Part VII: Learning from Data

VII.1: Deep Neural Networks

VII.2: Convolutional Neural Nets

VII.3: Backpropogation

VII.4: Hyperparameters

VII.5: Machine Learning

VII.1 The Construction of Deep Neural Networks

A neural network is a model $F: \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^k$ that makes a prediction $\hat{y} \in \mathbb{R}^k$ based on the parameters $x \in \mathbb{R}^n$ and the feature vector $v \in \mathbb{R}^m$:

$$\hat{y} = F(x, v).$$

Given the training data,

$$(v_i, y_i) \in \mathbb{R}^m \times \mathbb{R}^k, \quad i = 1, \ldots, N,$$

where v_i are features and y_i are labels, we optimize the parameters x so that the prediction of the model satisfies

$$y_i \approx F(x, v_i), \quad i = 1, \ldots, N.$$

In particular, neural networks are constructed from the following components:

① Affine functions: $v \mapsto Av + b$

② Activation functions: $w \mapsto \sigma(w)$

③ Composition: $F = F_L \circ \cdots \circ F_2 \circ F_1$

Note that the composition of affine functions is affine:

$$F_1(v) = A_1 v + b_1, \quad F_2(v) = A_2 v + b_2$$

$$\begin{aligned} \Rightarrow \quad (F_2 \circ F_1)(v) &= F_2(F_1(v)) \\ &= F_2(A_1 v + b_1) \\ &= A_2(A_1 v + b_1) + b_2 \\ &= A_2 A_1 v + (A_2 b_1 + b_2) \\ &= A v + b, \end{aligned}$$

where $A = A_2 A_1$ and $b = A_2 b_1 + b_2$.

The key to the modeling power of neural networks is the <u>activation function</u>.

The most popular activation function is the <u>rectified linear unit</u>:

$$\mathrm{ReLU}(w) = [w]_+ = \begin{cases} 0 & \text{if } w \leq 0 \\ w & \text{if } w > 0 \end{cases}$$

Some other activation functions are:

(1) Linear:

$$\sigma(w) = a^T w + b,$$

(2) Hyperbolic tangent:

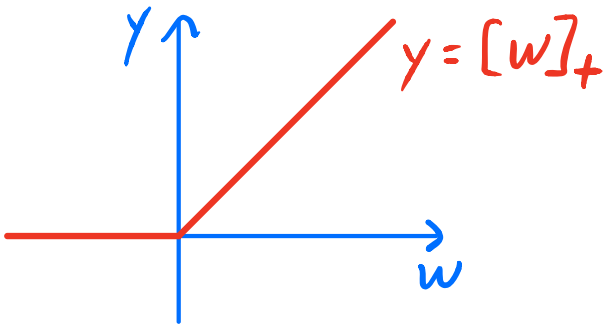
$$\sigma(w) = \tanh(w) \in (-1, 1),$$

(3) Logistic / Sigmoid:

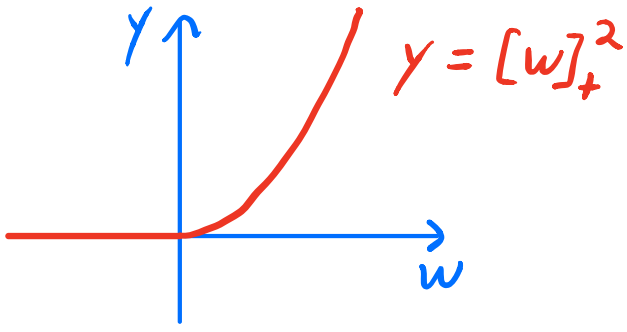
$$\sigma(w) = \frac{1}{1 + e^{-w}} \in (0, 1).$$

Historically, tanh and sigmoid were preferred for their smoothness.

ReLU makes $v \mapsto F(x, v)$ a continuous piecewise linear function.



nonsmooth



smooth

Example:

$$A_1 = \begin{bmatrix} -3 & -3 \\ 5 & 2 \\ -3 & 4 \end{bmatrix}, \quad b_1 = \begin{bmatrix} -2 \\ 1 \\ -1 \end{bmatrix}, \quad \sigma(w) = [w]_+$$

