ECE521 Lectures 9

Fully Connected Neural Networks

UNIVERSITY OF
TORONTO

Outline

- **Multi-class classification**
- Learning multi-layer neural networks

Measuring distance in probability space

- We learnt that the squared L2 distance is an important concept that captures the natural distance measure between the two points in Euclidean space. The Kullback-Leibler divergence is an equally important concept that is an appropriate distance measure between two probability distributions:

$$KL(Q\|P) = \sum_x Q(x) \log \frac{Q(x)}{P(x)}$$

  - KL divergence is an asymmetric distance function: $KL(Q\|P) \neq KL(P\|Q)$
  - KL divergence is also nonnegative for any two distributions

Binary cross-entropy loss

- Consider the *Q* distribution to be the discrete probability distribution of the observed binary labels $t \in \{0, 1\}$ in the training dataset
  - $Q(t = 1|\mathbf{x}) = t$ is either zero or one depending on the training example. (The dataset is observed, so there is no randomness)
- Choose the *P* distribution to be the model's prediction: $P(t = 1|\mathbf{x}) = \hat{y}(\mathbf{x})$. The KL divergence between *Q* and *P* is in fact the cross-entropy loss:

$$KL(Q\|P) = \sum_t Q(t|\mathbf{x}) \log \frac{Q(t|\mathbf{x})}{P(t|\mathbf{x})} = -\sum_t Q(t|\mathbf{x}) \log P(t|\mathbf{x}) + \boxed{\sum_t Q(t|\mathbf{x}) \log Q(t|\mathbf{x})}$$

$$= -Q(t = 1|\mathbf{x}) \log P(t = 1|\mathbf{x}) - Q(t = 0|\mathbf{x}) \log P(t = 0|\mathbf{x})$$

$$= \underbrace{-t \log \hat{y} - (1 - t) \log(1 - \hat{y})}_{\text{cross entropy}}$$

this is zero
0*log(0)→0

  - The cross-entropy loss function measures the distance between the empirical data distribution and the model predictive distribution

Multi-class cross-entropy loss and softmax

- It is easy to use the KL divergence interpretation to generalize the cross-entropy loss to a multi-class scenario:
  - Let there be *K* classes, with class labels $t \in \{1, \cdots, K\}$
  - The multi-class cross entropy loss can be written using indicator function I(.):

$$KL(Q\|P) = \sum_t Q(t|\mathbf{x}) \log \frac{Q(t|\mathbf{x})}{P(t|\mathbf{x})} = -\sum_{k=1}^{K} I(k, t_{class}) \log P(t = k|\mathbf{x})$$

  - Similarly, the multi-class generalization of the sigmoid function is the *softmax function*. The multi-class predictive distribution becomes:

$$P(t = k|\mathbf{x}) = \frac{e^{z_k}}{\sum_j e^{z_j}}$$

Here z_k are the outputs of a neural network or linear regression

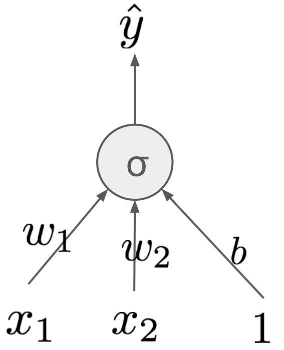
Outline

- Multi-class classification
- **Learning multi-layer neural networks**

Example 1: representing digital circuits with neural networks

- Let us look at a simple example of a soft OR gate simulated by a neural network:
  - Use a single sigmoid neuron with two inputs and a bias unit.

