Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2022

Comments

A few important side notes on training. First, on derivatives:

Concept 1

Sometimes, depending on your **loss** and **activation** function, it may be easier to directly compute

$$\frac{\partial \mathcal{L}}{\partial Z^L}$$

Than it is to find

$$\partial \mathcal{L}/\partial A^L \quad \text{and} \quad \partial A^L/\partial Z^L$$

So, our algorithm may change slightly.

Another thought: intialization.

Concept 2

We typically try to pick a **random initalization**. This does two things:

- Allows us to avoid weird **numerical** and **symmetry** issues that happen when we start with $W_{ij} = 0$.
- We can hopefully find different **local minima** if we run our algorithm multiple times.
  - This is also helped by picking **random data points** in **SGD** (our typical algorithm).

Here, we choose our **initialization** from a **Gaussian** distribution, if you know what that is.

If you do not know a gaussian distribution, that shouldn't be a problem. It is also known as a "normal" distribution.

Pseudocode

Our training algorithm for backprop can follow smoothly from what we've laid out.

SGD-NEURAL-NET($\mathcal{D}_n, T, L, (m^1, \ldots, m^L), (f^1, \ldots, f^L), \text{Loss}$)

for every **layer**:
    Randomly initialize
        the **weights** in every layer
        the **biases** in every layer

While **termination condition** not met:
    Get random data point i
    Kepp track of time t

    Do forward pass
        for every **layer**:
            Use previous **layer**'s **output**: get **pre-activation**
            Use **pre-activation**: get new output, **activation**

        Get **loss**: forward pass complete

    Do back-propagation
        for every **layer** in <u>reversed order</u>:
            If **final** layer: #Loss function
                Get $\partial\mathcal{L}/\partial A^L$

            Else:
                Get $\partial\mathcal{L}/\partial A^\ell$: #Link two layers
                    $(\partial Z^{\ell+1}/\partial A^\ell) * (\partial\mathcal{L}/\partial Z^{\ell+1})$

                Get $\partial\mathcal{L}/\partial Z^\ell$: #Within layer
                    $(\partial A^\ell/\partial Z^\ell) * (\partial\mathcal{L}/\partial A^\ell)$

            Compute weight gradients:
                Get $\partial\mathcal{L}/\partial W^\ell$: #Weights
                    $\partial Z^\ell/\partial W^\ell = A^{\ell-1}$
                    $(\partial Z^\ell/\partial W^\ell) * (\partial\mathcal{L}/\partial Z^\ell)$

                Get $\partial\mathcal{L}/\partial {W_0}^\ell$: #Biases
                    $\partial\mathcal{L}/\partial {W_0}^\ell = (\partial\mathcal{L}/\partial Z^\ell)$

            Follow Stochastic Gradient Descend (SGD): #Take step
                Update **weights**:
                    $W^\ell = W^\ell - \left(\eta(t) * (\partial\mathcal{L}/\partial W^\ell)\right)$

                Update **biases**:
                    ${W_0}^\ell = {W_0}^\ell - \left(\eta(t) * (\partial\mathcal{L}/\partial {W_0}^\ell)\right)$

Return final neural network with weights and biases