Formalize Deep Learning

Recall that a neural network approximates a function (e.g., the decision boundary between two classes) using piecewise linear functions (connected at "kinks"). In principle, we can approximate any linear or nonlinear functions using an infinite number of piecewise linear functions. That is, if $\mathbf{W}$ in $\mathbf{h}(\mathbf{x}) = \mathbf{U}\max(\mathbf{W}\mathbf{x} + \mathbf{b}, 0) + \mathbf{c}$ has shape $M \times d$, then we can make M very large.

☆ Key Points

Neural networks use piecewise linear functions to approximate decision boundaries.

It is possible to approximate any linear or nonlinear function.

Using fewer parameters allows for better generalizations.

Go Deep with Neural Networks

What if a decision boundary is very complicated and requires many (thousand or even million) linear pieces to be adequately approximated? To make neural networks more efficient, we can reuse parts of the decision boundary multiple times. This allows us to learn fewer parameters and generalize better.

Let us write a neural network as $\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x})$, where we absorb the biases into $\mathbf{W}$ by adding a new dimension to the input (just like what we did for the Perceptron) and where we let $\sigma(\mathbf{z}) = \max(\mathbf{z}, 0)$. This neural network learns M linear functions that are combined to a single nonlinear function. If we want to make this network more powerful, we can either increase M or *add more layers*.

More layers means that we take the output of $\mathbf{h}(\mathbf{x})$ as input to another similar network with different parameters. More concretely, we can write a generic multi-layer neural network as $\mathbf{h}(\mathbf{x}) = \mathbf{W}_l\sigma(\mathbf{W}_{l-1}\sigma(\cdots\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x})\cdots))$.

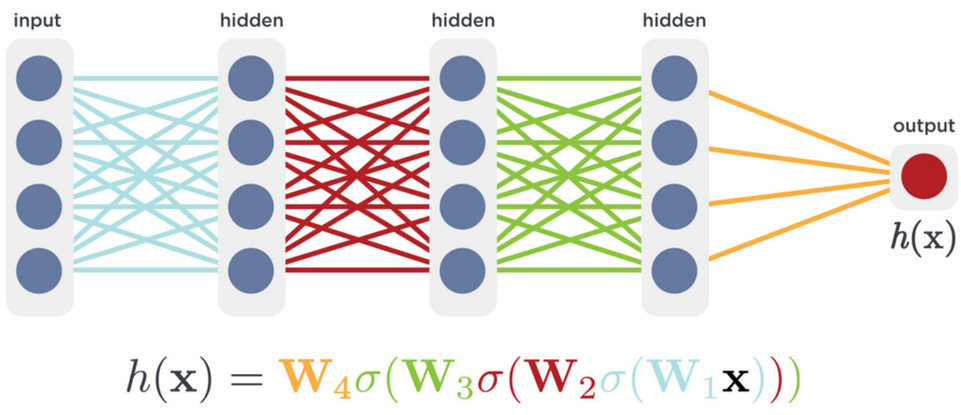
Let's unpack the equation a bit. The first layer in the network learns piecewise linear functions to approximate nonlinear boundaries. It does this in two steps:

1. $\mathbf{W}_1\mathbf{x}$. This learns M linear functions.
2. $\sigma(\mathbf{W}_1\mathbf{x})$. This sets every negative value equal to 0, creating the kinks.

Every subsequent layer in the network essentially does two things:

1. Combines linear combinations of the functions of the previous layer and aggregates them together. This is represented by $\mathbf{W}_k\sigma(\mathbf{W}_{k-1})$.
2. Set every negative value from $\mathbf{W}_k\sigma(\mathbf{W}_{k-1})$ to equal 0. This is represented by $\sigma(\mathbf{W}_k\sigma(\mathbf{W}_{k-1}))$.

Below is a pictorial representation of a neural network.



Create a Flexible Model

Intuitively, think of $\mathbf{W}_l$ as weights to combine the previous nonlinearities that we learned through $\mathbf{W}_{l-1}, \ldots, \mathbf{W}_1$. By increasing the number l, we can combine more varied shapes to form the decision boundaries and, thus, achieve a more flexible model.

Imagine that the network above $\mathbf{W}_1 \in \mathcal{R}^{M\times d}, \mathbf{W}_l \in \mathcal{R}^{1\times M}$, and $\mathbf{W}_k \in \mathcal{R}^{M\times M}$ for all $1 < k < l$. How many nonlinearities do we obtain in total?

Each dimension of layer 1 corresponds to a linear function.

Each dimension of layer 2 corresponds to a function with M piecewise linear pieces.

Each dimension of layer 3 corresponds to a function with M^2 piecewise linear pieces.

...

Each dimension of layer $l-1$ corresponds to a function with M^{l-2} piecewise linear pieces.

The output layer l corresponds to a function with M^{l-1} piecewise linear pieces.

Imagine $d = 2$, $M = 100$, and $l = 4$. Here, we obtain 1 million linear pieces but only have to train 20,300 parameters!

($\mathbf{W}_1$ is 2×100, $\mathbf{W}_2$ is 100×100, $\mathbf{W}_3$ is 100×100, $\mathbf{W}_4$ is 100×1 .)

Determine the Depth

To determine the number of hidden layers a network has, simply count the number of transformations the input goes through.

$$\mathbf{h}(\mathbf{x}) = \mathbf{W}_l\sigma(\mathbf{W}_{l-1}\sigma(\cdots\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x})\cdots))$$

has $l-1$ **hidden layers** because the input $\mathbf{x}$ goes through $l-1$ transformations. The σ function is applied at each hidden layer and so there are $l-1$ σ functions applied. **Since there are l tunable weight matrices, the depth of the network is l.**

In theory, a wide network (where $l = 2$ and M is very large) can approximate any functions to arbitrary accuracy; however, it might require many many parameters. To learn more efficiently, we instead go for deeper networks (where l is large) and make $\mathbf{W}_{l-1}, \ldots, \mathbf{W}_1$ of reasonable size.