NEURAL NETWORKS INTRODUCTION
THE MULTI-CLASS CROSS ENTROPY LOSS FUNCTION

MY PERSONAL NOTES ON

LIST OF TOPICS

BY

0xLeo (github.com/0xleo)

UDACITY *Udacity*

Deep Learning Nanodegree

APRIL 15, 2019

DRAFT X.YY

MISSING: ...

Contents

1 Multiclass and Binary Logistic Regression Activation and Error Functions

This is a template that includes all packages and styles I need for writing notes.

1.1 Perceptron binary classification drawbacks

So far, we've seen the perceptron, which was trying to classify a set of data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ in two classes – positive and negative using the prediction formula $\hat{y} = f(w_1x_1 + w_2x_2 + b)$, where f is the step function. For each misclassified point, it would adjust the weights and bias w_1, w_2, b based on the coordinates of the point. An error function wasn't defined but simply measuring the number of misclassified points is not good enough for the job.

1.2 A better activation function – the sigmoid

The closer to 0 the argument of the step function is, i.e. $\mathbf{w} \cdot \mathbf{x}$ the more uncertain we are about which class it belongs to and instead of assigning it a probability of 1 or 0, we assign it a continuous value from 0 to 1. A good probability (activation) function that is symmetric around 0 and scales the output from 0 to 1 is the sigmoid function. For values $x \to -\infty$, its value is close to 0, for values close to 0 its value is 0.5 and for values $x \to \infty$, its value is close to 1.

Definition 1.1 (sigmoid function). *The sigmoid activation function is continuous and differentiable and defined as*

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.1}$$

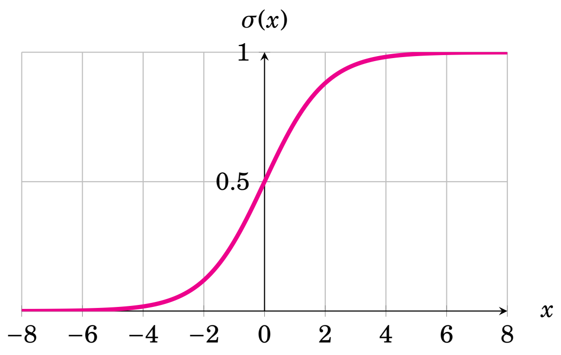

Fig. 1. The sigmoid function.

Suppose in the following figure that we classify some points based on their colour being blue/ not blue. Then $\hat{y} = f(\mathbf{w} \cdot \mathbf{x} + b) = 1$ for blue points that are certainly and $\hat{y} = 0$ for points that are certainly red. Each point in the plane will have a weighed sum $\mathbf{w} \cdot \mathbf{x} + b$, based on the weights and bias of the line. Using the sigmoid as the perceptron activation function, each point will be assigned a value $\in [0, 1]$ instead of 0 or 1.

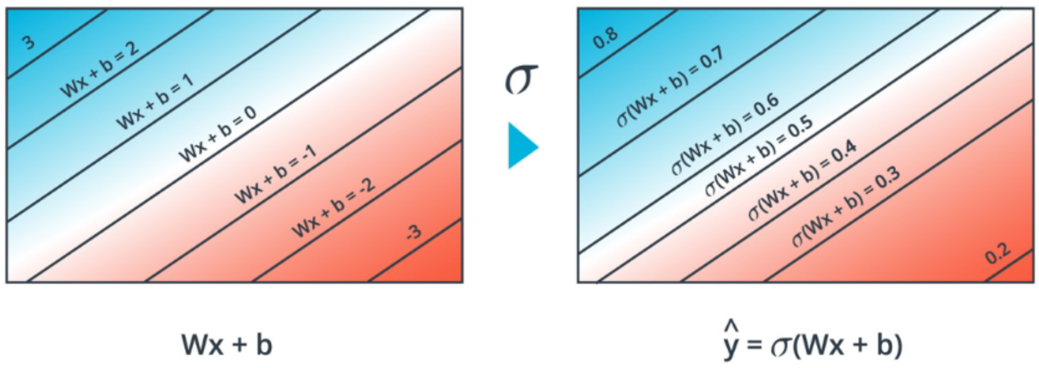

Fig. 2. The probability of being blue of each point in the x_1, x_2 space. Note that for each line the probability is the same.

