CME 216, ME 343 - Spring 2020

Eric Darve, ICME

Stanford University

In this lecture we are going to look at methods to compute the gradient of the loss function.

As we have seen, DL problems are formulated as a problem of minimizing a loss function L that depends on the weights and biases of the DNN.

To do that, we will use gradient-based algorithms. That is we update the weights and biases using

$$\Delta W^{(k)} = -\alpha \frac{\partial L}{\partial W^{(k)}}$$

$$\Delta b^{(k)} = -\alpha \frac{\partial L}{\partial b^{(k)}}$$

$\boldsymbol{W}^{(k)}$: weights of layer $\boldsymbol{k}$.

$\boldsymbol{b}^{(k)}$: biases of layer $\boldsymbol{k}$.

We will see more advanced optimization methods later, but they all require the computation of the gradient of the loss function with respect to the weights and biases.

Let's see how the gradient can be computed.

An efficient algorithm has been designed for this.

It's called the back-propagation algorithm.

As we have seen, a DNN is a function of the type

$$y(x) =$$

$$W^{(n)}\phi \odot W^{(n-1)} \dots \phi \odot W^{(2)}\phi \odot W^{(1)}x$$

$\phi \odot$

This corresponds to applying the non-linearity ϕ element-wise to a vector.

For example: $\phi \odot W^{(1)}x$

This corresponds to a multiplication by matrix $W^{(1)}$.

Then we apply the non-linear function ϕ element-wise.

In this formula we are seemingly not using any bias b.

But the following trick can be used:

$$w^T x + b = [w, b]^T \begin{pmatrix} x \\ 1 \end{pmatrix}$$

The bias b can be represented by adding an entry at the end of w and appending a 1 at the end of x.

So by changing our definition of x, we can represent our DNN using weights W only.

This will simplify our notations in what will follow.

Our DNN is the result of the following composition of functions:

$$W^{(n)}\phi \odot W^{(n-1)} \cdots \phi \odot W^{(2)}\phi \odot W^{(1)}x$$

We need to apply the chain rule to calculate the gradient.

It gets pretty complicated.

So grab a pencil and paper and please follow along the equations as we go.

The trick is to apply the chain rule starting from the left of this expression, progressively moving to the right.

$$W^{(n)}\phi \odot W^{(n-1)} \cdots \phi \odot W^{(2)}\phi \odot W^{(1)}x$$

You could do the same thing going from the right to the left but an analysis reveals that this is computationally more expensive.

$$W^{(n)}\phi \odot W^{(n-1)} \cdots \phi \odot W^{(2)}\phi \odot W^{(1)}x$$

Let us define:

$$a^{(k)} = \phi \odot W^{(k)} \cdots \phi \odot W^{(1)} x$$

So

$$y = W^{(n)} \phi \odot W^{(n-1)} \cdots \phi \odot W^{(k+1)} a^{(k)}$$

Recall:

$y(x) =$

$W^{(n)} \phi \odot W^{(n-1)} \cdots \phi \odot W^{(2)} \phi \odot W^{(1)} x$

We will use a recurrence relation.

It will start from $k = n$ and will go down to $k = 1$.

Let's start with

$$\frac{\partial y}{\partial W^{(n)}}$$

We can always assume that y is a scalar.

We have

$$y = W^{(n)} a^{(n-1)}$$

The trick is that $a^{(n-1)}$ is independent of $W^{(n)}$.

So y is a linear function of $W^{(n)}$.

Since y is a scalar, the last matrix $W^{(n)}$ must be a row vector.

Let's calculate a derivative with respect to a single component:

$$\frac{\partial y}{\partial [W^{(n)}]_j} = [a^{(n-1)}]_j$$

Using matrix notations, this becomes

$$\frac{\partial y}{\partial [W^{(n)}]} = [a^{(n-1)}]^T$$

where $[a^{(n-1)}]^T$ is a row vector.