Machine Learning: Sheet 4

Djordje Zivanovic

November 27, 2018

1.

$$\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{a}} = -\frac{\partial \log a_y}{\partial \mathbf{a}} \tag{1}$$

$$= \left[-\frac{\partial \log a_y}{\partial a_1}, ..., -\frac{\partial \log a_y}{\partial a_y}, ..., -\frac{\partial \log a_y}{\partial a_C} \right] \tag{2}$$

$$= \left[0, ..., -\frac{1}{a_y}, ..., 0 \right] \tag{3}$$

The equation 1 is a definition of the objective function for a given point. The equation 2 is a definition of a gradient of a scalar function (derivative of scalar by a vector). The equation 3 uses property of derivative of a logarithm function.

$$\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{z}} = \frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{a}} \cdot \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \tag{4}$$

From the equation 4 we can see that we need to calculate $\frac{\partial \mathbf{a}}{\partial \mathbf{z}}$

$$\frac{\partial \mathbf{a}}{\partial \mathbf{z}} = \frac{\partial \left[\frac{e^{z_1}}{\sum_{l=1}^{C} e^{z_l}}, ..., \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}}, ..., \frac{e^{z_C}}{\sum_{l=1}^{C} e^{z_l}} \right]}{\partial \mathbf{z}} \tag{5}$$

$$= \begin{bmatrix} \frac{\partial \frac{e^{z_1}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_1} & \frac{\partial \frac{e^{z_1}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_2} & \cdots & \frac{\partial \frac{e^{z_1}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_C} \\ \frac{\partial \frac{e^{z_2}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_1} & \frac{\partial \frac{e^{z_2}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_2} & \cdots & \frac{\partial \frac{e^{z_2}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_C} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial \frac{e^{z_C}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_1} & \frac{\partial \frac{e^{z_C}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_2} & \cdots & \frac{\partial \frac{e^{z_C}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_C} \end{bmatrix} \tag{6}$$

The equation 5 is softmax classifier equation given in the text. The equation 6 is a Jacobian of derivatives of vectors. Let us denote:

$$S_i = \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}}$$

Then the equation 6 becomes:

$$\begin{bmatrix} \frac{\partial S_1}{\partial z_1} & \frac{\partial S_1}{\partial z_2} & \cdots & \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} & \frac{\partial S_2}{\partial z_2} & \cdots & \frac{\partial S_2}{\partial z_C} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial S_C}{\partial z_1} & \frac{\partial S_C}{\partial z_2} & \cdots & \frac{\partial S_C}{\partial z_C} \end{bmatrix} \tag{7}$$

In the equation 7 we notice that we have only two "different" type of derivatives, $\frac{\partial S_i}{\partial z_i}, i \in \{1, ..., C\}$ and

$\frac{\partial S_i}{\partial z_j}, i \neq j, i \in \{1, ..., C\}, j \in \{1, ..., C\}$.

$$\frac{\partial S_i}{\partial z_i} = \frac{\partial \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_i} \tag{8}$$

$$= \frac{e^{z_i} \cdot \sum_{l=1}^{C} e^{z_l} - e^{z_i} e^{z_i}}{\left(\sum_{l=1}^{C} e^{z_l}\right)^2} \tag{9}$$

$$= \frac{e^{z_i} \cdot \left(\sum_{l=1}^{C} e^{z_l} - e^{z_i}\right)}{\left(\sum_{l=1}^{C} e^{z_l}\right)^2} \tag{10}$$

$$= \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}} \frac{\sum_{l=1}^{C} e^{z_l} - e^{z_i}}{\sum_{l=1}^{C} e^{z_l}} \tag{11}$$

$$= S_i \cdot (1 - S_i) \tag{12}$$

The equation 8 uses S_i definition. The equation 9 uses derivative of division rule. The equations 10, 11, 12 are the simple transformations of the expressions and usage of definition S_i.

