Hyperparameter Tuning with Keras Tuner

Getting the most out of your models

Cedric Conol Apr 30, 2020 · 5 min read ★

Photo by yinka adeoti on Unsplash

> Great data scientists do not settle with “okay”, they go beyond to achieve the extraordinary.

In this article, we’ll review techniques data scientists use to create models that work great and win competitions. Getting the most out of our models means choosing the optimal hyperparameters for our learning algorithm. This task is known as hyperparameter optimization or hyperparameter tuning. This is especially strenuous in deep learning as neural networks are full of hyperparameters. I’ll assume that you are

already familiar with common data science concepts like regression and mean squared error (MSE) metric and have experience building model using tensorflow and keras.

To demonstrate hyperparameter tuning methods, we'll use keras tuner library to tune a regression model on the Boston housing price dataset. This dataset contains 13 attributes with 404 and 102 training and testing samples respectively. We'll use tensorflow as keras backend so make sure you have tensorflow installed on your machines. I'm using tensorflow version '2.1.0' and kerastuner version '1.0.1'. Tensorflow 2.0.x comes up with keras so you don't need to install keras separately if you have version 2.0.x. You can check the version you have using the code below:

```
import tensorflow as tf
import kerastuner as kt

print(tf.__version__)
print(kt.__version__)
```

Load the dataset

Boston housing price regression dataset can be downloaded directly using keras. Here's a list of datasets that comes with keras. To load the dataset, run the following codes.

```
from tensorflow.keras.datasets import boston_housing

(x_train, y_train), (x_test, y_test) = boston_housing.load_data()
```

Note that if this is the first time you are using this dataset within keras, it will download the dataset from an external source.

This is the regression model I'll use in this demo. The code below shows how the model was built without any tuning.

```
from sklearn.preprocessing import StandardScaler
from tensorflow.keras import models, layers

# set random seed
from numpy.random import seed
seed(42)
import tensorflow
tensorflow.random.set_seed(42)

# preprocessing - normalization
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```
# model building
model = models.Sequential()
model.add(layers.Dense(8, activation='relu', input_shape=
(x_train.shape[1],)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.1))
model.add(layers.Dense(1))


# compile model using rmsprop
model.compile(optimizer='rmsprop',loss='mse',metrics=['mse'])


# model training
history = model.fit(x_train_scaled, y_train, validation_split=0.2,
epochs=10)


# model evaluation
model.evaluate(x_test_scaled, y_test)
```

This model has a MSE of around 434. I have set the random seed in numpy and tensorflow to 42 to get reproducible results. Despite doing so, I still get slightly different results every time I run the code. Let me know in the comments what else I missed make this reproducible.

Tuning with Keras Tuner

To start tuning the model in keras tuner, let's define a **hypermodel** first. **Hypermodel** is a keras tuner class that lets you define the model with a searchable space and build it.

Create a class that inherits from kerastuner.HyperModel, like so:

```
from kerastuner import HyperModel

class RegressionHyperModel(HyperModel):
    def __init__(self, input_shape):
        self.input_shape = input_shape

    def build(self, hp):
        model = Sequential()
        model.add(
            layers.Dense(
                units=hp.Int('units', 8, 64, 4, default=8),
                activation=hp.Choice(
                    'dense_activation',
                    values=['relu', 'tanh', 'sigmoid'],
                    default='relu'),
                input_shape=input_shape
            )
        )

        model.add(
            layers.Dense(
                units=hp.Int('units', 16, 64, 4, default=16),
                activation=hp.Choice(
                    'dense_activation',
                    values=['relu', 'tanh', 'sigmoid'],
                    default='relu')
            )
        )

        model.add(
```

```
            layers.Dropout(
                hp.Float(
                    'dropout',
                    min_value=0.0,
                    max_value=0.1,
                    default=0.005,
                    step=0.01)
            )
        )

        model.add(layers.Dense(1))

        model.compile(
            optimizer='rmsprop',loss='mse',metrics=['mse']
        )

        return model
```

This is the same model we built earlier, except that for every hyperparameter, we defined a search space. You may have noticed hp.Int, hp.Float, and hp.Choice, these are used to define a search space for a hyperparameter that accepts an integer, float and a category respectively. A complete list of hyperparameter methods can be found here. 'hp' is an alias for Keras Tuner's HyperParameters class.

