Introduction to Machine Learning
Neural Networks Problem Solutions

Prof. Sundeep Rangan and Yao Wang

1. Consider a neural network on a 3-dimensional input $\mathbf{x} = (x_1, x_2, x_3)$ with weights and biases:

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

Assume the network uses the threshold activation function (1)

$$g_{\rm act}(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0. \end{cases} \tag{1}$$

and the threshold output function (2):

$$\hat{y} = \begin{cases} 1 & z^{\rm O} \geq 0 \\ 0 & z^{\rm O} < 0. \end{cases} \tag{2}$$

(a) Write the components of $\mathbf{z}^{\rm H}$ and $\mathbf{u}^{\rm H}$ as a function of (x_1, x_2, x_3). For each component j, indicate where in the (x_1, x_2, x_3) hyperplane $u_j^{\rm H} = 1$.

(b) Write $z^{\rm O}$ as a function of (x_1, x_2, x_3). In what region is $\hat{y} = 1$? (You can describe in mathematical formulae).

Solution:

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

$$\mathbf{z}^{\rm H} = \mathbf{W}^{\rm H}\mathbf{x} + \mathbf{b}^{\rm H} = \begin{bmatrix} x_1 + x_3 \\ x_2 + x_3 \\ x_1 + x_2 - 1 \\ x_1 + x_2 + x_3 + 1 \end{bmatrix}.$$

Each component $u_j^{\rm H} = 1$ when $z_j^{\rm H} > 0$ so

$$\mathbf{u}^{\rm H} = \begin{bmatrix} \mathbb{1}_{\{x_1+x_3 \geq 0\}} \\ \mathbb{1}_{\{x_2+x_3 \geq 0\}} \\ \mathbb{1}_{\{x_1+x_2-1 \geq 0\}} \\ \mathbb{1}_{\{x_1+x_2+x_3+1 \geq 0\}} \end{bmatrix}.$$

(b) The output is,

$$\begin{aligned} z^{\mathrm{O}} &= W^{\mathrm{O}}\mathbf{u}^{\mathrm{H}} + b^{\mathrm{O}} \\ &= \mathbb{1}_{\{x_1+x_3\geq 0\}} + \mathbb{1}_{\{x_2+x_3\geq 0\}} - \mathbb{1}_{\{x_1+x_2-1\geq 0\}} - \mathbb{1}_{\{x_1+x_2+x_3+1\geq 0\}} - 1.5. \end{aligned}$$

Now, we have that $\hat{y} = 1$ when $z^{\mathrm{O}} > 0$. Since the indicator functions have values that are 0 or 1, the only way we can get $z^{\mathrm{O}} > 0$ is if the first two indicator functions are 1 and the second two are 0. This occurs when

$$x_1 + x_3 \geq 0, \quad x_2 + x_3 \geq 0 \qquad x_1 + x_2 - 1 < 0, \quad x_1 + x_2 + x_3 + 1 < 0.$$

You do not need to simplify this beyond this.

2. Consider a neural network used for regression with a scalar input x and scalar target y,

$$\begin{aligned} z_j^{\mathrm{H}} &= W_j^{\mathrm{H}} x + b_j^{\mathrm{H}}, \quad u_j^{\mathrm{H}} = \max\{0, z_j^{\mathrm{H}}\}, \quad j = 1, \ldots, N_h \\ z^{\mathrm{O}} &= \sum_{k=1}^{N_h} W_k^{\mathrm{O}} u_k^{\mathrm{H}} + b^{\mathrm{O}}, \quad \hat{y} = g_{\mathrm{out}}(z^{\mathrm{O}}). \end{aligned}$$

The hidden weights and biases are:

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

(a) What is the number N_h of hidden units? For each $j = 1, \ldots, N_h$, draw u_j^{H} as a function of x over some suitable range of values x. You may draw them on one plot, or on multiple plots.