$$\frac{\partial S_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\sum_{l=1}^{C} e^{z_l}}}{\partial z_j} \tag{13}$$

$$= -\frac{e^{z_i} \cdot e^{z_j}}{\left(\sum_{l=1}^{C} e^{z_l}\right)^2} \tag{14}$$

$$= -S_i \cdot S_j \tag{15}$$

The equation 13 uses S_i definition. The equations 14 and 15 are simple arithmetic transformations of expressions expressions and usage of definition S_i. Finally we have:

$$\frac{\partial \mathbf{a}}{\partial \mathbf{z}} = \begin{bmatrix} \frac{\partial S_1}{\partial z_1} & \frac{\partial S_1}{\partial z_2} & \dots & \frac{\partial S_1}{\partial z_1} \\ \frac{\partial S_2}{\partial z_1} & \frac{\partial S_2}{\partial z_2} & \dots & \frac{\partial S_2}{\partial z_C} \\ \dots & \dots & \dots & \dots \\ \frac{\partial S_C}{\partial z_1} & \frac{\partial S_C}{\partial z_2} & \dots & \frac{\partial S_C}{\partial z_C} \end{bmatrix}$$

$$= \begin{bmatrix} S_1 \cdot (1 - S_1) & -S_1 \cdot S_2 & \dots & -S_1 \cdot S_C \\ -S_2 \cdot S_1 & S_2 \cdot (1 - S_2) & \dots & -S_2 \cdot S_C \\ \dots & \dots & \dots & \dots \\ -S_C \cdot S_1 & -S_C \cdot S_1 & \dots & S_C \cdot (1 - S_C) \end{bmatrix} \tag{16}$$

The equation 16 uses equations 12 and 15. Further we have:

$$\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{z}} = \frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{a}} \cdot \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \tag{17}$$

$$= \left[0, ..., -\frac{1}{a_y}, ..., 0\right] \cdot \begin{bmatrix} S_1 \cdot (1 - S_1) & -S_1 \cdot S_2 & \dots & -S_1 \cdot S_C \\ -S_2 \cdot S_1 & S_2 \cdot (1 - S_2) & \dots & -S_2 \cdot S_C \\ \dots & \dots & \dots & \dots \\ -S_C \cdot S_1 & -S_C \cdot S_1 & \dots & S_C \cdot (1 - S_C) \end{bmatrix} \tag{18}$$

$$= \left[\frac{S_y \cdot S_1}{a_y}, \frac{S_y \cdot S_2}{a_y}, \dots, \frac{S_y \cdot (S_y - 1)}{a_y}, \dots, \frac{S_y \cdot S_C}{a_y}\right] \tag{19}$$

$$= [S_1, S_2, \dots, S_y - 1 \dots, S_C] \tag{20}$$

The equation 17 is just another way to write the same derivative. The equation 18 replaces derivatives using equations 3 and 16. The equation 19 is a matrix multiplication. The equation 20 is a a simplification of the equation 19 using the property that $a_i = S_i$ [1]. The Generalizing the formula $\frac{\partial \ell}{\partial w_{ij}^2} = \frac{\partial \ell}{\partial z_i^2} \cdot \frac{\partial z_i^2}{\partial w_{ij}^2} = \frac{\partial \ell}{\partial z_i^2} \cdot x_j$

[1] I saw really late that $S_i = a_i$

we get:

$$\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{W}} = \left(\mathbf{x} \frac{\partial \ell}{\partial \mathbf{z}} \right)^{\mathrm{T}} = \begin{bmatrix} x_1 \cdot S_1 & x_2 \cdot S_1 & \dots & x_D \cdot S_1 \\ x_1 \cdot S_2 & x_2 \cdot S_2 & \dots & x_D \cdot S_2 \\ \dots & \dots & \dots & \dots \\ x_1 \cdot (S_y - 1) & x_2 \cdot (S_y - 1) & \dots & x_D \cdot (S_y - 1) \\ \dots & \dots & \dots & \dots \\ x_1 \cdot S_C & x_2 \cdot S_C & \dots & x_D \cdot S_C \end{bmatrix} \tag{21}$$

The equation 21 is a matrix multiplication of matrices.

$$\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{b}} = \frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{b}} \tag{22}$$

The equation 22 is just another way to write derivative. If we notice that $\frac{\partial \mathbf{z}}{\partial \mathbf{b}}$ is an identity matrix (b_i appears only in equality with the z_i), from the equation 22 we get that $\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{b}}$ is the same as $\frac{\partial \ell(\mathbf{W}, \mathbf{b}, \mathbf{x}, y)}{\partial \mathbf{z}}$. Using gradient descent for minibatch of B training examples we get the following update equations:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell(\mathbf{x}_i, y_i, \mathbf{W}_t, \mathbf{b}_t)}{\partial \mathbf{w}}$$

$$\mathbf{b}_{t+1} = \mathbf{b}_t - \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \ell(\mathbf{x}_i, y_i, \mathbf{W}_t, \mathbf{b}_t)}{\partial \mathbf{b}}$$

We just need to adjust orientation of $\mathbf{b}$ and the derivative in the last update rule so the addition is possible.

