Introduction to Machine Learning
Neural Network Notes with Solved Problems

Prof. Sundeep Rangan

1 Description of Neural Networks

- Neural networks refer to a general class of models for making predictions on data. The key feature of neural networks is that the input data is processed in multiple stages, called *layers*. The parameters in each layer are *trained* or *learned* from examples. This multi-layer processing with trainable parameters in each layer is loosely inspired by biological neural systems.
- To describe the neural network, let $\mathbf{x}$ denote the input vector and let y denote the target variable that we wish to predict from $\mathbf{x}$.
- We consider networks for both **regression** or **classification** problems:
  - For regression problems, y is a scalar or vector and is typically continuous-valued. In this case, the neural network produces a prediction $\hat{y}$ of y of the same dimension as y.
  - For classification problems, the target variable $y \in \{1, \ldots, K\}$ and the neural network typically provides a soft prediction of the class label. Specifically, the networks makes a prediction of the probability $P(y = k|\mathbf{x})$ for each class label k given the input $\mathbf{x}$.
- For both regression and classification problems, we assume the input $\mathbf{x}$ is a vector of dimension N_i so that
$$\mathbf{x} = (x_1, \ldots, x_{N_i}).$$
- In this note, we look at neural networks with one hidden layer. In such a network, the neural network mapping is performed in two steps – each step is called a *layer*:
  - *Hidden layer* produces outputs $\mathbf{z}^{\rm H}$ and $\mathbf{u}^{\rm H}$ of dimension N_h.
  - *Output layer* produces outputs $\mathbf{z}^{\rm O}$ and $\mathbf{u}^{\rm O}$ of dimension N_o.

  The dimensions N_h and N_o will be discussed below.
- The equations for the two layers with a single input $\mathbf{x}$ are:

$$\text{Hidden layer:} \quad z_j^{\rm H} = \sum_{k=1}^{N_i} W_{jk}^{\rm H} x_k + b_j^{\rm H}, \quad u_j^{\rm H} = g_{\rm act}(z_j^{\rm H}), \quad j = 1, \ldots, N_h \tag{1a}$$

$$\text{Output layer:} \quad z_j^{\rm O} = \sum_{k=1}^{N_h} W_{jk}^{\rm O} u_k^{\rm H} + b_j^{\rm O}, \quad u^{\rm O} = g_{\rm out}(\mathbf{z}^{\rm O}). \quad j = 1, \ldots, N_o. \tag{1b}$$

- In the hidden layer, the function $g_{\rm act}(z)$ is called the *activation function*. There are three common choices:
  - Hard threshold:
$$g_{\rm act}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0. \end{cases} \tag{2}$$
  - Sigmoid: $g_{\rm act}(z) = 1/(1+e^{-z})$.
  - Rectified linear unit (ReLU): $g(z) = \max\{0, z\}$.
- The output map $g_{\rm out}(z)$ and dimension N_o depends on the type of prediction problem we are using the neural network for:
  - Binary classification: In this case, the response is $y = \{0, 1\}$. To predict the response, we typically take $N_o = 1$ and the output $u^{\rm O}$ is the class probability:
$$P(y = 1|\mathbf{x}) = u^{\rm O} = g_{\rm out}(z^{\rm O}) = \frac{1}{1+e^{-z^{\rm O}}}. \tag{3}$$
    The variable $z^{\rm O}$ is called the *logit* and the output mapping (3) is a sigmoid. We can also use $z^{\rm O}$ to make a hard decision by selecting the most likely class:
$$\hat{y} = \begin{cases} 1 & z^{\rm O} \geq 0 \\ 0 & z^{\rm O} < 0. \end{cases} \tag{4}$$
  - K-class classification: In this case, the target is a class label $y = 1, \ldots, K$. We typically take $N_o = K$ and use the soft-max function for the class probability:
$$P(y = k|\mathbf{x}) = u^{\rm O}_k = g_{\rm out,k}(z^{\rm O}) = \frac{e^{z^{\rm O}_k}}{\sum_{\ell=1}^K e^{z^{\rm O}_\ell}}. \tag{5}$$
    Again, we can make a hard decision on the class label by selecting the highest logit score:
$$\hat{y} = \arg\max_{k=1,\ldots,K} z^{\rm O}_k. \tag{6}$$
  - Regression: In this case, one is trying to predict $\mathbf{y} \in \mathbb{R}^d$, where d is the number of variables to predict and each component y_j is typically continuous-valued. For this problem, we take $N_o = d$ and $\mathbf{u}^{\rm O}$ is the prediction of $\mathbf{y}$,
$$\hat{\mathbf{y}} = \mathbf{u}^{\rm O} = g_{\rm out}(\mathbf{z}^{\rm O}) = \mathbf{z}^{\rm O}. \tag{7}$$
    In (7), we will either say there is no activation or an *identity* activation.
- The number, N_h of hidden variables (also called the number of *hidden units*) represents the model complexity. Higher values of N_h can express more complex mappings, but will need more data to train.

- When $N_o = 1$ (single output unit), we drop the subscript j in (1b) and write

$$\text{Output layer:} \quad z^{\mathrm{O}} = \sum_{k=1}^{N_h} W_k^{\mathrm{H}} u_k^{\mathrm{H}} + b^{\mathrm{O}}, \quad \hat{y} = g_{\mathrm{out}}(z^{\mathrm{O}}). \tag{8}$$

- Matrix form of the neural network with single sample input $\mathbf{x}$:

$$\begin{aligned} \mathbf{z}^{\mathrm{H}} &= \mathbf{W}^{\mathrm{H}}\mathbf{x} + \mathbf{b}^{\mathrm{H}}, \quad \mathbf{u}^{\mathrm{H}} = g_{\mathrm{act}}(\mathbf{z}^{\mathrm{H}}), \\ \mathbf{z}^{\mathrm{O}} &= \mathbf{W}^{\mathrm{O}}\mathbf{u}^{\mathrm{H}} + \mathbf{b}^{\mathrm{O}}, \quad \mathbf{u}^{\mathrm{O}} = g_{\mathrm{out}}(\mathbf{z}^{\mathrm{O}}). \end{aligned}$$

- Batch form: Often we have a batch of input-output samples, $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$ (e.g. a mini-batch in training or test). In this case, we need to add an index i to each sample and rewrite (1) as:

$$\text{Hidden layer:} \quad z_{ij}^{\mathrm{H}} = \sum_{k=1}^{N_i} W_{jk}^{\mathrm{H}} x_{ik} + b_j^{\mathrm{H}}, \quad u_{ij}^{\mathrm{H}} = g_{\mathrm{act}}(z_{ij}^{\mathrm{H}}), \quad j = 1, \ldots, N_h \tag{9a}$$

$$\text{Output layer:} \quad z_{ij}^{\mathrm{O}} = \sum_{k=1}^{N_h} W_{jk}^{\mathrm{O}} u_{ik}^{\mathrm{H}} + b_j^{\mathrm{O}}, \quad \mathbf{u}_j^{\mathrm{O}} = g_{\mathrm{out}}(\mathbf{z}_i^{\mathrm{O}}), \quad j = 1, \ldots, N_o. \tag{9b}$$

