Neural Networks (part 3)

Most of the materials are taken from here

Nastaran Okati

Outline

- ▷ Gradient Checks
- ▷ Sanity Checks in Learning
- ▷ Monitoring the Learning Process
- ▷ Parameter Updates

1.
Gradient Checks

Gradient checks

- Comparing the analytic gradient to the numerical gradient.

- use the centered difference formula of the form:

$$\frac{df(x)}{dx} = \frac{f(x+h) - f(x-h)}{2h}$$

- *h* is a very small number, in practice approximately 1e-5 or so

- How to compare the numerical and analytic gradient?
- Use the relative error: $$\frac{\mid f'_a - f'_n \mid}{\max(\mid f'_a \mid, \mid f'_n \mid)}$$
  - relative error > 1e-2 usually means the gradient is probably wrong
  - 1e-2 > relative error > 1e-4 should make you feel uncomfortable
  - 1e-4 > relative error is usually okay for objectives with kinks. But if there are no kinks (e.g. use of tanh nonlinearities and softmax), then 1e-4 is too high.
  - 1e-7 and less you should be happy.

Gradient checking in deeper networks

- The deeper the network, the higher the relative errors will be

- So if you are gradient checking the input data for a 10-layer network, a relative error of 1e-2 might be okay because the errors build up on the way

- An error of 1e-2 for a single differentiable function likely indicates incorrect gradient.

- Use double precision

Kinks in the objective

- Kinks refer to non-differentiable parts of an objective function
- Consider gradient checking the ReLU function at x=1e-6. What might happen?
- A Neural Network with an SVM classifier will contain many kinks due to ReLUs.
- It is possible to know if a kink was crossed in the evaluation of the loss. This can be done by keeping track of the identities of the parameters

Gradient Check Tips and Tricks

- Use only few datapoints.
- Be careful with the step size h
- Don’t let the regularization overwhelm the data.
- Remember to turn off any non-deterministic effects in the network, such as dropout, random data augmentations, etc.

- A gradient check is performed at a particular (and usually random), single point in the space of parameters.

- Even if the gradient check succeeds at that point, it is not immediately certain that the gradient is correctly implemented globally.

2.

Sanity Checks before Learning

Sanity Checks Tips

- Look for correct loss at chance performance.
  - It's best to first check the data loss alone
  - For example, for CIFAR-10 with a Softmax classifier we would expect the initial loss to be 2.302, because we expect a diffuse probability of 0.1 for each class (since there are 10 classes), and Softmax loss is the negative log probability of the correct class so: $-\ln(0.1) = 2.302$.
  - If you're not seeing these losses there might be issue with initialization.

Sanity Checks Tips

- Increasing the regularization strength should increase the loss

- Overfit a tiny subset of data
  - before training on the full dataset try to train on a tiny portion (e.g. 20 examples) and make sure you can achieve zero cost (set regularization to zero)

  - Unless you pass this sanity check with a small dataset it is not worth proceeding to the full dataset.

  - It may happen that you can overfit very small dataset but still have an incorrect implementation

3.

