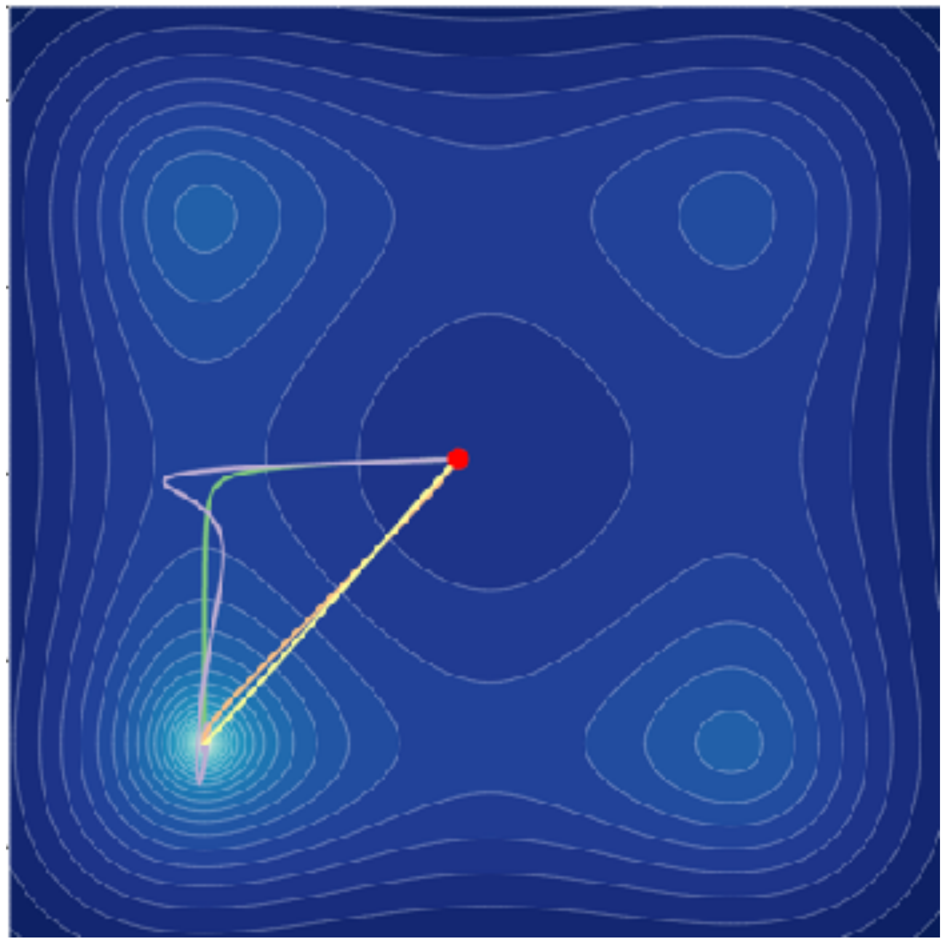
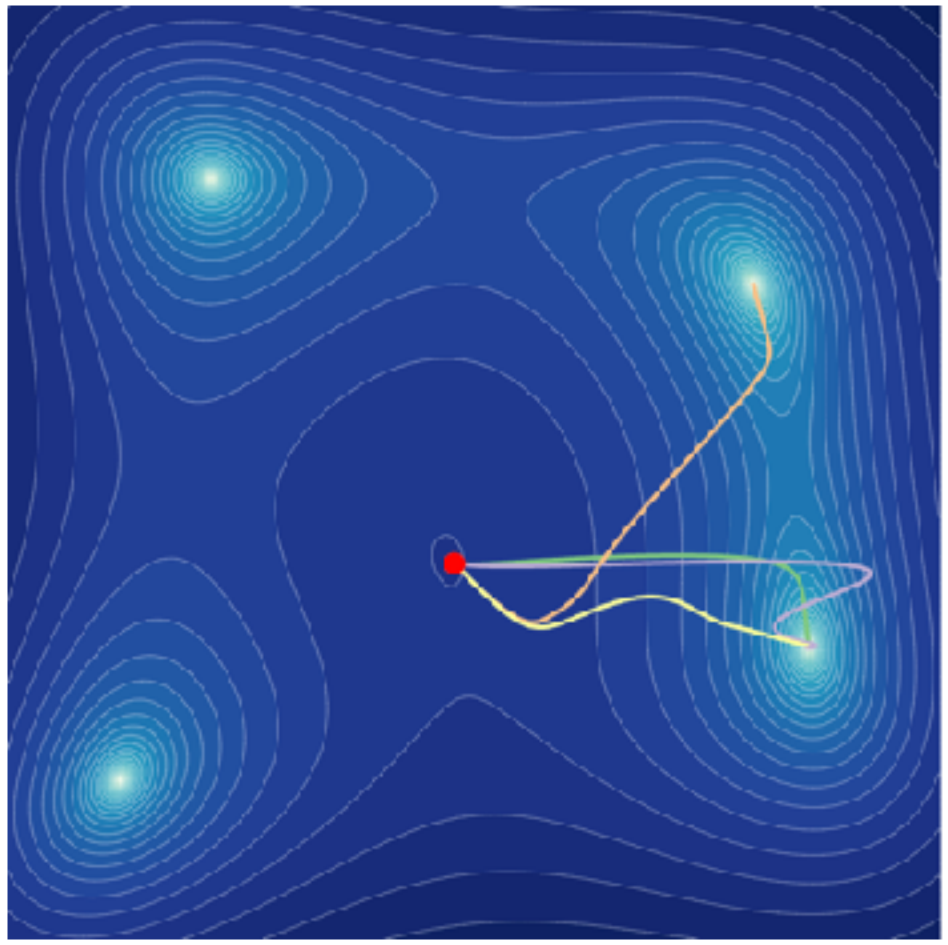
Optimization