2. 1. I do not agree with the suggestion for several reasons. Let us notice that loss function of one training example $(\hat{\mathbf{y}}_i - \mathbf{y}_i)^2$ is just a sum of squares of errors (component-wise) of wrongly calculated outputs. The first reason is that in the binary coding approach not all errors are equal. If we encode 0 as 0000 and our neural network outputs 7, the error will be 3. On the other hand for the output 1 the error will be 1. This means, not all errors are equal, and they are related to the encoding. This could lead to unnecessarily longer training. Moreover, it could lead to stimulating errors with "lower" value in order to decrease the number of errors with bigger values.
   The next problem is that our algorithm even when it is trained properly it could classify something incorrectly. For example in one hot encoding output with two ones is incorrect (we know there is some problem by seeing the output). On the other hand, in binary encoding we do not have "obvious" errors in encoding, because all outputs are valid (except the encodings for numbers bigger than 10). The final problem is that it could stimulate some neurons to have lower values because specific digit in encoding is not frequent enough. For example the first digit in binary encoding appears only in 2 numbers (9 and 10), but the network may believe it is not so important and it could decrease the value.

   2. On the figure 1 is shown the neural network that fulfills requested in the task. $p_i(x) = bs(x - p)$ is the activation function, which for $x > p$ returns 1, and for $x \leq p$ returns 0. We choose p that suits the best, $p = 0.5$ is a reasonable choice. Further, weight (p_i, d_j) is 1 only in those cases where in binary representation of number i on j-th place is 1. So for example for p_7 the weights towards d_1, d_2, d_3, d_4 will be $1, 1, 1, 0$, for p_9 the weights will be $1, 0, 0, 1$ etc. We can say we added two additional layers, because power layer has just activation functions binary step, and the input weights are all one. The output layer is just linear activation function, whose weights are previously stated. Why the network gives the correct output is obvious, because only in cases where $y_i > p$ p_i-th node will be 1, and further binary layer will generate corresponding representation of the i-th number.

   3. The same problem as for the first subproblem of this problem, because we minimize the objective functions, thus the different errors will have the different influence on nodes.

3. 1. We have the following equations for new neural network further called NNH1. The other neural network (without hidden layer) will be called NNH0.

