Benjamin Ye

CS/CNS/EE 156a: Learning Systems (Fall 2023)

November 17, 2023

**Bonus Exercise**

**1 Classifying digits with NNs**

**1.3 Number of Parameters**

What happens to the learning curve when you vary the number of hidden units? Specify the number of hidden units you use in each layer, and report learning curves for each different architecture you try. What trends do you notice? What is the smallest number of model parameters (not hidden units; view this via the summary function) for which you can achieve over 95% validation accuracy? An estimate is fine.

**1.4 Regularization**

Explore the usefulness of weight decay regularization (minimizing instead of just ) on the large neural net implementation give in the support code. You can do so changing the REGULARIZATION parameter in the IPython notebook, or by specifying the -r command line option in train.py in the command shell. Report learning curves for different choices of (just a few). Can you use regularization to get a similar result to one of the smaller neural nets you implemented in the previous part?

**1.5 Activations**

In class, we have primarily discussed using the and sigmoid activations. In practice nowadays, people often use the ReLU activation, which is defined as . We use this activation function in the support code, and in practice, people have observed that it results in much faster convergence. Verify this for yourself by replacing the ReLU activation with a activation. Why do you think this happens? (Hint: How does behave as ? Compare this to .)

**1.6 Different Architectures**

Now, let’s say that I give you a fixed budget of 200 hidden units. What is the best validation accuracy you can achieve? Feel free to vary the number of layers, the kind of regularization (e.g., which is based on the 1-norm of as opposed to which is the usual weight decay based on the 2-norm) and its strength (), the activation, and the optimizer you use.

**1.7 Convolutional Neural Nets**

By learning a few smaller convolutional filters instead of a series of huge matrices, a convolutional neural net performs much better on this image classification task with a fraction of the parameters. Verify this for yourself. Run the train script and the evaluate script using the convolutional neural net and no regularization. Compare the number of parameters between any of the convolutional and dense neural nets, and report learning curves.

**2 Generative Adversarial Network**

**2.2 How does a GAN work?**

1. Since this solution occurs at a saddle point, let’s take the derivative of the loss with respect to . Since is a function, take a functional derivative (like a vector derivative, don’t worry about being rigorous). Use the following loss,

Hint: We can interchange an integral and a functional derivative:

Don’t forget the chain rule!

1. Now that we have the derivative, we solve for in terms of the probability distributions and when . Hint: Move one integral to the other side of the equality. Notice that if the values under both integrals are the same pointwise, we also have equality with integrals. Solve with respect to .
2. You should get

Do you think this is reasonable?

Yes! Consider the case when , or when the distribution of images as classified by matches that of real and generated images generated by . As expected at this quasi-Nash-equilibrium state, the discriminator assigns a probability of to both real and generated images because it is unable to distinguish between them, indicating that the generator has created images realistic enough to not be detectable as fakes.

**2.4 Reflecting on GAN Behavior**

How would you describe the training process in terms of the images that the model generated?

At the beginning (epoch 0), the images are grayscale artifacts that do not resemble anything.

As training commences, the images start to resemble numbers (by around epoch 10) but are far from being able to be recognized as any specific number. Some simpler numbers, like 1, becomes legible due to their single stroke.

More complex numbers become distinguishable at around epoch 20. However, most look unnatural because of the gray uncertainty between the white digits and the black background and can easily be identified as generated images.

Further refinement occurs all the way up to the final epoch (150). In the final generated image, many of the digits look like they could have been written by a human (with bad handwriting), largely because the edges between the numbers and the backgrounds are much cleaner and pronounced.

What about the learning dynamics in the loss values for the generator and the discriminator?

A large concern in training GANs is a phenomenon known as “mode collapse”. Look up this term and explain it in your own words (it doesn’t have to be rigorous).

Skimming over the code, did the current generator’s implementation manage to avoid mode collapse?

**3 Feedback**