Neural Network Models for Combined Classification and Regression

by **Jason Brownlee** on April 5, 2021 in **Deep Learning**

Some prediction problems require predicting both numeric values and a class label for the same input.

A simple approach is to develop both regression and classification predictive models on the same data and use the models sequentially.

An alternative and often more effective approach is to develop a single neural network model that can predict both a numeric and class label value from the same input. This is called a **multi-output model** and can be relatively easy to develop and evaluate using modern deep learning libraries such as Keras and TensorFlow.

In this tutorial, you will discover how to develop a neural network for combined regression and classification predictions.

After completing this tutorial, you will know:

- Some prediction problems require predicting both numeric and class label values for each input example.
- How to develop separate regression and classification models for problems that require multiple outputs.
- How to develop and evaluate a neural network model capable of making simultaneous regression and classification predictions.

Let's get started.

Develop Neural Network for Combined Classification and Regression
Photo by Sang Trinh, some rights reserved.

Tutorial Overview

This tutorial is divided into three parts; they are:

1. Single Model for Regression and Classification
2. Separate Regression and Classification Models
    1. Abalone Dataset
    2. Regression Model
    3. Classification Model
3. Combined Regression and Classification Models

Single Model for Regression and Classification

It is common to develop a deep learning neural network model for a regression or classification problem, but on some predictive modeling tasks, we may want to develop a single model that can make both regression and classification predictions.

Regression refers to predictive modeling problems that involve predicting a numeric value given an input.

Classification refers to predictive modeling problems that involve predicting a class label or probability of class labels for a given input.

For more on the difference between classification and regression, see the tutorial:

- Difference Between Classification and Regression in Machine Learning

There may be some problems where we want to predict both a numerical value and a classification value.

One approach to solving this problem is to develop a separate model for each prediction that is required.

The problem with this approach is that the predictions made by the separate models may diverge.

An alternate approach that can be used when using neural network models is to develop a single model capable of making separate predictions for a numeric and class output for the same input.

This is called a multi-output neural network model.

The benefit of this type of model is that we have a single model to develop and maintain instead of two models and that training and updating the model on both output types at the same time may offer more consistency in the predictions between the two output types.

We will develop a multi-output neural network model capable of making regression and classification predictions at the same time.

First, let's select a dataset where this requirement makes sense and start by developing separate models for both regression and classification predictions.

Separate Regression and Classification Models

In this section, we will start by selecting a real dataset where we may want regression and classification predictions at the same time, then develop separate models for each type of prediction.

Abalone Dataset

We will use the "*abalone*" dataset.

Determining the age of an abalone is a time-consuming task and it is desirable to determine the age from physical details alone.

Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

This is a dataset that describes the physical details of abalone and requires predicting the number of rings of the abalone, which is a proxy for the age of the creature.

You can learn more about the dataset from here:

- Dataset (abalone.csv)
- Dataset Details (abalone.names)

The "*age*" can be predicted as both a numerical value (in years) or a class label (ordinal year as a class).

No need to download the dataset as we will download it automatically as part of the worked examples.

The dataset provides an example of a dataset where we may want both a numerical and classification of an input.

First, let's develop an example to download and summarize the dataset.

```
# load and summarize the abalone dataset
from pandas import read_csv
from matplotlib import pyplot
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
# summarize shape
print(dataframe.shape)
# summarize first few lines
print(dataframe.head())
```

Running the example first downloads and summarizes the shape of the dataset.

We can see that there are 4,177 examples (rows) that we can use to train and evaluate a model and 9 features (columns) including the target variable.

We can see that all input variables are numeric except the first, which is a string value.

To keep data preparation simple, we will drop the first column from our models and focus on modeling the numeric input values.

```
(4177, 9)
   0      1      2      3       4       5       6      7   8
0  M  0.455  0.365  0.095  0.5140  0.2245  0.1010  0.150  15
1  M  0.350  0.265  0.090  0.2255  0.0995  0.0485  0.070   7
2  F  0.530  0.420  0.135  0.6770  0.2565  0.1415  0.210   9
3  M  0.440  0.365  0.125  0.5160  0.2155  0.1140  0.155  10
4  I  0.330  0.255  0.080  0.2050  0.0895  0.0395  0.055   7
```

We can use the data as the basis for developing separate regression and classification Multilayer Perceptron (MLP) neural network models.