How do we optimize the parameters of our neural networks to achieve low loss on our training data?





- *Text from http://ruder.io/optimizing-gradient-descent/*
- *Images from Katanforoosh and Kunin 2018*

Optimization Methods

1. **Gradient Descent**:

   - Updates parameters in the opposite direction of their gradient.
   - three variants (**stochastic**, **mini batch**, **full batch**), which differ in how much data used to compute the gradient. Results in a trade-off between the accuracy and time

2. **Momentum**:

   - accelerates gradient descent in relevant directions and dampens oscillations by keeping track of previous updates which are added to the current update

4. **RMSprop**:

   - divides the learning rate by an average of squared gradients in order to adapt the learning rate to the parameters
   - smaller updates for parameters associated with large gradients and larger updates for parameters with small gradients

6. **Adam**:

   - Performs adaptive learning rate similar to RMSprop
   - Uses past gradients similar to momentum

Which of the following variants of gradient descent would lead to the most variance in the updates?

A. Stochastic Gradient Descent

B. Minibatch Gradient Descent

C. Full batch Gradient Descent

Which of the following is **not true** about classic gradient descent?

A. If the learning rate is too high gradient descent can diverge

B. For convex (bowl-shaped) losses gradient descent will never converge

C. When choosing stochastic and mini batches they should be chosen randomly from the training set

D. Without annealing the learning rate gradient descent can bounce around a local minimum

Which of the following optimizers will always converge to the global minimum of a loss landscape

A. RMSprop

B. Gradient Descent

C. Momentum

D. Adam

E. None of the above

Which method would you expect to be the **least** successful at navigating a ravine ("areas where the surface curves much more steeply in one dimension than in another")

