Sieci neuronowe

Sprawozdanie - Adam Dyszy

Zadania zapoznawcze z sieciami neuronowymi:

Intro:
Można było zobaczyć dane np. standardowo na początku 3,a przy zmianie na:
`example_label = mnist.train.labels[3]`
można było zobaczyć liczbę “6”, dzięki czemu mogliśmy zbadać wygląd danych wejściowych
Zadanie 1:
Po dokończeniu inicjalizacji funkcji udalo sie otrzymac wynik:
0.098
Ale przy zmianie argumentow cos bylo nie tak bo zawsze byl tak samo niski wynik, wiec zmienilem wczesniejsza werjse funkcji cross_entropy:

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(None,
reduction_indices=[1]))
```

Zmienilem na nastepującą:

```
cross_entropy =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y,
labels=y_))
```

I zadzialalo z wynikiem 0.9201

Zadanie 2:
Tym razem dodalismy warstwe relu jednak nie zadzialala zbyt dobrze przez niemozliwosc aktywacji spowodowaną wagami zinicjalizowanymi na 0 gdzie wpadliśmy w dołek.
step: 0, acc: 0.130
step: 50, acc: 0.150
step: 100, acc: 0.080
step: 150, acc: 0.180
step: 200, acc: 0.140
...
step: 700, acc: 0.170
step: 750, acc: 0.110
step: 800, acc: 0.160
step: 850, acc: 0.100
step: 900, acc: 0.130
step: 950, acc: 0.130
0.1135

Zadanie 3:
Przy zinicjalizowaniu wag na losowe wartosci z przedzialu -0.1, 0.1 z uzyciem random_uniform uzyskalismy taki wynik:
step: 0, acc: 0.4700
step: 50, acc: 0.9300
step: 100, acc: 0.9200
step: 150, acc: 0.9600
step: 200, acc: 0.9400
step: 250, acc: 0.9400
...
step: 700, acc: 0.9800
step: 750, acc: 1.0000
step: 800, acc: 1.0000
step: 850, acc: 0.9800
step: 900, acc: 1.0000
step: 950, acc: 1.0000
0.9611

Jednak widac ze dziala zbyt agresywnie i można zauważyć **overfitting** wiec sprobujemy zmniejszyc stopien uczenia sie w ciagu czasu zmieniając GradientDescent na MomentumOptimizer(lr =0.05,m=0.9) dodatkowo zwiekszymy ilosc iteracji i otrzymamy taki wynik:

step: 0, acc: 0.1560
step: 50, acc: 0.8770
step: 100, acc: 0.8960
step: 150, acc: 0.8960
step: 200, acc: 0.9170
...
step: 1650, acc: 0.9840
step: 1700, acc: 0.9860
step: 1750, acc: 0.9790
step: 1800, acc: 0.9790
step: 1850, acc: 0.9820
step: 1900, acc: 0.9790
step: 1950, acc: 0.9880
0.9737

Moznaby dalej optymalizować chociażby modyfikując warstwe ukrytą do 300neuronów i dodając kolejną warstwę z 300 neuronami otrzymalem taki wynik:
step: 0, acc: 0.0760
step: 50, acc: 0.9050
step: 100, acc: 0.9330
step: 150, acc: 0.9460
step: 200, acc: 0.9420
step: 250, acc: 0.9660

..
step: 1200, acc: 0.9920
step: 1250, acc: 0.9950
step: 1300, acc: 0.9920
step: 1350, acc: 0.9990
step: 1400, acc: 0.9990
step: 1450, acc: 0.9970
step: 1500, acc: 0.9980
step: 1550, acc: 1.0000
step: 1600, acc: 0.9980
step: 1650, acc: 1.0000
step: 1700, acc: 0.9980
step: 1750, acc: 0.9990
step: 1800, acc: 0.9990
step: 1850, acc: 0.9990
step: 1900, acc: 1.0000
step: 1950, acc: 1.0000
0.9787

Jak widać overfitting byl niższy ale również jest zauważalny, znowu można by pobawić się z learning rate oraz momentum, albo potestować inne optimalizatory np. Adam

Zadanie 4:
input_1 Tensor("input_1:0", shape=(?, 224, 224, 3), dtype=float32)
block1_conv1 Tensor("block1_conv1/Relu:0", shape=(?, 224, 224, 64), dtype=float32)
block1_conv2 Tensor("block1_conv2/Relu:0", shape=(?, 224, 224, 64), dtype=float32)
block1_pool Tensor("block1_pool/MaxPool:0", shape=(?, 112, 112, 64), dtype=float32)
block2_conv1 Tensor("block2_conv1/Relu:0", shape=(?, 112, 112, 128), dtype=float32)
block2_conv2 Tensor("block2_conv2/Relu:0", shape=(?, 112, 112, 128), dtype=float32)
block2_pool Tensor("block2_pool/MaxPool:0", shape=(?, 56, 56, 128), dtype=float32)
block3_conv1 Tensor("block3_conv1/Relu:0", shape=(?, 56, 56, 256), dtype=float32)
block3_conv2 Tensor("block3_conv2/Relu:0", shape=(?, 56, 56, 256), dtype=float32)
block3_conv3 Tensor("block3_conv3/Relu:0", shape=(?, 56, 56, 256), dtype=float32)
block3_pool Tensor("block3_pool/MaxPool:0", shape=(?, 28, 28, 256), dtype=float32)
block4_conv1 Tensor("block4_conv1/Relu:0", shape=(?, 28, 28, 512), dtype=float32)
block4_conv2 Tensor("block4_conv2/Relu:0", shape=(?, 28, 28, 512), dtype=float32)
block4_conv3 Tensor("block4_conv3/Relu:0", shape=(?, 28, 28, 512), dtype=float32)
block4_pool Tensor("block4_pool/MaxPool:0", shape=(?, 14, 14, 512), dtype=float32)
block5_conv1 Tensor("block5_conv1/Relu:0", shape=(?, 14, 14, 512), dtype=float32)
block5_conv2 Tensor("block5_conv2/Relu:0", shape=(?, 14, 14, 512), dtype=float32)
block5_conv3 Tensor("block5_conv3/Relu:0", shape=(?, 14, 14, 512), dtype=float32)
block5_pool Tensor("block5_pool/MaxPool:0", shape=(?, 7, 7, 512), dtype=float32)
flatten Tensor("flatten/Reshape:0", shape=(?, ?), dtype=float32)

Jak widać mamy tutaj konwolucje ze spłaszczenie do 1 wymiarowych danych

fc1 Tensor("fc1/Relu:0", shape=(?, 4096), dtype=float32)
fc2 Tensor("fc2/Relu:0", shape=(?, 4096), dtype=float32)
predictions Tensor("predictions/Softmax:0", shape=(?, 1000), dtype=float32)

Potem 2 wielkie Relu w których juz są pewnie zawarte dokładniejsze feature’y

Przy sprawdzaniu różnych obrazków algorytm działał dosyć dobrze.