Note: we are not trying to develop an optimal model for this dataset; instead we are demonstrating a specific technique: developing a model that can make both regression and classification predictions.

Regression Model

In this section, we will develop a regression MLP model for the abalone dataset.

First, we must separate the columns into input and output elements and drop the first column that contains string values.

We will also force all loaded columns to have a float type (expected by neural network models) and record the number of input features, which will need to be known by the model later.

```
...
# split into input (X) and output (y) variables
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]
```

Next, we can split the dataset into a train and test dataset.

We will use a 67% random sample to train the model and the remaining 33% to evaluate the model.

```
...
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st
```

We can then define an MLP neural network model.

The model will have two hidden layers, the first with 20 nodes and the second with 10 nodes, both using ReLU activation and “*he normal*” weight initialization (a good practice). The number of layers and nodes were chosen arbitrarily.

The output layer will have a single node for predicting a numeric value and a linear activation function.

```
...
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='he
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='linear'))
```

The model will be trained to minimize the mean squared error (MSE) loss function using the effective Adam version of stochastic gradient descent.

```
...
# compile the keras model
model.compile(loss='mse', optimizer='adam')
```

We will train the model for 150 epochs with a mini-batch size of 32 samples, again chosen arbitrarily.

```
...
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=2)
```

Finally, after the model is trained, we will evaluate it on the holdout test dataset and report the mean absolute error (MAE).

```
...
# evaluate on test set
yhat = model.predict(X_test)
error = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % error)
```

Tying this all together, the complete example of an MLP neural network for the abalone dataset framed as a regression problem is listed below.

```
# regression mlp model for the abalone dataset
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
dataset = dataframe.values
# split into input (X) and output (y) variables
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_s
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='h
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='linear'))
# compile the keras model
model.compile(loss='mse', optimizer='adam')
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=2)
# evaluate on test set
yhat = model.predict(X_test)
error = mean_absolute_error(y_test, yhat)
print('MAE: %.3f' % error)
```

Running the example will prepare the dataset, fit the model, and report an estimate of model error.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the model achieved an error of about 1.5 (rings).

```
...
Epoch 145/150
88/88 - 0s - loss: 4.6130
Epoch 146/150
88/88 - 0s - loss: 4.6182
Epoch 147/150
88/88 - 0s - loss: 4.6277
Epoch 148/150
88/88 - 0s - loss: 4.6437
Epoch 149/150
88/88 - 0s - loss: 4.6166
Epoch 150/150
88/88 - 0s - loss: 4.6132
MAE: 1.554
```

So far so good.

Next, let's look at developing a similar model for classification.

Classification Model

The abalone dataset can be framed as a classification problem where each "*ring*" integer is taken as a separate class label.

The example and model are much the same as the above example for regression, with a few important changes.

This requires first assigning a separate integer for each "*ring*" value, starting at 0 and ending at the total number of "*classes*" minus one.

This can be achieved using the LabelEncoder.

We can also record the total number of classes as the total number of unique encoded class values, which will be needed by the model later.

```
...
# encode strings to integer
y = LabelEncoder().fit_transform(y)
n_class = len(unique(y))
```

After splitting the data into train and test sets as before, we can define the model and change the number of outputs from the model to equal the number of classes and use the softmax activation function, common for multi-class classification.

```
...
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='he
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(n_class, activation='softmax'))
```

Given we have encoded class labels as integer values, we can fit the model by minimizing the sparse categorical cross-entropy loss function, appropriate for multi-class classification tasks with integer encoded class labels.

```
...
# compile the keras model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
```

After the model is fit on the training dataset as before, we can evaluate the performance of the model by calculating the classification accuracy on the hold-out test set.

```
...
# evaluate on test set
yhat = model.predict(X_test)
yhat = argmax(yhat, axis=-1).astype('int')
acc = accuracy_score(y_test, yhat)
print('Accuracy: %.3f' % acc)
```