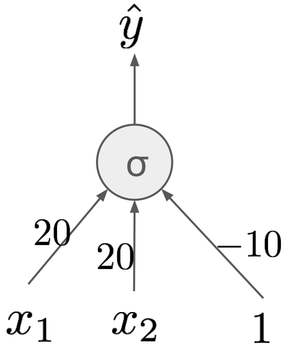
	x2=0	x2=1
x1=0	0	1
x1=1	1	1



Example 1: representing digital circuits with neural networks

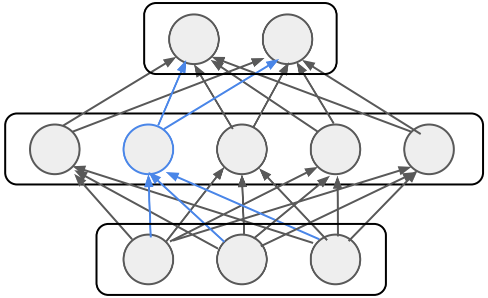
- Let us look at a simple example of a soft OR gate simulated by a neural network:
    - Use a single sigmoid neuron with two inputs and a bias unit.
    - One possible solution is to use the bias as a threshold while setting w_1 and w_2 to be large positive values. When either of the inputs is non-zero, the sigmoid neuron will be turned on and the output will be 1.

	x2=0	x2=1
x1=0	0	1
x1=1	1	1



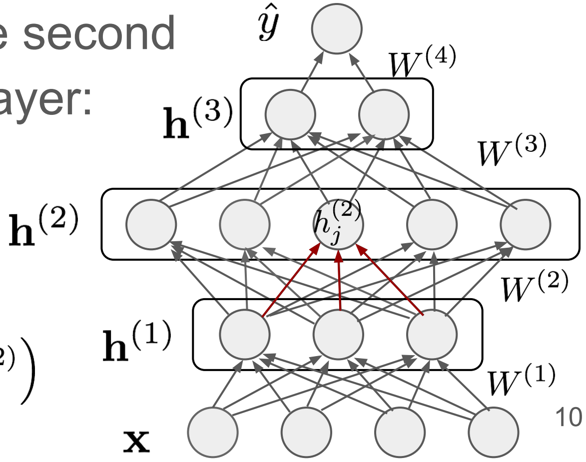
Learning fully connected multi-layer neural networks

- There are many choices when “crafting” the architecture of a neural network. The fully connected multi-layer NN is the most general multi-layer NN:
    - Each neuron has its incoming weights connected to *all* the neurons from the previous layer and its outgoing weights connected to *all* the neurons in the next layer.
- Fully connected network is the go-to architecture choice if we do not have any additional information about the dataset.
    - After choosing the network architecture, there are a few more engineering choices: #hidden units, #layers, the type of activation function.
    - The output units type: linear, logistic or softmax are determined by output tasks, i.e. regression or classification task



Learning fully connected multi-layer neural networks

- Consider a fully connected neural network with 3 hidden layers:
  - The input to the neural network is an *N*-dimensional vector $\mathbf{x}$. There are *H1, H2,* and *H3* hidden units in the three hidden layers. We use superscript to index the layers.
  - There are four weight matrices among the hidden layers, e.g. $W^{(2)} \in \mathbb{R}^{H_2 \times H_1}, b^{(2)} \in \mathbb{R}^{H_2}$
  - The *j*th row of the weight matrix $W^{(2)}$ is denoted as $W_j^{(2)} \in \mathbb{R}^{H_1}$
- The hidden activation of the *j*th hidden unit $h_j^{(2)}$ in the second hidden layer is the weighted sum of the first hidden layer:



$$h_j^{(2)} = \phi\left(z_j^{(2)}\right) = \phi\left(\sum_i w_{ij}^{(2)} h_i^{(1)} + b_j^{(2)}\right) = \phi\left({W_j^{(2)}}^T \mathbf{h}^{(1)} + b_j^{(2)}\right)$$

  - We can use vector notation to express the hidden vector:

$$\mathbf{h}^{(2)} = \begin{bmatrix} h_1^{(2)} \\ \vdots \\ h_{H_2}^{(2)} \end{bmatrix} = \begin{bmatrix} \phi\left(z_1^{(2)}\right) \\ \vdots \\ \phi\left(z_{H_2}^{(2)}\right) \end{bmatrix} = \phi\left(\begin{bmatrix} {W_1^{(2)}}^T \\ \vdots \\ {W_{H_2}^{(2)}}^T \end{bmatrix} \mathbf{h}^{(1)} + b^{(2)}\right) = \phi\left(W^{(2)}\mathbf{h}^{(1)} + b^{(2)}\right)$$