$$A_2 = \begin{bmatrix} 2 & 2 & 1 \end{bmatrix}, \quad b_2 = [3]$$

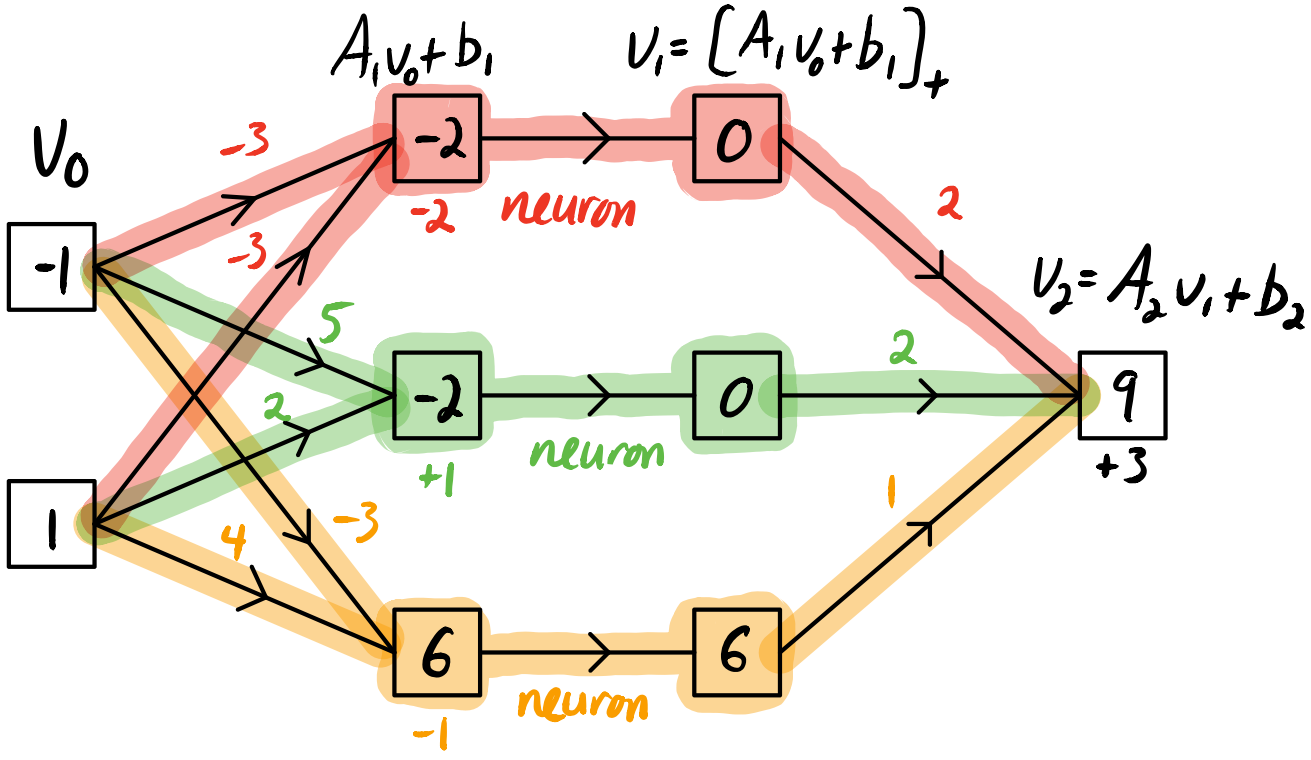
$$x = (A_1, b_1, A_2, b_2) \leftarrow \text{parameters}$$

$$F_1(v) = [A_1 v + b_1]_+$$

$$F_2(v) = A_2 v + b_2$$

$$v_0 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$F(x, v) = F(v) = F_2(F_1(v))$$



$$A_1 v_0 + b_1 = \begin{bmatrix} -3 & -3 \\ 5 & 2 \\ -3 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} -2 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \\ 6 \end{bmatrix}$$

$$v_1 = [A_1 v_0 + b_1]_+ = \begin{bmatrix} 0 \\ 0 \\ 6 \end{bmatrix}$$

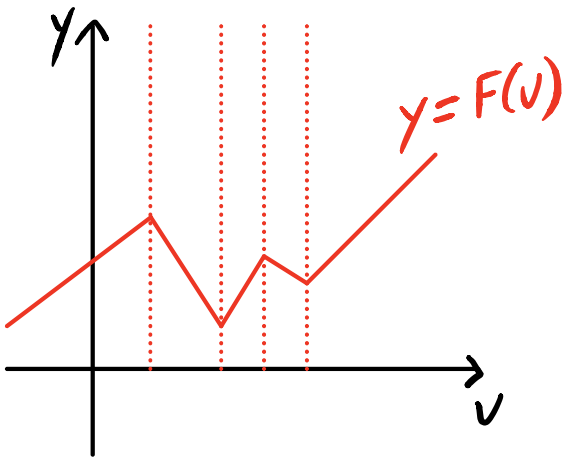
$$v_2 = A_2 v_1 + b_2 = \begin{bmatrix} 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 6 \end{bmatrix} + [3] = [9]$$