A. RMSprop

B. Gradient Descent

C. Momentum

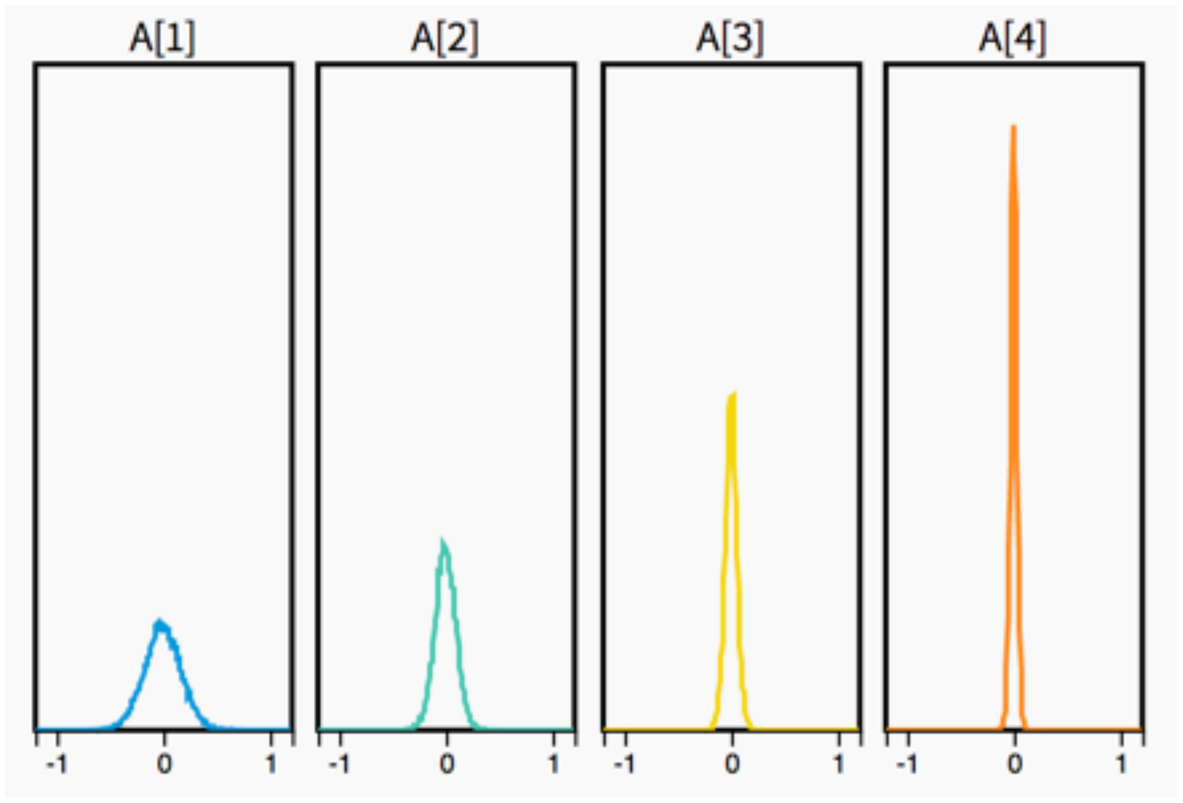
D. Adam

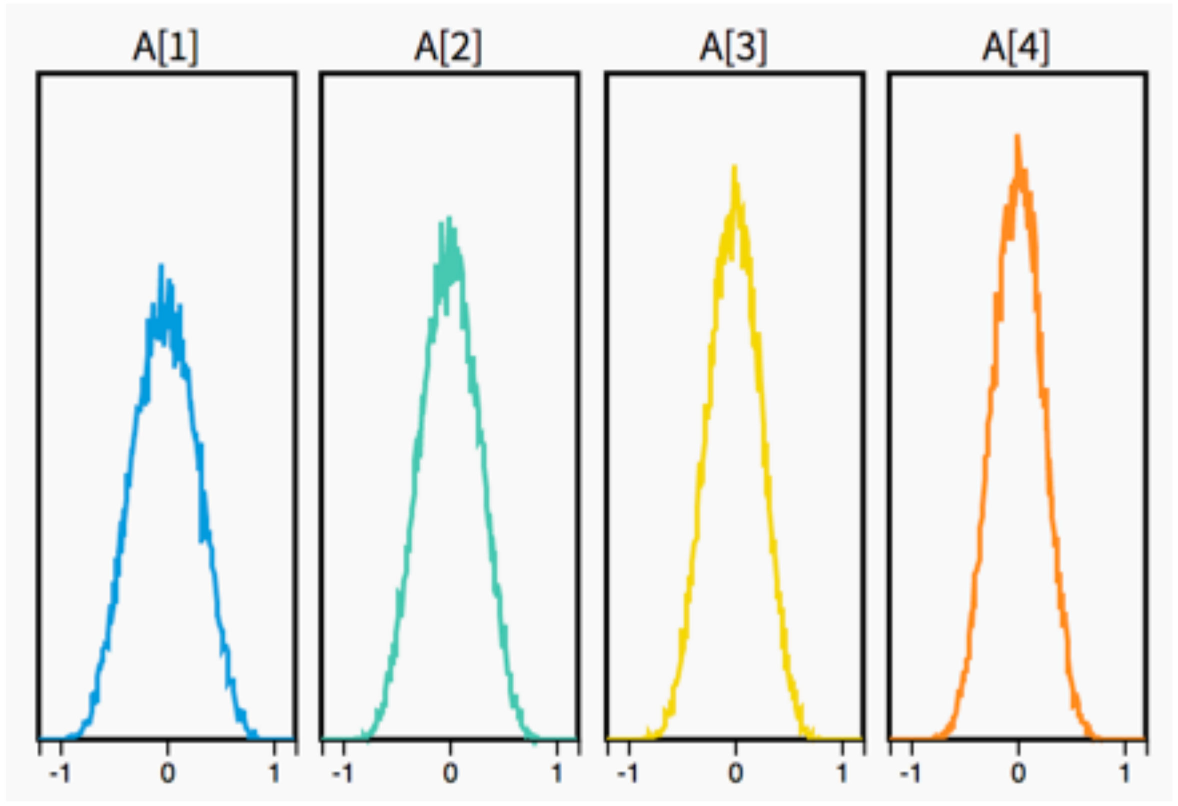
You have reason to believe that your parameters have very different magnitudes of influence on the loss function. Which method(s) would you consider using to train your network?

