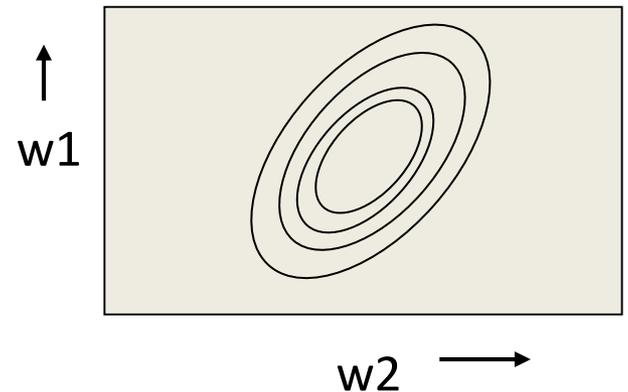
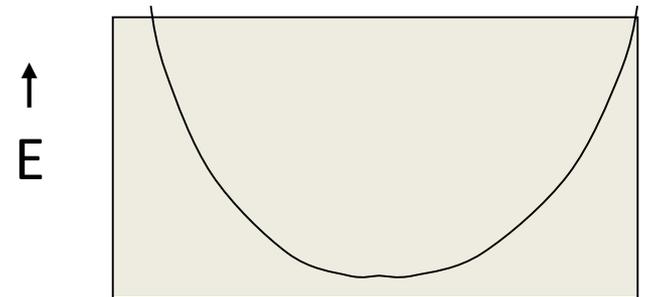


# Intro to Optimizing Neural Networks



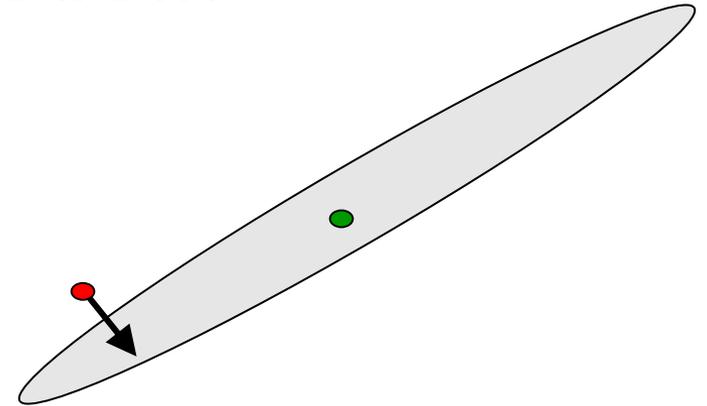
# The Surface Error For Neural Networks

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - For a linear neuron with a squared error, it is a quadratic bowl.
- For multi-layer, non-linear nets the error surface is much more complicated.
  - But locally, a piece of a quadratic bowl is usually a very good approximation.



# Convergence speed of full batch learning when the error surface is a quadratic bowl

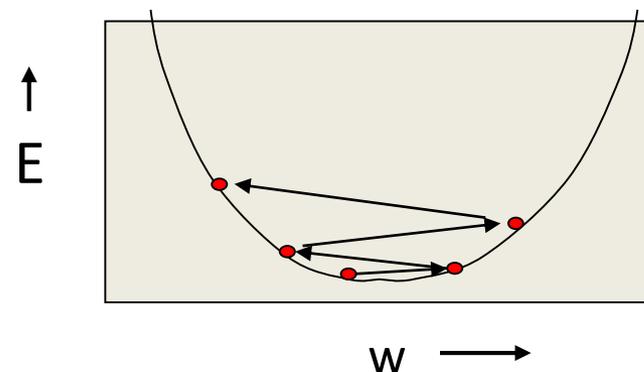
- Going downhill reduces the error, but the direction of steepest descent does not point at the minimum unless the ellipse is a circle.
  - The gradient is big in the direction in which we only want to travel a small distance.
  - The gradient is small in the direction in which we want to travel a large distance.



Even for non-linear multi-layer nets, the error surface is locally quadratic, so the same speed issues apply.

# How Learning Goes Wrong

- If the learning rate is big, the weights slosh to and fro across the ravine.
  - If the learning rate is too big, this oscillation diverges.
- What we would like to achieve:
  - Move quickly in directions with small but consistent gradients.
  - Move slowly in directions with big but inconsistent gradients.



# Mini-Batch Stochastic Gradient Descent

- Instead of minimizing the cost function  $\sum_{i=1}^M \mathcal{C}(y^{(i)}, f_{\theta}(x^{(i)}))$ , make a step along the gradient with respect to just a few examples
  - Repeat:
    - Select random mini-batch  $S$  of training examples (size e.g. 50, but could be 1)
    - $\theta \leftarrow \theta - \alpha \frac{\partial}{\partial \theta} \sum_{i \in S} \mathcal{C}(y^{(i)}, f_{\theta}(x^{(i)}))$
- (Perhaps) helps avoid bad local minima because the direction of the current gradient changes all the time
  - (Note: in deep neural networks, we're not so worried about bad minima)
- Don't need to store all the data in RAM
  - Useful a lot of the time!
- Minibatches need to be balanced for class
  - If a minibatch only contains images of class "Radcliffe," the network might decide to always output "Radcliffe" after the gradient update
  - Smaller alphas/smaller minibatches also help

# Adjusting the $\alpha$

- Idea: have each weight have its own individual  $\alpha$
- Set the so  $\alpha$ s that the optimization makes sense (i.e., if gradient updates make things worse, make  $\alpha$  smaller, if they make it better, make  $\alpha$  larger)

# rmsprop

- Keep a moving average of the squared gradient for each weight:

$$\begin{aligned} & \textit{MeanSquare}(w, t) \\ &= .9\textit{MeanSquare}(w, t - 1) + .1 \left( \frac{\partial E}{\partial w}(t) \right)^2 \end{aligned}$$

- Divide the gradient by  $\sqrt{\textit{MeanSquare}(w, t)}$

# Weight Initialization

- Extremely important for Multilayer Neural Networks!
- *Not* all zeros
  - If all the neurons in a layer are the same, they can only change in the same direction by the same amount
- Small random numbers
  - Not *too* small, since that might cause the gradient to be small
  - Called “symmetry breaking”
  - Good enough for CSC411
- Heuristic: random numbers that depend on the number of incoming weights:
  - $w \sim N(0,1)/\sqrt{n}$ . This makes the inputs to all the units initially be on approximately the same scale
- Can set biases to 0
  - Symmetry breaking provided by the weight initialization



**ML Hipster**  
@ML\_Hipster

 Follow

Everyone is all big data this and online that. My methods are small batch: they only handle a few instances but really look at them, y'know?

6:28 PM - 16 Aug 2012

  17  6