Forward-Feeding Neural Networks (FFNNs): Forward & Backward Passes

STUFF TO KNOW BY HEART - EVEN WHEN DRUNK!

*1. FFNNs are **supervised** learning models*

*2. **Cost functions** measure degree of prediction error*

*3. **Training** is mathematical optimization that makes cost decrease*

*4. An FFNN consists of **layers** of **transformation functions** and **weights***

*5. FFNNs' **forward pass** models **hypothesized output***

*6. FFNNs' **backward pass** computes **partial derivatives** of cost function with respect to weight layers, for use in mathematical optimization*

FFNNs: Supervised Learning Models with Continuous Real-Valued Scalar “Hypothesis-vs.-Target” Cost Functions

FFNNs are **supervised** learning models: given Input $\mathbf{X}$ - a matrix/array representing m cases of input features - and Target Output $\mathbf{Y}$ representing the m corresponding “**right answers**”, a Supervised Learning Model tries to learn a structured mapping that transforms $\mathbf{X}$ to Hypothesized Output $\mathbf{H}$ that is similar/close to $\mathbf{Y}$.

The extent of $\mathbf{H}$’s similarity/closeness to $\mathbf{Y}$ - which is the criterion to judge how well a Supervised Learning Model learns to mimick $\mathbf{Y}$ from knowing $\mathbf{X}$ - is numerically measured by a certain specified scalar cost function $c(\mathbf{H}, \mathbf{Y})$. This function c’s value should be small when $\mathbf{H}$ is very “similar” or “close” to $\mathbf{Y}$, and large otherwise. In almost all Supervised Learning Models in practical use nowadays, $c(\mathbf{H}, \mathbf{Y})$ is a continous real-valued function and its partial derivative $\frac{\partial c}{\partial \mathbf{H}}$ with respect to $\mathbf{H}$ is computable as a certain function $d(\mathbf{H}, \mathbf{Y})$. Cost functions are usually measured on an average per-case basis.

The task of helping a Supervised Learning Model learn - or so-called “**training**” it - involves mathematical optimization procedures that make the average per-case $\mathbf{H}$-vs.-$\mathbf{Y}$ cost decrease when we let the Model see more and more cases of inputs and corresponding “right-answer” target outputs. Once trained until its cost has decreased to an acceptably low level, a Model will make only small errors and hence be a good tool for predicting the output $\mathbf{y}$ from a not-yet-seen input $\mathbf{x}$.

Note that we are using the rather gentle phrase “acceptably low” instead of the stronger word “minimized”. This is because when a Supervised Learning Model does achieve the absolutely smallest possible error rate during the “training” process, it will have over-learned: not only will it have learned the overall rules of the game (which are useful when generalizing to new cases), it will have also **memorized various irrelevant idiosyncracies** specific to the training data (which **hurts** its generalization ability). We’ll discuss this so-called “**over-fitting**” issue separately.

Model Structure / Hypothesis: Forward Pass

In a generalized sense, an FFNN models Hypothesized Output $\mathbf{H} = h(\mathbf{X}, \mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \ldots, \mathbf{W}^{[L]})$ through 1 input layer and L additional layers of transformation functions and parameters (called "weights") in the following manner:

$$
\begin{aligned}
&\text{network layer 1: } \mathbf{A}^{[1]} = \text{Input } \mathbf{X} \\
&\text{network layer 2: } \mathbf{A}^{[2]} = f^{[1]}(\mathbf{A}^{[1]}, \mathbf{W}^{[1]}) \\
&\text{network layer 3: } \mathbf{A}^{[3]} = f^{[2]}(\mathbf{A}^{[2]}, \mathbf{W}^{[2]}) \\
&\ \ldots \\
&\text{network layer } (L+1)\text{: } \mathbf{H} = \mathbf{A}^{[L+1]} = f^{[L]}(\mathbf{A}^{[L]}, \mathbf{W}^{[L]})
\end{aligned}
$$

where:

- $\mathbf{A}$'s are called the layers' "**activations**" and inter-layer parameters $\mathbf{W}$'s are called "**weights**". The way the FFNN computes $\mathbf{H}$ from input $\mathbf{X}$ through layers of transformation functions and weights is called the "**forward pass**".

- Each "**forward function**" f is a structurally pre-defined transformation function $\mathbf{Output} = f(\mathbf{Input}, \mathbf{Parameter})$ such that, given partial derivative $\frac{\partial v}{\partial \mathbf{Output}}$ of a scalar variable v with respect to $\mathbf{Output}$, the following partial derivatives with respect to $\mathbf{Input}$ and $\mathbf{Parameter}$ can be computed by certain "**backward functions**" b_{Input} and $b_{Parameter}$:

$$
\begin{aligned}
&\frac{\partial v}{\partial \mathbf{Input}} = b_{Input}\left(\frac{\partial v}{\partial \mathbf{Output}}, \text{local state}\right) \\
&\frac{\partial v}{\partial \mathbf{Parameter}} = b_{Parameter}\left(\frac{\partial v}{\partial \mathbf{Output}}, \text{local state}\right) \\
&\text{where the term "local state" refers to current values} \\
&\text{of function } f\text{'s } \mathbf{Input}, \mathbf{Parameter} \text{ and } \mathbf{Output} \\
& \\
&\text{(for each neural network layer } l\text{, we henceforth denote} \\
&\text{its corresponding "backward functions" } b_A^{[l]} \text{ and } b_W^{[l]}\text{)}
\end{aligned}
$$

The purpose of knowing such partial derivatives will become clear later when we discuss the "**backward pass**" or "**backpropagation**" procedure.