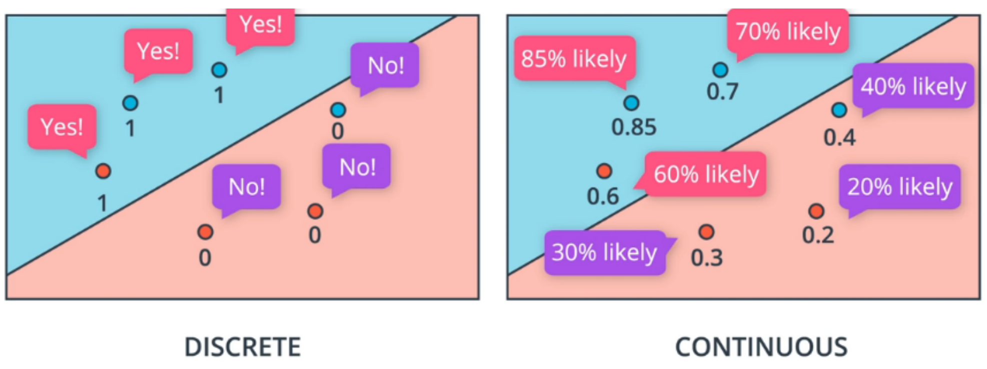

Fig. 3. Using the sigmoid, the perceptron outputs continuous values. Those on the black line are 0.5.

So far, the sigmoid-based perceptron outputs for each point a probility that it belongs to the blue class $P((blue))$ and one that doesn't, $P(\text{not}\quad\text{blue})$ such that $P(\text{blue}) + P(\text{not}\quad\text{blue}) = 1$.

1.2.1 Multi-class classification

1.2.2 One-hot encoding

The first difference between binary and multiclass classification is that the label y doe not only take the values 0 and 1, but it should take K different values, where K is the number of classes. Let's say we have some data $(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})$.

Suppose, for instance, that the data represent the output of an image classifier, and we have $K = 3$ classes.

Sigmoid doesn't work for more than two classes. In this section, we study softmax regression, which generalises logistic regression to classification problems where the class label y can take on more than two possible values. Suppose we have a set of n inputs $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}$ and we want to classify them in $K = 3$ classes – $y^{(i)} \in \{``person", ``hamster", ``capybara"\}$.

One way to encode them would be $y^{(i)} \in \{1, 2, 3\}$. However, that would imply that "hamster" is the average of "person" and "capybara" and wouldn't allow linear regression to run correctly. When classifying, there shouldn't be the concept of distance between classes. A better way to encode them would be using an one-hot vector for each class, particularly:

$$\text{person} \Rightarrow \mathbf{y}^{(1)} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top$$

$$\text{hamster} \Rightarrow \mathbf{y}^{(2)} = \begin{bmatrix}0 & 1 & 0\end{bmatrix}^\top$$

$$\text{capybara} \Rightarrow \mathbf{y}^{(3)} = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^\top$$

Definition 1.2 (one-hot encoding)**.** *If the number of classes is K, and a data point $\boldsymbol{x}^{(i)}$ belongs in the i-th class, then using one-hot encoding it is represented by:*

$$\boldsymbol{y}^{(i)} = \underbrace{\begin{bmatrix}0 & \dots & 0 & \underbrace{1}_{\text{index } i} & 0 & \dots & 0\end{bmatrix}^\top}_{K\times 1} \tag{1.2}$$

1.2.3 Activation function for multiclass model

The sigmoid activation only works for the binary model. In this section, we extend it for more than 2 classes, e.g. K classes. Once again, for each input data point $\mathbf{x}$ we need to compute the "pre-activation scores"

$$
\begin{aligned}
z^{(1)} &= \mathbf{x}\cdot\mathbf{w}^{(1)} + b^{(1)} = \mathbf{x}^\top\mathbf{w}^{(1)} + b^{(1)} \\
z^{(2)} &= \mathbf{x}^\top\mathbf{w}^{(2)} + b^{(2)} \\
&\dots \\
z^{(K)} &= \mathbf{x}^\top\mathbf{w}^{(K)} + b^{(K)}
\end{aligned}
$$

Each score is obtained by binary linear regression with weights $\mathbf{w}$. We now want one classifier such that:

1. Each output $\hat{\mathbf{y}}^k$ is non-negative.
2. The sum of $\hat{\mathbf{y}}^k$ over all K classes is 1.

The softmax function meets both of those requirements.

Definition 1.3 (softmax function). *Given a classifier of K classes, the softmax function outputs a $K \times 1$ vector where entry j has the value*

$$
softmax(\boldsymbol{z}^{(j)}) = \hat{\boldsymbol{y}}^{(j)} = \frac{\exp(\boldsymbol{z}_j)}{\sum\limits_{j=1}^{j=K}\exp(\boldsymbol{z}_j)} \tag{1.3}
$$

The softmax meets all the requirements of a probability function therefore the softmax of pre-activation score $\mathbf{z}^{(j)}$ represents the likelihood of the measurement $\mathbf{x}$ belonging in each class $y = j$.