Learning fully connected multi-layer neural networks

- For a single data point, we can write the the hidden activations of the fully connected neural network as a recursive computation using the vector notation:

$$\mathbf{z}^{(1)} = W^{(1)}\mathbf{x} + b^{(1)}, \quad \mathbf{h}^{(1)} = \phi\left(\mathbf{z}^{(1)}\right)$$

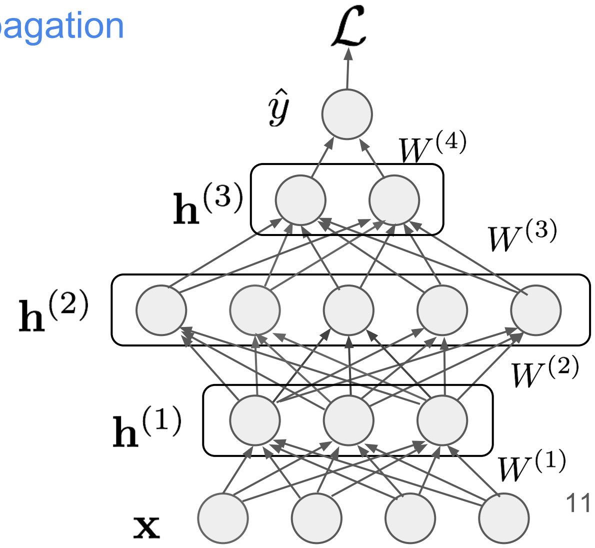
$$\mathbf{z}^{(2)} = W^{(2)}\mathbf{h}^{(1)} + b^{(2)}, \quad \mathbf{h}^{(2)} = \phi\left(\mathbf{z}^{(2)}\right)$$

$$\mathbf{z}^{(3)} = W^{(3)}\mathbf{h}^{(2)} + b^{(3)}, \quad \mathbf{h}^{(3)} = \phi\left(\mathbf{z}^{(3)}\right)$$

$$\mathbf{z}^{(4)} = W^{(4)}\mathbf{h}^{(3)} + b^{(4)}, \quad \hat{y} = f\left(\mathbf{z}^{(4)}\right)$$

  - *f()* is the output activation function
  - The output of the network is then used to compute the loss function on the training data

forward propagation

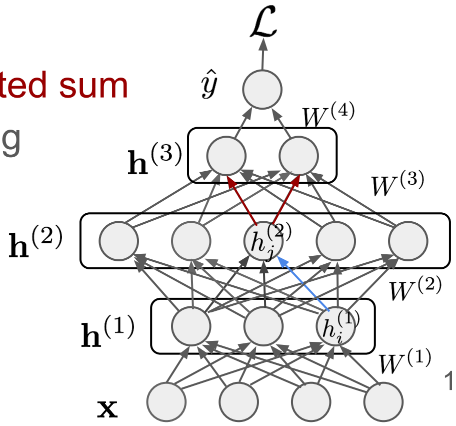


Learning fully connected multi-layer neural networks

- Learning neural networks using stochastic gradient descent requires the gradient of the weight matrices from each hidden layer.
    - Let us consider the gradient of the loss for a single training example. The gradient w.r.t. the incoming weights $w_{ij}^{(2)}$ of the *j*th hidden unit in the second layer is the *product* of the hidden activation from layer 1 *and* the partial derivative w.r.t. z_j. Remember: $h_j^{(2)} = \phi\left(z_j^{(2)}\right) = \phi\left(\sum_i w_{ij}^{(2)} h_i^{(1)} + b_j^{(2)}\right)$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial w_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(2)}} h_i^{(1)}$$

    - The partial derivative w.r.t. z_j in the second hidden layer is the weighted sum of the partial derivatives from the third layer, weighted by the outgoing weights of the *j*th hidden units:



$$\frac{\partial \mathcal{L}}{\partial z_j^{(2)}} = \frac{\partial \mathcal{L}}{\partial h_j^{(2)}} \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}} = \left(\sum_i \frac{\partial \mathcal{L}}{\partial z_i^{(3)}} \frac{\partial z_i^{(3)}}{\partial h_j^{(2)}}\right) \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}} = \boxed{\left(\sum_i \frac{\partial \mathcal{L}}{\partial z_i^{(3)}} w_{ji}^{(3)}\right)} \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}}$$

