YΣ19 Artificial Intelligence II (Deep Learning for Natural Language Processing)
Fall Semester 2020
Homework II

Nikolaos Galanis - sdi1700019

November 2020

1 Decision boundaries for linear binary classification

Our goal is to prove the following equations, using basic operations from linear algebra.

The 2 equations are on the subject on classification with 2 variables: $x_1 and x_2$. The simplest representation of a linear discriminant function is obtained by taking a linear function of the input vector so that :

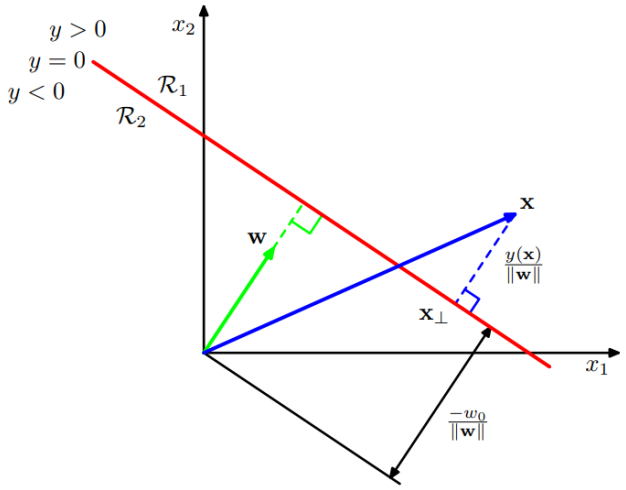
$$y(x) = w^T x + w_0$$

The first equation, takes the sub-case of x lying on the decision surface, where $y(x) = 0$. The equation can be easily proven by solving $y(x_0) = 0$ for the x_0 that satisfies the previously mentioned hypothesis, and then dividing both members by $||w||$, which as a vector length is a positive, non-zero number. Thus, the equation is proven as following:

$$\begin{aligned} y(x) = 0 \Leftrightarrow \\ w^T x + w_0 = 0 \Leftrightarrow \\ w^T x = -w_0 \Leftrightarrow \\ \frac{w^T x}{||w||} = -\frac{-w_0}{||w||} \end{aligned}$$

The second equation, takes an arbitrary point x in the hyperplane, and its orthogonal projection onto the decision surface x. Our goal is to prove that

$$x = x_\perp + r\frac{w}{||w||} \tag{1}$$



It is quite clear from (1) that $x - x_T = \vec{a}$. Because both $\vec{a}$ and $\vec{w}$ are vertical to the hyperplane $y = 0$, they are parallel, thus we can write $\vec{a}$ as:

$$\vec{a} = \lambda \times \vec{w} \tag{2}$$

Now our goal is to compute the λ value.

We know that r is the perpendicular distance of x from the hyperplane, thus the norm of the vector $\vec{a}$. Thus, we can write r as:

$$r = |\vec{a}| = |x - x_\perp|$$

Finally, by applying norms to the previous equation, we get that

$$\begin{aligned} |x - x_\perp| = \lambda|w| \Leftrightarrow \\ \lambda = \frac{|x - x_\perp|}{|w|} \Leftrightarrow \\ \lambda = \frac{r}{|w|} \end{aligned}$$

By substituting lambda value in (2), we get that

$$x - x_\perp = r\frac{w}{||w||} \Leftrightarrow x = x_\perp + r\frac{w}{||w||}$$

, which is what we wanted to prove.

2 Computing Partial Derivatives of the output layer

Let $x \in R^{1\times n}$ be a row vector, $W \in R^{n\times m}$ be a matrix and $z = xW$. Our goal is to compute $\frac{\partial z}{\partial x}$.

The result matrix after the multiplication of W with x, will be:

$$z = [x1 \quad x2 \quad \dots \quad x_n] \times \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1m} \\ \vdots & & & \\ w_{n1} & w_{n2} & \dots & w_{nm} \end{bmatrix} =$$

$$\Big[(w_{11}x1 + w_{21}x_2 + w_{n1}x_n) \quad \dots \quad (w_{1m}x1 + w_{2m}x_2 + w_{nm}x_n)\Big]$$

Thus, if we take the partial derivatives with respect to x, the result as we can easily see, is the initial matrix W. So, $\frac{\partial z}{\partial x} = W$

3 Computing Partial Derivatives of the logistic function

Let $\hat{y} = \sigma(x^T w)$, where $x, w \in R^{n\times 1}$ are column vectors and σ is the logistic function. Our goal is to compute $\frac{\partial \hat{y}}{\partial w}$

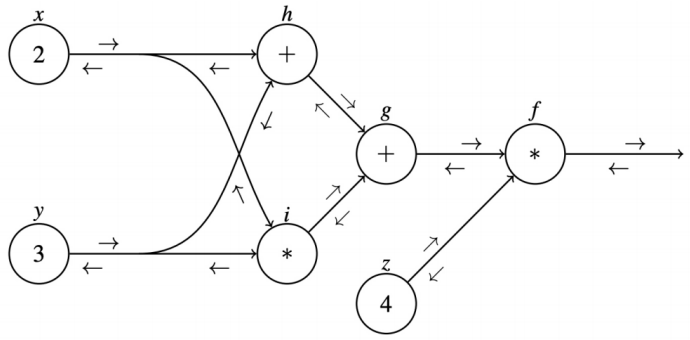
We can easily compute the partial derivatives by using the chain rule in the given equation.

