Report

Optimizing neural network on Keras

Samatov Almaz BS3-DS2

Motivation

In learning neural networks we usually face with low accuracy problem or want it to be more and we need to somehow change configuration of our network to achieve better results. This is what I will try to do in this report.

Body

```
## Build a CNN Model
model = keras.models.Sequential()

# model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
# model.add(MaxPooling2D(pool_size=(2, 2)))
# model.add(Dropout(0.1))

# a basic feed-forward model
model.add(keras.layers.Flatten())
# takes our 28x28 and makes it 1x784

model.add(keras.layers.Dense(784, activation=tf.nn.relu))
# a simple fully-connected layer, 128 units, relu activation

model.add(keras.layers.Dense(256, activation=tf.nn.relu))

#model.add(keras.layers.Dense(128, activation=tf.nn.relu))

model.add(Dropout(0.2))

#model.add(keras.layers.Dense(64, activation=tf.nn.relu))

# a simple fully-connected layer, 128 units, relu activation
model.add(keras.layers.Dense(10, activation=tf.nn.softmax))
# our output layer. 10 units for 10 classes. Softmax for probability distribution

model.compile(optimizer='adam',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "error." Neural network aims to minimize loss.
              metrics=['accuracy'])  # what to track

## Train the Model
model.fit(x_train, y_train, epochs=30, batch_size=100)
```

One of the ways how I do it: just increase number of neurons, then I have: 1 layer - 784 nodes with ReLU, 2 layer - 256 nodes with ReLU, then add Dropout layer with 0.2 (to avoid overfitting, because I use 30 epochs. It can overfit, because train accuracy increase all time when we train it) and then layer with softmax. Also I change batch size to 100 in order to process faster, but still learn on small changes.

This gave me 0.98 accuracy.

Continue on next page

```
x_train = x_train.reshape(60000,28,28,1)
x_test = x_test.reshape(10000,28,28,1)

## Build a CNN Model
model = keras.models.Sequential()

model.add(Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

# a basic feed-forward model
model.add(keras.layers.Flatten())
# takes our 28x28 and makes it 1x784

model.add(keras.layers.Dense(256, activation=tf.nn.relu))
# a simple fully-connected layer, 128 units, relu activation

model.add(keras.layers.Dense(128, activation=tf.nn.relu))

#model.add(keras.layers.Dense(128, activation=tf.nn.relu))

model.add(Dropout(0.2))

#model.add(keras.layers.Dense(64, activation=tf.nn.relu))

# a simple fully-connected layer, 128 units, relu activation
model.add(keras.layers.Dense(10, activation=tf.nn.softmax))
# our output layer. 10 units for 10 classes. Softmax for probability distribution

model.compile(optimizer='adam',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "error." Neural network aims to minimize loss.
              metrics=['accuracy'])  # what to track

## Train the Model
model.fit(x_train, y_train, epochs=3, batch_size=100)
```

Then I decided to do something smarter. Here I add convolution layer (with 64 channels and ReLU activation function), then do max pooling and then do Dropout (this is regularization method for training neural networks, which helps to avoid overfitting. It is easily implemented by randomly selecting nodes to be dropped-out with a given probability (e.g. 20%) each weight update cycle).

This gave me 0.9887 accuracy. And this is best result, which I achieve.

Also I try to change other things as optimizer function, but with Stochastic gradient descent accuracy decrease to 91%.

And also I try to change loss function, but results was bad. Only sparse categorical cross entropy shows good results (97% and more), but categorical and binary cross entropies shows bad results (around 90% accuracy).

And also I try to change lost function, but they does not help so much too. With sigmoid and tanh our model learns really slow and does not achieve so good results (it achieves only 96-97% accuracy).

Increasing of nodes help to increase accuracy, but your model can overfit.

Also increasing epoches help to increase accuracy, but again your model can overfit.

Conclusion

At the end, I can say that best parameters for learning neural networks in this task is using Adam optimization method, Sparse categorical cross entropy as loss function, ReLU as activation function, Increasing hidden layers does not help so much, but increasing epoches can help, but model can overfit and you need to use Dropout, keep batch size not so large, because your model need to learn fast. But to achieve really good results you should do some more smart things as adding convolution layers, max pooling layers than just increasing epoches and other numerical parameters.

Also I want to say that this parameters suit so good in my situations, with other methods you can also achieve good results.