(b) Since the network is for regression, you may choose the activation function $g_{\mathrm{out}}(z^{\mathrm{O}})$ to be linear. Given training data (x_i, y_i), $i = 1, \ldots, N$, formally define the loss function you would use to train the network?

(c) Using the output activation and loss function selected in part (b), set up the formulation to determine output weights and bias, $\mathbf{W}^{\mathrm{O}}$, b^{O}, for the training data below. You should be able to find a closed form solution. Write a few line of python code to solve the problem.

x_i	-2	-1	0	3	3.5
y_i	0	0	1	3	3

(d) Based on your solution for the output weights and bias, draw $\hat{y}$ vs. x over some suitable range of values x.

(e) Write a function `predict` to output $\hat{y}$ given a vector of inputs `x`. Assume `x` is a vector representing a batch of samples and `yhat` is a vector with the corresponding outputs. Use the activation function you selected in part (b), but your function should take the weights and biases for both layers as inputs. Clearly state any assumptions on the formats for the weights and biases. Also, to receive full credit, you must not use any for loops.

Solution:

(a) There are $N_h = 3$ hidden units. The linear outputs are:

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

After activations, you get

$$\mathbf{u}^{\mathrm{H}} = \max\{0, \mathbf{z}^{\mathrm{H}}\} = \begin{bmatrix} \max\{0, -x - 1\} \\ \max\{0, x + 1\} \\ \max\{0, x - 2\} \end{bmatrix}.$$

(b) Use MSE loss,

$$J = \sum_{i=1}^{n} (y_i - \widehat{y}_i)^2.$$

(c) Table 1 shows the output of the hidden layers u_j^{H} on the training data. If you take

$$W^{\mathrm{O}} = [0, 1, -1], b^{\mathrm{O}} = 0,$$

you get

$$\widehat{y}_i = u_{i2}^{\mathrm{H}} - u_{i3}^{\mathrm{H}}.$$

You can verify that with this setting, $\widehat{y}_i = y_i$ for all the training samples i.

(d) With the output weights and biases from the previous part we have that $\widehat{y}$ will be,

$$\begin{aligned} \widehat{y} = u_2^{\mathrm{H}} - u_3^{\mathrm{H}} &= \max\{0, x + 1\} - \max\{0, x - 2\} \\ &= \begin{cases} 0 & \text{if } x \leq -1 \\ x + 1 & \text{if } x \in [-1, 2] \\ 3 & \text{if } x \geq 2 \end{cases}. \end{aligned}$$

(e) One possible solution is as follows.

```
def predict(x,Wh,bh,Wo,bo):
    # Hidden layer
    zh = x[:,None]*Wh[None,:] + bh[None,:]
    uh = np.maximum(0, zh)

    # Output layer
    yhat = uh.dot(Wo) + bo
    return yhat
```

x_i	-2	-1	0	3	3.5
y_i	0	0	1	3	3
u_{i1}^{H}	-1	0	0	0	0
u_{i2}^{H}	0	0	1	4	4.5
u_{i3}^{H}	0	0	0	1	1.5

Table 1: Problem 2. Hidden layer outputs on the training data.