- Matrix form of batch processing in neural networks: Define the matrices,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^{\mathsf{T}} \\ \vdots \\ \mathbf{x}_N^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1,N_i} \\ \vdots & \vdots & \vdots \\ x_{N1} & \cdots & x_{N,N_i} \end{bmatrix}, \quad \mathbf{Z}^{\mathrm{H}} = \begin{bmatrix} (\mathbf{z}_1^{\mathrm{H}})^{\mathsf{T}} \\ \vdots \\ (\mathbf{z}_N^{\mathrm{H}})^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} z_{11}^{\mathrm{H}} & \cdots & z_{1,N_h}^{\mathrm{H}} \\ \vdots & \vdots & \vdots \\ z_{N1}^{\mathrm{H}} & \cdots & z_{N,N_h}^{\mathrm{H}} \end{bmatrix},$$

Similarly define

$$\mathbf{U}^{\mathrm{H}} = \begin{bmatrix} (\mathbf{u}_1^{\mathrm{H}})^{\mathsf{T}} \\ \vdots \\ (\mathbf{u}_N^{\mathrm{H}})^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} u_{11}^{\mathrm{H}} & \cdots & u_{1,N_h}^{\mathrm{H}} \\ \vdots & \vdots & \vdots \\ u_{N1}^{\mathrm{H}} & \cdots & u_{N,N_h}^{\mathrm{H}} \end{bmatrix}. \quad \mathbf{U}^{\mathrm{O}} = \begin{bmatrix} (\mathbf{u}_1^{\mathrm{O}})^{\mathsf{T}} \\ \vdots \\ (\mathbf{u}_N^{\mathrm{O}})^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} u_{11}^{\mathrm{O}} & \cdots & u_{1,N_o}^{\mathrm{H}} \\ \vdots & \vdots & \vdots \\ u_{N1}^{\mathrm{O}} & \cdots & u_{N,N_o}^{\mathrm{H}} \end{bmatrix}.$$

Hence each row contains all the components for the i-th sample. Then, we can write

$$\mathbf{Z}^{\mathrm{H}} = \mathbf{X}\left[\mathbf{W}^{\mathrm{H}}\right]^{\mathsf{T}} + \mathbf{B}^{\mathrm{H}}, \quad \mathbf{U}^{\mathrm{H}} = g_{\mathrm{act}}(\mathbf{Z}^{\mathrm{H}}) \tag{10a}$$

$$\mathbf{Z}^{\mathrm{O}} = \mathbf{U}^{\mathrm{H}}\left[\mathbf{W}^{\mathrm{O}}\right]^{\mathsf{T}} + \mathbf{B}^{\mathrm{O}} \quad \mathbf{U}^{\mathrm{O}} = g_{\mathrm{out}}\left(\mathbf{Z}^{\mathrm{O}}\right), \tag{10b}$$

where $\mathbf{B}^{\mathrm{H}}$ and $\mathbf{B}^{\mathrm{O}}$ repeat the bias vectors on the N rows

$$\mathbf{B}^{\mathrm{H}} = \begin{bmatrix} (\mathbf{b}^{\mathrm{H}})^{\mathsf{T}} \\ \vdots \\ (\mathbf{b}^{\mathrm{H}})^{\mathsf{T}} \end{bmatrix} \quad \mathbf{B}^{\mathrm{O}} = \begin{bmatrix} (\mathbf{b}^{\mathrm{O}})^{\mathsf{T}} \\ \vdots \\ (\mathbf{b}^{\mathrm{O}})^{\mathsf{T}} \end{bmatrix} \tag{11}$$

Problems

1. Consider the neural network (1) with a scalar input x and parameters

$$W^{\mathrm{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^{\mathrm{H}} = \begin{bmatrix} -1 \\ -3 \end{bmatrix} \quad W^{\mathrm{O}} = [-1, 2], \quad b^{\mathrm{O}} = 0.5,$$

using a hard threshold activation function (2) and threshold output function (4).

   (a) What is N_h, the number of hidden units? What is N_o, the number of output units?

   (b) Write $\mathbf{z}^{\mathrm{H}}$ in terms of x. Draw the functions for each component z_j^{H}.

   (c) Write $\mathbf{u}^{\mathrm{H}}$ in terms of x. Draw the functions for each component u_j^{H}.

   (d) Write z^{O} in terms of x.

   (e) What is $\hat{y}$ in terms of x?

2. Consider the data set for four points with scalar features x_i and binary class labels $y_i = 0, 1$.

x_i	0	1	3	5
y_i	0	0	1	0

   (a) Find a neural network with $N_h = 2$ units, $N_o = 1$ output units such that $\hat{y}_i = y_i$ on all four data points. Use a hard threshold activation function (2) and threshold output function (4). State the weights and biases used in each layer.

   (b) Compute the values of $\hat{y}_i$ and all the intermediate variables $\mathbf{z}_i^{\mathrm{H}}$, $\mathbf{u}_i^{\mathrm{H}}$ and z_i^{O} for each sample $x = x_i$.

   (c) Now suppose we are given a new sample, $x = 3.5$. What does the network predict as $\hat{y}$?

3. *Two-dimensional example:* Consider a neural network on a 2-dimensional input $\mathbf{x} = (x_1, x_2)$ with weights and biases:

$$W^{\mathrm{H}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad b^{\mathrm{H}} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad W^{\mathrm{O}} = [1, 1, -1], \quad b^{\mathrm{O}} = -1.5.$$

Assume the network uses hard a threshold activation function (2) and threshold output function (4).

   (a) Write the components of $\mathbf{z}^{\mathrm{H}}$ and $\mathbf{u}^{\mathrm{H}}$ as a function of (x_1, x_2). For each component j, indicate where in the (x_1, x_2) plane $u_j^{\mathrm{H}} = 1$.

   (b) Write z^{O} as a function of (x_1, x_2). In what region is $\hat{y} = 1$?

4. *Architecture choices for different problems:* For each problem, state possible selections for the dimensions N_i, N_h, N_o and the functions $g_{\mathrm{act}}(\cdot)$ and $g_{\mathrm{out}}(\cdot)$. Indicate which parameters are free to choose.

   (a) One wants a neural network to take as an input a 20×20 gray scale image and determine which letter ('a' to 'z') the image is of.

(b) One extracts 120 features of a sample of a speech recording (like the MFCCs). Based on the audio samples, the network is to determine if the speech is male or female.

