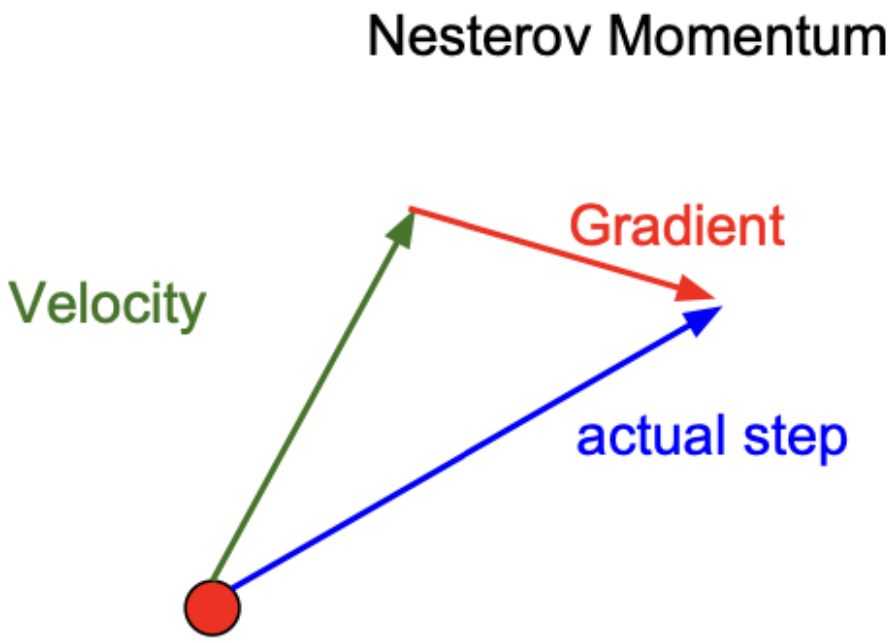
Training Neural Networks II

Fancier Optimization

1. SGD
    a. problems
        i. what if loss changes quickly in one direction and slowly in another?
            - very slow progress along shallow dimension
            - jitter along steep direction
        ii. loss function has high condition number
            - ratio of largest to smallest singular value of the Hessian matrix is large
        iii. what if the loss function has a local minima or saddle point?
            - zero gradient
            - gradient descent gets stuck
        iv. can be noisy
            - gradients come from minibatches
2. SGD + Momentum
    a. Momentum
        i. updates using velocity, gradient
        ii. actual step is the weighted average of the two (velocity, gradient)
    b. Nesterov momentum
        i. gradient is calculated at the point velocity ends
        ii. actual step is the mix of the two (velocity, gradient)



3. AdaGrad
    a. add element-wise scaling of the gradient based on the historical sum of squares in each dimension

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

    b. what happens to the step size over long time?
        i. step size gets smaller over time
        ii. for convex case, this is ok but in non-convex case, this is problematic
4. RMSProp
    a. improve the AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

5. Adam

a. sort of like RMSProp with momentum

   i. almost

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment + (1 - beta1) * dx # momentum
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx # AdaGrad / RMSProp
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7)
```

   ii. full form

```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment + (1 - beta1) * dx # momentum
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx # AdaGrad / RMSProp
  first_unbias = first_moment / (1 - beta1 ** t) # bias correction
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7)
```

b. what happens at first timestep?

   i. the second moment is initialized to 0 and updated, still near 0

   ii. divide by the second moment, so large step size at the beginning

c. bias correction

   i. b/c first and second moment estimates start at zero

d. starting point for many models

   i. beta1 = 0.9, beta2 = 0.999, and learning rate = 1e-3 or 5e-4

6. Learning rate decay

   a. all optimization methods have learning rate as a hyperparameter

   b. need to find the best learning rate

   c. learning rate decay over time

      i. more critical with SGD + momentum

      ii. less common with Adam

7. First-order optimization

    a. use gradient form linear approximation
    b. step to minimize the approximation
8. Second-order optimization
    a. use gradient and Hessian to form quadratic approximation
    b. step to the minima of the approximation
    c. Newton parameter update
        i. no hyperparameters, no learning rate for vanila version
        ii. this is bad because the required memory is too big
    d. BFGS (Quasi-Newton method)
        i. approximate inverse Hessian with rank 1 updates over time, instead of inverting the Hessian
    e. L-BFGS
        i. does not store the full inverse Hessian
        ii. usually works very well in full batch, deterministic mode
        iii. does not transfer very well to mini-batch setting
9. Better optimization algorithms help reduce training loss
    a. but we really care about error on new data
    b. how to reduce the gap?
        i. model ensembles (multiple independent models)
        ii. regularization (single model)

Model Ensembles

1. Overall process
    a. train multiple independent models
    b. at test time, average their results
2. use multiple snapshots of a single model during training
    a. instead of training independent models
3. keep a moving average of the parameter vector and use that at test time (Polyak averaging)
    a. instead of using actual parameter vector

Regularization

1. Common use
    a. L2 regularization (weight decay)
    b. L1 regularization
    c. Elastic net (L1 + L2)
2. Dropout
    a. summary
        i. drop in forward pass
        ii. scale at test time
    b. in each forward pass, randomly set some neurons to zero
        i. commonly use 0.5 as a probability of dropping hyperparameter
    c. prevents co-adaptation of features
        i. forces the network to have a redundant representation
    d. dropout is training a large ensemble of models (that share parameters)
    e. approximate the integral
        i. integral is used to average out the randomness at test-time
        ii. multiply by dropout probability at test time
    f. inverted dropout
        i. at forward pass, divide by p
        ii. test time is unchanged
3. Batch normalization
    a. training
        i. normalize using stats from random minibatches
    b. testing
        i. use fixed stats to normalize
4. Data augmentation
    a. transform image without changing the label
    b. example

        i. horizontal flip
        ii. random crop and scale
        iii. color jitter
5. Other examples
    a. DropConnect
        i. zero out from the weight matrix
    b. Fractional max pooling
    c. Stochastic depth

Transfer Learning

1. Summary
    a. train on Imagenet
    b. apply to small dataset (C classes)
        i. reinitialize the last layer randomly
        ii. train with the weights of all other layers freezed
    c. apply to larger dataset
        i. reinitialize a few layers randomly
        ii. train with the weights of all other layers freezed
        iii. lower training rate when finetuning 1/10 of original learning rate is good starting point
2. useful when the size of training dataset is too small
    a. find a very large dataset that has similar data
    b. train a big ConvNet there
    c. transfer learn to the dataset