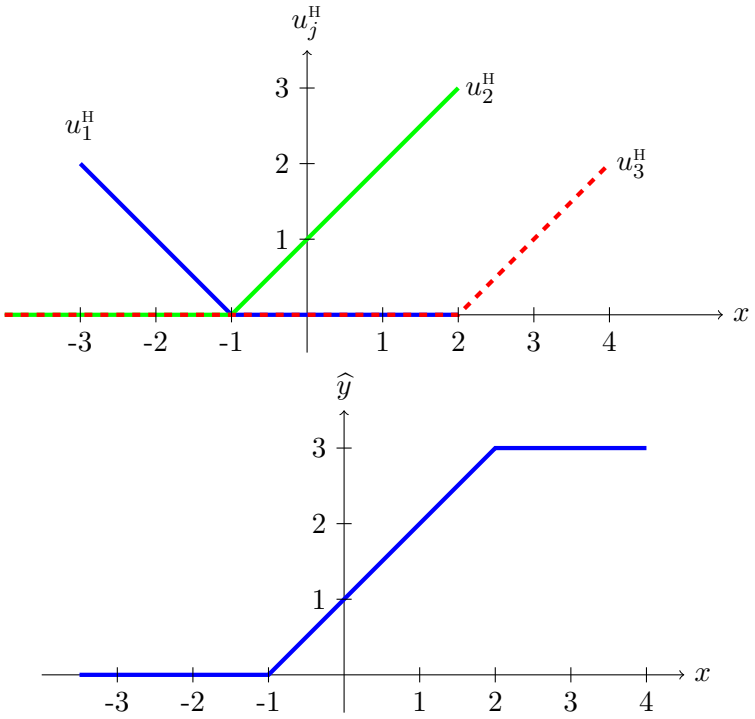

Figure 1: Problem 2. Top: Hidden layer activations, u_j^{H} vs. x for $j = 1, 2, 3$. Bottom: Output $\widehat{y}$ vs. x.

3. Consider a neural network that takes each input $\mathbf{x}$ and produces a prediction $\hat{y}$ given

$$\begin{aligned} z_j &= \sum_{k=1}^{N_i} W_{jk} x_k + b_j, \quad u_j = 1/(1+\exp(-z_j)), \quad j = 1, \ldots, M, \\ \hat{y} &= \frac{\sum_{j=1}^{M} a_j u_j}{\sum_{j=1}^{M} u_j}, \end{aligned} \tag{3}$$

where M is the number of hidden units and is fixed (i.e. not trainable). To train the model, we get training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$.

(a) Rewrite the equations (4) for the batch of inputs $\mathbf{x}_i$ from the training data. That is, correctly add the indexing i, to the equations.

(b) If we use a loss function,

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

draw the computation graph describing the mapping from $\mathbf{x}_i$ and parameters to L. Indicate which nodes are trainable parameters.

(c) Compute the gradient $\partial L/\partial \hat{y}_i$ for all i.

(d) Suppose that, in backpropagation, we have computed $\partial L/\partial \hat{y}_i$ for all i, represented as $\partial L/\partial \hat{\mathbf{y}}$. Describe how to compute the components of the gradient $\partial L/\partial \mathbf{u}$.

(e) Suppose that we have computed the gradient $\partial L/\partial \mathbf{u}$, describe how would you compute the gradient $\partial L/\partial \mathbf{z}$.

(f) Suppose that we have computed the gradient $\partial L/\partial \mathbf{z}$, describe how would you compute the gradient $\partial L/\partial W_{jk}$ and $\partial L/\partial b_j$.

(g) Put all above together, describe how would you compute the gradient $\partial L/\partial W_{jk}$ and $\partial L/\partial b_j$.

(h) Write a few lines of python code to implement the gradients $\partial L/\partial \mathbf{u}$ (as in part (d)), given the gradient $\partial L/\partial \hat{\mathbf{y}}$. Indicate how you represent the gradients. Full credit requires that you avoid for-loops.

Solution:

(a) To add the index i for the training samples we get,

$$\begin{aligned} z_{ij} &= \sum_{k=1}^{N_i} W_{jk} x_{ik} + b_j, \quad u_{ij} = 1/(1+\exp(-z_{ij})), \quad j = 1, \ldots, M, \\ \widehat{y}_i &= \frac{\sum_{j=1}^{M} a_j u_{ij}}{\sum_{j=1}^{M} u_{ij}}, \end{aligned} \tag{4}$$

(b) The computation graph is shown in Fig. 2.