(c) One wants a neural network to predict the stock price based on the average stock price of the last five days.

5. *Implementation in python:* Write python code for implementing the following steps for a batch of samples:

   (a) The hidden layer step (10a).

   (b) The output layer step (10b) for binary classification with a sigmoid output (3).

   (c) The output layer step (10b) for K-class classification with a softmax output (5).

   For all examples, try to avoid for-loops.

Solutions

1. (a) Since $\mathbf{W}^{\text{H}}$ has 2 ouptuts, $N_h = 2$. Since W^{O} has 1 output, $N_o = 1$.

   (b) The components of the linear variables $\mathbf{z}^{\text{H}}$ in the hidden layer are:

$$\mathbf{z}^{\text{H}} = \mathbf{W}^{\text{H}}\mathbf{x} + \mathbf{b}^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x + \begin{bmatrix} -1 \\ -3 \end{bmatrix} = \begin{bmatrix} x-1 \\ x-3 \end{bmatrix}.$$

   (c) The activation outputs in the hidden layer are

$$\mathbf{u}^{\text{H}} = g_{\text{act}}(\mathbf{z}^{\text{H}}) = \begin{bmatrix} g_{\text{act}}(x-1) \\ g_{\text{act}}(x-3) \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{\{x\geq 1\}} \\ \mathbb{1}_{\{x\geq 3\}} \end{bmatrix}.$$

   (d) The linear variable in the output layer is

$$\begin{aligned} z^{\text{O}} = W^{\text{O}}\mathbf{u}^{\text{H}} + b^{\text{O}} &= [-1, 2] \begin{bmatrix} \mathbb{1}_{\{x\geq 1\}} \\ \mathbb{1}_{\{x\geq 3\}} \end{bmatrix} + 0.5 \\ &= -\mathbb{1}_{\{x\geq 1\}} + 2\mathbb{1}_{\{x\geq 3\}} + 0.5 = \begin{cases} 0.5 & \text{when } x < 1 \\ -0.5 & \text{when } x \in [1,3) \\ 1.5 & \text{when } x \geq 3 \end{cases} \end{aligned}$$

   (e) The predicted output $\hat{y}$ is

$$\hat{y} = \mathbb{1}_{\{z^{\text{O}}\geq 0\}} = \begin{cases} 0 & \text{if } x \in [1,3) \\ 1 & \text{else} \end{cases}$$

2. (a) We can use a network similar in structure to the previous problem. In the hidden layer, we want to find features that can distinguish between $x = 3$ which belongs to class $y = 1$, and $x = \{0, 1, 5\}$ which belong to class $y = 0$. There are lots of choices, but we will extract two features: whether $x \geq 2$ and $x \geq 4$. We can do this with the linear transforms:

$$\mathbf{W}^{\text{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^{\text{H}} = \begin{bmatrix} -2 \\ -4 \end{bmatrix},$$

	Training				Test
x_i	0	1	3	5	3.5
$z_{i1}^{\mathrm{H}} = x - 2$	-2	-1	1	3	1.5
$z_{i2}^{\mathrm{H}} = x - 2$	-4	-3	-1	1	0.5
$u_{i1}^{\mathrm{H}} = g_{\mathrm{act}}(z_{i1}^{\mathrm{H}})$	0	0	1	1	1
$u_{i1}^{\mathrm{H}} = g_{\mathrm{act}}(z_{i2}^{\mathrm{H}})$	0	0	0	1	0
$z_i^{\mathrm{O}} = u_{i1}^{\mathrm{H}} - 2u_{i1}^{\mathrm{H}} - 0.5$	-0.5	-0.5	0.5	-1.5	0.5
$\hat{y}_i = g_{\mathrm{out}}(z_i^{\mathrm{O}})$	0	0	1	0	1
y_i	0	0	1	0	

Table 1: Computations for the neural network in Problem 2

so that

$$\mathbf{z}^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}}\mathbf{x} + \mathbf{b}^{\mathrm{H}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x + \begin{bmatrix} -2 \\ -4 \end{bmatrix} = \begin{bmatrix} x-2 \\ x-4 \end{bmatrix}.$$

Then, the activation outputs in the hidden layer are

$$\mathbf{u}^{\mathrm{H}} = g_{\mathrm{act}}(\mathbf{z}^{\mathrm{H}}) = \begin{bmatrix} g_{\mathrm{act}}(x-2) \\ g_{\mathrm{act}}(x-4) \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{\{x\geq 2\}} \\ \mathbb{1}_{\{x\geq 4\}} \end{bmatrix}.$$

Now we need to combine these functions in a way that they are negative for samples x_i when $y_i = 0$ and positive on sample x_i when $y_i = 1$. Similar to the previous problem, take

$$W^{\mathrm{O}} = [1, -2], \quad b^{\mathrm{O}} = -0.5.$$

Then,

$$\begin{aligned} z^{\mathrm{O}} = W^{\mathrm{O}}\mathbf{u}^{\mathrm{H}} + b^{\mathrm{O}} &= [1, -2] \begin{bmatrix} \mathbb{1}_{\{x\geq 2\}} \\ \mathbb{1}_{\{x\geq 4\}} \end{bmatrix} - 0.5 \\ &= \mathbb{1}_{\{x\geq 2\}} - 2\mathbb{1}_{\{x\geq 4\}} - 0.5 = \begin{cases} -0.5 & \text{when } x < 2 \\ 0.5 & \text{when } x \in [2, 4) \\ -1.5 & \text{when } x \geq 4. \end{cases} \end{aligned}$$

The predicted output $\hat{y}$ is

$$\hat{y} = \mathbb{1}_{\{z^{\mathrm{O}}\geq 0\}} = \begin{cases} 1 & \text{if } x \in [2, 4) \\ 0 & \text{else} \end{cases}$$

This matches the training data.

(b) Table 1 computes the values for all intermediate variables for the four training data points. We can see that $\hat{y}_i = y_i$ for all four points.

(c) The value for $x = 3.5$ is shown in the final column of Table 1. For this value of $x = 3.5$, we get $\hat{y} = 1$.

