1. Layers

Conventionally, a layer in ML is an affine transformation of the input followed by a nonlinear function: mathematically,

$$\mathrm{x} \xmapsto{\text{affine}} \mathrm{W}\,\mathrm{x} + \mathrm{b} \xmapsto{\text{nonlinear}} f(\mathrm{W}\,\mathrm{x} + \mathrm{b})\,, \tag{1.1}$$

where W is a matrix called *weight*, b is a vectcalled *bias*, and f is a nonlinear function called *activation function*. Typical choices of an activation function are sigmoid, tanh, ReLU and so on.

We often represent this map via the following diagram:

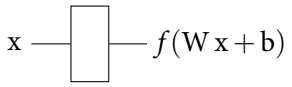

Figure 1: Single layer in ML

When we have multiple layers, it is conventional to omit the intermediate outputs $f(\mathrm{W}\,\mathrm{x} + \mathrm{b})$, or *hidden states*, and simply draw the lines and boxes:

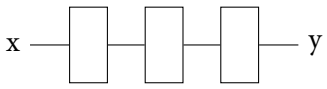

Figure 2: Multiple layers in ML

2. Skip Connection

Skip connection is a technique in machine learning (ML) that allows a data to bypass one or more layers. Concretely, we combine the output of a layer with the outputs of previous layers. There are two major types of skip connection: addition and concatenation. For now, let us focus on the former case.

An *addition-based* skip connection combines outputs, as its name suggests, by addition. When dealing with a single layer, this can be represented diagrammatically by:

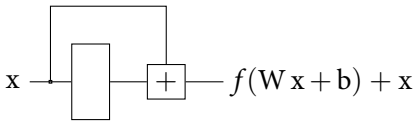

Figure 3: Single addition-based skip connection

In other words, we are adding the input x to the output $f(\mathrm{W}\,\mathrm{x} + \mathrm{b})$ of the layer without skip connection. We can of course have multiple layers with multiple skip connections:

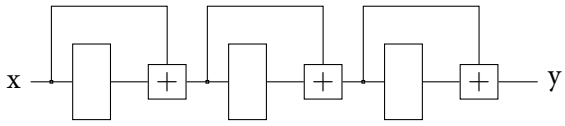

Figure 4: Multiple addition-based skip connections

Sometimes we can skip connect multiple layers:

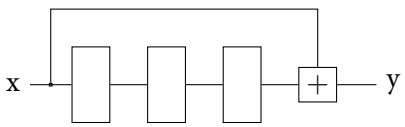

Figure 5: Skipping through multiple layers

Remark.
Often times it is desirable to normalize layer outputs.[1] That is, given $\mathrm{y} = f(\mathrm{W}\,\mathrm{x} + \mathrm{b}) + \mathrm{x}$, normalizing y is a transformation on it so that its entries (i.e. *features*) are on a similar scale. Most normalizations are nonlinear transformations. For simplicity, we are not going to include normalization in our layers.

[1] In the presence of a skip connection, normalization is usually done afterwards. After all, what's the point of doing normalization before the skip connection?

Now one may ask,

what's the point of skip connection?

The goal of ML seems to be finding a suitable transformation that maps inputs x to desired outputs y. At first glance, this seems to suggest that skip connection may be useless since we are sending an input without processing it. But let's take a closer look at the layers. As mentioned before, it is a composition of two transformations: an affine transformation $\mathrm{x} \mapsto \mathrm{W}\,\mathrm{x} + \mathrm{b}$ followed by a nonlinear transformation $\mathrm{W}\,\mathrm{x} + \mathrm{b} \mapsto f(\mathrm{W}\,\mathrm{x} + \mathrm{b})$. Usually we make the nonlinear transformation f to be fixed, so let's assume this is the case. With f fixed, we can still adjust the output of the layer by adjusting W, b, and almost always this is done by *backpropagation*.

3. Backpropagation

To make things more concrete, let us consider posing an ML problem as an optimization problem. Assume that there is an unknown function $F : X \to Y$ that we want to figure out. Although we do not know what F is, we have some partial information: say we know a set

$$\mathcal{S} = \left\{ \left(\mathrm{x}_i, \mathrm{y}_i \right) \right\}_{i \in I} \tag{3.1}$$

for some finite subset $\{\mathrm{x}_i\}_{i \in I} \subseteq X$. With this partial information in hand, our job is to best approximate F, say by coming up with a map $G : X \to Y$.

There could be various ways that we can tackle this task, but one particular way that people have been found to be useful is to utilize a *cost function*. That is, we consider a nonnegative map C over all possible maps $G : X \to Y$[1] such that $C(G)$ measures how far G is from F. Of course, due to the limitation of our knowledge in F, the best we can try would be in terms of the only data we know of: $\left(\mathrm{x}_i, \mathrm{y}_i \right)$ for $i \in I$. A typical choice would be the mean square of 2-norm (so called *mean-squared-error*),

$$C(G) = \frac{1}{|I|} \sum_{i \in I} \left\| F(\mathrm{x}_i) - G(\mathrm{x}_i) \right\|_2^2 = \frac{1}{|I|} \sum_{i \in I} \left\| \mathrm{y}_i - G(\mathrm{x}_i) \right\|_2^2, \tag{3.2}$$

among many others. Therefore, our problem would be

$$\text{minimize } C(G) .^{2} \tag{3.3}$$

Now that we have an actual problem, let us see how we can tackle it. As an analogy, consider yourself standing on a mountain that you never went before on a very foggy day. You want to go back to your base, which is located at the bottom of the mountain. But because of the fog, you can only see regions around you. Then probably the most sensible thing that you could do is to take steps towards the direction where the elevation seems to decreasing, at least in the vicinity of yourself.

We can try to solve minimization problem like [3.3] in an analogous way. The global information about the cost function C is usually unaccessible (just like you cannot see far in a foggy day), so we have to work with the local information. We can use gradient of C to find out the direction of steepest descent, so that we can take a step in that direction.

[1] That is, C would be a *functional* in the applied mathematician's lingo.

[2] One may notice that, while $G = F$ implies minimum $C(G)$ for a reasonably defined C, the converse is mostly not the case. This is what is underlying the problem of *overfitting*.