Monitoring the Learning Process

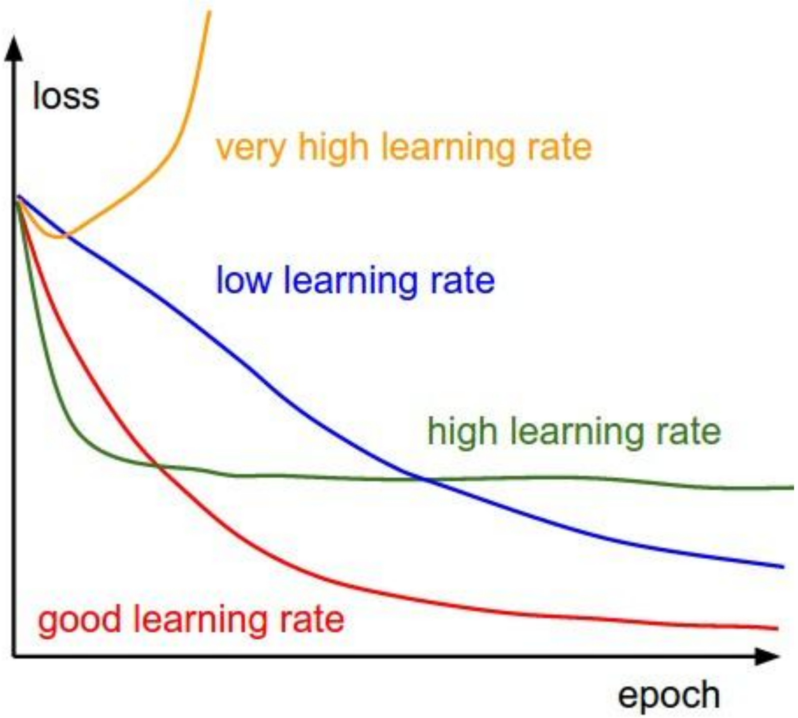
Monitoring the Learning Process

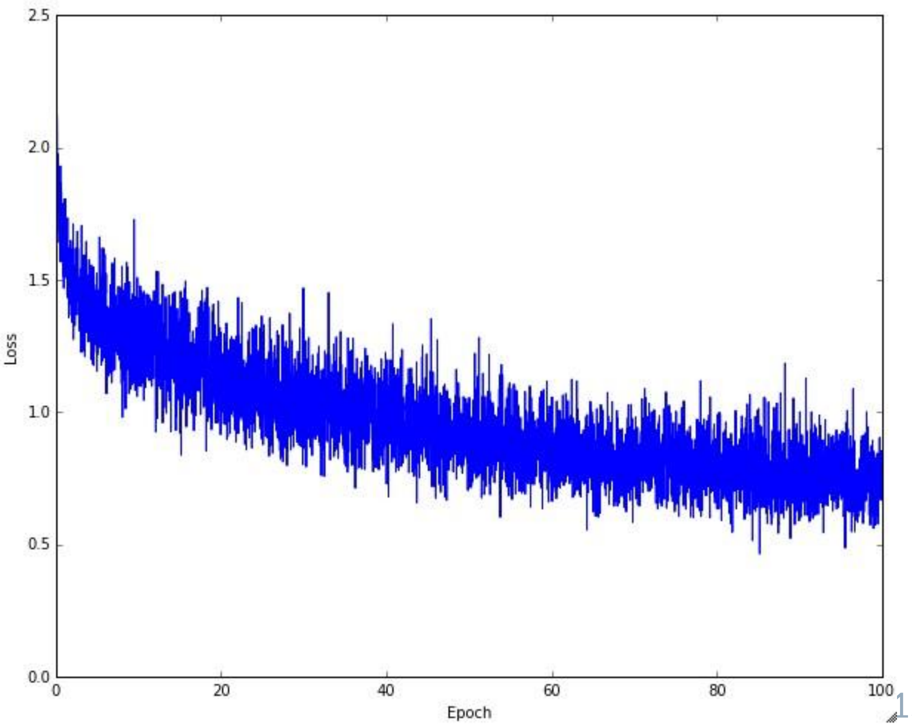
- There are multiple useful quantities you should monitor during training of a neural network
  - Loss function
  - Train/Val accuracy
  - Ratio of weights:updates

- The x-axis of the plots below are always in units of epochs

Monitoring Loss Function

The first quantity that is useful to track during training is the loss, as it is evaluated on the individual batches during the forward pass.





Monitoring Loss Function

- With low learning rates the improvements will be linear.