3. (a) The linear functions in the hidden layer are:

$$\mathbf{z}^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}}\mathbf{x} + \mathbf{b}^{\mathrm{H}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 + x_2 - 1 \end{bmatrix}$$

Hence, the activation functions are

$$\mathbf{u}^{\mathrm{H}} = g_{\mathrm{act}}(\mathbf{z}^{\mathrm{H}}) = \begin{bmatrix} g_{\mathrm{act}}(x_1) \\ g_{\mathrm{act}}(x_2) \\ g_{\mathrm{act}}(x_1 + x_2 - 1) \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{\{x_1 \geq 0\}} \\ \mathbb{1}_{\{x_2 \geq 0\}} \\ \mathbb{1}_{\{x_1 + x_2 \geq 1\}} \end{bmatrix}.$$

(b) The output z^{O} is

$$\begin{aligned} z^{\mathrm{O}} = W^{\mathrm{O}}\mathbf{u}^{\mathrm{H}} + b^{\mathrm{O}} &= [1, 1, -1] \begin{bmatrix} \mathbb{1}_{\{x_1 \geq 0\}} \\ \mathbb{1}_{\{x_2 \geq 0\}} \\ \mathbb{1}_{\{x_1 + x_2 \geq 1\}} \end{bmatrix} - 1.5 \\ &= \mathbb{1}_{\{x_1 \geq 0\}} + \mathbb{1}_{\{x_2 \geq 0\}} - \mathbb{1}_{\{x_1 + x_2 \geq 1\}} - 1.5. \end{aligned}$$

It is best to draw this. If you do that (I will add the figures later), you will see that in the triangle defined by the lines

$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_1 + x_2 < 1, \tag{12}$$

we have

$$z^o = (1) + (1) - (0) - 1.5 = 0.5.$$

Outside this region, $z^{\mathrm{O}} < 0$. Hence, $z^O \geq 0$ only in the region (12). Therefore,

$$\hat{y} = \mathbb{1}_{\{z^{\mathrm{O}} \geq 0\}} = \begin{cases} 1 & \text{if } x_1 \geq 0, x_2 \geq 0, x_1 + x_2 < 1 \\ 0 & \text{else} \end{cases}$$

4. (a) Assuming we represent the image pixels as a vector, the input size is $N_i = 20 \times 20 = 400$. Since this is a multi-class classification problem with 26 classes, one would generally take an output size of $N_o = 26$ and use a soft-max for the output map $g_{\mathrm{out}}(\cdot)$. The number of hidden units N_h and the activation $g_{\mathrm{act}}(\cdot)$ are free to choose

(b) The input size is $N_i = 120$. Since this is a binary classification problem, we would generally take $N_o = 1$ and use a sigmoid output for $g_{\mathrm{out}}(\cdot)$. The number of hidden units N_h and the activation $g_{\mathrm{act}}(\cdot)$ are free to choose.

(c) We would take $N_i = 5$ for the stock prices for the last five days. Since this is a regression problem predicting a scalar value, we would take $N_o = 1$ and use an identity activation, $g_{\mathrm{out}}(z) = z$ (i.e. there is no activation). Again, the number of hidden units N_h and the activation $g_{\mathrm{act}}(\cdot)$ are free to choose.

5. (a) The operations in the hidden layer in (10) can be implemented as:

```
Zh = X.dot(Wh.T) + bh[None,:]
Uh = 1/(1+np.exp(-Zh))
```

where `Wh` and `bh` are the weight matrix and bias vector. Note that we have used python broadcasting on the bias vector `bh`

(b) For a binary classification problem, $N_o = 1$ and we can represent $\mathbf{Z}^{\mathrm{O}}$ as an N-dimensional vector, `Zo`. Also, we can represent the weight $\mathbf{W}^{\mathrm{H}}$ as an N_h-dimensional vector `Wh` and the bias b^{H} as a scalar `bh`. In this case,

```
Zo = Uh.dot(Wo) + bo
Uo = 1/(1+np.exp(-Zo))
```

(c) For K-class classification, we perform

```
Zo = Uh.dot(Wo.T) + bo[None,:]
expZo = np.exp(Zo)
Uo = expZo / np.sum(expZo,axis=1)[None,:]
```

Note the method for normalizing the outputs in a softmax classifier.

2 Multivariate Gradients and Chain Rule

- The parameter learning for a neural network is based on gradient-descent. Before looking at the learning rules, we need to review some concepts from multivariable calculus.
- *Tensors:* A tensor is a multi-dimensional array,

$$X_{i_1,i_2,\ldots,i_d}, \quad 0 \le i_1 < N_1, \ldots, 0 \le i_d < N_d.$$

  - The number of dimensions d is called the *order*. It is also sometimes called the *rank* (esp. in Tensorflow documentation), but this can be confusing since rank has other meanings in linear algebra.
  - The elements are accessed by a *multi-index*, $(i_1, \ldots, i_d)$.
  - The *shape* or *size* of the tensor is $(N_1, \ldots, N_d)$.
- It will be sometimes more convenient to index the elements of a tensor with brackets: $X[i_1, \ldots, i_d]$ instead of $X_{i_1,\ldots,i_d}$.
- *Examples:*
  - A vector is an order 1 tensor.
  - A matrix is an order 2 tensor.
  - A color image can be represented as an order 3 tensor (height, width, color). For example, a 512×512 with three color channels (RGB) would have shape $(512, 512, 3)$.
  - A batch of color images can be represented as an order 4 tensor (batch size, height, width, color). For example, a mini-batch of 32 color images as above would have shape $(32, 512, 512, 3)$.
- *Gradient tensors:* Suppose that $\mathbf{Y} = f(\mathbf{X})$ where:
  - The input $\mathbf{X}$ is a tensor of size $(N_1, \ldots, N_r)$, and

– The output $\mathbf{Y}$ is a tensor of size $(M_1, \ldots, M_s)$.

The *gradient* $\partial \mathbf{Y}/\partial \mathbf{X}$ is a tensor of shape $(M_1, \ldots, M_s, N_1, \ldots, N_r)$ with components,

$$\left[\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}\right]_{i_1,\ldots,i_s,j_1,\ldots,j_r} = \frac{\partial Y_{i_1,\ldots,i_r}}{\partial X_{j_1,\ldots,j_s}}.$$

- *Jacobian*: Consider the special case that $\mathbf{y} = f(\mathbf{x})$ where $\mathbf{x}$ is a vector of dimension N and $\mathbf{y}$ is a vector of dimension M. The gradient tensor $\partial \mathbf{y}/\partial \mathbf{x}$ is a matrix of shape $M \times N$. This is called the *Jacobian*.

- *Gradients for scalar-valued functions:* Suppose $J = f(\mathbf{X})$, where the output J is a scalar and the input $\mathbf{X}$ has shape $(N_1, \ldots, N_r)$. In this case, the gradient tensor $\partial J/\partial \mathbf{X}$ will have the shape $(1, N_1, \ldots, N_r)$. It is sometimes convenient to regard $\partial J/\partial \mathbf{X}$ has a $(N_1, \ldots, N_r)$ tensor and drop the index over the single output components. That is, $\partial J/\partial \mathbf{X}$ has the same shape as $\mathbf{X}$ with components,

$$\frac{\partial J}{\partial X_{1_1,\ldots,1_r}}.$$

- *Tensor chain rule:* Consider the sequence $\mathbf{X} \mapsto \mathbf{Y} \mapsto \mathbf{Z}$ where

  – The input $\mathbf{X}$ is a tensor of size $(K_1, \ldots, K_t)$;
  – The intermediate variable $\mathbf{Y} = g(\mathbf{X})$ is a tensor of size $(N_1, \ldots, N_r)$;
  – The output $\mathbf{Z} = h(\mathbf{Y})$ is a tensor of size $(M_1, \ldots, M_s)$.