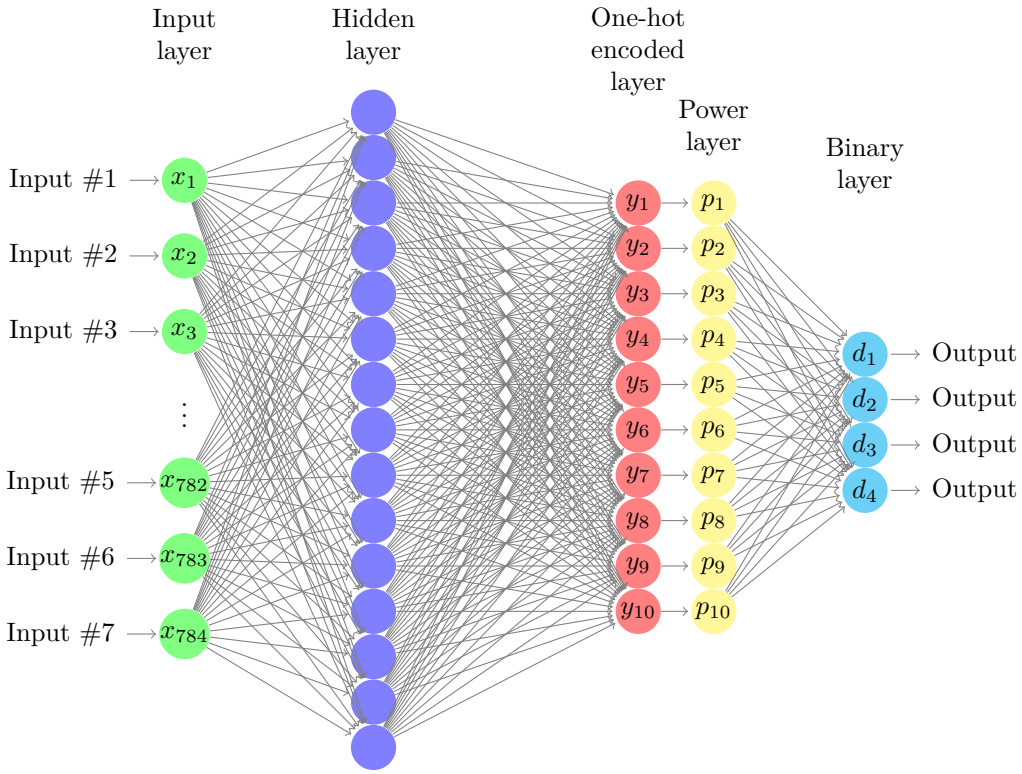

Figure 1: Neural network that transforms one hot-encoded output to binary output

The forward equations for NNH0 are:

$$\mathbf{z}^1 = \mathbf{a}^1 = \mathbf{x} \tag{23}$$
$$\mathbf{z}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2 \tag{24}$$
$$\mathbf{z}^2 = \mathbf{W}^2\mathbf{x} + \mathbf{b}^2 \tag{25}$$
$$\mathbf{a}^2 = \text{softmax}(\mathbf{z}^2) \tag{26}$$

The forward equations for NNH1 are:

$$\mathbf{z}^1 = \mathbf{a}^1 = \mathbf{x} \tag{27}$$
$$\mathbf{z}^2 = \mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2 \tag{28}$$
$$\mathbf{a}^2 = \mathbf{z}^2 \tag{29}$$
$$\mathbf{z}^3 = \mathbf{W}^3\mathbf{a}^2 + \mathbf{b}^3 \tag{30}$$
$$\mathbf{z}^3 = \mathbf{W}^3\mathbf{W}^2\mathbf{x} + \mathbf{W}^3\mathbf{b}^2 + \mathbf{b}^3 \tag{31}$$
$$\mathbf{a}^3 = \text{softmax}(\mathbf{z}^3) \tag{32}$$

Assume NNH1 has some representation that cannot be represented using NNH0. This would mean there is some combination on input $\mathbf{x}$ that doesn't give the same output. But this is incorrect because if we set $\mathbf{W}^2$ in NNH0 to be $\mathbf{W}^3\mathbf{W}^2$ from NNH1 and set $\mathbf{b}^2$ from NNH0 to be $\mathbf{W}^3\mathbf{b}^2 + \mathbf{b}^3$ from the NNH1, we get that $\mathbf{z}^2$ from NNH0 will always have the same values as $\mathbf{z}^3$ from NNH1. Thus, the assumption is incorrect.
Assume NNH0 has some representation that cannot be represented using NNH1. If we set up $\mathbf{b}^2$ from NNH1 to be zero vector and $\mathbf{b}^3$ from NNH1 to be $\mathbf{b}^2$ from NNH0 and $\mathbf{W}^3$ and $\mathbf{W}^2$ from NNH1

to be:

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & & \dots & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & 0 \end{bmatrix}, \mathbf{W}^2 = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \dots & w_{1,783} & w_{1,784} \\ w_{2,1} & w_{2,2} & w_{2,3} & \dots & w_{2,783} & w_{2,784} \\ w_{3,1} & w_{3,3} & w_{3,3} & \dots & w_{3,783} & w_{3,784} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{9,1} & w_{9,2} & w_{9,3} & \dots & w_{9,783} & w_{9,784} \\ w_{10,1} & w_{10,2} & w_{10,3} & \dots & w_{109,783} & w_{10,784} \\ 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix},$$

where $w_{i,j}$ in $\mathbf{W}^2$ is corresponding w from $\mathbf{W}^2$ from NNH0 we will get that our assumption is incorrect. We see that $\mathbf{W}^3$ is "expanded" identity matrix. Dimensions of identity matrix are 10×10 (without "unnecessary zeros"), and $\mathbf{W}^2$ is "expanded" $\mathbf{W}^2$ from NNH0 matrix. If we multiply these matrices, we will get the same $\mathbf{W}^2$ from NNH0.
This means our models can represent the same relationships between inputs and outputs.

