    

Overfitting

* Use a **validation set** to see how your model, which was trained on the training data only, generalizes to other data to prevent overfitting. You should calculate loss here for logging, but don’t use the loss to teach the neural network. *Use this separate validation set in our calculateLoss fcn.*
* You can also reduce the complexity of the model by reducing the number of layers or neurons in the layer to better generalize to unseen data

**Regularization**: reduce overfitting and variance in network by penalizing for complexity

* Trade accuracy on training data for ability to predict better on data it hasn’t seen before
* Add a term to the loss function that penalizes for large weights (loss -> loss + x). This influences our weights to be closer to 0 and effectively simplifies the model by reducing the impact of some of the weights.
* L2 Regularization: x = sum of squares of norms of the weight matrices for each layer \* lambda / 2 \* # inputs, where lambda is the regularization term
  + Norm of a matrix = square root of the sum of the squares of its elements
  + See here for matrix norm: http://mathworld.wolfram.com/FrobeniusNorm.html
  + See here for L2 regularization formula: <https://www.youtube.com/watch?v=iuJgyiS7BKM&index=12&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU>

Learning Rate

* Large learning rate can overshoot the minimum loss

Batch Size: the number of training samples we pass through the network at a time

* Larger batch size -> faster training, less quality *(Why ???)*

Vanishing/Exploding Gradient Problem

* Earlier layers’ gradients are calculated using a product of derivatives from earlier layers. If these multiplicands are small, the gradient becomes very small (vanishes) and is unable to significantly affect the value of the weight. If these multiplicands are large, the gradient becomes very large (explodes) and continually overshoots the optimal value for reducing loss

Weight Initialization

* If first layer of weights is initialized with a random normalized distribution around a mean of zero, the variance of the input layer is 1. But the variance of the next layer is the sum of the variances from the input layer. Thus, if there are 250 inputs, the variance of the nodes in the next layer is 250. When this is transformed in a sigmoid activation function, it can make the weights so close to 0 or 1 that they cannot be adjusted significantly.
* Ideal variance is 1/n, so multiply weights by 1/sqrt(n) (2/sqrt(n) for ReLu activation fcn), where n is # nodes in the previous layer

Reinforcement Learning

* Instead of actual values of y as training data, we use an estimate of the actual value function: actual\_y = V(s) = *r* + γ*V*(*s*’;*θ*)
* So the loss function becomes sum(*r* + γ*V*(*s*’;*θ*) – V(s) )^2)