  Then, the components of the gradient tensor $\partial \mathbf{Z}/\partial \mathbf{X}$ are given by:

$$\frac{\partial Z_{i_1,\ldots,i_r}}{\partial X_{k_1,\ldots,k_t}} = \sum_{j_1=0}^{N_1-1} \cdots \sum_{j_r=0}^{N_r-1} \frac{\partial Z_{i_1,\ldots,i_r}}{\partial Y_{j_1,\ldots,j_s}} \frac{\partial Y_{j_1,\ldots,j_s}}{\partial X_{k_1,\ldots,k_t}}.$$

  That is, one sums over all the intermediate variables. We will often use the shorthand,

$$\frac{\partial \mathbf{Z}}{\partial \mathbf{X}} = \frac{\partial \mathbf{Z}}{\partial \mathbf{Y}} \cdot \frac{\partial \mathbf{Y}}{\partial \mathbf{X}},$$

  where $\cdot$ is called the *tensor dot-product*.

- *Gradients of linear functions:* In neural networks, we often have linear layers where

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b},$$

  meaning that $z_i = \sum_j W_{ij} x_j + b_i$. Therefore, the components of the gradients of $\partial \mathbf{z}/\partial \mathbf{W}$ and $\partial \mathbf{z}/\partial \mathbf{b}$ and $\partial \mathbf{z}/\partial \mathbf{x}$ are:

$$\frac{\partial z_i}{\partial W_{ij}} = x_j, \quad \frac{\partial z_i}{\partial b_i} = 1, \quad \frac{\partial z_i}{\partial x_j} = W_{ij}. \tag{13}$$

  This can often be combined with chain rule.

Problems

1. Compute the gradient $\partial f(\mathbf{x})/\partial \mathbf{x}$ of the function,

$$f(\mathbf{x}) = (x_1 + x_2 x_3, x_1^2 + 3x_2).$$

2. Consider the logistic loss function,

$$J = \sum_{i=1}^{N} \left[\ln(1 + e^{z_i}) - y_i z_i \right], \quad z_i = \sum_{j=1}^{p} x_{ij} w_j + b.$$

   (a) What is the gradient $\partial J/\partial \mathbf{z}$? That is, find the components $\partial J/\partial z_i$.

   (b) What are the gradient $\partial \mathbf{z}/\partial \mathbf{w}$ and $\partial \mathbf{z}/\partial b$?

   (c) Use the chain rule to compute the gradients $\partial J/\partial \mathbf{w}$ and $\partial J/\partial b$.

3. Suppose that

$$\mathbf{u} = f(\mathbf{z}) = (f(z_1), \ldots, f(z_N)), \quad f(z_i) = \frac{1}{1 + e^{-z_i}},$$

   so that both $\mathbf{u}$ and $\mathbf{z}$ are N-dimensional vectors.

   (a) What is the gradient $\partial \mathbf{u}/\partial \mathbf{z}$?

   (b) If $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, what are the gradients $\partial \mathbf{u}/\partial \mathbf{W}$ and $\partial \mathbf{u}/\partial \mathbf{b}$?

   (c) In part (b), what is $\partial \mathbf{u}/\partial \mathbf{x}$?

4. Let

$$L = \|\mathbf{y} - \mathbf{z}\|^2, \quad \mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b},$$

   (a) What is the gradient $\partial L/\partial \mathbf{z}$?

   (b) What are the gradients $\partial L/\partial \mathbf{W}$ and $\partial L/\partial \mathbf{b}$?

   (c) What is the gradient $\partial L/\partial \mathbf{x}$?

Solutions

1. This function has two outputs and three inputs. Therefore, the gradient can be represented as 2×3 matrix with the partial derivatives:

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} 1 & x_3 & x_2 \\ 2x_1 & 3 & 0 \end{bmatrix}$$

2. (a) The components of the gradient $dJ/\partial \mathbf{z}$ are given by the partial derivatives,

$$\frac{\partial J}{\partial z_i} = -y_i + \frac{e^{z_i}}{1 + e^{z_i}}.$$

   (b) The components of the gradient $\partial \mathbf{z}/\partial \mathbf{w}$ and $\partial \mathbf{z}/\partial b$ are

$$\frac{\partial z_i}{\partial w_j} = x_{ij}, \quad \frac{\partial z_i}{\partial b} = 1.$$

(c) Using chain rule,

$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial w_j} = \frac{\partial J}{\partial z_i}x_{ij},$$
$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial b} = \frac{\partial J}{\partial z_i},$$

where $\partial J/\partial z_i$ is given in part (a).

3. (a) Since u_i only depends on z_i, we have that

$$\frac{\partial u_i}{\partial z_j} = 0 \text{ for } i \neq j.$$

For $i = j$,

$$\frac{\partial u_i}{\partial z_i} = f'(z_i) = \frac{e^{-z_i}}{(1+e^{-z_i})^2}.$$

Therefore, the gradient is the diagonal matrix,

$$\frac{\partial \mathbf{u}}{\partial \mathbf{z}} = \begin{bmatrix} f'(z_1) & 0 & \cdots & 0 \\ 0 & f'(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'(z_N) \end{bmatrix} = \text{diag}(f'(z_1), f'(z_2), \cdots, f'(z_N)).$$

(b) Using chain rule with (13),

$$\frac{\partial u_i}{\partial W_{ij}} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = f'(z_i)x_j,$$

and

$$\frac{\partial u_i}{\partial b_i} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} = f'(z_i).$$

(c) We again use chain rule with (13),

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial x_j} = f'(z_i)W_{ij}.$$

4. (a) We have

$$L = \|\mathbf{y} - \mathbf{z}\|^2 = \sum_{j=1}^{N}(y_i - z_i)^2.$$

Hence, the components of the gradient $\partial J/\partial \mathbf{z}$ are,

$$\frac{\partial L}{\partial z_j} = 2(z_i - y_i).$$

(b) Using chain rule with (13),

$$\frac{\partial u_i}{\partial W_{ij}} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial W_{ij}} = 2(z_i - y_i)x_j,$$

and

$$\frac{\partial u_i}{\partial b_i} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} = 2(z_i - y_i).$$

(c) We again use chain rule with (13),

$$\frac{\partial u_i}{\partial x_j} = \frac{\partial u_i}{\partial z_i}\frac{\partial z_i}{\partial x_j} = 2(z_i - y_i)W_{ij}.$$

3 Gradient Descent Training of Neural Nets

- Problem: Given data $(\mathbf{x}_i, y_i)$, we wish to learn the parameters

$$\theta = (\mathbf{W}^{\mathrm{H}}, \mathbf{b}^{\mathrm{H}}, \mathbf{W}^{\mathrm{O}}, \mathbf{b}^{\mathrm{O}}). \tag{14}$$

- We will learn the parameters through the minimization

$$\hat{\theta} = \arg\min_{\theta} L(\theta), \tag{15}$$

where $L(\theta)$ is a *loss* function:

$$L(\theta) := \sum_{i=1}^{N} g_{\mathrm{loss}}(\mathbf{z}_i^{\mathrm{O}}, y_i), \tag{16}$$

where $\mathbf{z}_i^{\mathrm{O}}$ is the output of the neural network (1) with the input $\mathbf{x} = \mathbf{x}_i$ and parameters θ.

