Perceptron (Recap)

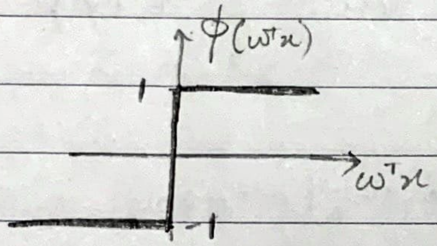
In the last class, we considered the perceptron, for which, given data x, and corresponding label z, the classification rule was:

Decision Rule: $\quad w^T x \geq 0 \quad$ if $\quad z = 1$

$\qquad\qquad\qquad\quad w^T x < 0 \quad$ if $\quad z = -1$

Hence, we could write the predicted label as:

$$\hat{z} = \sigma(w^T x) = \begin{cases} 1 & \text{if } w^T x \geq 0 \\ -1 & \text{if } w^T x < 0 \end{cases}$$



sign function

And our loss function (which describes any discrepancy between predicted $\hat{z}$ and actual z, and which we desire to minimize) is defined by all the data points for which $\hat{z} = \sigma(w^T x)$ does not match z

$$J_i(z_i, \sigma(w^T x_i)) = \begin{cases} 0 & \text{if } z_i = \hat{z} = \sigma(w^T x) \\ 1 & \text{if } z_i = \sigma(w^T x) \end{cases}$$

We noted that this objective could be re-written as

$$J_i = \max(0, -z_i w^T x_i)$$

[Minimizing this equivalent to minimizing $c\,(\max(0, 1 - z_i \hat{z}_i))$]

And in fact, $J = \sum_{i=1}^{N} J_i = -\sum_{i \in \mathcal{M}} z_i w^T x$

where $\mathcal{M} \subseteq \{1, \ldots, N\}$.

But in minimizing J over w by gradient descent we needed to update $\mathcal{M}$ in our gradient descent algorithm

$$w_{(k+1)} \leftarrow w_{(k)} + \alpha_k \sum_{i \in \mathcal{M}_k} z_i x_i$$

As such, the minimization of J may take a long number of steps to converge.

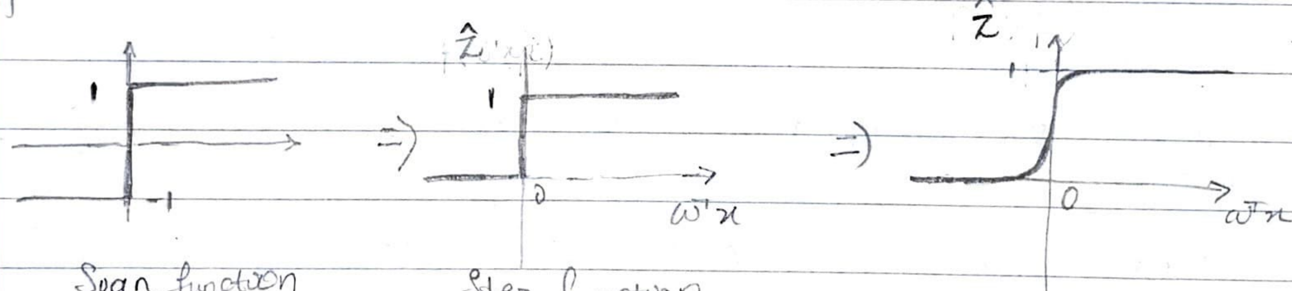
One way around this is to define our $\hat{z}$, or predicted label by a continuous function: one that is continuously differentiable.

Logistic Regression

We can re-write our classification task using a step function rather than a sign

$$\sigma(w^T x) = \begin{cases} 1 & \text{if } w^T x > 0 \\ 0 & \text{if } w^T x < 0 \end{cases}$$

With the corresponding class labels $z \in \{0, 1\}$. Because $\sigma(w^T x)$ assumes values $\{0, 1\}$, we can think of σ as a probability function



Sign function   Step function

Since this step function is not differentiable, we can replace it with a continuously differentiable function.

$$\hat{z} = \sigma(w^T x) = \frac{1}{1 + e^{-w^T x}} \quad \leftarrow \text{Sigmoid function}$$

This function has interesting properties

1. $x \to \infty \quad \sigma(x) \to 1$

   $x \to -\infty \quad \sigma(x) \to 0$

2. $\sigma(-x) = 1 - \sigma(x)$

3. $\dfrac{d\sigma(x)}{dx} = (1 - \sigma(x))\,\sigma(x) = \sigma(-x)\,\sigma(x)$

Now our objective is to find a loss function $J(\hat{w})$ that minimizes (or maximizes) how much $\hat{z}$ differs (or is similar) from z. Consider the candidate

$$p(z|x;w) = \hat{z}^{z}(1-\hat{z})^{1-z}$$

When $z=1$, $p(z|x) = \hat{z}$, and we want this to be as large as possible (cannot be > 1).

