1. Fully connected network

What is momentum?

* Build up speed along directions of gradient that do not change: allows you to converge much faster, also plow through local minima even if you encounter them (not likely)
* Variations: nesterov momentum, adam
  + Adam is preferred method, in adam we combine RMSprop with momentum (avg squared gradients and build up speed)
  + In full version of adam we use a bias correction mechanism that divides by 1 - beta\*\*t, so in the beginning the momentum and avg gradients terms are boosted up. This is done to compensate for slower learning in the beginning, since we initialize m and v with zeros.

What are adaptive learning rate methods?

* Ways to help the model converge faster
* Adagrad, RMSprop
* RMSprop is less volatile because it uses a moving average of squared gradients instead of just the squared gradients
* Goal: implement modular versions of the huge files for neural networks that we wrote last time. Broken up into forward pass, backward pass, and ReLU forward pass / ReLU backward pass. We also compute the backward pass for each of the above (pretty easy -- make sure to store the relevant variables in the cache).
* Mistakes:
  + did not look at the model setup, missed that the last layer was just affine instead of affine\_relu.
    - Soln: Read the documentation closely and articulate your understanding of it (out loud, typed, etc)
  + Update the iteration variable t before using it in calculations, another issue with reading but more about not really understanding Adam and why it works
    - Need to read closer about the update methods, batchnorm, all of this stuff.

1. Batch Normalization and Layer Norm

What is batchnorm? -- solves the unstable gradient problem

* Forces the activations to take on a unit gaussian distribution at each layer for each mini-batch
  + Implementation wise -- we calculate the normalization statistics based on the mini batches (down 0 axis)
  + What is internal covariate shift?
    - Refers to the changing of the input distributions as the parameters change
    - Other problems in training include neuron saturation -- in sigmoid for example, large activations and small activations will cause the gradient to shrink towards zero
    - We use reLU to partly solve this problem, as well as careful weight initialization and small learning rates
    - We also know that using whitening helps train the network faster, but whitening is hard to do at every layer
  + Why can we do this?
    - We can do this relatively easily because normalization can be differentiated (though it took me the better part of two days!)
  + Why is this useful?
    - Training is made more efficient when the train and test data have the same distribution
    - We can use higher learning rates, and we don’t have to worry about things like gradients exploding (because we normalize the activations)
    - Batchnorm also regularizes the model, reducing need for ex. Dropout
* What is layer norm? -- removes mini-batch dependence (apparently useful for recurrent nets, haven’t learned about those yet)
  + Same as batchnorm, but instead of using the mini-batch, we remove dependence on mini-batches by using the individual data points to calculate the normalization statistics -- calculate sideways along the 1 axis

Bugs

* For some reason, doing the x[:, None] indexing to do the subtraction and division caused x\_hat to have shape (4, 1, 1, 5) instead of (4, 5). Fixed it by just doing x - mean and x / sqrt(var + eps). This also fixed the gradients for gamma and beta.
* Initialized the biases with a random distribution centered at zero with std = weight\_scale instead of to zeros. Found error because of loss=1.9 instead of 2.3 and only one bias ended up with a strange relative error
* Not a mistake, but another weird thing about the biases -- my implementation of the alternative batchnorm backward pass is slightly different from the ones online ( I followed Kevin Zakka’s notes). This resulted in dout being slightly different during the backward pass (comparing 1e-12 to 1-13). Since db is one to one with dout, db ended up being slightly different. Also changed the cache to include the square root term to reduce a bit of redundancy, and after that change (not sure if that change was the cause, didn’t check) everything just started working perfectly. Wow.

1. Dropout

* What is dropout?
  + Randomly drop neurons with probability p, this increases robustness to noise by forcing the neuron to not rely on the neurons around it
  + In practice, works way better
  + Analogous to ensemble methods, because it’s like training a bunch of neural networks at once

Tomorrow: the big stuff

1. Convolutional Net

* Mostly pretty good. CNN implementation is tough, but doable. Did not look at im2col, that is the next level up to optimize the convolution operation
* Max pool is kind of like ReLU so that is fine.
* No idea how to implement groupnorm. In the future -- understand numpy and matrix dimensionalities better so that implementing data processing is easier, going to have to do a lot of it.

1. Pytorch

* What is pytorch?
  + Numpy but can use GPU for speed
  + Autograd -- does backpropagation for you with the requires\_grad keyword arg, set that to True so a computational graph gets built up in the background
* With torch.no\_grad() disables sets all the requires\_grad of operations within that ‘context manager’ to false
* In the class defining your network, you should set up the layers in the init, and define the connectivity in forward. Inherit from nn.Module, and make sure to call the super().\_\_init\_\_