Tying this all together, the complete example of an MLP neural network for the abalone dataset framed as a classification problem is listed below.

```
# classification mlp model for the abalone dataset
from numpy import unique
from numpy import argmax
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
# load dataset
```

```
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
dataset = dataframe.values
# split into input (X) and output (y) variables
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]
# encode strings to integer
y = LabelEncoder().fit_transform(y)
n_class = len(unique(y))
# split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_s
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=n_features, activation='relu', kernel_initializer='h
model.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(n_class, activation='softmax'))
# compile the keras model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam')
# fit the keras model on the dataset
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=2)
# evaluate on test set
yhat = model.predict(X_test)
yhat = argmax(yhat, axis=-1).astype('int')
acc = accuracy_score(y_test, yhat)
print('Accuracy: %.3f' % acc)
```

Running the example will prepare the dataset, fit the model, and report an estimate of model error.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the model achieved an accuracy of about 27%.

```
...
Epoch 145/150
88/88 - 0s - loss: 1.9271
Epoch 146/150
88/88 - 0s - loss: 1.9265
Epoch 147/150
88/88 - 0s - loss: 1.9265
Epoch 148/150
88/88 - 0s - loss: 1.9271
Epoch 149/150
88/88 - 0s - loss: 1.9262
Epoch 150/150
88/88 - 0s - loss: 1.9260
Accuracy: 0.274
```

So far so good.

Next, let's look at developing a combined model capable of both regression and classification predictions.

Combined Regression and Classification Models

In this section, we can develop a single MLP neural network model that can make both regression and classification predictions for a single input.

This is called a multi-output model and can be developed using the functional Keras API.

For more on this functional API, which can be tricky for beginners, see the tutorials:

- TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras
- How to Use the Keras Functional API for Deep Learning

First, the dataset must be prepared.

We can prepare the dataset as we did before for classification, although we should save the encoded target variable with a separate name to differentiate it from the raw target variable values.

```
...
# encode strings to integer
y_class = LabelEncoder().fit_transform(y)
n_class = len(unique(y_class))
```

We can then split the input, raw output, and encoded output variables into train and test sets.

```
...
# split data into train and test sets
```

Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep

```
X_train, X_test, y_train, y_test, y_train_class, y_test_class = train_test_split(X,
```

Next, we can define the model using the functional API.

The model takes the same number of inputs as before with the standalone models and uses two hidden layers configured in the same way.

```
...
# input
visible = Input(shape=(n_features,))
hidden1 = Dense(20, activation='relu', kernel_initializer='he_normal')(visible)
hidden2 = Dense(10, activation='relu', kernel_initializer='he_normal')(hidden1)
```

We can then define two separate output layers that connect to the second hidden layer of the model.

The first is a regression output layer that has a single node and a linear activation function.

```
...
# regression output
out_reg = Dense(1, activation='linear')(hidden2)
```

The second is a classification output layer that has one node for each class being predicted and uses a softmax activation function.

```
...
# classification output
out_clas = Dense(n_class, activation='softmax')(hidden2)
```

We can then define the model with a single input layer and two output layers.

```
...
# define model
model = Model(inputs=visible, outputs=[out_reg, out_clas])
```

Given the two output layers, we can compile the model with two loss functions, mean squared error loss for the first (regression) output layer and sparse categorical cross-entropy for the second (classification) output layer.

```
...
# compile the keras model
model.compile(loss=['mse','sparse_categorical_crossentropy'], optimizer='adam')
```

We can also create a plot of the model for reference.

This requires that pydot and pygraphviz are installed. If this is a problem, you can comment out this line and the import statement for the *plot_model()* function.

```
...
# plot graph of model
plot_model(model, to_file='model.png', show_shapes=True)
```

Each time the model makes a prediction, it will predict two values.

Similarly, when training the model, it will need one target variable per sample for each output.

As such, we can train the model, carefully providing both the regression target and classification target data to each output of the model.

```
...
# fit the keras model on the dataset
model.fit(X_train, [y_train,y_train_class], epochs=150, batch_size=32, verbose=2)
```

The fit model can then make a regression and classification prediction for each example in the hold-out test set.

```
...
# make predictions on test set
yhat1, yhat2 = model.predict(X_test)
```

The first array can be used to evaluate the regression predictions via mean absolute error.

```
...
# calculate error for regression model
error = mean_absolute_error(y_test, yhat1)
print('MAE: %.3f' % error)
```

The second array can be used to evaluate the classification predictions via classification accuracy.

```
...
# evaluate accuracy for classification model
```

```
yhat2 = argmax(yhat2, axis=-1).astype('int')
acc = accuracy_score(y_test_class, yhat2)
print('Accuracy: %.3f' % acc)
```

And that's it.