When $z=0$, $p(z|x) = (1-\hat{z})$, and we want this to be as large as possible (cannot be > 1), or $\hat{z}$ as small as possible (cannot be < 0).

Taking the log of $p(z|x)$ eliminates the powers

$$\log p(z|x;w) = z\log\hat{z} + (1-z)\log(1-\hat{z}).$$ ← log likelihood

Note that whatever maximizes $p(z|x;w)$ does the same to $\log p(z|x;w)$

We can now write our loss function as is

$$J_{CE}(w) = -\log p(z|x;w) = -z\log\hat{z} - (1-z)\log(1-\hat{z})$$

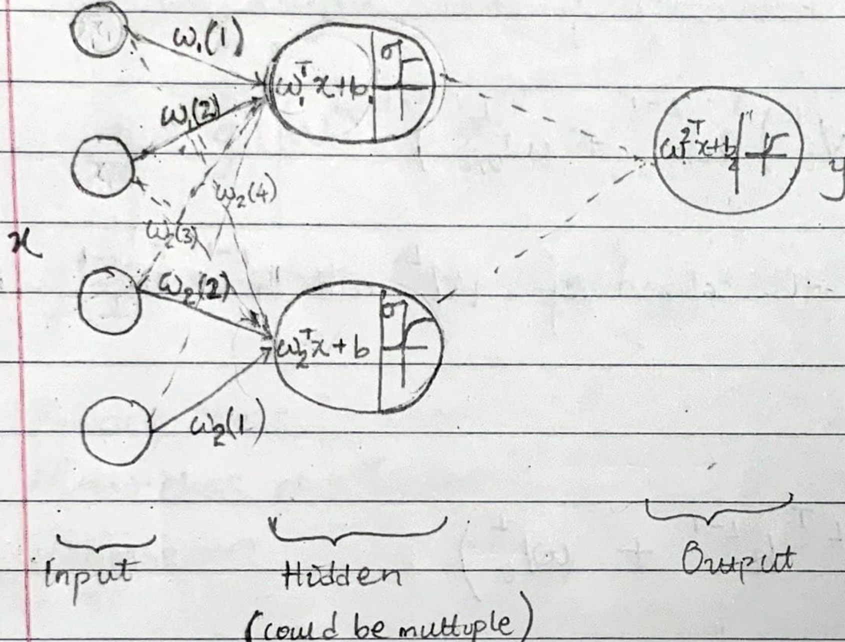
Cross entropy loss

And,

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{N} J_{CE,i}(w).$$

NEURAL NETWORKS

A neural network takes a logistic regression or perceptron to the 'next level'.



← Sometimes called Multi-Layer Perceptron

Here, our input data is transformed into an abstract feature space by multiple perception units or neurons. and this can be stacked layer after layer. The result is that for sufficiently large number of parameters the network becomes a universal approximator between x and y characterized by its weights;

$$y = f(x; \theta)$$

where θ represents the set of all weights vectors, w's used in the network. The weights between layers: for example in the above diagram

$$W_1' = \begin{bmatrix} | & | \\ w_1 & w_2 \\ | & | \end{bmatrix}$$ is the weight matrix between the input and hidden layer

Hidden Layers

Each node in an hidden layer is a single perceptron connected to all nodes in the previous layer. Its value is the projection of the previous layer onto its weight plus a bias, all passed through an activation function

$$h_i^L = \sigma\left((W_i^L)^T h^{L-1} + w_{0,i}^L\right)$$

where W_i is the ith column of W^L and $h^{L-1} = [h_1^{L-1}, \ldots h_{N_{L-1}}^{L-1}]^T$

and thus

$$h^L = \sigma\left(W^{L^T} h^{L-1} + w_0^L\right)$$

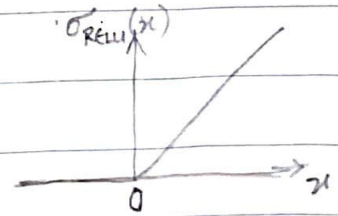
where $w_0^L = [w_{0,1}^L, \ldots, w_{0,N_L}^L]^T$

and $\sigma(x)$ is applied element-wise

Activation Functions

Several activation functions can be used in addition to the sigmoid function. One of the most popular is the ReLU (Rectified Linear Unit)

$$\sigma_{RELU}(x) = \max(0, x)$$



ReLU is more commonly used in training NN, where gradient stability and sparse activation is needed

Output Layer

$$y = g\left((W^{L_o-1})^T h^{L_o-1} + w_0^{L_o-1}\right)$$

g is the output (activation) function that depends on the task

- Binary classification
- Multi-class classification
- Regression

For binary classification we can use the sigmoid function
In general, it is common to use a softmax function for multiclass classification

$$y_k = \frac{\exp\left[(W_k^{L_o-1})^T h^{L_o-1} + w_{0,k}^{L_o-1}\right]}{\sum_{k=1}^{K} \exp\left[(W_k^{L_o-1})^T h^{L_o-1} + w_{0,k}^{L_o-1}\right]}$$