Lernverfahren autonomer Roboter - Übung 11

G10
Andre Osse
Waldemar Stockmann
Markus Strehling
Tobias Hahn

January 31, 2017

1 Convolutional Neural Networks

1.1 Softmax activation function and Cross entropy error

Beide Funktionen wurden in tools.py wie gewünscht implementiert. Die entsprechenden Codeausschnitte sind im folgenden wiedergegeben, gefolgt von der Konsolenausgabe für den Test:

1.1.1 Code

Listing 1: Softmax

```
1 def softmax(A):
2     A = np.exp(A - np.amax(A, axis=1, keepdims=True))
3     return A / np.sum(A, axis=1, keepdims=True)
```

Listing 2: Cross entropy error

```
1 # Compute error of the dataset
2 if self.error\_function == "ce":
3    return -np.sum(np.log(self.predict(X)) * T)
```

1.2 Output

Listing 3: MLNN Classification Test Output

```
1 Fully connected layer (20 nodes, 20 x 11 weights)
2 Fully connected layer (10 nodes, 10 x 21 weights)
3 Fully connected layer (3 nodes, 3 x 11 weights)
4 Checking gradients up to 6 positions after decimal point...
5 OK
```

1.3 Convolutional Layer

Der Convolutional Layer wurde mit Numba implementiert. Die relevanten Codestellen sind angegeben, gemeinsam mit der Ausgabe des Tests.

1.3.1 Code

Listing 4: Convolve

```
1 for n in range(N):
2    for j in range(J):
3       for y_o in range(Y_o):
4                   for x_o in range(X_o):
5                       for i in range(I):
6                           for y_k in range(Y_k):
7                               for x_k in range(X_k):
8                                   A[n,j,y_o,x_o] += kernels[j,i,y_k,x_k] * feature_maps[n,
                                        i,y_o*stride_y+y_k, x_o*stride_x+x_k]
9                           A[n,j,y_o,x_o] += bias[j,i]
```

Listing 5: Output Convolutional Test

```
1 Convolutional layer (2 x 8 x 8 nodes, 2 x 1 x 3 x 3 weights)
2 Convolutional layer (2 x 6 x 6 nodes, 2 x 2 x 3 x 3 weights)
3 Convolutional layer (2 x 4 x 4 nodes, 2 x 2 x 3 x 3 weights)
4 Fully connected layer (10 nodes, 10 x 33 weights)
5 Fully connected layer (2 nodes, 2 x 11 weights)
6 Checking gradients up to 6 positions after decimal point...Max error: 1.0920861268457293e-08
7 OK
```

1.4 Connection calculations

1.4.1 Input to First Convolutional

Pixel Alle Pixelanzahlen sind immer berechnet auf ein Bild. Anzahl der Pixel in den Featuremaps: Ein Featurelayer hat $((Original_x - Kernel_x div 2)/Stride_x)) * ((Original_y - Kernel_y div 2)/Stride_y) = 144$. Von diesen Feauturelayers gibt es 6, also ist die Gesamtpixelanzahl $144 * 6 = 864$.

Connections Eine Verbindung sei zwischen zwei Pixeln dann gegeben, wenn der erste Pixel mit seinem Wert Einfluss auf den zweiten Pixel nimmt. Dann haben die Pixel, die mindestens 9 Pixel Abstand von jedem Rand haben, eine Verbindung mit $3 * 3 = 9$ Pixel in eine Featuremap, Pixel die weniger Abstand haben $(Abstand_X div 3 + 1) * (Abstand_Y div 3 + 1)$. Damit ergeben sich bei $28 * 28$ Pixel Ausgangsbild und 6 Featuremaps $2702 * 6 = 16212$ Verbindungen.

Weights Für jede Featuremap gibt es im Layer einen Kernel, der für jeden Pixel Gewichte hat. In den Convolutional Layers hat der Kernel jeweils 25 Pixel, das multipliziert mit 6 ergibt 150 Gewichte für den ersten Layer.

1.4.2 First Convolutional to second Convolutional

Pixel Anzahl der Pixel in den Featuremaps: Eine Featuremap hat $((Original_x - Kernel_x div 2)/Stride_x)) * ((Original_y - Kernel_y div 2)/Stride_y) = 16$ Pixel. Von diesen Feauturelayers gibt es 50, also ist die Gesamtpixelanzahl $16 * 50 = 800$.