Hyperparameter such as the number of units in a dense layer accepts an integer, hence, hp.Int is used to define a range of integers to try. Similarly, the dropout rate accepts a float value so hp.Float is used. Both hp.Int and hp.Float requires a name, minimum value and maximum value, while the step size and default value is optional.

The hp.Int search space below is named, "units", and will have values from 8 to 64 in multiples of 4, and a default value of 8. hp. Float is used similarly as hp.Int but accepts float values.

```
hp.Int('units', 8, 64, 4, default=8)
```

hp.Choice is used to define a categorical hyperparameter such as the activation function. The search space below, named "dense_activation", will choose between "relu", "tanh", and "sigmoid" functions, with a default value set to "relu".

```
hp.Choice('dense_activation', values=['relu', 'tanh', 'sigmoid'],
default='relu')
```

Instantiate HyperModel

Let's instantiate a hypermodel object. Input shape varies per dataset and the problem you are trying to solve.

```
input_shape = (x_train.shape[1],)
hypermodel = RegressionHyperModel(input_shape)
```

Let's start tuning!

Random Search

As the name suggests, this hyperparameter tuning method randomly tries a combination of hyperparameters from a given search space. To use this method in keras tuner, let's define a tuner using one of the available Tuners. Here's a full list of Tuners.

```
tuner_rs = RandomSearch(
            hypermodel,
            objective='mse',
            seed=42,
            max_trials=10,
            executions_per_trial=2)
```

Run the random search tuner using the *search* method.

```
tuner_rs.search(x_train_scaled, y_train, epochs=10,
validation_split=0.2, verbose=0)
```

Select the best combination of hyperparameters the tuner had tried and evaluate.

```
best_model = tuner_rs.get_best_models(num_models=1)[0]
loss, mse = best_model.evaluate(x_test_scaled, y_test)
```

Random search's MSE is 53.48, a very big improvement from not performing any tuning at all.

Hyperband

Hyperband is based on the algorithm by Li et. al. It optimizes random search method through adaptive resource allocation and early-stopping. Hyperband first runs random hyperparameter configurations for one iteration or two, then selects which configurations perform well, then continues tuning the best performers.

```
tuner_hb = Hyperband(
```

```
            hypermodel,
            max_epochs=5,
            objective='mse',
            seed=42,
            executions_per_trial=2
        )

tuner_hb.search(x_train_scaled, y_train, epochs=10,
validation_split=0.2, verbose=0)

best_model = tuner_hb.get_best_models(num_models=1)[0]
best_model.evaluate(x_test_scaled, y_test)
```

The resulting MSE is 395.19 which is a lot worse when compared to random search but a little bit better than not tuning at all.

Bayesian Optimization

Bayesian optimization is a probabilistic model that maps the hyperparameters to a probability score on the objective function. Unlike Random Search and Hyperband models, Bayesian Optimization keeps track of its past evaluation results and uses it to build the probability model.

```
tuner_bo = BayesianOptimization(
            hypermodel,
            objective='mse',
            max_trials=10,
            seed=42,
            executions_per_trial=2
        )

tuner_bo.search(x_train_scaled, y_train, epochs=10,
validation_split=0.2, verbose=0)

best_model = tuner_bo.get_best_models(num_models=1)[0]
best_model.evaluate(x_test_scaled, y_test)
```

Best model MSE tuned using Bayesian optimization is 46.47, better than the first two tuners we have tried.

Conclusion

We were able to show that indeed, tuning helps us get the most out of our models. Discussed here are just 3 of the many methods of hyperparameter tuning. When trying out the codes above, we may get slightly different results, for some reason, despite setting numpy, tensorflow, and keras tuner random seeds, results per iteration still differ slightly. The notebook is uploaded in my github repo.

