Neural Network Choices Discussed so far

- cost functions
- activation function

  sigmoid

  tanh

  ReLU - typically most common now in hidden layers

  ↳ reduced likelihood of vanishing gradient

  ↳ sparsity → more likely to have inputs for which output is 0

  ↳ more computationally efficient

Learning Rate:

- as mentioned last week:
  a) can try range of learning rates
  b) can do some sort of decreasing/adaptive learning rate
     - use function of error to determine learning rate
     - specific learning rates for each parameter
     - set scaling to decrease learning rate every iteration or epoch

Commonly used adaptive learning rates

- Adagrad → takes into account update history of parameters
  decay learning rate for parameters updated a lot
  ↳ diminishes rapidly
- RMSprop → modifies adagrad to not decay quite so rapidly
- Adam → like RMS prop with momentum (look at cumulative history of gradients)
  estimate 1st and 2nd moments of gradients and use to update

Momentum

- another way to improve convergence
- idea: don't want to drastically change directions each time step
- so, keep in aspect of direction that previous gradients sent you

$$\Delta w_i^t = -\alpha \frac{dE^t}{dw_i} + \beta \Delta w_i^{t-1}$$

But $\Delta w_i^{t-1} = w_i^{t-2} + -\alpha \frac{dE^t}{dw_i} + \beta \Delta w_i^{t-2}$

$\vdots$

so keeps some (small) dependence all the way back through weight updates

Prevent Overfitting / overtraining

- neural networks often have lots of parameters
  - d = input dimension
  - H = hidden layer dimension
  - k = # of outputs
  - ↳ fully connected neural net has $H(d+1) + k(H+1)$ parameters
    - ↳ often quite a few particularly for datasets with many features

want to make sure we don't overfit
- more parameters (hidden units) aren't always better

overtraining
- ↳ one issue training for too long. actually start overfitting

idea: monitor error (every epoch or couple epochs) on validation set
- ↳ training error will keep going down, error will eventually start going

Other Validation Uses:

(validation set or cross validation)

- determining # of hidden units
- determining learning rate

What structure to use:

- often 1 hidden layer is sufficient
  - ↳ good purposes/applications to use more (topic for next week)
- number of hidden units
  - between size of input and size of output layer
  - no good be all/end all rule of thumb
    - average of $n_{in} + n_{out}$
    - scaled sum of $(n_{in} + n_{out}) \cdot$ factor e.g. $2/3(n_{in}+n_{out})$
  - limiting # of hidden units will help prevent overfitting

Optimizing Network Configuration

- pruning: techniques to identify and remove nodes from network that can be removed without a large impact on network performance
  - penalty methods -> add penalty term to objective function to try to drive small weights to 0
  - remove connections/nodes with small weights
  - many other methods/active research area
- growing: start with simple, small structure and progressively add on neurons
- sometimes they are combined

How else to improve performance?

prevent overfitting:

a) stop early to prevent overtraining

b) limit # of hidden units

c) limit size of weights

↳ regularization

↳ add on term to cost function

$$L2 = \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

$$L1 = \lambda \|\mathbf{w}\|_1$$

d) dropout

↳ more common in deep learning which we'll talk about next week

Data Preprocessing

- important for good performance
- normalization - feature scaling

continuous

$$\hat{X}_i = \frac{X_i - \mu}{\sigma^2}$$

discrete

$$\hat{X}_i = \frac{X_i - X_{min}}{X_{max} - X_{min}}$$

- encoding of categorical data
  - ↳ same choices as before