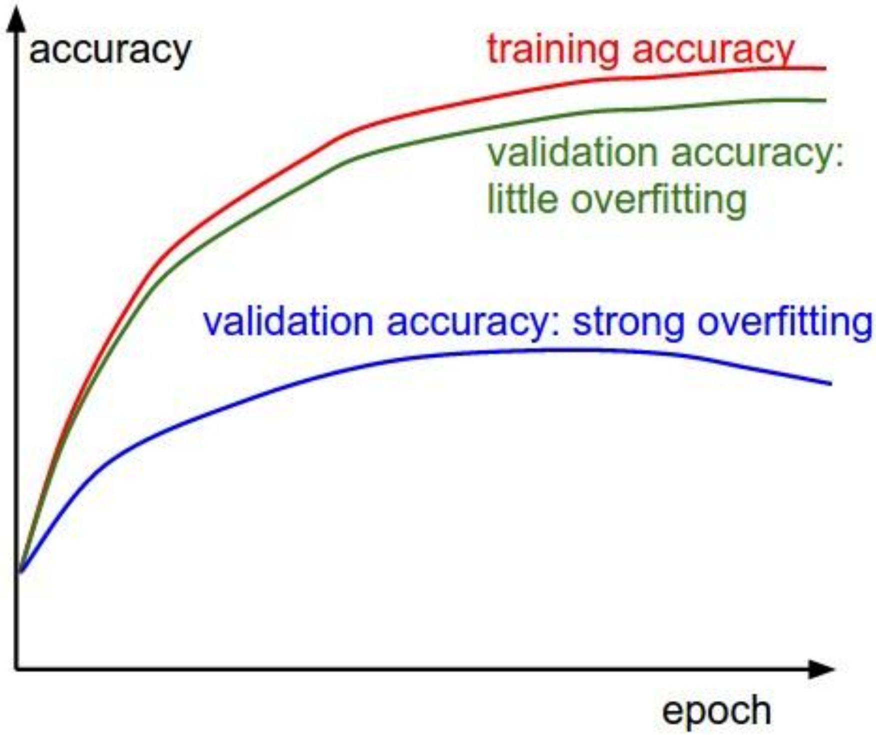
- With high learning rates they will start to look more exponential

- Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (parameters are bouncing around chaotically, unable to settle in a nice spot)

- Too much noise in the loss function may indicates that the batch size is a little too low

Monitoring Train/Val Accuracy

The second important quantity to track while training a classifier is the validation/training accuracy. This plot can give you valuable insights into the amount of overfitting in your model:



Monitoring the Ratio of weights:updates

- The last quantity you might want to track is the ratio of the update magnitudes to the value magnitudes

- What are the updates?

- A rough heuristic is that this ratio should be somewhere around 1e-3.

- If it is lower than this then the learning rate might be too low. If it is higher then the learning rate is likely too high.

4.
Parameter Updates

Annealing Learning Rate

- With a high learning rate, the system contains too much kinetic energy and the parameter vector bounces around chaotically, unable to settle down into deeper, but narrower parts of the loss function.

- When to decay the learning rate?

- Decay it slowly or aggressively?

Learning Rate Decay

- Step Decay: Reduce the learning rate by some factor every few epochs.
  - Typical values might be reducing the learning rate by a half every 5 epochs, or by 0.1 every 20 epochs.
  - Watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving.

Per-parameter adaptive learning rate methods

- All previous approaches we've discussed so far manipulated the learning rate globally and equally for all parameters.

- Tuning the learning rates is an expensive process, so much work has gone into devising methods that can adaptively tune the learning rates and even do so per parameter
  - Adagrad
  - RMSprop
  - Adam

Adagrad

- The weights that receive high gradients will have their effective learning rate reduced, while weights that receive small or infrequent updates will have their effective learning rate increased.

- Keep track of per-parameter sum of squared gradients then normalize the parameter update step, element-wise

```
>>>cache += dx**2
>>>x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

RMSprop

- Adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate.

- Modulates the learning rate of each weight based on the magnitudes of its gradients, which has a beneficial equalizing effect, but unlike Adagrad the updates do not get monotonically smaller.

