CS 5600/6600: F20: Intelligent Systems

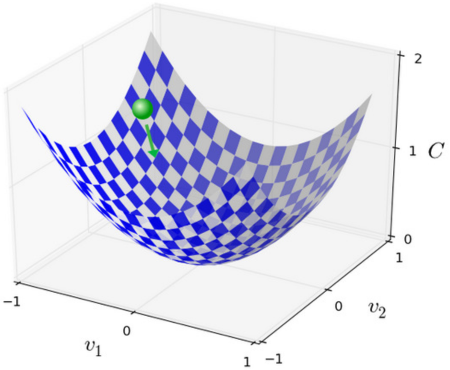
Gradient Descent and Backpropagation: Part 01

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

Cost Function Minimization

Suppose that we have to minimize some cost function $C(v_1, v_2)$.



Ball Analogy

We would like to find the point where C achieves its global minimum.

We can think of an optimization function, such as C, as a valley and imagine a ball rolling down the slope of the valley.

Theoretically, the ball will eventually roll down to the bottom of the valley. We can randomly choose a starting point for this imaginary ball and make the ball roll down and hope that the ball will reach the actual bottom.

Gradient of C

What happens when we move the ball a small amount Δv_1 in the v_1 direction and a small amount Δv_2 in the v_2 direction. Here is what calculus tells us:

$$\Delta C \approx \frac{\partial C}{\partial v_1} \Delta v_1 + \frac{\partial C}{\partial v_2} \Delta v_2.$$

We need to find a way of choosing Δv_1 and Δv_2 to make ΔC negative, i.e., to make the ball roll down.

Let's define two mathematical objects that we'll help us do that.

Gradient of C

We define the **vector of changes** Δv:

$$\Delta v \equiv (\Delta v_1, \Delta v_2).$$

We define the **gradient** of C to be the vector of partial derivatives:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2} \right).$$

Expressing ΔC in Terms of ∇C and Δv

We can rewrite ΔC in terms of ∇C and Δv.

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 = \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right) \cdot (\Delta v_1, \Delta v_2)^T = \nabla C \cdot \Delta v^T.$$

We can now choose Δv to be negative, e.g., $\Delta v = -\eta \nabla C$, where η is a small positive parameter known as the *learning rate*. Then

$$\Delta C \approx \nabla C \cdot -\eta \nabla C = -\eta ||\nabla C||^2,$$

which guarantees that $\Delta C \leq 0$, because $||\nabla C||^2 \geq 0$.

The Law of Ball Motion

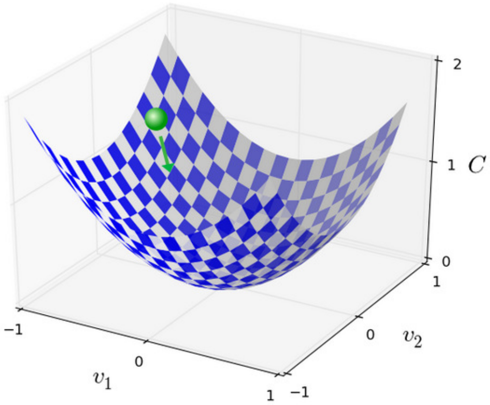
The law of motion for our imaginary ball is to update the ball's position $v = (v_1, v_2)$ as follows:

$$v \to v' = v - \eta \nabla C = \left(v_1 - \eta \frac{\partial C}{\partial v_1}, v_2 - \eta \frac{\partial C}{\partial v_2} \right).$$

In other words, we repeatedly compute ∇C and use the above update rule to move the ball down.

The Law of Ball Motion

We keep moving this imaginary ball over and over and decreasing C until we reach a global minimum.



Minimization is Expensive

Minimizing a 2D cost function, this may be as easy as eyeballing its graph.

We can use calculus to find the minimum analytically and compute derivatives to find where C has an extremum.

But, ANN cost functions are functions of dozen variables, at least. Deeper ANNs may have cost functions that depend on billions of weights and biases. Recall **the curse of dimensionality**.

Moving to Optimization Functions of Many Variables

The ball analogy works when C is a function of many more variables. Let C be a function of m variables $v_1, ..., v_m$. The change ΔC in C produced by a small $\Delta v = (\Delta v_1, ..., \Delta v_m)$ is

$$\Delta C \approx \nabla C \cdot \Delta v,$$

where

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, ..., \frac{\partial C}{\partial v_m} \right),$$

$$\Delta v = -\eta \nabla C,$$

and

$$v \to v' = v - \eta \nabla C.$$

Outline

Gradient Descent

Gradient descent algorithm is a computational method of changing the position v to find a minimum of the function C by using the equations on the previous slide.

Caveat: The update rule doesn't always work - there are cases when the global minimum of C is not reached. In practice, gradient descent works well to help ANNs to learn.

Mean Squared Error (MSE)

A common way to measure how far the actual output of the network a is from the desired output $y(x)$, aka *target*, is to use the quadratic cost function, aka *mean squared error* (MSE), where x ranges over all inputs, i.e., training examples, $w = (w_1, ..., w_n)$ is a vector of ANN weights, and $b = (b_1, ..., b_n)$ is a vector of ANN biases.

$$C(w, b) = \frac{1}{2n} \sum_x ||y(x) - a||^2 = \frac{1}{n} \sum_x \frac{||y(x) - a||^2}{2} = \frac{1}{n} \sum_x C_x,$$

where

$$C_x = \frac{||y(x) - a||^2}{2}.$$

The aim of ANN training is to minimize the cost $C(w, b)$ as a function of the weights and biases.

Modifying the Update Rules for MSE

The new position $C(w, b)$ has two components: the weights $w = (w_1, ..., w_n)$ and the biases $b = (b_1, ..., b_n)$. Let's rewrite the gradient descent rule in terms of these components:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k},$$

$$b_j \rightarrow b'_j = b_j - \eta \frac{\partial C}{\partial b_j}.$$

A Challenge for the Gradient Descent Algorithm

Let's take a look at the gradient descent rule $C(w, b)$ one more time.

$$C(w, b) = \frac{1}{n} \sum_x \frac{||y(x) - a||^2}{2} = \frac{1}{n} \sum_x C_x,$$

where

$$C_x = \frac{||y(x) - a||^2}{2}.$$

In practice, to compute ∇C we have to compute ∇C_x for each input x separately and then average them as $\nabla C = \frac{1}{n} \sum_x \nabla C_x$. It gets expensive when we have lots of training inputs.