Learning fully connected multi-layer neural networks

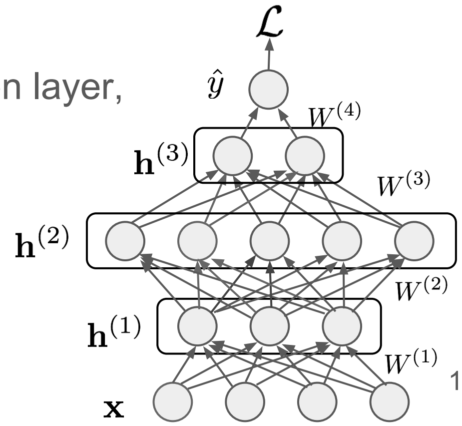
- Similar to the hidden-activation computation (slide 10), the weighted sum of the partial derivatives can be rewritten using vector notation:

$$\frac{\partial \mathcal{L}}{\partial z_j^{(2)}} = \left(\sum_i \frac{\partial \mathcal{L}}{\partial z_i^{(3)}} w_{ji}^{(3)}\right) \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}} = \left(\mathcal{W}_{j\cdot}{}^{(3)T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}}\right) \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}}$$

$$\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{z}^{(2)}} = \begin{bmatrix} \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} 0 & \cdots & 0 \\ \vdots & \frac{\partial h_j^{(2)}}{\partial z_j^{(2)}} & \vdots \\ 0 & \cdots & \frac{\partial h_{H_2}^{(2)}}{\partial z_{H_2}^{(2)}} \end{bmatrix} = \mathrm{diag}\left\{\begin{bmatrix} \frac{\partial h_1^{(2)}}{\partial z_1^{(2)}} \\ \vdots \\ \frac{\partial h_{H_2}^{(2)}}{\partial z_{H_2}^{(2)}} \end{bmatrix}\right\}$$

  - Here, $\mathcal{W}_{j\cdot}^{(3)}$ is the *j*th column of the weight matrix $W^{(3)}$

  - To express the partial derivatives w.r.t. *z* for the entire second hidden layer, we can use a matrix-vector product:



$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial z_1^{(2)}} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial z_{H_2}^{(2)}} \end{bmatrix} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{z}^{(2)}} \left(\begin{bmatrix} \mathcal{W}_{1\cdot}^{(3)T} \\ \vdots \\ \mathcal{W}_{H_2\cdot}^{(3)T} \end{bmatrix} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}}\right) = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{z}^{(2)}} \left(W^{(3)T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}}\right)$$

Learning fully connected multi-layer neural networks

- For a single training datum, computing the gradient w.r.t. the weight matrices is also a recursive procedure:
  - Remember: $\mathbf{z}^{(4)} = W^{(4)}\mathbf{h}^{(3)} + b^{(4)}, \quad \hat{y} = f\left(\mathbf{z}^{(4)}\right)$
  - Back-propagation is similar to running the neural network backwards using the transpose of the weight matrices

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(4)}} = \frac{\partial \hat{y}}{\partial \mathbf{z}^{(4)}} \frac{\partial \mathcal{L}}{\partial \hat{y}}, \quad \frac{\partial \mathcal{L}}{\partial W^{(4)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(4)}} \mathbf{h}^{(3)T}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}} = \frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{z}^{(3)}} \left(W^{(4)T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(4)}} \right), \quad \frac{\partial \mathcal{L}}{\partial W^{(3)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}} \mathbf{h}^{(2)T}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{z}^{(2)}} \left(W^{(3)T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(3)}} \right), \quad \frac{\partial \mathcal{L}}{\partial W^{(2)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \mathbf{h}^{(1)T}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{z}^{(1)}} \left(W^{(2)T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(2)}} \right), \quad \frac{\partial \mathcal{L}}{\partial W^{(1)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{(1)}} \mathbf{x}^{T}$$

back-propagation

What about the expression for the bias units?