(c) The gradient is,

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

(d) We first compute,

$$\frac{\partial \widehat{y}_i}{\partial u_{ij}} = \frac{1}{(\sum_{k=1}^{M} u_{ik})^2}\left[a_j \sum_{k=1}^{M} u_{ik} - \sum_{k=1}^{M} a_k u_{ik}\right]$$
$$= \frac{1}{(\sum_{k=1}^{M} u_{ik})^2}\left[\sum_{k=1}^{M} (a_j - a_k) u_{ik}\right].$$

Note that, to compute the derivative with respect to u_{ij}, we changed the summation index to k. Next, we apply multi-variable chain rule,

$$\frac{\partial L}{\partial u_{ij}} = \frac{\partial L}{\partial \widehat{y}_i}\frac{\partial \widehat{y}_i}{\partial u_{ij}}.$$

You do not need to simplify further.

(e) We have

$$\frac{\partial u_{ik}}{\partial z_{ij}} = \frac{\exp(-z_{ij})}{(1+\exp(-z_{ij}))^2} = u_{ij}(1-u_{ij}).$$

Applying multi-variable chain rule,

$$\frac{\partial L}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}}\frac{\partial u_{ik}}{\partial z_{ij}} = \frac{\partial L}{\partial u_{ij}} u_{ij}(1-u_{ij}).$$

(f) Similarly,

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

So,

$$\frac{\partial L}{\partial W_{jk}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ij}}\frac{\partial z_{ik}}{\partial W_{jk}} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ij}} x_{ij}$$
$$\frac{\partial L}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ij}}\frac{\partial z_{ik}}{\partial b_j} = \sum_{i=1}^{N} \frac{\partial L}{\partial z_{ij}}.$$

(g) Compute the forward pass in part (a) and then perform back-propagation steps in (c) to (f).

(h) To compute the gradient $\partial L/\partial \hat{\mathbf{u}}$ from $\partial L/\partial \mathbf{y}$ you can use the following python code:

```
usum = np.sum(u,axis=1)
dyhat_du = a[None,:]/usum[:,None] - u*a[None,:]/(usum[:,None]**2)
grad_u = grad_yhat[:,None]*dyhat_du
```

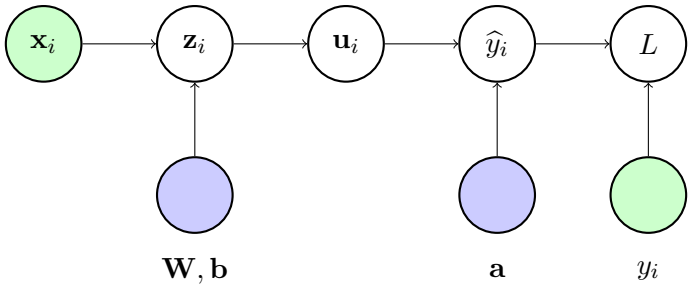

Figure 2: Computation graph for the neural network mapping the training data $(\mathbf{x}_i, y_i)$ and parameters to the loss function L. Parameters are shown in light blue and data in light green.

The problem does not ask for it, but the full back-propagation (except for the gradient with respect to $\mathbf{a}$) is:

```
def loss_eval(X,y,W,b,a):

    # Forward pass
    z = X.dot(W.T) + b[None,:]
    u = 1/(1 + np.exp(-z))
    usum = np.sum(u,axis=1)
    yhat = u.dot(a) / usum

    # Loss
    loss = np.sum((y-yhat)**2)

    # Gradient wrt yhat
    grad_yhat = 2*(yhat-y)

    # Gradient wrt to u
    dyhat_du = a[None,:]/usum[:,None] - u*a[None,:]/(usum[:,None]**2)
    grad_u = grad_yhat[:,None]*dyhat_du

    # Gradient wrt to z
    grad_z = grad_u*u*(1-u)

    # Gradient wrt to W
    grad_W = np.sum( grad_z[:,:,None]*X[:,None,:], axis=0)
    grad_b = np.sum( grad_z[:,:,None]*X[:,None,:], axis=0)

    return loss, grad_W, grad_b
```