Backward Pass / Backpropagation procedure to derive $\frac{\partial c}{\partial \mathbf{W}^{[l]}}$ for each layer l, to be used in optimization

With the structure of the transformation functions f's fixed, in the learning/training process, our job is to adjust/update the values of weight layers $\mathbf{W}^{[1]}$, $\mathbf{W}^{[2]}$, …, $\mathbf{W}^{[L]}$ so as to make the cost function $c(\mathbf{H}, \mathbf{Y})$ decrease. This invariably requires us to know or be able to estimate the partial derivative $\frac{\partial c}{\partial \mathbf{W}^{[l]}}$ for each layer l. We can compute such partial derivatives through the following "backpropagating" procedure:

$$
\begin{array}{lcl}
\dfrac{\partial c}{\partial \mathbf{A}^{[L+1]}} = \dfrac{\partial c}{\partial \mathbf{H}} = d(\mathbf{H}, \mathbf{Y}) & \Rightarrow & \dfrac{\partial c}{\partial \mathbf{W}^{[L]}} = b_W^{[L]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[L+1]}}, \text{local state}\right) \\
\Downarrow & & \\
\dfrac{\partial c}{\partial \mathbf{A}^{[L]}} = b_A^{[L]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[L+1]}}, \text{local state}\right) & \Rightarrow & \dfrac{\partial c}{\partial \mathbf{W}^{[L-1]}} = b_W^{[L-1]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[L]}}, \text{local state}\right) \\
\Downarrow & & \\
\dfrac{\partial c}{\partial \mathbf{A}^{[L-1]}} = b_A^{[L-1]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[L]}}, \text{local state}\right) & \Rightarrow & \dfrac{\partial c}{\partial \mathbf{W}^{[L-2]}} = b_W^{[L-2]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[L-1]}}, \text{local state}\right) \\
\Downarrow & & \\
\dots & & \\
\Downarrow & & \\
\dfrac{\partial c}{\partial \mathbf{A}^{[2]}} = b_A^{[2]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[3]}}, \text{local state}\right) & \Rightarrow & \dfrac{\partial c}{\partial \mathbf{W}^{[1]}} = b_W^{[1]}\left(\dfrac{\partial c}{\partial \mathbf{A}^{[2]}}, \text{local state}\right)
\end{array}
$$

Illustration of Forward and Backward Passes

The following diagram illustrates an FFNN's forward and backward passes:

$$\mathbf{X} = \mathbf{A}^{[1]} \xrightarrow[(W^{[1]})]{f^{[1]}} \mathbf{A}^{[2]} \xrightarrow[(W^{[2]})]{f^{[2]}} \mathbf{A}^{[3]} \xrightarrow[(W^{[3]})]{f^{[3]}} \cdots \xrightarrow[(W^{[L-2]})]{f^{[L-2]}} \mathbf{A}^{[L-1]} \xrightarrow[(W^{[L-1]})]{f^{[L-1]}} \mathbf{A}^{[L]} \xrightarrow[(W^{[L]})]{f^{[L]}} \mathbf{A}^{[L+1]} = \mathbf{H} \longrightarrow \text{cost: } c(\mathbf{H}, \mathbf{Y})$$

(FORWARD PASS)

layer #1	layer #2	layer #$(L-2)$	layer #$(L-1)$	layer #L

(BACKWARD PASS)

$$\begin{array}{ccccccccccccc}
\dfrac{\partial c}{\partial \mathbf{W}^{[1]}} & & \dfrac{\partial c}{\partial \mathbf{W}^{[2]}} & & \dfrac{\partial c}{\partial \mathbf{W}^{[3]}} & & \dfrac{\partial c}{\partial \mathbf{W}^{[L-2]}} & & \dfrac{\partial c}{\partial \mathbf{W}^{[L-1]}} & & \dfrac{\partial c}{\partial \mathbf{W}^{[L]}} & & \\
b_W^{[1]} \nwarrow & & b_W^{[2]} \nwarrow & & b_W^{[3]} \nwarrow & & b_W^{[L-2]} \nwarrow & & b_W^{[L-1]} \nwarrow & & b_W^{[L]} \nwarrow & & \\
& \dfrac{\partial c}{\partial \mathbf{A}^{[2]}} & \xleftarrow[b_A^{[2]}]{} & \dfrac{\partial c}{\partial \mathbf{A}^{[3]}} & \xleftarrow[b_A^{[3]}]{} & \cdots & \xleftarrow[b_A^{[L-2]}]{} & \dfrac{\partial c}{\partial \mathbf{A}^{[L-1]}} & \xleftarrow[b_A^{[L-1]}]{} & \dfrac{\partial c}{\partial \mathbf{A}^{[L]}} & \xleftarrow[b_A^{[L]}]{} & \dfrac{\partial c}{\partial \mathbf{A}^{[L+1]}} & = \dfrac{\partial c}{\partial \mathbf{H}} = d(\mathbf{H}, \mathbf{Y})
\end{array}$$