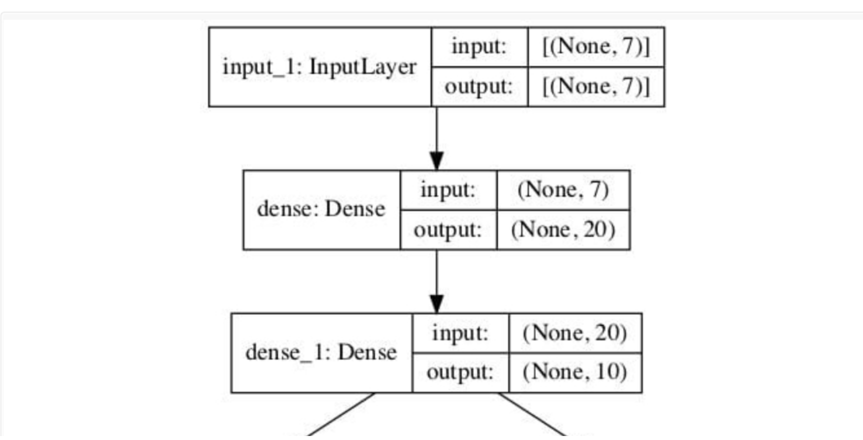
Tying this together, the complete example of training and evaluating a multi-output model for combiner regression and classification predictions on the abalone dataset is listed below.

```
# mlp for combined regression and classification predictions on the abalone datase
from numpy import unique
from numpy import argmax
from pandas import read_csv
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import plot_model
# load dataset
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/abalone.csv'
dataframe = read_csv(url, header=None)
dataset = dataframe.values
# split into input (X) and output (y) variables
X, y = dataset[:, 1:-1], dataset[:, -1]
X, y = X.astype('float'), y.astype('float')
n_features = X.shape[1]
# encode strings to integer
y_class = LabelEncoder().fit_transform(y)
n_class = len(unique(y_class))
# split data into train and test sets
X_train, X_test, y_train, y_test, y_train_class, y_test_class = train_test_split(X
# input
visible = Input(shape=(n_features,))
hidden1 = Dense(20, activation='relu', kernel_initializer='he_normal')(visible)
hidden2 = Dense(10, activation='relu', kernel_initializer='he_normal')(hidden1)
# regression output
out_reg = Dense(1, activation='linear')(hidden2)
# classification output
out_clas = Dense(n_class, activation='softmax')(hidden2)
# define model
model = Model(inputs=visible, outputs=[out_reg, out_clas])
# compile the keras model
model.compile(loss=['mse','sparse_categorical_crossentropy'], optimizer='adam')
# plot graph of model
plot_model(model, to_file='model.png', show_shapes=True)
# fit the keras model on the dataset
model.fit(X_train, [y_train,y_train_class], epochs=150, batch_size=32, verbose=2)
# make predictions on test set
yhat1, yhat2 = model.predict(X_test)
# calculate error for regression model
error = mean_absolute_error(y_test, yhat1)
print('MAE: %.3f' % error)
# evaluate accuracy for classification model
yhat2 = argmax(yhat2, axis=-1).astype('int')
acc = accuracy_score(y_test_class, yhat2)
print('Accuracy: %.3f' % acc)
```

Running the example will prepare the dataset, fit the model, and report an estimate of model error.

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

A plot of the multi-output model is created, clearly showing the regression (left) and classification (right) output layers connected to the second hidden layer of the model.



Deep Learning Models in Python With Keras

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

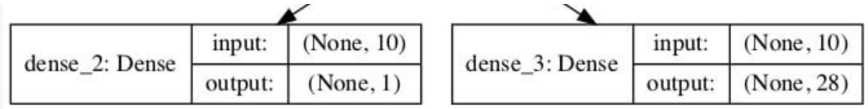
Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep



Plot of the Multi-Output Model for Combine Regression and Classification Predictions

