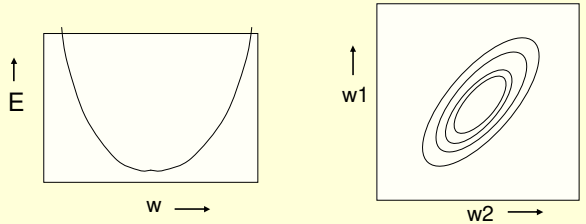
CSC321: Neural Networks

Lecture 10: Speeding up the Learning

Geoffrey Hinton

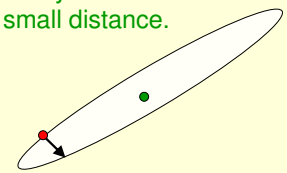
The error surface for a linear neuron

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - It is a quadratic bowl.
    - i.e. the height can be expressed as a function of the weights without using powers higher than 2. Quadratics have constant curvature (because the second derivative must be a constant)
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses.



Convergence speed

- 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.



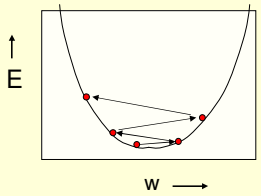
  - The gradient is small in the direction in which we want to travel a large distance.

$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

This equation is sick. The RHS needs to be multiplied by a term of dimension w^2 to make the dimensions balance.

How the learning goes wrong

- If the learning rate is big, it sloshes to and fro across the ravine. If the rate is too big, this oscillation diverges.
- How can we move quickly in directions with small gradients without getting divergent oscillations in directions with big gradients?



Five ways to speed up learning

- Use an adaptive global learning rate
  - Increase the rate slowly if its not diverging
  - Decrease the rate quickly if it starts diverging
- Use separate adaptive learning rate on each connection
  - Adjust using consistency of gradient on that weight axis
- Use momentum
  - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
- Use a stochastic estimate of the gradient from a few cases
  - This works very well on large, redundant datasets.
- Don't go in the direction of steepest descent.
  - The gradient does not point at the minimum.
    - Can we preprocess the data or do something to the gradient so that we move directly towards the minimum?

The momentum method

Imagine a ball on the error surface with velocity v.

- It starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
  - It damps oscillations by combining gradients with opposite signs.
  - It builds up speed in directions with a gentle but consistent gradient.
  - On an inclined plane it reaches a terminal velocity.

$$v(t) = \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$\Delta w(t) = v(t)$$

$$= \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$= \alpha\, \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

$$v(\infty) = \frac{1}{1-\alpha}\left(-\varepsilon \frac{\partial E}{\partial w}\right)$$

Adaptive learning rates on each connection

- Use a global learning rate multiplied by a local gain on each connection.
- Increase the local gains if the gradient does not change sign.
- Use additive increases and multiplicative decreases.
  - This ensures that big learning rates decay rapidly when oscillations start.

$$\Delta w_{ij} = -\varepsilon\, g_{ij} \frac{\partial E}{\partial w_{ij}}$$

$$if \quad \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1)\right) > 0$$

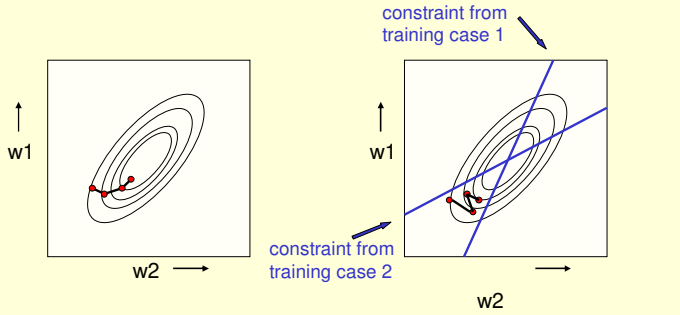
$$then \quad g_{ij}(t) = g_{ij}(t-1) + .05$$

$$else \quad g_{ij}(t) = g_{ij}(t-1) * .95$$

Online versus batch learning

- Batch learning does steepest descent on the error surface

Online learning updates the weights after each training case. It zig-zags around the direction of steepest descent.



Stochastic gradient descent

- If the dataset is highly redundant, the gradient on the first half is almost identical to the gradient on the second half.
  - So instead of computing the full gradient, update the weights using the gradient on the first half and then get a gradient for the new weights on the second half.
  - The extreme version is to update the weights after each example, but balanced mini-batches are just as good and faster in matlab.

Newton's method

- Newton did not have a computer. So he had a lot of motivation to find efficient numerical methods.
- The basic problem is that the gradient is not the direction we want to go in.
  - If the error surface had circular cross-sections, the gradient would be fine.
  - So lets apply a linear transformation that turns ellipses into circles.
- The covariance matrix of the input vectors determines how elongated the ellipse is.

Covariance Matrices

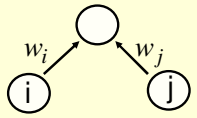
- For each pair of input dimensions we can compute the covariance of their values over a set of data vectors.
- These covariances can be arranged in a symmetric matrix.
- The terms along the diagonal are the variances of the individual input dimensions.

$$Cov(x_i, x_j) = \frac{1}{N}\sum_{n=1}^{N}(x_{i,n} - \mu_i)(x_{j,n} - \mu_j)$$

	i	j	k
i		c_{ij}	
j	c_{ji}		
k			V_k

$$Variance(x_i) = \frac{1}{N}\sum_{n=1}^{N}(x_{i,n} - \mu_i)^2$$

Fixing up the error surface



Suppose that inputs i and j are highly correlated. Changing wi will change the error in the same way as changing wj.

So changes in wi will have a big effect on $\partial E / \partial w_j$

- So multiply each input vector by the inverse of the covariance matrix.
  - The preprocessed input vectors will now have uncorrelated components.
  - So the error surface will have circular cross-sections

Curvature Matrices (optional material!)

- Each element in the curvature matrix specifies how the gradient in one direction changes as we move in some other direction.
- For a linear network with a squared error, the curvature matrix of the error, E, is the covariance matrix of the inputs.
- The reason steepest descent goes wrong is that the ratio of the magnitudes of the gradients for different weights changes as we move down the gradient.

	i	j	k
i		$\frac{\partial\left(\partial E/\partial w_i\right)}{\partial w_j}$	
j	$\frac{\partial\left(\partial E/\partial w_j\right)}{\partial w_i}$		
k			$\frac{\partial^2 E}{\partial w_k^2}$

Another, more general way to fix up the error surface

- We can leave the error surface alone, but apply the correction to the gradient vector.
- We multiply the vector of gradients by the inverse of the curvature matrix. This produces a direction that points straight at the minimum for a quadratic surface.
- The curvature matrix has too many terms to be of use in a big network. Maybe we can get some benefit from just using the terms along the leading diagonal (Le Cun).

Extra problems that occur in multilayer non-linear networks

- If we start with a big learning rate, the bias and all of the weights for one of the output units may become very negative.
  - The output unit is then very firmly off and it will never produce a significant error derivative.
  - So it will never recover (unless we have weight-decay).
- In classification networks that use a squared error, the best guessing strategy is to make each output unit produce an output equal to the proportion of time it should be a 1.
  - The network finds this strategy quickly and takes a long time to improve on it. So it looks like a local minimum.