Verbindungen Die Rechnung ist die selbe von oben, allerdings ist es nun so dass die Verbindungen nun nicht mehr von einem Ausgangsbild gerechnet werden sondern von jeder der Featuremaps. Die Berechnung für ein Pixel in einer Featuremap ist noch die gleiche, da sich die Größe des Kernels nicht geändert hat. Jedoch muss diese Pixelanzahl nun mal die Anzahl der Ausgangsfeaturemaps und die Anzahl der Eingangsfeaturemaps gerechnet werden. Damit ergibt sich $242 * 6 * 50 = 72600$.

Weights Für jede jede Ausgangsfeaturemap gibt es einen Kernel, dessen Pixel immer Gewichte sind. Damit ergibt sich für diesen Layer mit 50 Featuremaps folgende Gewichtsanzahl $25 * 50 = 1250$ Gewichte.

1.4.3 Second Convolutional to Last (Fully Connected) Layer

Verbindungen Jeder Pixel in den letzten Featuremaps ist verbunden mit jedem Neuron im Layer. Damit ergibt sich folgende Anzahl der Verbindungen: $800 * 100 = 80000$.

Gewichte Die Anzahl der Gewichte ist gleich der Anzahl der Verbindungen, da jede Verbindung ein eigenes Gewicht hat.

1.4.4 Fully connected layer

Sei jedes Pixel in den feature maps der Layer ein Neuron, dann gibt es in jedem Layer des Fully Connected Net so viele Verbindungen, die jeweils einem Gewicht entsprechen:

Eingabe zu erstem Hidden Layer $28 * 28 * 864 = 677376$ Verbindungen

Erster Hidden zu zweitem Hidden $864 * 800 = 691200$ Verbindungen

Zweiter Hidden zu Letztem Hidden $800 * 100 = 80000$ Verbindungen

Insgesamt Insgesamt wäre die Anzahl der Verbindungen: $677376 + 691200 + 80000 = 1448576$

1.5 Training CNN

Das Netzwerk wurde mit den vorgegebenen Parametern trainiert. Die relevante Codestelle dafür ist unten aufgeführt, anschließend die Accuracy und die falsch vorhergesagten Ziffern.

1.5.1 Code

Train CVNN

```
    ################################################################################
    # Here you should define and train 'mlnn' (type: MultilayerNeuralNetwork)
    layers = \
        [
            {
                "type": "convolutional",
                "num_feature_maps": 6,
                "kernel_shape": (5, 5),
                "strides": (2, 2)
            },
            {
                "type": "convolutional",
                "num_feature_maps": 50,
                "kernel_shape": (5, 5),
                "strides": (2, 2)
            },
            {
                "type": "fully_connected",
                "num_nodes": 100
            }
        ]

    mlnn = MultilayerNeuralNetwork(
        D=(1, 28,28), F=10, layers=layers, std_dev=0.01,
        verbose=1)
    mbsgd = MiniBatchSGD(net=mlnn, epochs=15, batch_size=32, alpha=0.01, alpha_decay=0.9999,
         min_alpha=0.00005, eta=0.5, eta_inc=0.00001, max_eta=0.9, random_state=0, verbose
        =1)
    mbsgd.fit(train_images, train_targets)

    ################################################################################
```

../code/error.py

```
import pickle
import numpy as np
from mnist import read, scale, generate_targets, model_accuracy
import matplotlib.pyplot as plt

if __name__ == "__main__":
    categories = 10
    test_images, test_labels = read(range(categories), "testing")
    test_images = scale(test_images)[:, None, :, :]
    test_targets = generate_targets(test_labels)

    mlnn = pickle.load(open("mnist_model.pickle", "rb"))

    accuracy = 100 * model_accuracy(mlnn, test_images, test_labels)
    error = mlnn.error(test_images, test_targets)
    print("Accuracy on test set: %.2f %%" % accuracy)
    print("Error = %.3f" % error)

    predict = np.argmax(mlnn.predict(test_images), axis=1)
    errors = []
    for i, x in enumerate(test_labels):
        if (x != predict[i]):
            errors.append(i)

    for row in range(8):
        for col in range(16):
            if (row == 7 and col == 12):
                break
            plt.subplot(8,16,row*16+col+1)
            plt.title("T:{0}/P:{1}".format(test_labels[errors[row*16+col],0], predict[errors
                [row*16+col]]))
            plt.imshow(test_images[errors[row*16+col],0])
            plt.axis('off')
```

```
33      plt.show()
```

1.5.2 Output

Console Output

```
1  Accuracy on test set: 98.76 %
2  Error = 476.353
```

1.5.3 Results