In this case, we can see that the model achieved both a reasonable error of about 1.495 (rings) and a similar accuracy as before of about 25.6%.

```
...
Epoch 145/150
88/88 - 0s - loss: 6.5707 - dense_2_loss: 4.5396 - dense_3_loss: 2.0311
Epoch 146/150
88/88 - 0s - loss: 6.5753 - dense_2_loss: 4.5466 - dense_3_loss: 2.0287
Epoch 147/150
88/88 - 0s - loss: 6.5970 - dense_2_loss: 4.5723 - dense_3_loss: 2.0247
Epoch 148/150
88/88 - 0s - loss: 6.5640 - dense_2_loss: 4.5389 - dense_3_loss: 2.0251
Epoch 149/150
88/88 - 0s - loss: 6.6053 - dense_2_loss: 4.5827 - dense_3_loss: 2.0226
Epoch 150/150
88/88 - 0s - loss: 6.5754 - dense_2_loss: 4.5524 - dense_3_loss: 2.0230
MAE: 1.495
Accuracy: 0.256
```

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Tutorials

- Difference Between Classification and Regression in Machine Learning
- TensorFlow 2 Tutorial: Get Started in Deep Learning With tf.keras
- Best Results for Standard Machine Learning Datasets
- How to Use the Keras Functional API for Deep Learning

Summary

In this tutorial, you discovered how to develop a neural network for combined regression and classification predictions.

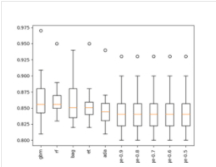
Specifically, you learned:

- Some prediction problems require predicting both numeric and class label values for each input example.
- How to develop separate regression and classification models for problems that require multiple outputs.
- How to develop and evaluate a neural network model capable of making simultaneous regression and classification predictions.

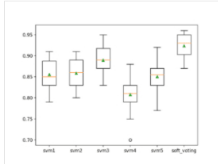
Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

Tweet Tweet Share Share

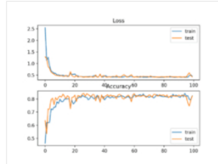
More On This Topic



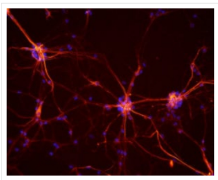
How to Develop a Framework to Spot-Check Machine…



How to Develop Voting Ensembles With Python

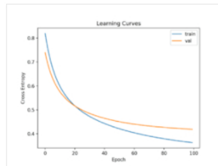


How to Choose Loss Functions When Training Deep…



How to Code a Neural Network with Backpropagation In…

Blending Ensemble Machine Learning With Python



TensorFlow 2 Tutorial: Get Started in Deep Learning…

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

< Iterated Local Search From Scratch in Python

Develop a Neural Network for Cancer Survival Dataset >

35 Responses to *Neural Network Models for Combined Classification and Regression*

Ravi Mariwalla April 6, 2021 at 11:22 pm # REPLY

Hi, Are there examples from healthcare where neural networks can solve the problem of both regression and classification?

Jason Brownlee April 7, 2021 at 5:10 am # REPLY

Sure. You can search scholar.google.com

Rajan Prasad April 15, 2021 at 7:29 pm # REPLY

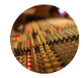
hi jason
can you explain one example of classification model with microarray gene expression dataset

Jason Brownlee April 16, 2021 at 5:29 am # REPLY

Thanks for the suggestion.


Keras Deep Learning Library

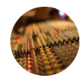

How to Save and Load Your Keras Deep Learning Model


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


Regression Tutorial with the Keras Deep Learning Library in Python


Multi-Class Classification Tutorial with the Keras Deep Learning Library

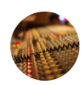

How to Save and Load Your Keras Deep Learning Model


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


Regression Tutorial with the Keras Deep Learning Library in Python


Multi-Class Classification Tutorial with the Keras Deep Learning Library

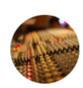

How to Save and Load Your Keras Deep Learning Model


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


Regression Tutorial with the Keras Deep Learning Library in Python


Multi-Class Classification Tutorial with the Keras Deep Learning Library


How to Save and Load Your Keras Deep Learning Model


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


Regression Tutorial with the Keras Deep Learning Library in Python


Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep

Sepideh April 21, 2021 at 5:53 am #

REPLY

Hi Jason,

this model can be performed on data with multi-outputs?