Often we denote the softmax function as h as it represents the hypothesis given the weights and biases $\mathbf{w}, \mathbf{b}$.

Corollary 1.1. *The probabity of the data point $\boldsymbol{x}^{(i)}$ belonging in each class $1, 2, \dots, K$ is given by the softmax function:*

$$
\begin{bmatrix} P(1 \mid \boldsymbol{x}^{(i)}) \\ P(2 \mid \boldsymbol{x}^{(i)}) \\ \dots \\ P(K \mid \boldsymbol{x}^{(i)}) \end{bmatrix} = softmax(\boldsymbol{z}^{(i)}), \quad \boldsymbol{z}^{(i)} = \begin{bmatrix} \boldsymbol{w}_1^\top \boldsymbol{x}^{(i)} + b_1 \\ \boldsymbol{w}_2^\top \boldsymbol{x}^{(i)} + b_2 \\ \dots \\ \boldsymbol{w}_K^\top \boldsymbol{x}^{(i)} + b_K \end{bmatrix} \tag{1.4}
$$

Equivalently,

$$
P(k \mid \boldsymbol{x}^{(i)}) = \frac{\exp(\boldsymbol{w}_k^\top \boldsymbol{x}^{(i)} + b_k)}{\sum_{j=1}^{j=K}\exp(\boldsymbol{w}_j^\top \boldsymbol{x}^{(i)} + b_j)} \tag{1.5}
$$

$\sum_{j=1}^{j=K}\exp(\mathbf{w}_j^\top\mathbf{x}^{(i)} + b_j)$ is just the normalisation term so requirement (2) can be fulfiled. To summarise, the order of operation before the softmax of each class is computed is shown below.

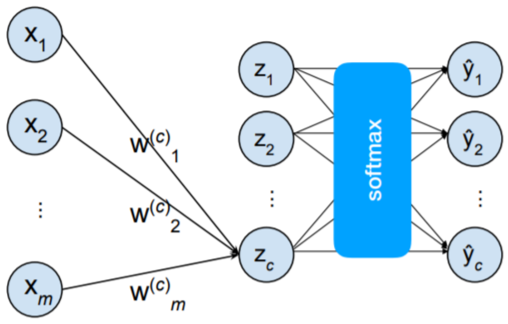

Fig. 4. The steps of softmax computation.

Example 1.1. *Let $m = 2$ be the number of features, $K = 3$ the classes and $\boldsymbol{x} = \begin{bmatrix}1 & 1\end{bmatrix}^\top$ and the weights of each class be $\boldsymbol{w}^{(1)} = \begin{bmatrix}-2.5 & -1\end{bmatrix}$, $\boldsymbol{w}^{(2)} = \begin{bmatrix}1 & 2\end{bmatrix}$, $\boldsymbol{w}^{(3)} = \begin{bmatrix}1 & 0\end{bmatrix}$. Which class does $\boldsymbol{x}$ belong to?*

Solution 1.1. *Find the $\boldsymbol{z}$ vector.*

$$\boldsymbol{z}^{(1)} = \boldsymbol{w}^{(1)\top}\boldsymbol{x} = \begin{bmatrix}-2.5 & -1\end{bmatrix}\begin{bmatrix}-1\\ 1\end{bmatrix} = 1.5$$

$$\boldsymbol{z}^{(2)} = \boldsymbol{w}^{(2)\top}\boldsymbol{x} = 1$$

$$\boldsymbol{z}^{(3)} = \boldsymbol{w}^{(3)\top}\boldsymbol{x} = -1$$

Using the softmax for each score we can find the prediction vector:

$$\hat{\boldsymbol{y}} = \begin{bmatrix}.592\\ .359\\ .049\end{bmatrix}$$

The prediction with highest probability is for class 1. ■

Corollary 1.2. *For $K = 2$ classes, the softmax and sigmoid functions for classification are equivalent.*

In the next section, we examine how to measure the correctness of the whole model, i.e. how $\mathbf{w}^{(1)}, \dots, \mathbf{w}^{(K)}$ have ben tuned such that $\hat{\mathbf{y}}$ is as close as possible to the ground truth $\mathbf{y}$.