- The loss function factor $g_{\mathrm{loss}}(z_i^{\mathrm{O}}, y_i)$ compares each neural network output z_i^{O} with the observed y_i. Its specific form depends on the nature of the response variable y_i to be predicted by the network:

  – Binary classification: In this case $y_i = \{0, 1\}$, and z_i^{O} is the logit and we use the binary cross entropy,

$$g_{\mathrm{loss}}(z_i^{\mathrm{O}}, y_i) = \ln\left[1 + e^{z_i^{\mathrm{O}}}\right] - y_i z_i. \tag{17}$$

  – Multi-class classification: In this case, $y_i \in \{1, \ldots, K\}$ and $\mathbf{z}_i^{\mathrm{out}} \in \mathbb{R}^K$ are a set of K logits– one logit for each class. For the loss function, we use the categorial cross-entropy,

$$g_{\mathrm{loss}}(\mathbf{z}_i^{\mathrm{O}}, y_i) := \ln\left[\sum_{\ell=1}^{K} e^{z_{i\ell}^{\mathrm{O}}}\right] - \sum_{k=1}^{K} r_{ik} z_{ik}^{\mathrm{O}}, \tag{18}$$

where r_{ik} is the one-hot coded version of y_i:

$$r_{ik} = \begin{cases} 1 & \text{if } y_i = k, \\ 0 & \text{if } y_i \neq k. \end{cases} \tag{19}$$

– Regression: If $\mathbf{y}_i \in \mathbb{R}^d$, then we typically take the squared loss function,

$$g_{\text{loss}}(\mathbf{z}_i^{\text{O}}, \mathbf{y}_i) = \|\mathbf{z}_i^{\text{O}} - \mathbf{y}_i\|^2 = \sum_{k=1}^{d} (z_{ik}^{\text{O}} - y_{ik})^2. \tag{20}$$

- The loss function is typically minimized by some variant of stochastic gradient descent (SGD). In SGD, we compute a sequence of estimates of the minimizer (15) by

$$\theta^{t+1} = \theta^t - \alpha \partial L(\theta^t)/\partial \theta^t, \tag{21}$$

where t is the training iteration index, $\alpha > 0$ is the step size (called the *learning rate*) and the gradient $\partial L(\theta)/\partial\theta$ is evaluated on some mini-batch. The gradient update is to be performed for each component of θ in (14),

$$\begin{aligned}
(\mathbf{W}^{\text{H}})^{t+1} &= (\mathbf{W}^{\text{H}})^t - \alpha \partial L(\theta^t)/\partial \mathbf{W}^{\text{H}}, \\
(\mathbf{W}^{\text{O}})^{t+1} &= (\mathbf{W}^{\text{O}})^t - \alpha \partial L(\theta^t)/\partial \mathbf{W}^{\text{O}}, \\
(\mathbf{b}^{\text{H}})^{t+1} &= (\mathbf{b}^{\text{H}})^t - \alpha \partial L(\theta^t)/\partial \mathbf{b}^{\text{H}}, \\
(\mathbf{b}^{\text{O}})^{t+1} &= (\mathbf{b}^{\text{O}})^t - \alpha \partial L(\theta^t)/\partial \mathbf{b}^{\text{O}}.
\end{aligned}$$

The only challenging part is computing the gradients in an efficient way. We discuss this now.

- *Computation graph:* We can write the computation of the loss function as a sequence of operations represented schematically in Fig. 1. This representation is called the *computation graph*. In Fig. 1, the data $\mathbf{x}_i$ and y_i are represented as light green circles and the parameters to be estimated are represented in light blue circles. The other circles are computational nodes that use the outputs from previous nodes to compute outputs for subsequent nodes. Each computational node thus represents one computation. For the neural network, the equations for the computational nodes are:

$$\begin{aligned}
\mathbf{z}_i^{\text{H}} &= \mathbf{W}^{\text{H}} \mathbf{x}_i + \mathbf{b}^{\text{H}}, \quad \mathbf{u}_i^{\text{H}} = g_{\text{act}}(\mathbf{z}_i^{\text{H}}), \\
\mathbf{z}_i^{\text{O}} &= \mathbf{W}^{\text{O}} \mathbf{u}_i^{\text{H}} + \mathbf{b}^{\text{O}}, \quad \hat{y}_i = g_{\text{out}}(\mathbf{z}_i^{\text{O}}). \\
L &= \sum_{i=1}^{N} g_{\text{loss}}(\mathbf{z}_i^{\text{O}}, y_i).
\end{aligned}$$

Here, the summation is over all N samples in some mini-batch.

- *Back propagation:* Once data is represented in a computation graph, we can compute gradients of any of the output variables from the parameters by a procedure called *back propagation*. Since we want to compute the gradient of the loss L, we start at that node. Then, we find the path going backwards in the computation graph towards all the parameters to compute the gradients. Note that the data $(\mathbf{x}_i, y_i)$ is fixed, so we don't take the gradients with respect to $\mathbf{x}_i$ or $\mathbf{y}_i$. As we move backwards in the path, we compute the gradients via chain rule.

  For the computation graph in Fig. 1, we could follow the following steps:

  – $L \to \mathbf{z}^{\text{O}}$: Compute the gradient $\partial L/\partial \mathbf{z}^{\text{O}}$.

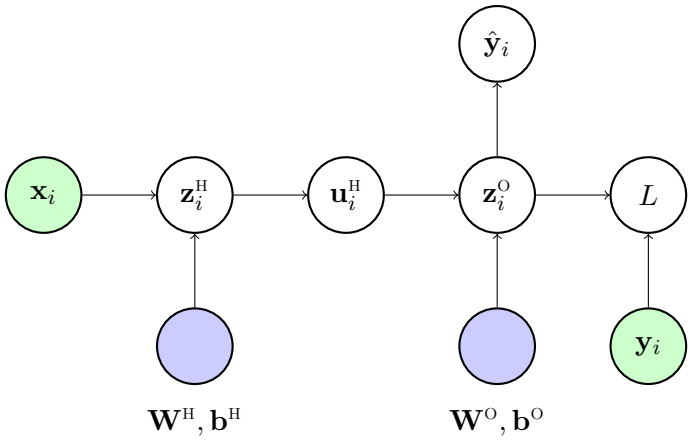

Figure 1: Computation graph for the neural network mapping the the training data $(\mathbf{x}_i, y_i)$ and parameters to the loss function L. Parameters are shown in light blue and data in light green.

