to follow tutorial! One question: if you call native Keras `model.fit(X,y)` you can also supply `validation_data`, such that validation score is printed during training (if `verbose=1`). Do you know how to switch this feature on in the pipeline? `sklearn` creates the split automatically within the `cross_val_score` step, but how to pass this on to the Keras fit method...?

Thanks a lot!



Jason Brownlee September 20, 2017 at 5:55 am #

REPLY ↩

No and I would not recommend it. I think it would cause more problems.



B G SINGH September 20, 2017 at 3:38 am #

REPLY ↩

Hi Jason,

Is there any way to use `class_weight` parameter in this code?

Thanks,
Biswa



Jason Brownlee September 20, 2017 at 6:02 am #

REPLY ↩

Yes, set `class_weight` in the `fit()` function.

More help here:

Get Your Start in Machine Learning

<https://keras.io/models/sequential/>**Luis Ernesto** October 19, 2017 at 2:53 pm #

REPLY ↩

Good day interesting article. I am currently doing an investigation, it is a comparative study of three types of artificial neural network algorithms: multilayer perceptron, radial and recurrent neural networks. Well I already work the algorithms and I'm in training time, everything is fine until I start this stage unfortunately I can not generalize the network, and try changing parameters such as learning reason and number of iterations, but the result remains the same. The input data (dataset) that input are binary ie a pattern for example has (1,0,0,1,1,0,0,1,0,1,1,1) the last indicator being the desired output , I also noticed that when the weights converge and I use them in the validation stage, all the results are almost the same is as if there would be no difference in the patterns. Well now I am doing cross validation hoping to solve this problem or to realize what my error may be. I would appreciate your help or advice

**Jason Brownlee** October 19, 2017 at 4:03 pm #

REPLY ↩

Generally, I would recommend this process for evaluating your model:

<https://machinelearningmastery.com/evaluate-skill-deep-learning-models/>

Perhaps you can calculate some diagnostics like learning rate on a training and validation datasets?

Leave a Reply

Name (required)

[Get Your Start in Machine Learning](#)

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](#)

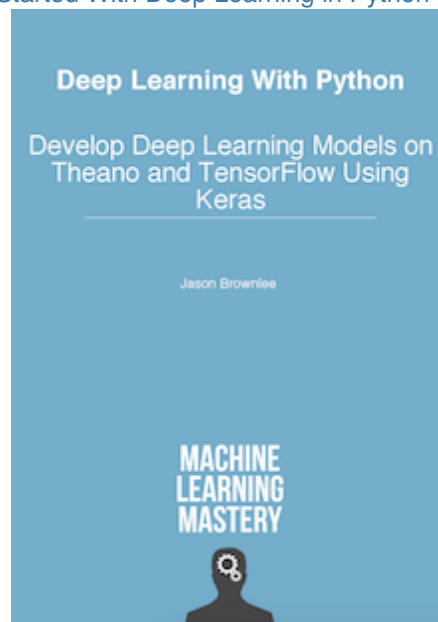
Finally Get Started With Deep Learning

Sick of the fancy math and need for super computers?











Looking for step-by-step tutorials?

Want end-to-end projects?

[Get Started With Deep Learning in Python Today!](#)



Get Your Start in Machine Learning

POPULAR	
	<div>Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras</div> <div>JULY 21, 2016</div>
	<div>Your First Machine Learning Project in Python Step-By-Step</div> <div>JUNE 10, 2016</div>
	<div>Develop Your First Neural Network in Python With Keras Step-By-Step</div> <div>MAY 24, 2016</div>
	<div>How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda</div> <div>MARCH 13, 2017</div>
	<div>Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras</div> <div>JULY 26, 2016</div>
	<div>Time Series Forecasting with the Long Short-Term Memory Network in Python</div> <div>APRIL 7, 2017</div>
	<div>Multivariate Time Series Forecasting with LSTMs in Keras</div> <div>AUGUST 14, 2017</div>
	<div>Multi-Class Classification Tutorial with the Keras Deep Learning Library</div> <div>JUNE 2, 2016</div>
	<div>Regression Tutorial with the Keras Deep Learning Library in Python</div> <div>JUNE 9, 2016</div>
	<div>How to Implement the Backpropagation Algorithm From Scratch In Python</div> <div>NOVEMBER 7, 2016</div>

Get Your Start in Machine Learning

© 2017 Machine Learning Mastery. All Rights Reserved.

[Privacy](#) | [Contact](#) | [About](#)

Get Your Start in Machine Learning