1.3 Logistic regression error functions

1.3.1 Coming up with an error function – intuition

Let's consider the binary logistic regression problem again. Suppose we want to classify points as "blue" and "red" and we have the linear classifiers in the Fig. 5. Note that in Fig. 5, red points are *actually* red and so are the blue ($y^{(i)} = 1$ for blue, $y^{(i)} = 0$ for red). The colour gradient illustrates what the classifier has predicted for the points above or below it, i.e. the estimated probability of the point being blue $\hat{y}^{(i)} = P\left(y^{(i)} \mid \mathbf{x}^{(i)}\right) \in [0,1]$. Of course along the line we have $\hat{y}^{(i)} = 0.5$.

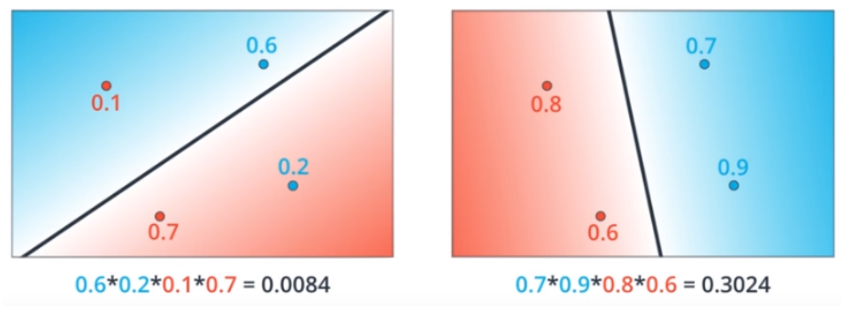

Fig. 5. Two linear binary classification models. The colour of the points shows what they *actually* are and the colour gradient what the predictor has *estimated*.

As shown in Fig. 5, a good way to measure the performance of each classifier w.r.t. the actual label is by measuring the total probability – $0.6 \cdot 0.2 \cdot 0.1 \cdot 0.7 = 0.0084$ and $0.7 \cdot 0.9 \cdot 0.8 \cdot 0.6 = 0.3024$ for the first and second model. The higher the total probability, the better the predictions. What's an analytical expression for this product?

The model outputs a value between 0 and 1, with 1 for points predicted as "blue" and 0 those as "red". For each term in the product, given that it's a blue, i.e. $y^{(i)} = 1$, we want $y^{(i)}\ \hat{y}^{(i)}$ to be large. Given that it's red, i.e. $y^{(i)} = 0$, we want $\left(1 - y^{(i)}\right)\ \left(1 - \hat{y}^{(i)}\right)$ to be large. Thus, the larger the product, the better the predictions.

1.3.2 Binary regression log loss cost function

When designing an error function, we want it to fulfil the following requirements.

- To be large when the prediction $\hat{y}$ is different from the actual label y.
- To be convex so that it can be optimised.

As motivated in the previous section, we want to find model weights and bias (denote them all w for convenience) that maximise the probability of the actual labels $y^{(i)}$ given the input $\mathbf{x}^{(i)}$:

$$w = \arg\max_w P_w\left(\left(y^{(1)}, \ldots, y^{(n)}\right) \,\middle|\, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\right)$$

Since each label $y^{(i)}$ is independent of the others as it depends only on the features of the corresponding example, this can expanded as products:

$$w = \arg\max_w P_w\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}\right) P_w\left(y^{(2)} \,\middle|\, \mathbf{x}^{(2)}\right) \ldots P_w\left(y^{(n)} \,\middle|\, \mathbf{x}^{(n)}\right)$$

This seems to be a good loss function but as shown in the previous section, especially for large numbers of data points, this may output a probability very close to 0, which can be numerically unstable. It's easier to work with sums, i.e. take the log of the product:

$$w = \arg\max_w \left(\log P_w\left(y^{(1)} \,\middle|\, \mathbf{x}^{(1)}\right) + \log P_w\left(y^{(2)} \,\middle|\, \mathbf{x}^{(2)}\right) + \ldots + \log P_w\left(y^{(n)} \,\middle|\, \mathbf{x}^{(n)}\right)\right)$$

The part inside the max is the objective function, or cost function. Because we usually want to minimise it, we flip the sign behind each term so:

$$w = \arg\min_w \left(-\sum_{i=1}^{n} \log P_w\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}\right)\right) \tag{1}$$

As established in the previous section $y^{(i)} = 1$ if $\mathbf{x}^{(i)}$ belongs to the "positive" class, therefore then we have $P_w\left(y^{(i)} \mid \mathbf{x}^{(i)}\right) = \hat{y}^{(i)}$. If $\mathbf{x}^{(i)}$ belongs to the "negative" class, then $y^{(i)} = 0$ therefore $P_w\left(y^{(i)} \mid \mathbf{x}^{(i)}\right) = 1 - \hat{y}^{(i)}$, i.e.