2. $\mathbf{z}^2$ in NNH0 has stronger representation abilities than NNH1 for some weights, but NNH1 doesn't have stronger for some combinations. The part that all representations of weights from NNH1 can be represented using weights from NNH0 is the same as in the previous subproblem.
$\mathbf{W}^2$ and $\mathbf{W}^3$ are two matrices of dimensions 10×4 and 4×784. Because there is unlimited number of inputs, and there are limited number of parameters - weights and biases from NNH1 that could map these inputs - as in the previous problem a product of these two matrices has to be $\mathbf{W}^2$ from NNH0. But this is problematic because this means we would need to have decomposition of a matrix of dimensions 784×10 whose rank can be bigger than 4, into the product of two matrices of rank the most 4. This is impossible for all matrices of $\mathbf{W}^2$ (Echart-Young theorem).
The last question is what about softmax functions, maybe distinct inputs give as the result the same vector? This is impossible because, if we fix all outputs except one, the function will be monotonically increasing. This means function is increasing for all inputs (variables). Thus the function always generate unique values, and inputs have to be the same.

3. Let us assume that the cross entropy loss function is convex. Then let $(\mathbf{W}^{2:3*}, \mathbf{b}^{2:3*})$ be a minimal solution. This minimal solution and other ones have to give unique output values (convexity consequence). If we swap units (neurons) in the same layers, we will still have the same network, and the minimal solution will be the same. But because we swapped the units, we are allowed to swap weights belonging to swapped units going out of the same inputs and we now have a new solution that is minimal, but not the one from beginning. This means in some segment weights of a neural network need to generate the same values. Otherwise there would be a gap between two minimal solutions and all points between them would be above hyperplane that these two solutions would form. (this could be written $f(\lambda w_l + (1-\lambda)w_m, ...) \leq \lambda f(w_l, ...) + (1-\lambda)f(w_m) \leq f(w_m)$ which is not true, because $f(w_m, ...)$ is minimal solution). This would mean our function is concave, which is not what we assumed.
Hence, our weights generate the same loss function values in specific segments. Due to previous, for the NN given in text we can notice that all the way up to softmax we will have the same values at each output of neurons in the same layer, for minimal solution. This means minimal cross entropy has a value

$$-\log \frac{e^{z_y}}{\sum_{l=1}^{C} e^{z_l}} = -\log \frac{1}{C} = \log C$$

But if we take a solution where all the values of weights and biases coming to other outputs except C are 0, and ones coming to C are 1 (between input and hidden layer is unimportant, set them to positive) then we will have loss function whose value is 0 and that is less than our function. This is contradiction, because we assumed our minimal solution is minimal and outputs the unique value, which it does not.
Hence, we cannot get a convex optimization problem.