We know by applying basic differentiation rules, that $\sigma'(t) = \frac{e^{-t}}{(1+e^{-t})^2}$. Thus, the result will be:

$$\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w}(\sigma(x^T w)) = \frac{\partial \sigma}{\partial w} \times \frac{\partial (x^T w)}{\partial w} = \frac{e^{-x^T w}}{(1 + e^{-x^T w})^2} \times x^T$$

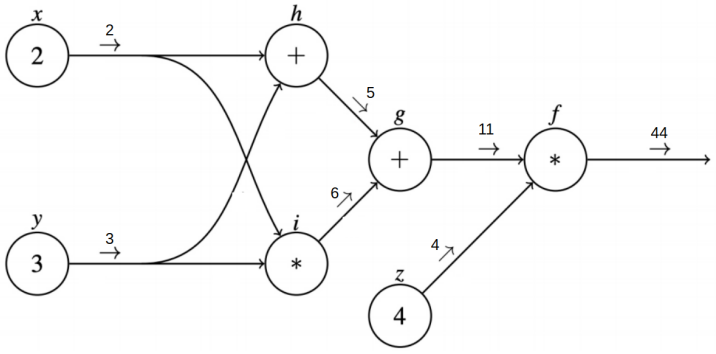
4 Applying Forward and Backward Propagation in a computational graph

Our goal is to apply both forward and backward propagation in the following graph:



4.1 Forward Propagation

The application of forward Propagation is rather simple: We just follow the rules of the nodes and the inputs, in order to produce the output. The results are shown in the following graph, the computations are not shown due to their simplicity.



We observe, that the final output of the graph is 44

4.2 Backward Propagation

In order to compute the results of the Backward Propagation method, we must first define the functions that we observe in each node of the graph. Starting from the

lower layers, we have the following functions:

$$h = x + y$$
$$i = x \times y$$
$$g = h + i = x + y + x \times y$$
$$f = 4 \times g = 4 \times g = 4 \times (x + y) \times (x \times y)$$

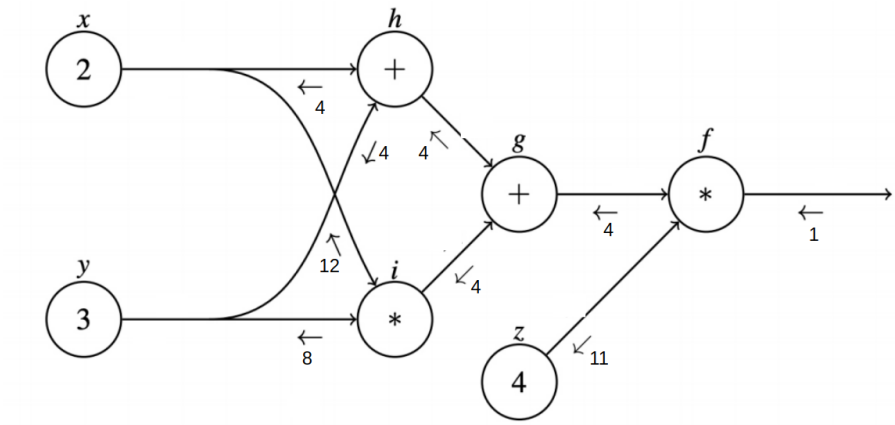
We must now define the local derivatives of each node. We will do that with respect to their input

$$\begin{aligned}
h: &\quad \frac{\partial h}{\partial x} = 1 &\quad \frac{\partial h}{\partial y} = 1 \\
i: &\quad \frac{\partial i}{\partial x} = y &\quad \frac{\partial i}{\partial y} = x \\
g: &\quad \frac{\partial g}{\partial h} = 1 &\quad \frac{\partial g}{\partial i} = 1 \\
f: &\quad \frac{\partial f}{\partial g} = z &\quad \frac{\partial f}{\partial z} = g
\end{aligned}$$

We are going to compute all the gradients, based on the rule $downstream = local \times upstream$, and by starting at the output layer of the graph, who's gradient is 1. Thus, for each node we have:

$$\begin{aligned}
f: &\quad down_z = 1 \times \frac{\partial f}{\partial z} = g = 11 \\
f: &\quad down_g = 1 \times \frac{\partial f}{\partial g} = z = 4 \\
g: &\quad down_h = 4 \times \frac{\partial g}{\partial h} = 4 \times 1 = 4 \\
g: &\quad down_i = 4 \times \frac{\partial g}{\partial i} = 4 \times 1 = 4 \\
h: &\quad down_x = 4 \times \frac{\partial h}{\partial x} = 4 \times 1 = 4 \\
h: &\quad down_y = 4 \times \frac{\partial h}{\partial y} = 4 \times 1 = 4 \\
i: &\quad down_x = 4 \times \frac{\partial i}{\partial x} = 4 \times y = 12 \\
i: &\quad down_y = 4 \times \frac{\partial i}{\partial y} = 4 \times x = 8
\end{aligned}$$

Thus, the final look of the graph with the back-propagation results(downstream gradients) is the following:



5 Sentiment Classifier using feed-forward NNs

Two different python notebooks were created, and can be found in the parent directory. I chose to construct models using 2 different types of features:

- TF-IDF features
- GloVe features, using the pre-trained vectors

Both of the notebooks are well-documented. In general, I observed that the model which used GloVe (along with an embedding layer), behaved better than the TF-IDF one. I compared all the models with F1, accuracy, precision and recall, and structured the appropriate plots.

Note: In the first notebook, i only used a 10% of the dataset to train the model, because colab's RAM could not handle all of it.