$$\therefore F(v_0) = [9]$$

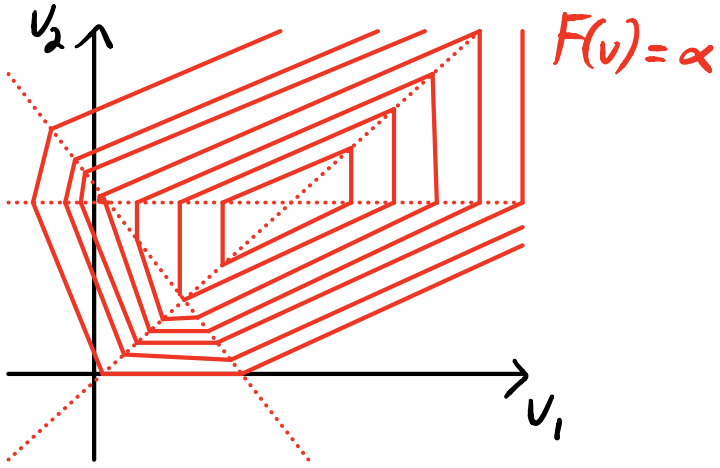
The Graph of $F(v)$

Since $F(v)$ is a continuous piecewise linear function, its graph is made up of many flat pieces that are joined without any gaps. It is like origami with many folds. The folds come from the ReLU function.

$F: \mathbb{R} \to \mathbb{R}$



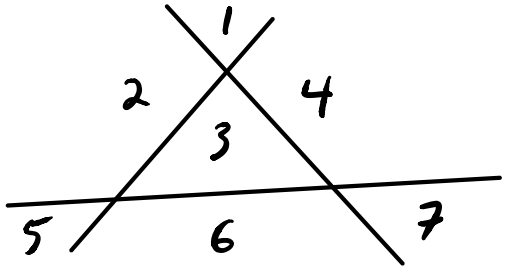
$F: \mathbb{R}^2 \to \mathbb{R}$



For a neural network with one hidden layer (like the example above) we can count the number of flat pieces.

n neurons $\Rightarrow$ folds along n hyperplanes in $\mathbb{R}^m$ $(a_i^T v + b_i = 0)$

eg $m = 2$, $n = 3$ $\Rightarrow$ 7 flat pieces



In general, the number of flat pieces is:

$$r(n,m) = \sum_{k=0}^{m} \binom{n}{k}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ for $k \le n$

and $\binom{n}{k} = 0$ for $k > n$.

eg $r(3,2) = \binom{3}{0} + \binom{3}{1} + \binom{3}{2}$

$= 1 + 3 + 3 = 7.$

For $n \le m$, we have

$$r(n,m) = 2^n.$$

For $n \gg m$, we have

$$r(n,m) \approx \frac{n^m}{m!}.$$

$r(32,2)$

$= \binom{32}{0} + \binom{32}{1} + \binom{32}{2}$

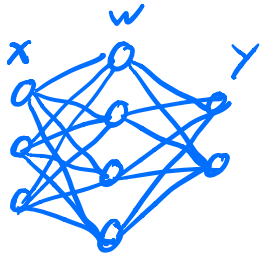
$= 1 + 32 + 496$

$= 529$

Neural Nets are Universal Approximators

Let $f: \mathbb{R}^m \to \mathbb{R}^k$ be a <u>continuous function</u> and let $K \subseteq \mathbb{R}^m$ be compact. Then for every $\varepsilon > 0$, there is a <u>neural net</u>

$$F(x) = A_2[A_1 x + b_1]_+ + b_2$$



$w = [A_1 x + b_1]_+$

$y = A_2 w + b_2$

such that

$$\max_{x \in K} \|f(x) - F(x)\| < \varepsilon.$$

Note that the number of neurons in the hidden layer (ie n where A_1 is $n \times m$) may need to be very large to obtain a very accurate approximation of f.

Reducing the dimension in neural nets

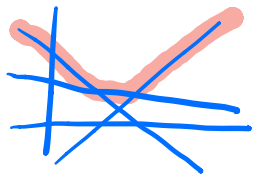
① Subsample

stride = 2 : use every second output

stride = 3 : use every third output

This might destroy important information.

② Max-pooling



Take the maximum value of each group of outputs. There is less risk of destroying important information, and can help reduce the computation and reduce the possibility of overfitting.

③ Average pooling (linear map)

Take the average of each group of outputs.

The initial parameters x_0

The minimizer we obtain depends on where we start the optimization algorithm. We should choose x_0 randomly. Two requirements:

① x_0 has a carefully chosen variance σ^2;

② the hidden layers in the neural net have enough neurons.

A good choice of σ^2 is

$$\sigma^2 = \frac{2}{\text{fan-in}}$$

where fan-in is the maximum number of inputs to neurons.