Furthermore, tuners can also be tuned! Yes, you read that right, tuning the tuners. Tuners accept values such as max_trials and execution per trial and are can, therefore, be tuned as well. Try changing these parameters and see

if you get further improvements.

References

[1] F. Chollet, *Deep Learning with Python* (2018), Manning Publications Inc.

[2] Keras Tuner Documentation, https://keras-team.github.io/keras-tuner/

[3] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization (2018),* https://arxiv.org/abs/1603.06560

Hyperparameter Tuning | Keras | TensorFlow | Deep Learning | Data Science

Photo by Yassine Khalfalli on Unsplash

On January 19, 2020, the first case of novel coronavirus (COVID-19) reached the United States when a 35 year old man in Washington state walked into a clinic with what we now understand as common symptoms of the virus: respiratory inflammation and a fever. By April 30, 2020, there were more than 1 million cases of COVID-19 in the U.S. Most Americans were truly shocked by the spread of the virus.

COVID-19 Growth Dynamics

. . .

As is the case with epidemics, COVID-19's growth was exponential at first. Imagine a person with COVID-19 going to a party and spreading the disease to…

Read more · 9 min read

157 3

Post a quick thought or a long story. It's easy and free. Write on Medium

Nabanita Roy · Apr 30, 2020 ★

How to Build your own Domain-Focused Datasets or Corpora from Wikipedia

Wikipedia (Python Library) | Beautiful Soup | MediaWikiAPI | DBpedia | SPARQL

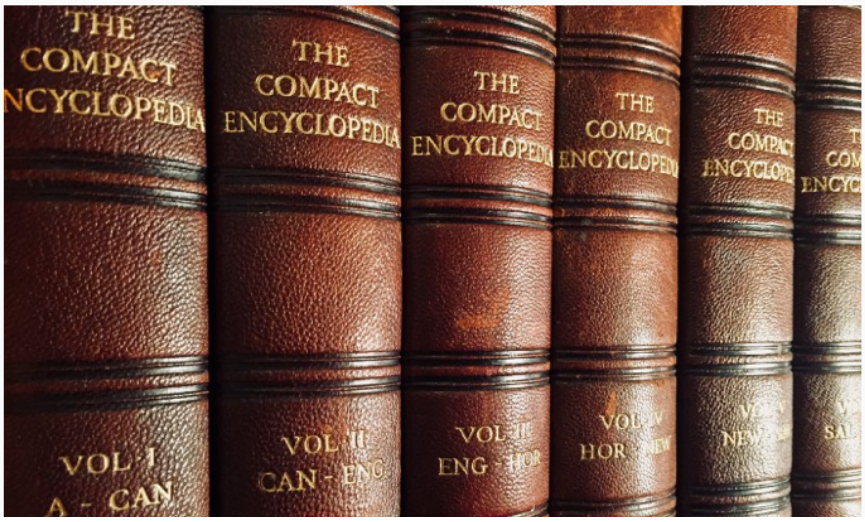

Photo by James L.W on Unsplash

Wikipedia is the 21st century's most-trusted and crowd-sourced digital wealth of information which we refer to for answers to questions, in case of any doubt, to know the plot of web series, to find biodata of the casts of a movie, to know more about the great leaders of the world or the disasters that shook the earth, to learn about the past and what the future holds for us.

Too dramatic, eh?

Wikipedia is a great platform for creating rich datasets or corpora because it has natural language content as well as semantically structured database called the DBpedia.

This…

Read more · 5 min read

53

That Data Bloke · Apr 30, 2020 ★

Gesture Recognition for Beginners with CNN

A fun experiment using Python & SqueezeNet

. . .

Photo by Osman Rana on Unsplash

The CNN or convolutional neural networks are the most commonly used algorithms for image classification problems. An image classifier takes a photograph or video as an input and classifies it into one of the possible categories that it was trained to identify. They have applications in various fields like driver less cars, defense, healthcare etc. There are many algorithms for image classification and in this experiment, we are going to look at one of the most popular algorithms in this genre, called ***SqueezeNet*** by DeepScale.

. . .

The Objective:
Our goal is to design an application that will use a webcam(or an external camera)…

Read more · 8 min read