A. RMSprop
B. Gradient Descent
C. Momentum
D. Adam

Initialization

How do we choose an *appropriate* initialization point so we will train quickly and effectively?

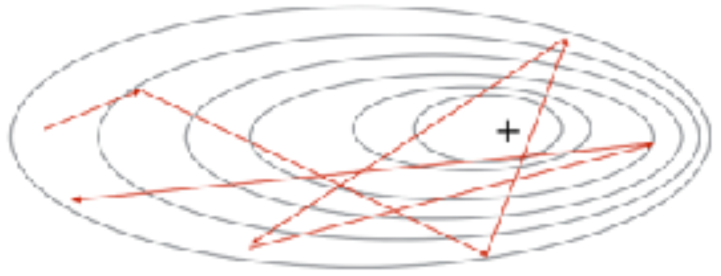




- *Images from Katanforoosh and Kunin 2018*

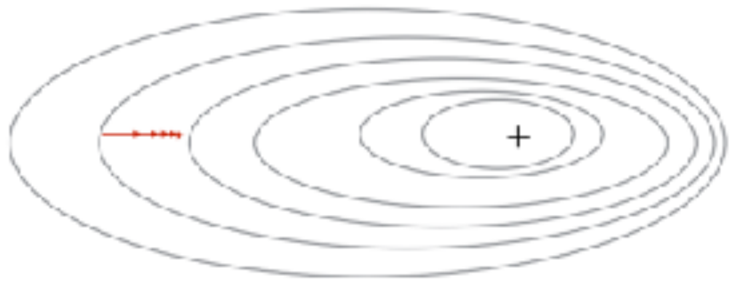
Motivation: Prevent exploding or vanishing gradients when training deeper networks.

Exploding



$$W^{[1]} = W^{[2]} = \cdots = W^{[L-1]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

Vanishing



$$W^{[1]} = W^{[2]} = \cdots = W^{[L-1]} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

Goal: The variance should stay the same across every layer to prevent the signal from vanishing or exploding.

Assumptions:

1. Weights are normally distributed around zero and biases are zero
2. Weights are independent and identically distributed
3. Inputs are independent and identically distributed
4. Weights and inputs are mutually independent

Xavier Initialization

$$Var(a_i^{[\ell-1]}) = Var(a_i^{[\ell]})$$

$$= Var(z_i^{[\ell]}) \longleftarrow$$

linearity of **tanh** around zero

$$tanh(z) \approx z$$

$$= Var(\sum_{j=1}^{n^{[\ell-1]}} w_{ij}^{[\ell]} a_j^{[\ell-1]})$$

$$= \sum_{j=1}^{n^{[\ell-1]}} Var(w_{ij}^{[\ell]} a_j^{[\ell-1]}) \longleftarrow$$

variance of independent sum

$$Var(X+Y) = Var(X) + Var(Y)$$

$$= \sum_{j=1}^{n^{[\ell-1]}} E[w_{ij}^{[\ell]}]^2 Var(a_j^{[\ell-1]}) + E[a_j^{[\ell-1]}]^2 Var(w_{ij}^{[\ell]}) + Var(w_{ij}^{[\ell]}) Var(a_j^{[\ell-1]}) \longleftarrow$$

variance of independent product

$$Var(XY) = E[X]^2 Var(Y) + E[Y]^2 Var(X) + Var(X)Var(Y)$$

$$= n^{[\ell-1]} Var(w_{ij}^{[\ell]}) Var(a_j^{[\ell-1]}) \implies Var(W) = \frac{1}{n^{[\ell-1]}}$$

Your friend suggests a new goal for initialization. Instead of keeping the variance of the activations constant we should keep the variance of the weights constant between layers. Is this a good idea?

No, the motivation of keeping the variance of the activations the same between layers is to prevent an exploding or vanishing gradient by limiting how much the activations can change. Keeping the variance of the weights the same will not limit the change in the activations between layers.

The proof of Xavier Initialization assumed we used **tanh** as our non-linear function. What assumption would we have to change in the proof if we used the **relu** function?

We can no longer assume that the activations are linear around zero because the relu function is not symmetric. This is actually the motivation for *He initialization.*