Thanks

Jason Brownlee April 21, 2021 at 5:58 am #

REPLY

Perhaps this would be a more appropriate model:
https://machinelearningmastery.com/multi-output-regression-models-with-python/

sepideh radhoush April 24, 2021 at 2:06 am #

REPLY

Thanks,

I will check.

Jason Brownlee April 24, 2021 at 5:22 am #

REPLY

You're welcome.

Tom April 26, 2021 at 10:37 am #

REPLY

Hi Jason,
Would it be possible to use the Sequential method for the "Combined Regression and Classification Models" ? (Like in the Regression and Classification Model examples).

y_values could be a numpy array array([y, y_class])
The unit of last Dense layer would be equal to 2.

But in that case, what would be the activation of this last layer ?
And what would be the loss attribute in the compile function ?

Thank you very much

Jason Brownlee April 27, 2021 at 5:13 am #

REPLY

I don't see why not.

You may have to make large changes to the model.

JG April 27, 2021 at 8:25 pm #

REPLY

Hi Jason,

Very interesting tutorial !. Specially to get familiar with API Model Keras. Thank you.
Anyway, I think both problems are better analyzed in a separate way.

I share my code experiment.

– It was simply to use the own model.evaluate () method to get out the metrics. "mae" for regression part of the full model and 'accuracy' for the classification part.

Previously I have to add, at the compilation model method, the 'metrics' argument equal to ['mae', 'accuracy'].

The only confusing thing is when you get the output of model.evaluate() you have all cross combinations for 'mae' and 'accuracy', even for the opposite layer of classifier and regressor.
So I decided to put names on the outputs layers (at model creation) to get a better identification of the right ones!

Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

Jason Brownlee April 28, 2021 at 6:01 am #

REPLY

Thanks.

Hmmm, perhaps evaluate() is not a good tool for such a complex model.

Reza Behzadpour May 14, 2021 at 7:52 pm #

REPLY

I think the code needs a little revision:
1) The name should be dataframe instead of dataset
2) We should use iloc to access the dataframe in index format

```
>>> X, y = dataframe.iloc[:, 1:-1], dataframe.iloc[:, -1]
>>> X, y = X.astype('float'), y.astype('float')
>>> n_features = X.shape[1]
```

Jason Brownlee May 15, 2021 at 6:32 am #

REPLY

I prefer to work with numpy arrays directly.

Shruti Upadhyaya July 2, 2021 at 2:29 am #

REPLY

Hello Jason, Your blogs are great. This is what exactly I was looking for. This works great but I need a slight modification can you please suggest me the best solution.

I have a classification and regression problem for the image dataset. My regression is dependent on the classification, i.e. for example: If Class=1, then Regression prediction should be = 0. How will I force my ML model to predict zero when Class=1 without building different classification model.

Thank you so much.

Jason Brownlee July 2, 2021 at 5:22 am #

REPLY

Thanks!

Perhaps you can chain the models sequentially, perhaps manually.

Shruti Upadhyaya July 9, 2021 at 5:08 am #

REPLY

Thanks Jason, Is it possible to train both models together if it is sequentially chained?

sinfer July 5, 2021 at 4:24 am #

REPLY

Hi Jason,

I have a scenario where i need to forecast a single sequence of time series values first based on the last 10 sequences and based on the forecasted value sequence i need to check under which class label that single sequence falls out of 7 classes.

Thanks

Jason Brownlee July 5, 2021 at 5:09 am #

REPLY

I recommend testing a suite of models in order to discover what works well or best for your dataset.

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

sinfer July 5, 2021 at 1:06 pm #

REPLY

Can you please provide me an example or a brief explanation on what you meant by suit of models ?

Thanks

Jason Brownlee July 6, 2021 at 5:45 am #

REPLY

Sorry, I mean many different model types or configurations.

sinfer July 8, 2021 at 1:37 am #

REPLY

Thanks got it 🙂

Venkata August 17, 2021 at 7:28 pm #

REPLY

Hi Jason,

Thank you for the insightful article on combined regression and classification neural network model. I had few queries from my side concerning the model.

1. Does the model capture the dependency between the two output variables or do we assume they are independent of each other?

2. In the given example, it seems that the same output variable is being classified and same output variable is used for regression. Can we use two different output variables, one for regression and other for classification and run the model ?