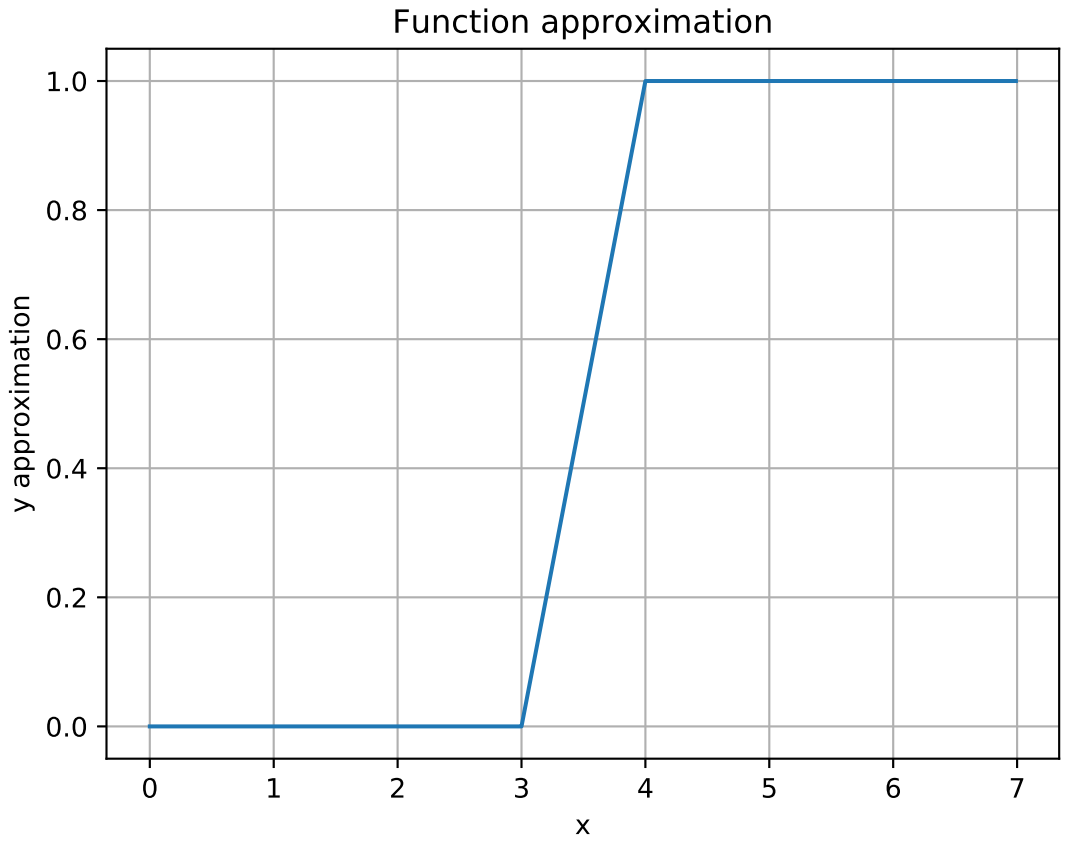

Figure 2: h(x)

4. 1.

$$
\begin{aligned}
a_0^1 &= z_0^1 = x \\
z_0^2 &= b_0^2 + w + 00^2 x \\
z_1^2 &= b_1^2 + w_{10}^2 x \\
a_0^2 &= \max(0, z_0^2) = \max(0, b_0^2 + w_{00}^2 x) \\
a_1^2 &= \max(0, z_1^2) = \max(0, b_1^2 + w_{10}^2 x) \\
z_0^3 &= a_0^2 w_{00}^3 + a_1^2 w_{01}^3 + b_0^3 \\
\hat{y} &= a_0^3 = z_0^3 = a_0^2 w_{00}^3 + a_1^2 w_{01}^3 + b_0^3 \\
&= \max(0, b_0^2 + w_{00}^2 x) w_{00}^3 + \max(0, b_1^2 + w_{10}^2 x) w_{01}^3 + b_0^3
\end{aligned}
$$

2. We can notice that the function $h(x) = \max(0, x-3) - \max(0, x-4)$.For $x < 3$ function is 0, for $x > 4$ function is $x - 4 + x - 3 = 1$ and for $3 \leq x \leq 4$ is $x - 3$In points 3 and 4 it has values 0 and 1.
3. By setting $b_0^2 = -c, b_1^2 = -d, b_0^3 = 0, w_{00}^2 = 1, w_{10}^2 = 1, w_{00}^3 = \frac{1}{d-c}, w_{01}^3 = -\frac{1}{d-c}$ we get

$$\hat{y} = \frac{\max(0, x-c)}{d-c} - \frac{\max(0, x-d)}{d-c}.$$

   For $x < c, \hat{y} = 0$. For $c \leq x \leq d$ $\hat{y} = \frac{x-c}{d-c} = \frac{1}{d-c}x - \frac{c}{d-c}$. For $x > d$ we have $\frac{x-c-(x-d)}{d-c} = 1$
4. First, let us prove that the $[0, 1]$ can be divided on a finite number of equal length subintervals where the difference between a minimum and maximum function value in that each interval is not bigger than ε.

   According to the definition of the continuous function: given any $\varepsilon > 0$, a $\delta > 0$ can be found such that for every x within the neighborhood of x_0 of radius δ ($|x - x_0| < \delta$) such that $|f(x) - f(x_0)| < \varepsilon$.

Let us assume otherwise, that $[0,1]$ cannot be divided into equal length intervals $[0,\delta,...,\frac{1}{\delta}]$ whose $|f(x) - f(\frac{i\delta+(i+1)\delta}{2})| < \varepsilon$.