$$P_w\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}\right) = \begin{cases} \hat{y}^{(i)} & y^{(i)} = 1 \\ 1 - \hat{y}^{(i)} & y^{(i)} = 0 \end{cases}$$

This can be written in a more compact form as

$$P_w\left(y^{(i)} \,\middle|\, \mathbf{x}^{(i)}\right) = (\hat{y}^{(i)})^{y^{(i)}} (1 - \hat{y}^{(i)})^{1 - y^{(i)}}$$

Expanding its log in the sum in Eq. (1), the objective function to minimise is:

$$J(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \tag{2}$$

Often we divide the sum by the number of samples n to scale the cost function. It also implicitly depends on $\mathbf{w}$.

DEFINITION 1.4. *The cost function for binary logistic regression is defined as*

$$J(\boldsymbol{w}, \boldsymbol{b}) = -\frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \tag{1.6}$$

, where $y^{(i)}$ is the actual label for sample $\boldsymbol{x}^{(i)}$ and $\hat{y}^{(i)} = \sigma(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + \boldsymbol{b})$ the prediction that it belongs to the "positive" class.

$-y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$ is called error term and if we denote it as J_i, $J(\mathbf{w})$ makes sense as a loss function as

- When we have a positive prediction, i.e. $y^{(i)} = 1$, then we only care about the term $J_i = -y^{(i)} \log \hat{y}^{(i)}$) inside the sum. If the prediction is completely correct, $\log \hat{y}^{(i)} = 1$, then $J_i = 0$. If is it completely incorrect ($\log \hat{y}^{(i)} = 0$), then $J_i \to +\infty$.
- Similarly, for a negative prediction ($y^{(i)} = 0$), we care about $J_i = -(1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) = -\log(1 - \hat{y}^{(i)})$. if the prediction is completely correct ($\hat{y}^{(i)} = 0$), then $J_i = 0$ and if it completely wrong then $J_i \to +\infty$.
- It can be proven that it's convex therefore has a global minimum.

The dependance of the cost function J on the weights can be seen if we let $y := y^{(i)}$, $z := y^{(i)}$ and substutute $\hat{y}$ with $\sigma(z)$ in the error J_i inside the sum in Eq. (1.6):

$$
\begin{aligned}
J_i(y\,|\mathbf{x},\mathbf{w}) = -y\log(\sigma(z)) - (1-y)\log(1-\sigma(z)) = \\
-y\log(\frac{1}{1+e^{-z}}) - (1-y)\log(1-\frac{1}{1+e^{-z}}) = \\
y\log(1+e^{-z}) - (1-y)\log(\frac{e^{-z}}{1+e^{-z}}) = \\
y\log(1+e^{-z}) + (1-y)\log(e^{z}+1)
\end{aligned}
$$

1.3.3 The multi-class cross entropy loss function

When we have a classifier of $K > 2$ classes, it turns out the cost function can be derived in a similar way with the binary case – using the maximum likelihood principle and assuming all examples are mutually independent.

The binary class loss function can be written as a special of the following definition, for $K = 2$.

DEFINITION 1.5. *The multi-class logistic regression cost function with K classes and input data $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$ is defined as*

$$J(\boldsymbol{w},\boldsymbol{b}) = -\sum_{i=1}^{m}\sum_{k=1}^{K} y_{ki}\log\left(p_{ki}\right) \tag{1.7}$$

, where $p_{ki} = \log h_k(\boldsymbol{w}^T\boldsymbol{x}_i + \boldsymbol{b})$ is the estimated probability and h_k the k-th entry of the softmax hypothesis function.

This formula is not as complicated as it seems, as shown in the example below.

EXAMPLE 1.2. *Let's say we have 3 doors and behind each door there can be an animal. Suppose we have a model that has estimated the probabilities of each animal behind each door as follows. Find the cross entropy for door 1.*

Table 1: Multi-class probability table.

Animal	Door 1	Door 2	Door 3
Duck	0.7	0.3	0.1
Beaver	0.2	0.4	0.5
Walrus	0.1	0.3	0.4

SOLUTION 1.2.
For door 1:

$$
\begin{aligned}
CE &= -\sum_{k=1}^{3}\log p_{k1} \\
&= -\log 0.7 - \log 0.2 - \log 0.1 \\
&= 2.48
\end{aligned}
$$

The next section shows how the cost function J can be minimised.

A Appendices