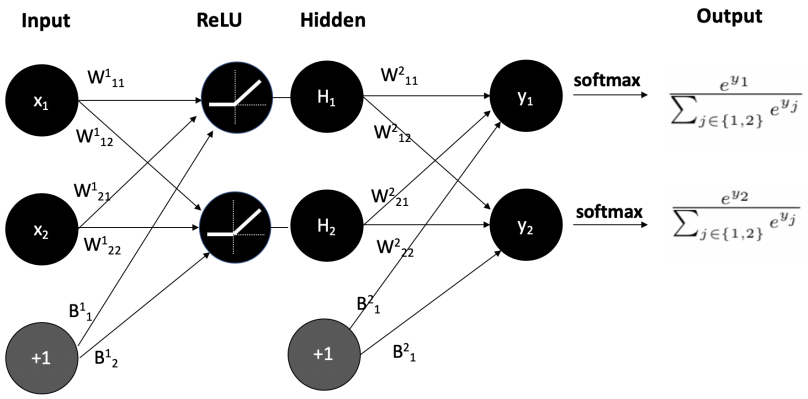


Figure 1: Fully-connected feed-forward neural net with one hidden layer. Also known as two-layer multilayer perceptron (MLP). Includes bias weights for each unit.

1 Example weights and input.

Above we've drawn a fully-connected feed-forward neural net with one hidden layer. We're using ReLU as the activation function for the first layer, and softmax to give the output probability of each class. If you had more hidden layers, you would also use something like ReLU for each hidden layer. Only the output layer uses softmax since this is where you have to compute the probability of the output classes. (Here, we're using y_j to denote the linear scores of the last layer before you apply the softmax. Just wanted to point out this notation since in the past sometimes y_j has been the actual label.)

Note, we're using only two classes for simplicity, but the formula used for softmax is the multi-class logistic regression formula which generalizes to any number of output classes. In the case of your homework data, you would have 10 outputs, one probability for each digit class. Similarly, for simplicity, we've let x be two dimensional. Your homework will have a 784 dimensional x.

Now, to compute the numerical result of the neural net with specific weights (suppose you got these weights after training with stochastic gradient) on a specific input x, we need to specify the actual numerical values. Below we've set values for all the parameters in the neural net, along with specifying the

value of some input example, x.

$$
\begin{aligned}
x &= (x_1, x_2)^\top = (-1, 1)^\top \\
W^1 &= \begin{bmatrix} W^1_{11} & W^1_{12} \\ W^1_{21} & W^1_{22} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.8 \\ 0.6 & 0.1 \end{bmatrix} \\
W^2 &= \begin{bmatrix} W^2_{11} & W^2_{12} \\ W^2_{21} & W^2_{22} \end{bmatrix} = \begin{bmatrix} 0.3 & 0.2 \\ 0.7 & 0.4 \end{bmatrix} \\
B^1 &= \begin{bmatrix} B^1_1 \\ B^1_2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix} \\
B^2 &= \begin{bmatrix} B^2_1 \\ B^2_2 \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}
\end{aligned}
$$

2 Forward computation

Finally, we provide the explicit computation that this neural net with the parameters and input above would compute in order to get the probability of the two output classes. The class predict by the neural net in this example is the output class with highest probability – which is also the output class with largest linear score y_j before applying the softmax. The computation is also depicted in the last figure. Note, the final application of softmax in the equations below is a slight abuse of notation – really, the softmax has to take in both y_1 and y_2 since it needs to normalize to make a probability distribution. In code, you would just call the softmax function on the vector y to get a vector of output probabilities. Or, during training, if you're using the CrossEntropyLoss function, you would just call that to compute your final loss function.

$$z_1 = W^1_{11}x_1 + W^1_{21}x_2 + B^1_1 = 0.5 * -1 + 0.6 * 1 + 0.5 = 0.7$$

$$z_2 = W^1_{12}x_1 + W^1_{22}x_2 + B^1_2 = 0.8 * -1 + 0.1 * 1 + 0.1 = -0.6$$

$$H_1 = ReLU(z_1) = max(z_1, 0) = 0.7$$

$$H_2 = ReLU(z_2) = max(z_2, 0) = 0$$

$$y_1 = W^2_{11}H_1 + W^2_{21}H_2 + B^2_1 = 0.3 * 0.7 + 0.2 * 0 + 0.3 = 0.51$$

$$y_2 = W^2_{12}H_1 + W^2_{22}H_2 + B^2_2 = 0.7 * 0.7 + 0.4 * 0 + 0.6 = 1.09$$

$$out_1 = softmax(y_1) = \frac{e^{y_1}}{\sum_{j\in\{1,2\}} e^{y_j}} = \frac{e^{0.51}}{e^{0.51} + e^{1.09}} = 0.3589$$

$$out_2 = softmax(y_2) = \frac{e^{y_2}}{\sum_{j\in\{1,2\}} e^{y_j}} = \frac{e^{1.09}}{e^{0.51} + e^{1.09}} = 0.6411$$

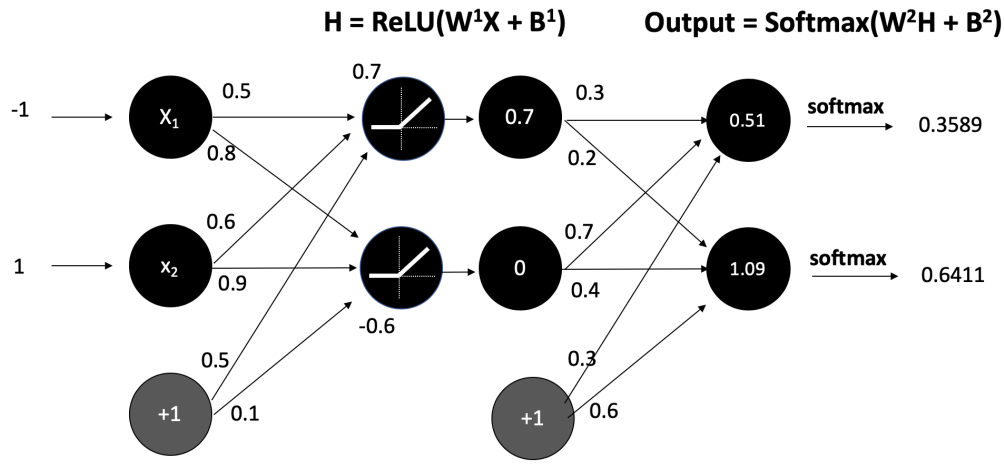


Figure 2: Numbers included to indicate computations of all hidden units and outputs for given parameters and input x.