ConvNetJS MNIST demo

Description

This demo trains a Convolutional Neural Network on the MNIST digits dataset in your browser, with nothing but Javascript. The dataset is fairly easy and one should expect to get somewhere around 99% accuracy within few minutes. I used this python script to parse the original files into batches of images that can be easily loaded into page DOM with img tags.

This network takes a 28x28 MNIST image and crops a random 24x24 window before training on it (this technique is called data augmentation and improves generalization). Similarly to do prediction, 4 random crops are sampled and the probabilities across all crops are averaged to produce final predictions. The network runs at about 5ms for both forward and backward pass on my reasonably decent Ubuntu+Chrome machine.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

Training Stats

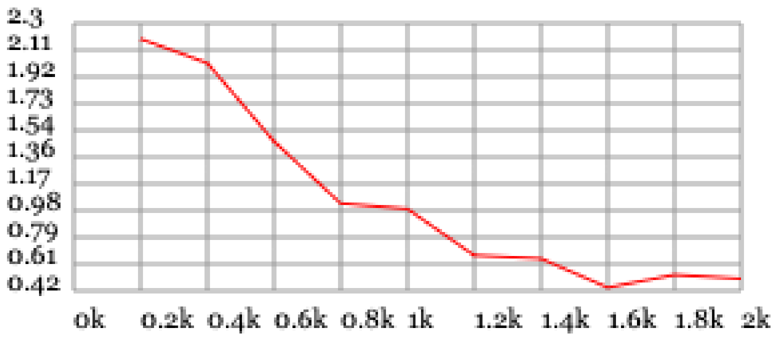
pause

Forward time per example: 2ms
Backprop time per example: 2ms
Classification loss: 0.50576
L2 Weight decay loss: 0.00135
Training accuracy: 0.87
Validation accuracy: 0.78
Examples seen: 2000
Learning rate: 0.01 change
Momentum: 0.9 change
Batch size: 20 change
Weight decay: 0.001 change

save network snapshot as JSON

init network from JSON snapshot

Loss:



clear graph

Instantiate a Network and Trainer

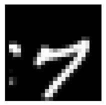
```
layer_defs = [];
layer_defs.push({type:'input', out_sx:24, out_sy:24, out_depth:1});
layer_defs.push({type:'conv', sx:5, filters:8, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:2, stride:2});
layer_defs.push({type:'conv', sx:5, filters:16, stride:1, pad:2, activation:'relu'});
layer_defs.push({type:'pool', sx:3, stride:3});
layer_defs.push({type:'softmax', num_classes:10});

net = new convnetjs.Net();
net.makeLayers(layer_defs);

trainer = new convnetjs.SGDTrainer(net, {method:'adadelta', batch_size:20, l2_decay:0.001});
```

change network

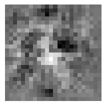
Network Visualization

input (24x24x1)
max activation: 1, min: 0
max gradient: 0.46304, min: -0.3949

Activations:

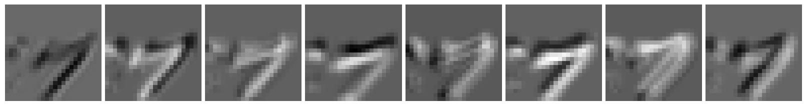


Activation Gradients:

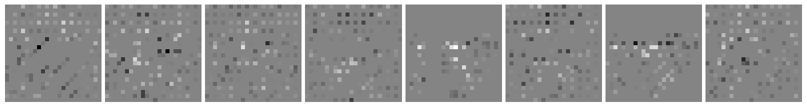


conv (24x24x8)
filter size 5x5x1, stride 1
max activation: 2.47644, min: -1.43224
max gradient: 0.23071, min: -0.24962
parameters: 8x5x5x1+8 = 208

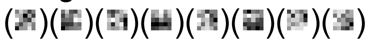
Activations:



Activation Gradients:



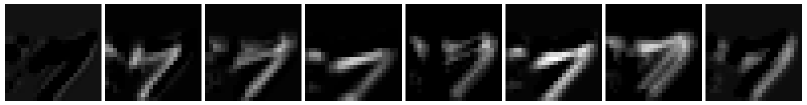
Weights:



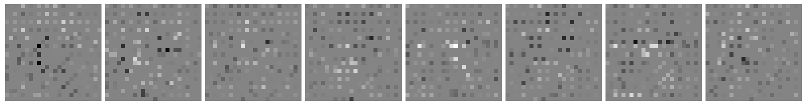
Weight Gradients:

relu (24x24x8)
max activation: 2.47644, min: 0
max gradient: 0.23071, min: -0.24962

Activations:

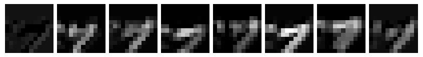


Activation Gradients:

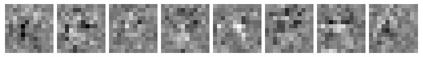


pool (12x12x8)
pooling size 2x2, stride 2
max activation: 2.47644, min: 0
max gradient: 0.23071, min: -0.24962

Activations:

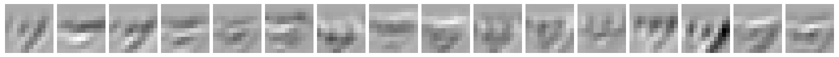


Activation Gradients:

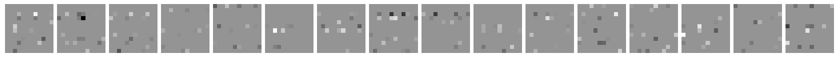


conv (12x12x16)
filter size 5x5x8, stride 1
max activation: 4.07014, min: -8.45024
max gradient: 0.28477, min: -0.39567
parameters: 16x5x5x8+16 = 3216

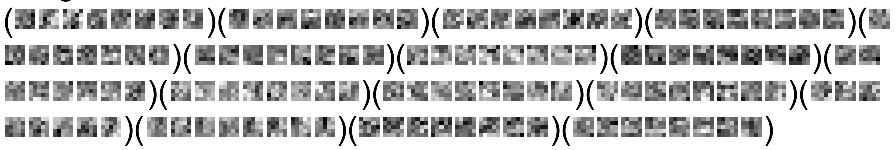
Activations:



Activation Gradients:



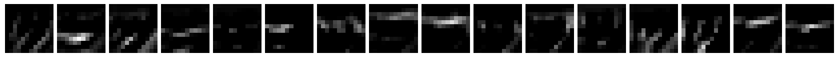
Weights:



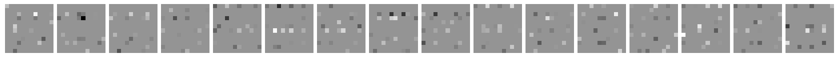
Weight Gradients:

relu (12x12x16)
max activation: 4.07014, min: 0
max gradient: 0.28477, min: -0.39567

Activations:



Activation Gradients:



pool (4x4x16)
pooling size 3x3, stride 3
max activation: 4.07014, min: 0
max gradient: 0.28477, min: -0.39567

Activations:

Activation Gradients:

fc (1x1x10)
max activation: 2.31402, min: -5.56164
max gradient: 0.75314, min: -0.89211
parameters: 10x256+10 = 2570

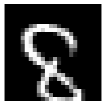
Activations:

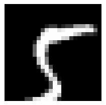
Activation Gradients:

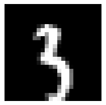
softmax (1x1x10)
max activation: 0.75314, min: 0.00028
max gradient: 0, min: 0

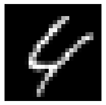
Activations:

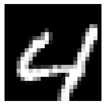
Example predictions on Test set

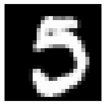

8
2
9

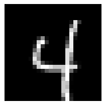

5
0
8

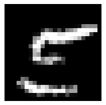

3
9
4

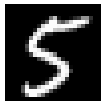

4
6
9

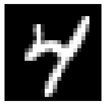

4
0
9

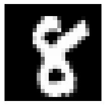

0
5
3

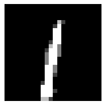

4
9
6

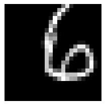

5
0
8

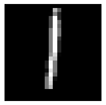

5
2
3

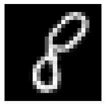

4
7
9

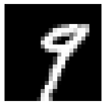

8
5
4

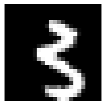

1
6
4

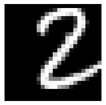

6
2
4

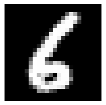

1
6
4

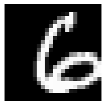

8
0
5

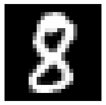

7
9
1

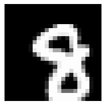

2
3
5

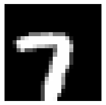

2
0
8

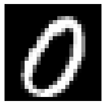

6
2
8

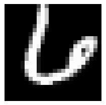

6
4
2

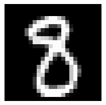

8
2
3

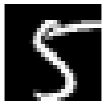

8
9
2

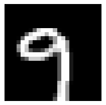

7
9
4

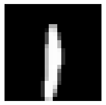

0
2
9

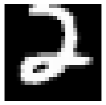

0
6
4

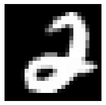

8
3
2

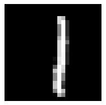

5
9
8

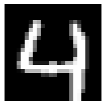

9
7
4

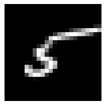

1
4
9

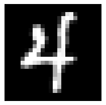

2
3
0

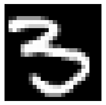

2
0
3

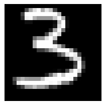

1
6
8

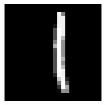

4
9
2

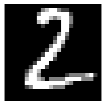

5
4
0

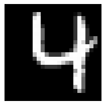

4
9
6

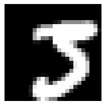

3
2
5

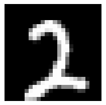

3
2
5

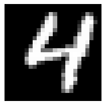

1
6
4

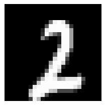

2
8
0

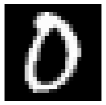

4
9
6

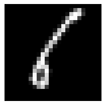

2
5
3

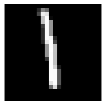

2
3
7

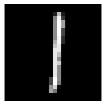

4
9
0

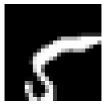

2
3
6

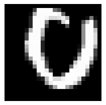

0
2
3


1
6
0


1
4
6


1
6
4


5
0
4


0
4
2