FNLP Tutorial 2

1 The Softmax Function

The softmax function takes an arbitrary vector $\mathbf{v}$ as input, with $|\mathbf{v}|$ dimensions. It computes an output vector, also of $|\mathbf{v}|$ dimensions, whose ith element is given by:

$$\text{softmax}(\mathbf{v})_i = \frac{\exp(\mathbf{v}_i)}{\sum_{j=1}^{|\mathbf{v}|} \exp(\mathbf{v}_j)}$$

1. What is the purpose of the softmax function?

   Solution The softmax converts an arbitrary vector of $|\mathbf{v}|$ dimensions into a valid categorical probability distribution over $|\mathbf{v}|$ possible outcomes. In particular it ensures that all individual elements (probabilities) are non-negative and sum to one.

2. What is the purpose of the expression in the numerator?

   Solution The numerator ensures that all values are positive. Note that this is stronger than needed: the axioms of probability simply require all values to be non-negative. But exponentiation is only zero in the (negative) limit.

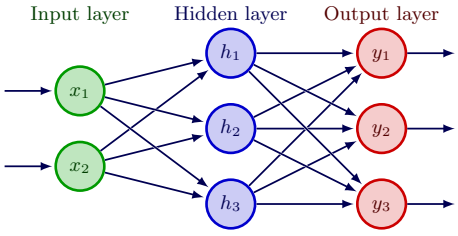
3. What is the purpose of the expression in the denominator?

   Solution The denominator normalises the distribution so that all individual probabilities sum to one.

2 Feed-forward neural networks

Consider a two-layer neural network with the topology visualised below, with the corresponding weights and bias values in the table. The hidden layer is followed by a non-linear function: the ReLU. The output layer is followed by a non-linear function too: the softmax. Read up on those functions and how to work with feedforward neural networks in sections 7.1 to 7.3 from J&M (only available in the 3rd edition!).

The network can be used for simple classification for three output classes. An input (consisting of features x_1 and x_2) belongs to one of the three classes, and you will classify an example input.



$w_{x_1,h_1} = 0.23$	$b_{h_1} = 0.13$	$w_{h_1,y_1} = 0.02$	$b_{y_1} = 0.31$
$w_{x_1,h_2} = -2.90$	$b_{h_2} = 0.90$	$w_{h_1,y_2} = -0.13$	$b_{y_2} = -0.18$
$w_{x_1,h_3} = -0.45$	$b_{h_3} = 0.24$	$w_{h_1,y_3} = 0.47$	$b_{y_3} = -0.56$
$w_{x_2,h_1} = 1.10$		$w_{h_2,y_1} = -0.88$	
$w_{x_2,h_2} = 0.05$		$w_{h_2,y_2} = 1.74$	
$w_{x_2,h_3} = -0.01$		$w_{h_2,y_3} = -0.09$	
		$w_{h_3,y_1} = -0.55$	
		$w_{h_3,y_2} = 0.72$	
		$w_{h_3,y_3} = -2.22$	

1. Compute the class an input with $x_1 = 1.50$, $x_2 = 3.11$ would belong to. Show the intermediate computations, not just the final class.

   Solution Firstly, let's compute the activation of the nodes in the hidden layers. This requires combining the inputs with the weights, adding the bias, and applying the ReLU function:
   $a_{h_1} = \text{ReLU}(1.50 \cdot 0.23 + 3.11 \cdot 1.10 + 0.13) = \text{ReLU}(3.896) = 3.896$
   $a_{h_2} = \text{ReLU}(1.50 \cdot -2.90 + 3.11 \cdot 0.05 + 0.90) = \text{ReLU}(-3.2945) = 0$
   $a_{h_3} = \text{ReLU}(1.50 \cdot -0.45 + 3.11 \cdot -0.01 + 0.24) = \text{ReLU}(-0.4661) = 0$

   Secondly, let's compute the activation of the output nodes **before** computing the softmax. We need all three outputs to be able to apply the softmax:
   $a_{y_1} = 3.896 \cdot 0.02 + 0 + 0 + 0.31 = 0.38792$
   $a_{y_2} = 3.896 \cdot -0.13 + 0 + 0 + -0.18 = -0.68648$
   $a_{y_3} = 3.896 \cdot 0.47 + 0 + 0 + -0.56 = 1.27112$
   Now, as a final step, we can apply the softmax to turn the outputs into probabilities:
   $p_1 = \frac{\exp(0.38792)}{\exp(0.38792)+\exp(-0.68648)+\exp(1.27112)} \approx 0.266$
   $p_2 = \frac{\exp(-0.68648)}{\exp(0.38792)+\exp(-0.68648)+\exp(1.27112)} \approx 0.091$
   $p_3 = \frac{\exp(1.27112)}{\exp(0.38792)+\exp(-0.68648)+\exp(1.27112)} \approx 0.643$

   This input is categorised with class 3!

2. Now imagine that you want to perform classification, but one input can belong to multiple classes. For example, when classifying a sentence with an emotion, that sentence can capture both anger and despair. To enable multi-class classification in this network, what adaptation would you make to its structure or the non-linear functions it uses?

   Solution A rather complicated solution would be creating output classes that represent multiple classes. For example, for 3 output classes, we would create separate output nodes for classes (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2, 3). This is the *wrong* solution due to the complexity it adds to the network.
   The most straightforward solution is replacing the non-linear function of the output layer with a sigmoid function, that computes a value between 0 and 1 for each class, and by thresholding that value, one would know whether an input belongs to that class or not.