– $\mathbf{z}^{\mathrm{O}} \to \mathbf{W}^{\mathrm{O}}, \mathbf{b}^{\mathrm{O}}, \mathbf{u}^{\mathrm{H}}$: Compute the gradients:

$$\frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{W}^{\mathrm{O}}}, \quad \frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{b}^{\mathrm{O}}}, \quad \frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{u}^{\mathrm{H}}}.$$

Then apply chain rule to compute the gradient of L,

$$\frac{\partial L}{\partial \mathbf{W}^{\mathrm{O}}} = \frac{\partial L}{\partial \mathbf{z}^{\mathrm{O}}} \cdot \frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{W}^{\mathrm{O}}}, \quad \frac{\partial L}{\partial \mathbf{b}^{\mathrm{O}}} = \frac{\partial L}{\partial \mathbf{z}^{\mathrm{O}}} \cdot \frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{b}^{\mathrm{O}}}, \quad \frac{\partial L}{\partial \mathbf{u}^{\mathrm{H}}} = \frac{\partial L}{\partial \mathbf{z}^{\mathrm{O}}} \cdot \frac{\partial \mathbf{z}^{\mathrm{O}}}{\partial \mathbf{u}^{\mathrm{H}}},$$

where the multiplication $\cdot$ represents a tensor dot-product.

– $\mathbf{u}^{\mathrm{H}} \to \mathbf{z}^{\mathrm{H}}$: Compute $\partial \mathbf{u}^{\mathrm{H}} / \partial \mathbf{z}^{\mathrm{H}}$. Then, we apply chain rule,

$$\frac{\partial L}{\partial \mathbf{z}^{\mathrm{H}}} = \frac{\partial L}{\partial \mathbf{u}^{\mathrm{H}}} \cdot \frac{\partial \mathbf{u}^{\mathrm{H}}}{\partial \mathbf{z}^{\mathrm{H}}}.$$

– $\mathbf{z}^{\mathrm{H}} \to \mathbf{W}^{\mathrm{H}}, \mathbf{b}^{\mathrm{H}}$: We have $\mathbf{z}_i^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}} \mathbf{x}_i + \mathbf{b}^{\mathrm{H}}$. Compute the gradients:

$$\frac{\partial \mathbf{z}^{\mathrm{H}}}{\partial \mathbf{W}^{\mathrm{H}}}, \quad \frac{\partial \mathbf{z}^{\mathrm{H}}}{\partial \mathbf{b}^{\mathrm{H}}}.$$

Then apply chain rule to compute the gradient of L,

$$\frac{\partial L}{\partial \mathbf{W}^{\mathrm{H}}} = \frac{\partial L}{\partial \mathbf{z}^{\mathrm{H}}} \cdot \frac{\partial \mathbf{z}^{\mathrm{H}}}{\partial \mathbf{W}^{\mathrm{H}}}, \quad \frac{\partial L}{\partial \mathbf{b}^{\mathrm{H}}} = \frac{\partial L}{\partial \mathbf{z}^{\mathrm{H}}} \cdot \frac{\partial \mathbf{z}^{\mathrm{H}}}{\partial \mathbf{b}^{\mathrm{H}}}.$$

We illustrate the details in the problems below.

Problems

1. Consider a neural network (1) acting on a mini-batch of training samples $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$. Suppose that $y_i \in \{1, \ldots, K\}$ so that the problem is K-class classification, we use the categorial cross-entropy loss (18) and a ReLU activation. Derive the back-propagation equations for all steps.

2. Suppose that we are given a mini-batch of training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$, and we try to fit a model to the data of the form,

$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}, \quad \hat{y}_i = \sum_{j=1}^{M} \alpha_j \exp(z_{ij}),$$

for unknown parameters $\theta = (\mathbf{W}, \mathbf{b}, \alpha)$. Here M is the dimension of the hidden variables $\mathbf{z}_i$. We use the squared error loss function,

$$L = \sum_{i=1}^{N} L_i, \quad L_i = (y_i - \hat{y}_i)^2.$$

(a) Add an intermediate variable $\mathbf{u}_i = \exp(\mathbf{z}_i)$, by which we mean $u_{ij} = \exp(z_{ij})$. Draw the computation graph showing the mapping from the inputs $\mathbf{x}_i$ and the parameters θ to the loss function, L. Also, write the equations for each step in the computation graph.

(b) Write the back-propagation equations to obtain the gradients with respect to the parameters θ.

Solutions

1. We work our way backward starting at L along paths to get to the parameters:

   - $L \to \mathbf{z}^{\mathrm{O}}$: The loss function is the categorial cross-entropy (18). Therefore, the gradient components are

   $$\frac{\partial L}{\partial z_{ij}^{\mathrm{O}}} = \frac{\partial g_{\mathrm{loss}}(\mathbf{z}_i^{\mathrm{O}}, y_i)}{\partial z_{ij}^{\mathrm{O}}} = \frac{e^{z_{ij}^{\mathrm{O}}}}{\sum_{\ell=1}^{K} e^{z_{i\ell}^{\mathrm{O}}}} - r_{ij}.$$

   - $\mathbf{z}^{\mathrm{O}} \to \mathbf{W}^{\mathrm{O}}, \mathbf{b}^{\mathrm{O}}, \mathbf{u}^{\mathrm{H}}$: We have $\mathbf{z}_i^{\mathrm{O}} = \mathbf{W}^{\mathrm{O}}\mathbf{u}_i^{\mathrm{H}} + \mathbf{b}^{\mathrm{O}}$. Therefore,

   $$z_{ij}^{\mathrm{O}} = \sum_k W_{jk}^{\mathrm{O}} u_{ik}^{\mathrm{H}} + b_j^{\mathrm{O}}.$$

   Hence, we have the derivatives,

   $$\frac{\partial z_{ij}^{\mathrm{O}}}{\partial W_{jk}^{\mathrm{O}}} = u_{ik}^{\mathrm{H}}, \quad \frac{\partial z_{ij}^{\mathrm{O}}}{\partial b_j^{\mathrm{O}}} = 1, \quad \frac{\partial z_{ij}^{\mathrm{O}}}{\partial u_{ik}^{\mathrm{H}}} = W_{jk}.$$