This means no matter how small δ we set, there will always exist some i such $|f(x)-f(\frac{i\delta+(i+1)\delta}{2})| \geq \varepsilon$, but this means as $\delta \to 0$ there will be always some value in $[0,1]$ interval that doesn't fulfill condition of continuality. Contradiction!

Thus, let us divide our $[0,1]$ interval into equal length subintervals with δ that such that

$$\left| f(x) - f\left(\frac{i\delta + (i+1)\delta}{2}\right)\right| < \varepsilon, i \in \left\{0,, \frac{1-\delta}{\delta}\right\}$$

We will prove that we can make a sliced linear function in such a way that it traverses through these limitations (which in reality are rectangles), and changes its shape at the beginning of each subinterval. Because the function that we need to approximate is bounded and it can be negative, first, we will set b_0^3 such that $f(x) + b_0^3$ is always positive (in the worst case $b_0^3 = c$), this will be called $f_{positive}$. This means our sliced function has only to approximate positive bounded function. Moreover, if we concentrate on one specific rectangle (subinterval of function $f(x)$), we just need for a part of $h(x)$function to pass through it, so it starts in $f(i\delta)$ and it finishes in $f((i+1)\delta)$.

Lemma 1. *A unit in the hidden layer can produce function such that it has values* 0 *before some value* $x_0 \geq 0$ *and it behaves as any linear function that passes through* x_0 *after that.*

Proof. Let us denote with w an input weight to this hidden unit, and b a bias and w_o weight to the output layer of this unit. For a linear function $kx + n$ to pass through x_0: $kx_0 + n = 0$, $k = -\frac{n}{x_0}$. Thus we can set k to be positive as much as we want by setting n to negative values. This is easily achieved by setting $b = -kx_0$ and setting $w = k$ as we want, and setting $w_o = 1$. This only works for $k \geq 0$. If we need to set $k \leq 0$, we use the same trick with absolute value of k and set w_0 to -1. Thus $b = -|k|x_0, w = |k|, w_o = sgn(k), k \neq 0$, in the second case $b = |c|, w = 0, w_o = sgn(c), k = 0$. □

Using lemma 1 we can just adjust the flow of our function so they "adjust" current linear function so it starts in $f(i\delta)$ and ends in $f((i+1)\delta)$. For the first interval using lemma 1 $k = \frac{f(\delta)-f(0)}{\delta}$ and we should increase b_0^3 so that $h(0) = f_{positive}(0)$. So until the "end" of the first subinterval our function is in the ε limits of $f(x)$. When it enters the second subinterval, we need to correct its coefficient in the following way: if the desired coefficient for new subinterval is k_r than k in a new unit is $k_r - k_{previous}$, because $(k_r - k_{previous})x + k_{previous} = k_r x$. By applying this rule to every subinterval and generating new units in hidden layer consecutively, we will get the neural network that produces function $h(x)$, which approximates $f(x)$ in limit ε.

5. Let us notice:

$$\hat{y} = b_0^3 + \sum_{i=1}^{N} \max(0, b_i^2 + w_{i0}^2 x) w_{0i}^3$$

If we notice that $b_i^2 + w_{i0}^2 x$ are linear functions and they will only cut x axis only once (unless they are parallel, but they won't cut the axis then, or in case when the line is axis, but then it won't have any influence on output because the value is 0). Thus, after the point of intersection (if one exists) their activation functions are either linear or constant. Because we have finite number of neurons, there is a maximal number x in these cuts. After that point all functions will behave in the same way (either as linear function or constant). If we sum functions we will only have $a_i x + c_i$ and some constants. When we group coefficients next to x and replace their sum with m and the sum of constants and b_0^3 with n we get the claim from the subproblem.

6. I wouldn't invest for the following reasons. First, there is an infinite number of prime and composite numbers. Let us assume there is limited number of primes. If we multiple all primes and add 1 to the product, we will get number that is coprime to each of the primes, and thus each of the composite numbers.

   The second, because the function, which is generated using types of NNs in this problem, after some time becomes linear (previous subproblem). They cannot derive anything from this function, because it is always increasing at the same pace, but number of prime numbers is unlimited, as the number of composite numbers.