3. Going back to the original example with $x_1 = 1.50$, $x_2 = 3.11$, use back-propagation to compute the derivative of each parameter if the gold label for that example was $y = 1$ (corresponding with the first class). Assume we use cross-entropy as the loss function:

$$\mathbf{L}(x) = -\sum_i p_i(x) \log q_i(x)$$

   Where $p_i(x)$ is the target probability of the i-th class for the gold labels (in this case, $p_1(x) = 1$, $p_2(x) = 0$, $p_3(x) = 0$) and $q_i(x)$ is the probability the network assigns to the i-th class.

   Solution We will use the notation that $\hat{y}_i$ is the output after the softmax function, and y_i is before the softmax. Similarly, we will use h_i to refer to the hidden layer value before the ReLU, and $\hat{h}_i$ after the ReLU

   We start with the first partial derivative:

$$\frac{d\mathbf{L}}{d\hat{y}_i} = \begin{cases} -\frac{1}{\hat{y}_1} & i = 1 \\ 0 & i = 2, 3 \end{cases}$$

From our lectures, $\frac{d\text{softmax}(x)_i}{dx_j} = \text{softmax}(x)_i[\delta_i(j) - \text{softmax}(x)_j]$. Applying the chain rule $\frac{d\mathbf{L}}{dy_i} = \frac{d\mathbf{L}}{d\hat{y}_1}\frac{\hat{y}_1}{dy_i}$, we have:

$$\frac{d\mathbf{L}}{dy_i} = \begin{cases} -(1-\hat{y}_1) & i = 1 \\ \hat{y}_i & i = 2, 3 \end{cases}$$

We can now compute the derivative for the bias and parameters of the last layer:

$$\frac{d\mathbf{L}}{dw_{\hat{h}_i,y_j}} = \frac{d\mathbf{L}}{dy_j}\hat{h_i} = \begin{pmatrix} -3.896(1-0.266) & 3.896 \cdot 0.091 & 3.896 \cdot 0.643 \\ 0(1-0.266) & 0 \cdot 0.091 & 0 \cdot 0.643 \\ 0(1-0.266) & 0 \cdot 0.091 & 0 \cdot 0.643 \end{pmatrix} = \begin{pmatrix} -2.8597 & 0.3545 & 2.5051 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

For the bias, we have $\frac{d\mathbf{L}}{db_{y_i}} = \frac{d\mathbf{L}}{dy_i}$, therefore $\frac{d\mathbf{L}}{db} = \begin{pmatrix} -0.734 & 0.091 & 0.643 \end{pmatrix}$.

We have that $\frac{dy_j}{d\hat{h}_i} = w_{\hat{h}_i,y_j}$; the ReLU derivative (again, from the lectures) is

$$\frac{d\text{ReLU}(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

so, we only need to compute $\frac{d\mathbf{L}}{dh_1}$ since $a_{h_2} = a_{h_3} = 0$, therefore $\frac{d\mathbf{L}}{dh_2} = \frac{d\mathbf{L}}{dh_3} = 0$. We have that $\frac{dy_i}{dh_1} = w_{\hat{h}_1,y_i}$

$$\frac{d\mathbf{L}}{dh_1} = \sum_j \frac{d\mathbf{L}}{dy_j}\frac{dy_j}{dh_1} = -0.734 \cdot 0.02 - 0.091 \cdot 0.13 + 0.643 \cdot 0.47 = 0.2757$$

Computing the gradient for the weights of the first layer becomes very similar to the previous step:

$$\frac{d\mathbf{L}}{dw_{x_i,h_j}} = \frac{d\mathbf{L}}{dh_j}x_i = \begin{pmatrix} 0.2757 \cdot 1.5 & 0 \cdot 1.5 & 0 \cdot 1.5 \\ 0.2757 \cdot 3.11 & 0 \cdot 3.11 & 0 \cdot 3.11 \end{pmatrix} = \begin{pmatrix} 0.4136 & 0 & 0 \\ 0.8574 & 0 & 0 \end{pmatrix}$$

For the bias, we have $\frac{d\mathbf{L}}{db_{h_1}} = \frac{d\mathbf{L}}{dh_1}$, therefore $\frac{d\mathbf{L}}{db_{h_1}} = 0.2757$. We also have that $\frac{d\mathbf{L}}{db_{h_2}} = \frac{d\mathbf{L}}{db_{h_3}} = 0$.

4. What are the benefits of back-propagation?

Solution Notice that we have just computed the partial derivatives of 22 parameters, which involved as well computing the partial derivatives of h_i and y_i before and after the activation function (so, in total, 34 partial derivatives!). In back-propagation, we iterate backwards from the last layer to avoid redundant calculations of intermediate terms in the chain rule, thus resulting in very efficient compute.

5. Update the parameters of the first layer, following simple gradient descent. Assume a learning rate $\mu = 0.1$.

Solution For w_{x_i,h_j}:

$$w_{x_i,h_j} = \begin{pmatrix} 0.23 & 2.90 & 0.45 \\ 1.10 & 0.05 & 0.01 \end{pmatrix} - 0.1 \begin{pmatrix} 0.4136 & 0 & 0 \\ 0.8574 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.19 & 2.90 & 0.45 \\ 1.01 & 0.05 & 0.01 \end{pmatrix}$$

For b_{h_i}:

$$b_{h_i} = \begin{pmatrix} 0.13 & 0.90 & 0.24 \end{pmatrix} - 0.1 \begin{pmatrix} 0.2575 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0.10 & 0.90 & 0.24 \end{pmatrix}$$