Adrian Tam August 18, 2021 at 3:14 am #

REPLY

The model should not assume (1) but it will learn it. Remember, the network model is non-linear. The dependency in that case is not as trivial as linear dependency. For (2), I don't see that is the same. Note that there are y and y_class, which the latter is a transformed result of the former.

Venkata August 18, 2021 at 3:30 pm #

REPLY

Hi Adrian,

Thank you for your response. In the model I am working on, I have 2 different variables dependent on each other and are outputs. One variable is numeric and other variable is categorical. The rest of the variables are inputs.

I tried running the combined model and found that the regression aspect of the model was working well returning almost the same outputs as compared to a neural network model that does only regression (The second categorical output variable that I mentioned earlier was input to this model).

However, when it comes to classification, I am getting very low numbers (of the order 10^-5) instead of class labels(6 classes). Any insights from your side would help. Otherwise, can we connect on LinkedIn or email?

Venkata August 19, 2021 at 1:14 am #

REPLY

When I tried to run the code given on this page as it is, the regression part is giving good predictions. But the classification part is giving predictions of the order of 10^-5 or even lower. The loss function values for both regression and classification were exactly the same values which are shown in this web page. Is there anything else I need to do for the classification part?

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Adrian Tam August 19, 2021 at 3:58 am #

REPLY

It is not my case. You can try add these two lines at the end after you print the accuracy to compare the output side by side:

```
import pandas as pd
print(pd.DataFrame({"y":y_test.ravel(), "yhat":yhat1.ravel(), "y_cls":
y_test_class.ravel(), "yhat_cls":yhat2.ravel()}))
```

Venkata August 21, 2021 at 3:53 am #

REPLY

Thanks for your insights Adrian. The code is working fine now.

For general insight, could you elaborate a bit on how the model predicts 2 output variables. Let us say we have inputs x1….x7 & output variables y1 and y2. For predicting y1, does the model take x1…x7 as input or does the model take x1…x7 including y2 as input. Similarly how does it do for y2 ?

Can I use this technique to predict on data across multiple quarters (time-series). If yes, in what way can I modify it ?

Adrian Tam August 21, 2021 at 5:22 am #

Usually we will not have y1 and y2 in the input, because if you call it the output, that means we should not know it beforehand. But for time-series, you may include the lagged version of y1 and y2. For example, predicting GDP of next quarter, you may use the GDP of this quarter, stock market performance, unemployment rate of today, etc.

Erick Rodriguez September 30, 2021 at 4:59 am #

REPLY

Hello Jason, thanks for your example.

My question is related with the Loss, is there a problem if you are combining differents functions to calculate it?

To calculate the weights of the NN, how can be affected if you are using different losses at the same time? or simply the NN will adjust the weights onto minimizes the general output error?

Adrian Tam October 1, 2021 at 12:14 pm #

REPLY

You can define your own loss function but you use one loss function to train the neural network (or otherwise, gradient descent won't work). You can make use of a different loss function to keep track on the performance of partially-trained network, e.g., in the validation steps.

Adrian Tam October 6, 2021 at 5:06 am #

REPLY

If you are to train a model, there should only be one loss function for the training. This is the loss that training will try to minimize. For validation, however, you can use many different functions.

Ngoc-Tuan Do December 31, 2021 at 2:42 am #

REPLY

Thank you for very interesting post. Could you please explain how the dense layer (hidden1 layer) weights are updated in the training phase?

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

James Carmichael December 31, 2021 at 10:15 am #

REPLY

Hi Ngoc-Tuan Do…Weights and bias are adjusted during training via backgpropagation utilizing a form of gradient descent for optimization. The following may be of interest to you:

https://machinelearningmastery.com/a-gentle-introduction-to-the-challenge-of-training-deep-learning-neural-network-models/

Ogawa January 27, 2022 at 9:32 pm #

REPLY

In the above example, you can do one regression, one classification, but got multiple outputs in a similar way.
(The labels to classify are biased (in many cases), but in the case of multiple outputs, how do you stratified sampling?

Leave a Reply

Name (required)

Email (will not be published) (required)

SUBMIT COMMENT

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

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

Regression Tutorial with the Keras Deep Learning Library in Python

Multi-Class Classification Tutorial with the Keras Deep Learning Library

How to Save and Load Your Keras Deep Learning Model