09/19/2017

I’m getting eerily similar training results every time with the pretraining step of the SAE. This is 400 epochs, with reporting every 50. Mb size = 20.

89.6588363201

27.936811581

21.1432677307

17.2972330443

14.6389792783

12.7702663755

11.3860413865

10.3059933832

Also, it seems that the full-batch training method is working a fair bit better than the minibatch approach. I wonder if I need to randomize the minibatches. But minibatch is faster.

In fact, I think the pretraining may be hurting the performance of the SAE. Additionally, the pretrained first depth layer outperforms the stacked finetuning, at least at first. And if I “donate” those extra epochs from the fine-tune to the first pretraining, it’s better than the SAE.

Compare:

For a 50-20 SAE. Mb size = 20, trainingdata = 100, epochs = 400, learning rate = 5, no weight decay or sparsity.

|  |  |  |  |
| --- | --- | --- | --- |
| Pretrained 1st layer | Pretraining+fine-tuning | Just fine-tuning | FB approach eta = 1 |
| 89.6825230642  28.2885931141  21.1949529425  17.2707212921  14.737571636  12.930437667  11.5639083091  10.4899190334 | 13.3338828229  11.8658261455  11.0304312536  10.3635327607  9.80453370975  9.31546000795  8.88513386833  8.49709778234 | 16.5568680662  7.08634231798  6.53417265913  6.17620641104  5.89721820236  5.66277923129  5.45239412989  5.26413091542 | 115.539420641  25.4352064244  17.7598807455  14.0159246566  11.7452038558  10.1729344543  9.04660925228  8.18084998942 |

Training with more input images (10000) for SAE seemed to do a little better in terms of not jaggedly overfittting, but I think I cut the epochs down too much.

09/20/2017

I’m not seeing that to be the case anymore. The pretraining is definitely helping a lot

Same eta, mb, epochs as above, except every layer got 400 pretraining epochs

|  |  |
| --- | --- |
| With pretrainin | Without pretraining |
| 11.6816940302  10.4389943367  9.68439988193  9.04859831825  8.54778993231  8.13098201607  7.76296984762  7.44082861193 | 69.3857057151  49.375759012  49.2166619098  45.4833144464  43.6114881369  42.4636631167  41.4267167562  40.6541265606 |

Oh wow, I just started using the test data to evaluate it, and it’s not doing very well actually. I need to be using the test data from now on to do evaluations.

Same settings, evaluating on test data

33.4512055228

33.5519314699

33.7931798149

33.8852607681

33.9634980623

34.0422506919

34.1295740102

34.2172509573

We see that the error actually increases! It’s clearly overfitting.

I should be regularizing.

Now using test data. Something is still weird! The single layer is working better in terms of error than the SAE

Note this is all within a single pretraining/training sequence:

Pretraining outer layer

98.8554596448

37.2866874788

32.9964377937

30.603882659

29.1153054016

28.2686920626

27.711597101

27.3635089561

pretraining middle layer

9.08849662884

1.54076943416

1.08756853745

0.891112617864

0.773532415944

0.701588393488

0.646517964197

0.604187839278

initial error avg SSE between non-test data and prediction = 11.7774877672

33.6450021098

34.5235426293

34.7993055102

35.0460709851

35.2555583574

35.4003992388

35.4214690585

35.4746671149

It’s odd to me that the SAE is so much worse? Oh, actually it makes sense, as it’s further compressing the data! That’s probably why.

Hmm. Not necessarily. It’s still doing worse than the pretrain even with 50-50 setup.

98.772925893

36.908439225

31.8217481501

29.2119670043

27.9276889189

27.272858041

27.0065754094

26.8679562134

8.42716769653

0.237351101667

0.111123122748

0.0659286365242

0.0487428267369

0.0388934564738

0.0326909467414

0.0284308021774

avg SSE between data and prediction = 10.0053371174

30.0897187526

30.8118082633

31.3866478584

31.7597261016

32.0261561475

32.2394051917

32.413522633

32.5744939006

Looking into dr milchenko’s SAE design: gradient boosted SAE.

It’s producing lots of nonsense.

It may be because we need to downscale the “deeper” residuals. Right now summing all errors for a single residual series gave this for the four levels. Actually, you can’t just sum the residuals. You gotta square them first.

[ 3.83148236 1.15968813 3.24094709 1.67424373]

Actually, you can’t just sum the residuals. You gotta square them first. That looks like

[ 5.7716879 5.80802388 5.79478967 5.79269591]

so it seems they are all similar in magnitude… I don’t think this should be the case, and if it somehow is, I don’t think they should contribute equally. I would think the error would decay, as each time you start with smaller and smaller values… Maybe my gradient descent is totally random?