```
>>>cache = decay_rate * cache + (1 - decay_rate) * dx**2
>>>x += - learning_rate * dx / (np.sqrt(cache) + eps)
```

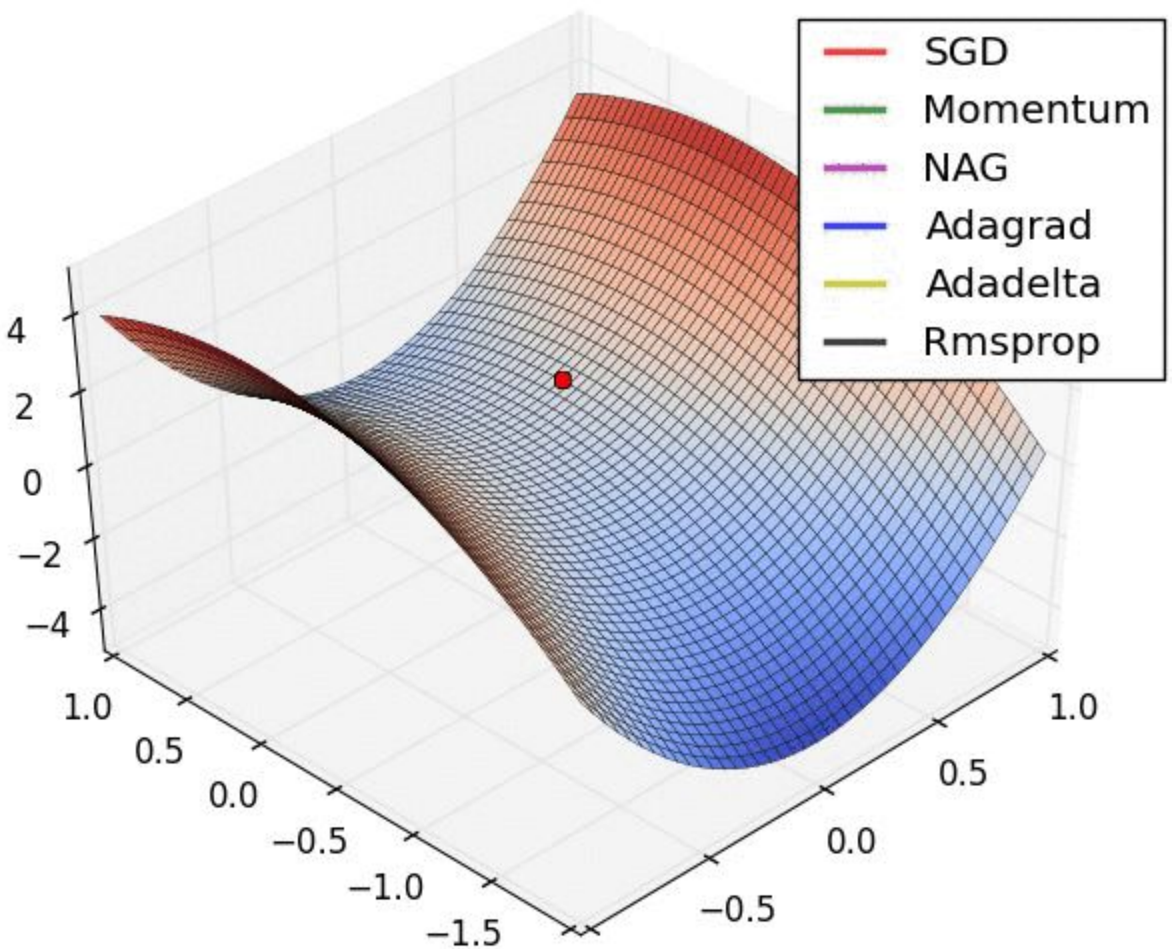
Adam

- The update looks exactly as RMSProp update, except the “smooth” version of the gradient m is used instead of the raw gradient vector dx

- Recommended values in the paper are eps = 1e-8, beta1 = 0.9, beta2 = 0.999

- In practice Adam is currently recommended as the default algorithm to use

```
>>>m = beta1*m + (1-beta1)*dx
>>>v = beta2*v + (1-beta2)*(dx**2)
>>>x += - learning_rate * m / (np.sqrt(v) + eps)
```


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

Hyperparameter Optimization

- Be careful with parameter ranges
- Prefer random search to grid search
- Stage your search from coarse to fine

4.

Evaluation

Model Ensembles

- One reliable approach to improving the performance of Neural Networks by a few percent is to train multiple independent models, and at test time average their predictions
  - Same model, different initializations
  - Top models discovered during cross-validation
  - Running average of parameters during training

- Disadvantage of model ensembles is that they take longer to evaluate on test example.

What are Convolutional Neural Networks?

Wait until next session! 😉

Thanks!

Any questions?