   Using chain rule,

   $$\begin{aligned}
   \frac{\partial L}{\partial W_{jk}^{\mathrm{O}}} &= \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{O}}} \frac{\partial z_{ij}^{\mathrm{O}}}{\partial W_{jk}^{\mathrm{O}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{O}}} u_{ik}^{\mathrm{H}}, \\
   \frac{\partial L}{\partial b_j^{\mathrm{O}}} &= \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{O}}} \frac{\partial z_{ij}^{\mathrm{O}}}{\partial b_j^{\mathrm{O}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{O}}}, \\
   \frac{\partial L}{\partial u_{ik}^{\mathrm{H}}} &= \sum_{j=1}^{N_o} \frac{\partial L}{\partial z_{ij}^{\mathrm{O}}} \frac{\partial z_{ij}^{\mathrm{O}}}{\partial u_{ik}^{\mathrm{H}}} \sum_{j=1}^{N_o} \frac{\partial L}{\partial z_{ij}^{\mathrm{O}}} W_{jk}.
   \end{aligned}$$

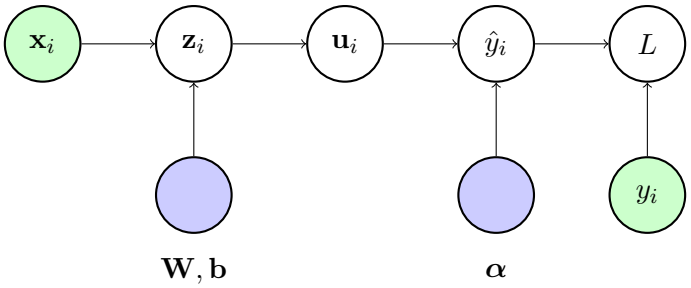

Figure 2: Computation graph for Problem 2 mapping the the training data $(\mathbf{x}_i, y_i)$ and parameters to the loss function L. Parameters are shown in light blue and data in light green.

- $\mathbf{u}^{\mathrm{H}} \to \mathbf{z}^{\mathrm{H}}$: We have
$$u_{ij}^{\mathrm{H}} = g_{\mathrm{act}}(z_{ij}^{\mathrm{H}}) = \max(0, z_{ij}^{\mathrm{H}}).$$
Hence,
$$\frac{\partial u_{ij}^{\mathrm{H}}}{\partial z_{ij}^{\mathrm{H}}} = \begin{cases} 1, & \text{if } z_{ij}^{\mathrm{H}} > 0 \\ 0, & \text{if } z_{ij}^{\mathrm{H}} < 0. \end{cases}.$$
Then, we apply chain rule,
$$\frac{\partial L}{\partial z_{ij}^{\mathrm{H}}} = \frac{\partial L}{\partial u_{ij}^{\mathrm{H}}} \frac{\partial u_{ij}^{\mathrm{H}}}{\partial z_{ij}^{\mathrm{H}}}.$$
- $\mathbf{z}^{\mathrm{H}} \to \mathbf{W}^{\mathrm{H}}, \mathbf{b}^{\mathrm{H}}$: We have $\mathbf{z}_i^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}}\mathbf{x}_i + \mathbf{b}^{\mathrm{H}}$. Therefore,
$$z_{ij}^{\mathrm{H}} = \sum_k W_{jk}^{\mathrm{H}} x_{ik} + b_j^{\mathrm{H}}.$$
Hence, we have the derivatives,
$$\frac{\partial z_{ij}^{\mathrm{H}}}{\partial W_{jk}^{\mathrm{H}}} = x_{ik}, \quad \frac{\partial z_{ij}^{\mathrm{H}}}{\partial b_j^{\mathrm{H}}} = 1.$$
Using chain rule,
$$\frac{\partial L}{\partial W_{jk}^{\mathrm{H}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} \frac{\partial z_{ij}^{\mathrm{H}}}{\partial W_{jk}^{\mathrm{H}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} x_{ik},$$
$$\frac{\partial L}{\partial b_{jk}^{\mathrm{H}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}} \frac{\partial z_{ij}^{\mathrm{H}}}{\partial b_j^{\mathrm{H}}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ik}^{\mathrm{H}}}.$$

2. (a) If we substitute $\mathbf{u}_i = \exp(\mathbf{z}_i)$ we can write the mapping from the set of $\mathbf{x}_i$'s to L as
$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}$$
$$\mathbf{u}_i = \exp(\mathbf{z}_i), \quad \hat{y}_i = \boldsymbol{\alpha}^{\mathsf{T}}\mathbf{u}_i$$
$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$
The computation graph is shown in Fig. 2.

(b) We perform back-propagation in the following steps:

- $L \to \hat{\mathbf{y}}$: The components of the gradient are:

$$\frac{\partial L}{\partial \hat{y}_i} = 2(\hat{y}_i - y_i).$$

- $\hat{\mathbf{y}} \to \boldsymbol{\alpha}, \mathbf{u}$: We have

$$\hat{y}_i = \boldsymbol{\alpha}^\mathsf{T} \mathbf{u}_i = \sum_{j=1}^{M} \alpha_j u_{ij}.$$

Therefore,

$$\frac{\partial \hat{y}_i}{\partial u_{ij}} = \alpha_j, \quad \frac{\partial \hat{y}_i}{\partial \alpha_j} = u_{ij}.$$

Applying chain rule,

$$\frac{\partial L}{\partial \alpha_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial \alpha_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial \hat{y}_i} u_{ij}$$
$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial u_{ij}} = \frac{\partial L}{\partial \hat{y}_i} \alpha_j.$$

- $\mathbf{u} \to \mathbf{z}$: Since $u_{ij} = \exp(z_{ij})$, we have the derivative,

$$\frac{\partial u_{ij}}{\partial z_{ij}} = \exp(z_{ij}).$$

Using chain rule:

$$\frac{\partial L}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}} \frac{\partial u_{ij}}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}} \exp(z_{ij}).$$

- $\mathbf{z} \to \mathbf{W}, \mathbf{b}$: Since $\mathbf{z}_i = \mathbf{W}\mathbf{x}_i + \mathbf{b}$, Therefore,

$$z_{ij} = \sum_k W_{jk} x_{ik} + b_j.$$

Hence, we have the derivatives,

$$\frac{\partial z_{ij}}{\partial W_{jk}} = u_{ik}, \quad \frac{\partial z_{ij}}{\partial b_j} = 1.$$

Applying chain rule,

$$\frac{\partial L}{\partial W_{jk}} = \sum_i \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial W_{jk}} = \sum_i \frac{\partial L}{\partial z_{ij}} u_{ik},$$
$$\frac{\partial L}{\partial b_j} = \sum_i \frac{\partial L}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial b_j} = \sum_i \frac{\partial L}{\partial z_{ij}}.$$