Summary

May 26, 2024

1 1000-719bMSB Modeling of Complex Biological Systems

2 Deep Neural Network: Supervised Learning

2.1 Homework - Krzysztof Łukasz - SUMMARY

3 TENSORFLOW

3.1 1.1 Densly connected

Architecture of the model:

[ ]:
```
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
loss='mean_squared_error',
metrics=['accuracy'])
```

Training accuracy:

- 0.8906

Test accuracy:

- 0.8553

3.2 1.2 Convolutional

Architecture of the model:

[ ]:
```
model2 = models.Sequential()
model2.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,␣
↪1)))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(layers.MaxPooling2D((2, 2)))
model2.add(layers.Conv2D(64, (3, 3), activation='relu'))
model2.add(layers.Flatten())
model2.add(layers.Dense(64, activation='relu'))
model2.add(layers.Dense(10, activation='softmax'))
```

Training accuracy:

- 0.9580

Test accuracy:

- 0.9112

3.3 1.3 Improved

Architecture of the model:

[ ]:
```
better_model = models.Sequential()

better_model.add(Conv2D(32, 3, padding='same',␣
 ↪activation='relu',kernel_initializer='he_normal', input_shape=(28,28, 1)))
better_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))


better_model.add(Conv2D(64, 3, padding='same', activation='relu'))
better_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

better_model.add(Dropout(0.3))
better_model.add(BatchNormalization())
better_model.add(Conv2D(128, 3, padding='same', activation='relu'))
better_model.add(Conv2D(128, 3, padding='same', activation='relu'))
better_model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

better_model.add(Dropout(0.2))
better_model.add(Flatten())
better_model.add(BatchNormalization())
better_model.add(Dense(512, activation='relu'))

better_model.add(Dropout(0.25))
better_model.add(Dense(10, activation='softmax'))
```

Training accuracy:

- 0.95018333

Test accuracy:

- 0.9224

4 PyTORCH

4.1 1.1 Dense

Architecture of the model:

[ ]:
```
class MnistModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, xb):
        # view xb with two dimensions, 28 * 28(i.e 784)
        # One argument to .reshape can be set to -1(in this case the first dimension),
        # to let PyTorch figure it out automatically based on the shape of the original tensor.
        xb = xb.reshape(-1, 784)
        print(xb)
        out = self.linear(xb)
        print(out)
        return(out)

model = MnistModel()
```

Training accuracy:

- 0.7182

Test accuracy:

- 0.709179

4.2 1.2 Convolutional

Architecture of the model:

[ ]:
```
# We construct a fundamental CNN class.
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
```

```
            nn.MaxPool2d(2),
        )
        # fully connected layer, output 10 classes
        self.out = nn.Linear(32 * 7 * 7, 10)
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x    # return x for visualization

cnn = CNN()
loss_func = nn.CrossEntropyLoss()
loss_func

# unlike earlier example using optim.SGD, we use optim.Adam as the optimizer
# lr(Learning Rate): Rate at which our model updates the weights in the cells␣
 ↪each time back-propagation is done.
optimizer = optim.Adam(cnn.parameters(), lr = 0.01)
optimizer
```

Training accuracy:

- 0.89

Test accuracy:

- 0.88

4.3 1.3 Improved

Architecture of the model:

[ ]:
```
# We construct a fundamental CNN class.
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=32,
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.BatchNorm2d(32),

            nn.ReLU(),
```

```
            nn.MaxPool2d(kernel_size=2),

        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout2d(p=0.2),
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 1, 1),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )

        # Adjust the input size of the fully connected layer
        self.out = nn.Linear(128  * 3 * 3, 10)  # 3x3 is the output size after␣
 ↪conv3

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)  # Pass through the new layer
        x = x.view(x.size(0), -1)
        output = self.out(x)
        return output, x
cnn = CNN()

###
optimizer = optim.Adam(cnn.parameters(), lr = 0.001)

###
train(num_epochs=15, cnn=cnn, loaders=train_loader)
```

Training accuracy:

- 0.96

Test accuracy:

- 0.94

[ ]: