Neural Networks
Part 4 – Training with Backpropagation

CSE 4311 – Neural Networks and Deep Learning

Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Review: A Multiclass Example

- Suppose we have this training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad q_1 = \text{dog}$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad q_2 = \text{dog}$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad q_3 = \text{cat}$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad q_4 = \text{fox}$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad q_5 = \text{cat}$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad q_6 = \text{fox}$
- In this training set:
  - We have three classes.
  - Each training input is a five-dimensional vector.

Review: Generating One-Hot Vectors

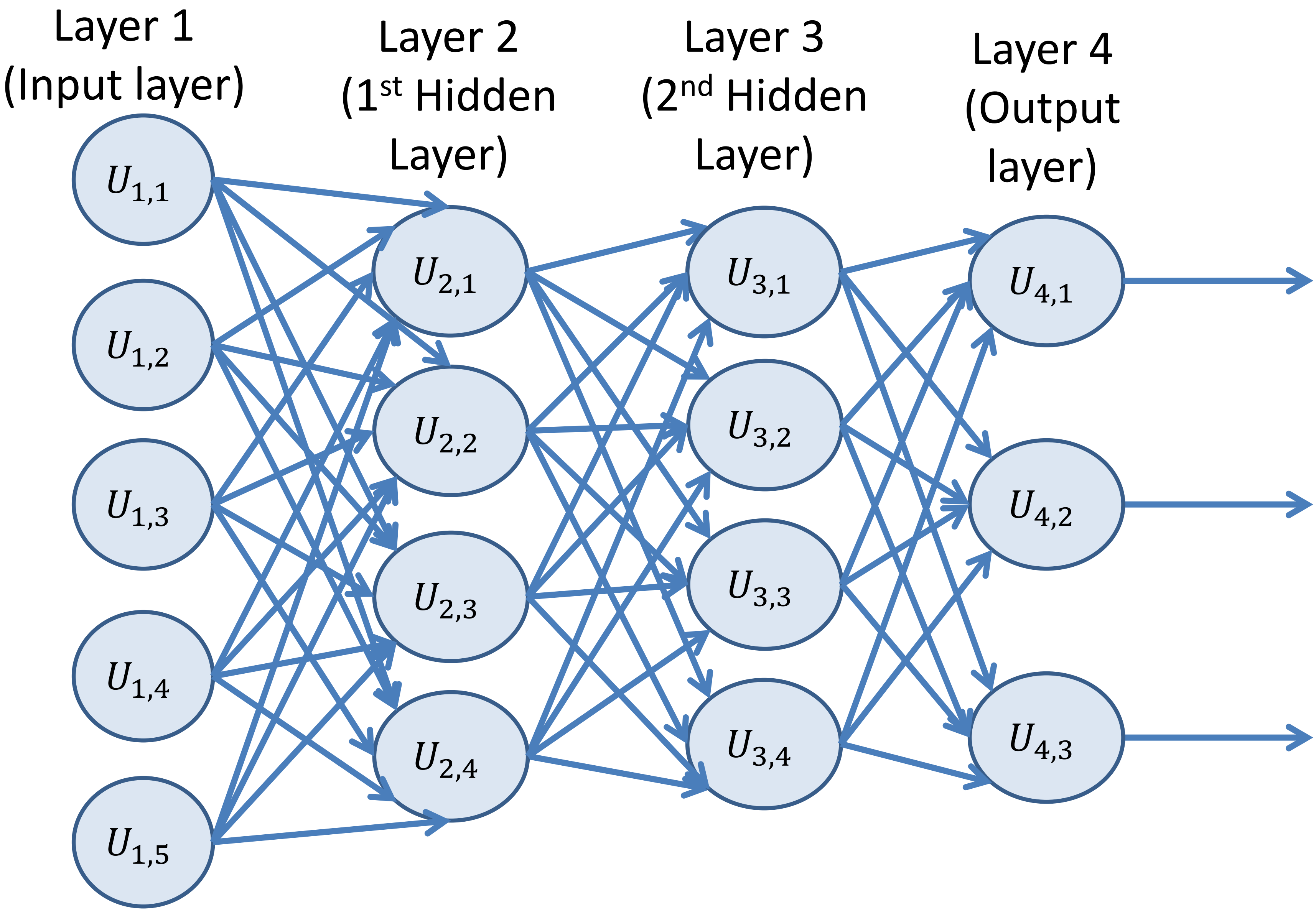
- Before we train a neural network, we must convert class labels to one-hot-vectors.
- Step 1: convert class labels q_n to new class labels s_n, which are integers between 1 and K.
  - K is the number of classes, $K = 3$ in our example.
- Step 2: convert labels s_n to K-dimensional one-hot vectors $\boldsymbol{t}_n$.

- Result: training set with new class labels s_n and one-hot vectors $\boldsymbol{t}_n$:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T$, $s_1 = 1$ $\boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T$, $s_2 = 1$ $\boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T$, $s_3 = 2$ $\boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T$, $s_4 = 3$ $\boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T$, $s_5 = 2$ $\boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T$, $s_6 = 3$ $\boldsymbol{t}_6 = (0, 0, 1)^T$

Review: Multiclass Neural Networks

- For perceptrons, we saw that we can perform multiclass (i.e., for more than two classes) classification by training one perceptron for each class.
- For neural networks, we will train a SINGLE neural network, with MULTIPLE output units.
  - The number of output units will be equal to the number of classes.

Review: A Network for Our Example



Neural Network Notation

- L is the total number of layers in the neural network.
- D is the number of dimensions of the input.
- K is the number of classes we want to recognize.
- Each unit, is denoted as $U_{l,i}$, where :

  $1 \leq l \leq L$, and l is the layer index

  $1 \leq i$, and i is the index of the unit within layer l.

  Layer 1 is the input layer. Units $U_{1,1}, \ldots, U_{1,D}$ are the **input units**.

  Layer L is the output layer. Units $U_{L,1}, \ldots, U_{L,K}$ are the **output units**.
- We denote by $w_{l,i,j}$ the weight of the edge connecting the output of unit $U_{l-1,j}$ to an input of unit $U_{l,i}$.

  j is the index of the unit in layer $l - 1$.

  i is the index of the unit in layer l.
- We denote by $b_{l,i}$ the bias weight of $U_{l,i}$.

Neural Network Notation

- We denote by J_l the number of units in layer l.
  - For the input layer, $J_1 = \mathrm{D}$.
  - For the output layer, $J_L = \mathrm{K}$ (the number of classes).
  - For each hidden layer l, J_l is a hyperparameter.
- We denote by $a_{l,i}$ the weighted sum that is calculated at $U_{l,i}$.

$$a_{l,i} = b_{l,i} + \sum_{j=1}^{J_{l-1}} \left(w_{l,i,j} * z_{l-1,j}\right)$$

  - Note that this weighted sum is NOT applicable when $l = 1$, it only starts getting calculated for $l \geq 2$. So, $a_{1,i}$ is not defined.
- We denote by $z_{l,i}$ the output of unit $U_{l,i}$.
  - If $l = 1$, then $z_{1,i} = x_i$.
  - If $l \geq 2$, then $z_{l,i} = \sigma\left(a_{l,i}\right) = \frac{1}{1+e^{-a_{l,i}}}$

Neural Network Notation

- We denote by J_l the number of units in layer l.
  - For the input layer, $J_1 = \mathrm{D}$.
  - For the output layer, $J_L = \mathrm{K}$ (the number of classes).
  - For each hidden layer l, J_l is a hyperparameter.
- We denote by $a_{l,i}$ the weighted sum that is calculated at $U_{l,i}$.

$$a_{l,i} = b_{l,i} + \sum_{j=1}^{J_{l-1}} \left(w_{l,i,j} * z_{l-1,j} \right)$$

  - Note that this weighted sum is NOT applicable when $l = 1$, it only starts getting calculated for $l \geq 2$. So, $a_{1,i}$ is not defined.
- We denote by $z_{l,i}$ the output of unit $U_{l,i}$.
  - If $l = 1$, then $z_{1,i} = x_i$.
  - If $l \geq 2$, then $z_{l,i} = \sigma(a_{l,i}) = \frac{1}{1+e^{-a_{l,i}}}$

In these slides, we assume that we are using the sigmoid as activation function.

Squared Error for Neural Networks

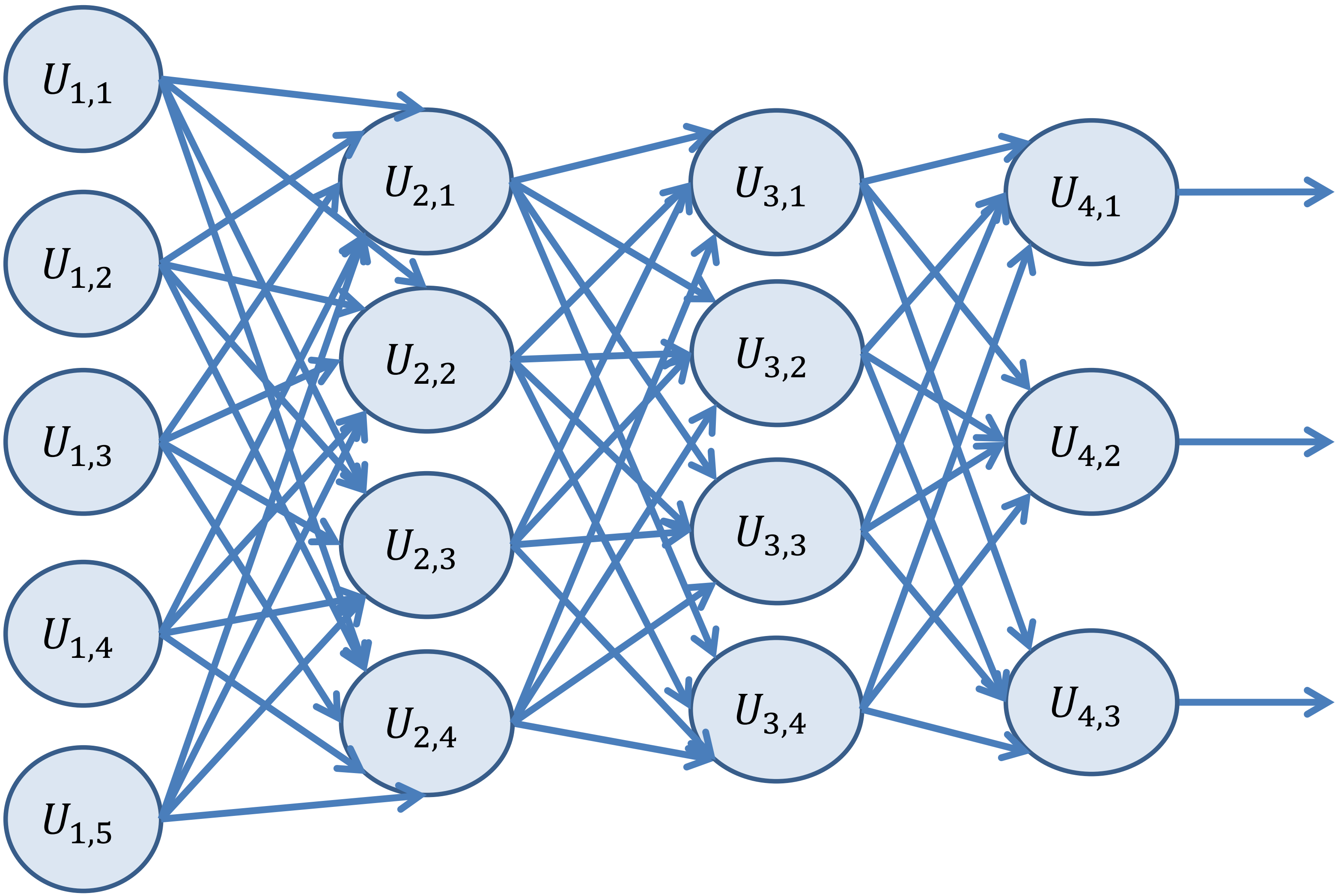
- In neural networks, the optimization criterion is usually called the **loss function**.
  - In these slides, the loss function will be the sum of squared differences.
- We denote by $\boldsymbol{b}$ the vector of all bias weights $b_{l,i}$.
- We denote by $\boldsymbol{w}$ the vector of all weights $w_{l,i,j}$.
- We denote by $E_n(\boldsymbol{b}, \boldsymbol{w})$ the contribution that training input $\boldsymbol{x}_n$ makes to the overall loss function, given $\boldsymbol{b}, \boldsymbol{w}$.

$$E_n(\boldsymbol{b}, \boldsymbol{w}) = \frac{1}{2} \sum_{c=1}^{K} \left\{ \left(t_{n,c} - z_{L,c} \right)^2 \right\}$$

- Remember: target output $\boldsymbol{t}_n$ is a one-hot vector.
  - We denote by $t_{n,c}$ the c-th dimension of target output $\boldsymbol{t}_n$.

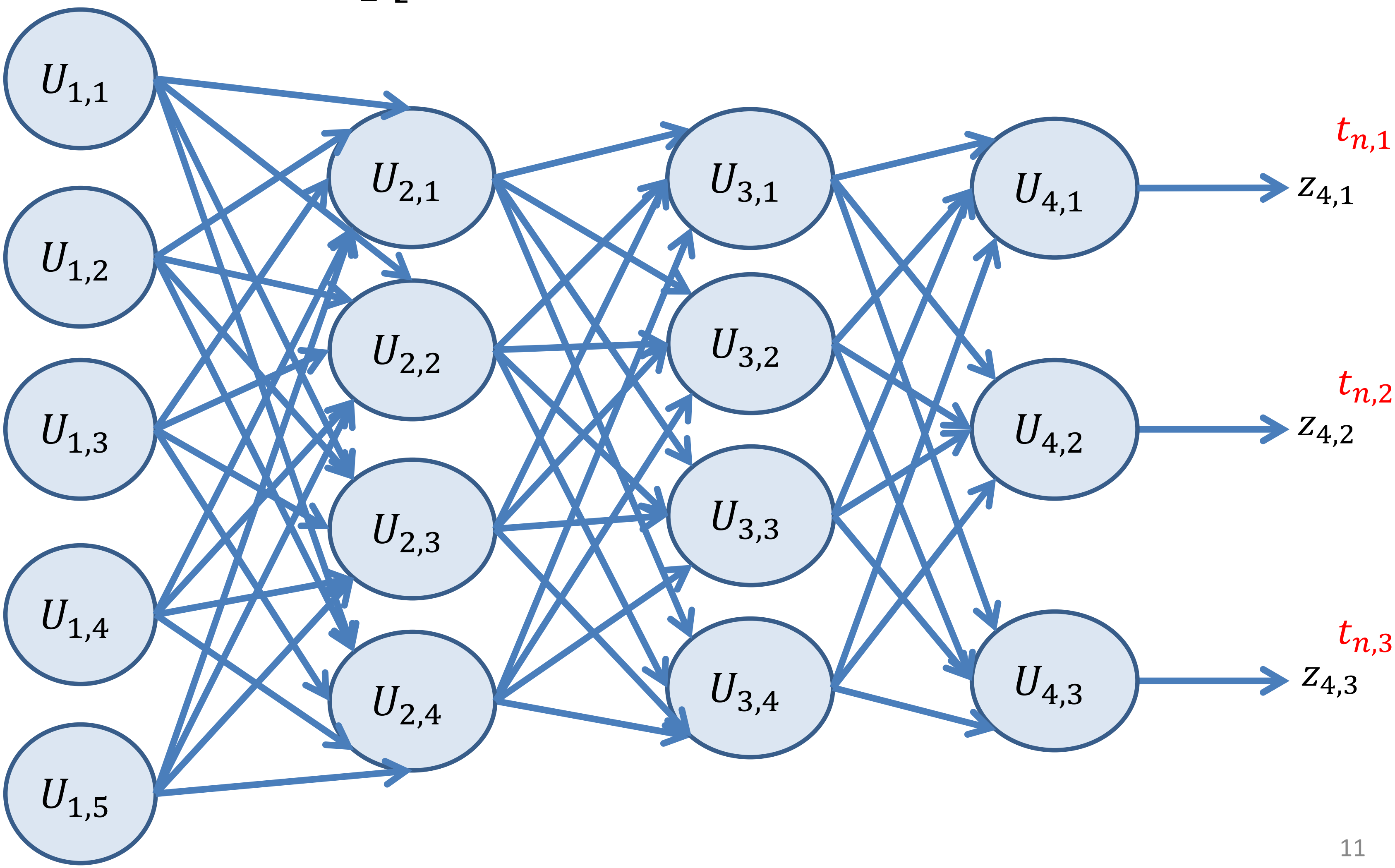
In our three-class example:

- $E_n = \frac{1}{2}\sum_{c=1}^{K}\left\{\left(t_{n,c} - z_{L,c}\right)^2\right\}$
- What does each $z_{L,c}$ correspond to?



In our three-class example:

- $E_n = \frac{1}{2}\sum_{c=1}^{K}\left\{\left(t_{n,c} - z_{L,c}\right)^2\right\}$
- $E_n = \frac{1}{2}\left[\left(t_{n,1} - z_{4,1}\right)^2 + \left(t_{n,2} - z_{4,2}\right)^2 + \left(t_{n,3} - z_{4,3}\right)^2\right]$



Squared Error for Neural Networks

- We denote by $E(\boldsymbol{b}, \boldsymbol{w})$ the overall loss (or error) over all training examples for the network specified by $\boldsymbol{b}, \boldsymbol{w}$.

$$E(\boldsymbol{b}, \boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{b}, \boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{c=1}^{K} \left\{ \left(t_{n,c} - z_{L,c} \right)^2 \right\}$$

- This is now a double summation.
  - We sum over all training examples $\boldsymbol{x}_n$.
  - For each $\boldsymbol{x}_n$, we sum over all perceptrons in the output layer.

Training Neural Networks

- To train a neural network, we use gradient descent.
  - We follow the same approach of sequential learning that we followed for training single perceptrons.
- Given a training example $\boldsymbol{x}_n$ and target output $\boldsymbol{t}_n$:
  - Compute the training loss $E_n(\boldsymbol{b}, \boldsymbol{w})$.
  - Compute the gradients $\frac{\partial E_n}{\partial \boldsymbol{b}}$ and $\frac{\partial E_n}{\partial \boldsymbol{w}}$.
  - Based on the gradients, we can update all weights $\boldsymbol{b}$ and $\boldsymbol{w}$.
- The process of computing the gradient and updating neural network weights is called **backpropagation**.
- We will see the solution when we use the sigmoidal function as activation function h.

Computing the Gradient

- Overall, we want to compute $\frac{\partial E_n}{\partial \boldsymbol{b}}$ and $\frac{\partial E_n}{\partial \boldsymbol{w}}$.
- This is the same as computing:
  - For each bias weight $b_{l,i}$, the partial derivative $\frac{\partial E_n}{\partial b_{l,i}}$
  - For each $w_{l,i,j}$, the partial derivative $\frac{\partial E_n}{\partial w_{l,i,j}}$.
- To compute $\frac{\partial E_n}{\partial b_{l,i}}$ and $\frac{\partial E_n}{\partial w_{l,i,j}}$, we will use this strategy:
  - Decompose E_n into a composition of simpler functions.
  - Compute the derivative of each of those simpler functions.
  - Apply the chain rule to obtain $\frac{\partial E_n}{\partial b_{l,i}}$ and $\frac{\partial E_n}{\partial w_{l,i,j}}$.

Decomposing the Loss Function

- Let $U_{l,i}$ be a perceptron in the neural network.
- Define $a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})$ to be the weighted sum of the inputs of $U_{l,i}$, given input $\boldsymbol{x}_n$ and given the current values of $\boldsymbol{b}$ and $\mathbf{w}$.

$$a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) = b_{l,i} + \sum_{j=1}^{J_{l-1}} \left(w_{l,i,j} * z_{l-1,j}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) \right)$$

- Remember that $z_{l,i}$ is the output of unit $U_{l,i}$, and it is obtained by applying the sigmoid function on top of $a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})$.

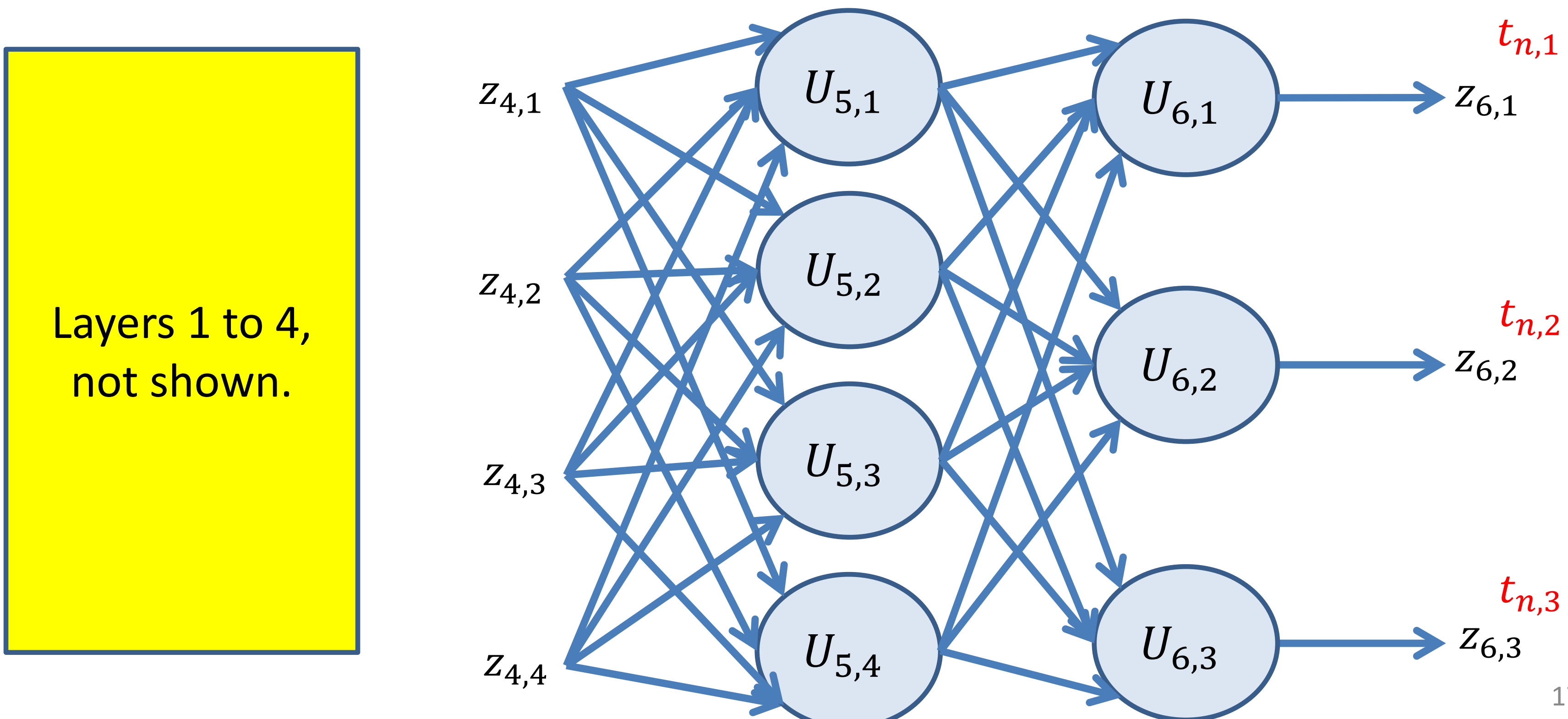
$$z_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) = \sigma\left(a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right) = \frac{1}{1 + e^{-a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})}}$$

Decomposing the Loss Function

- Define $\mathbf{z}_l$ to be a vector containing the outputs of all units at layer l.
  - Using our notation, $\mathbf{z}_l = \left(z_{l,1}, z_{l,2}, \dots, z_{l,J_l}\right)^T$, where J_l is the number of units at layer l.
- Define function $E_{n,l}(\mathbf{z}_l, \mathbf{b}, \mathbf{w})$ to be the loss of the network given outputs $\mathbf{z}_l$.
- Intuition for $E_{n,l}(\mathbf{z}_l, \mathbf{b}, \mathbf{w})$:
  - Suppose that you know $\mathbf{z}_l$, and the weights for all layers after layer l.
  - Then, you can still compute the output of the network, and the loss $E_n(\mathbf{b}, \mathbf{w})$.

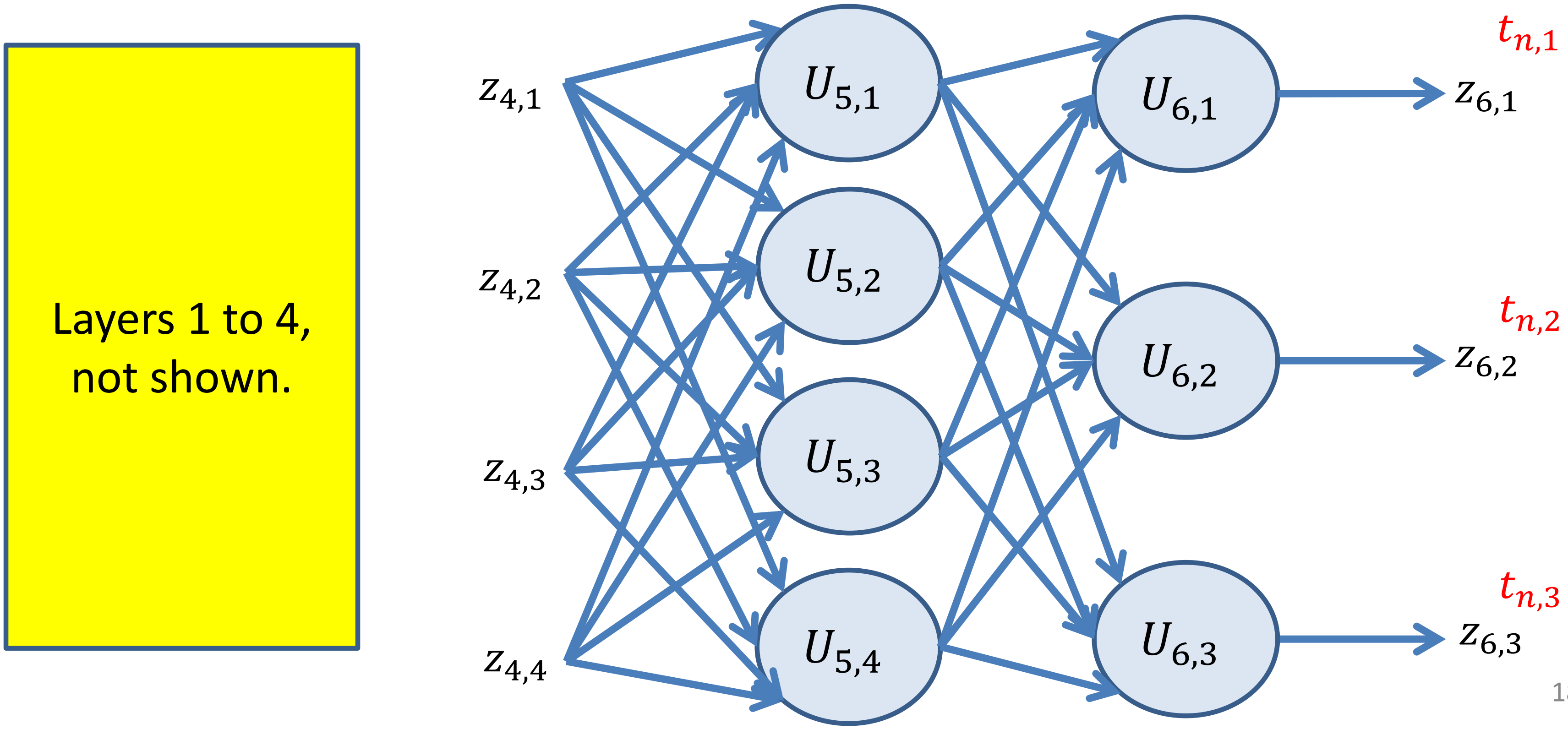
Visualizing Function $E_{n,l}$

- Suppose we know the target output $\mathbf{t}_n$, and all weights $\boldsymbol{b}$ and $\boldsymbol{w}$.
- If we know the output $\mathbf{z}_l$ of layer l:
  - Can we compute the output of the network?
  - Can we compute the loss value E_n?



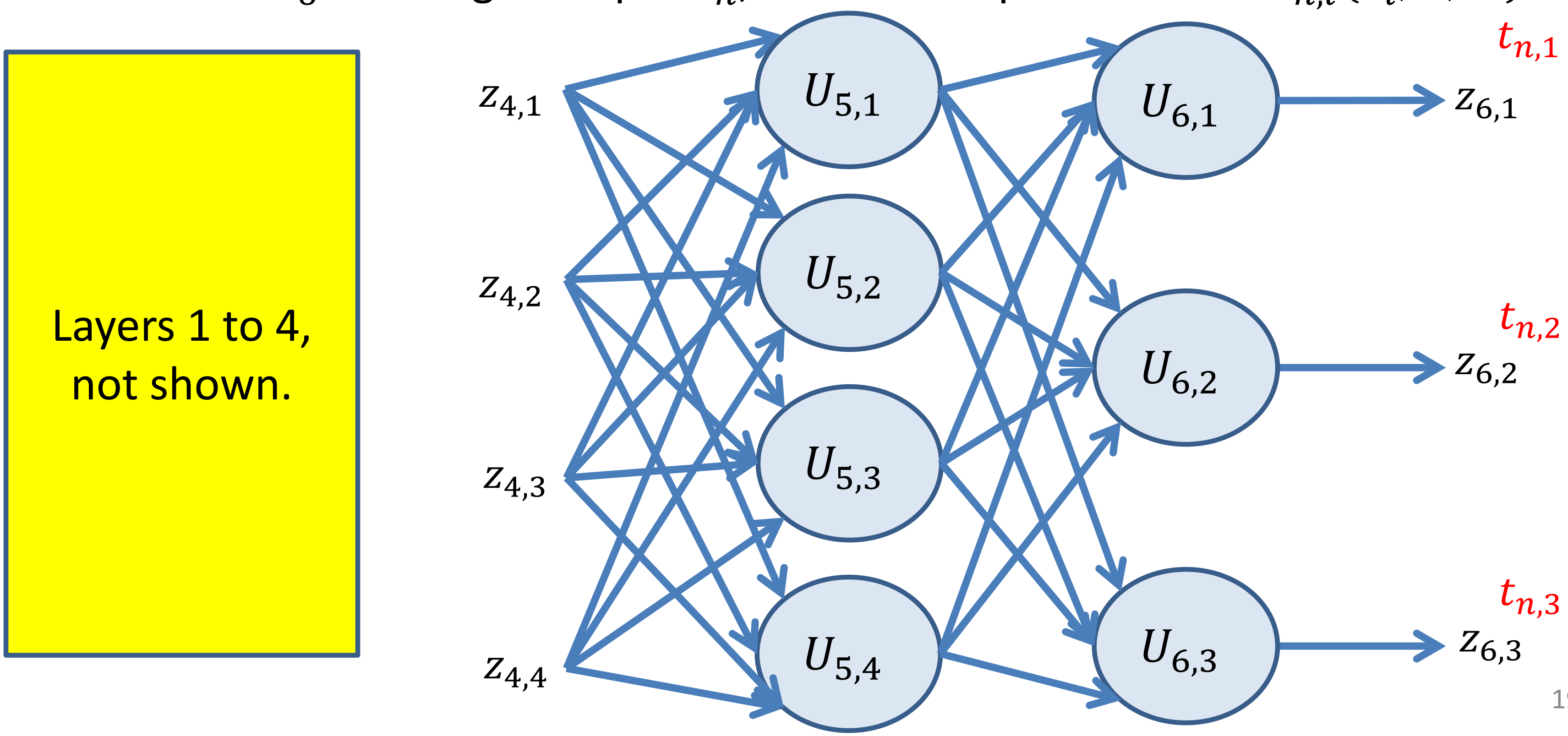
Visualizing Function $E_{n,l}$

- In this example, the network has six layers.
  - We have no idea what happens in layers 1 to 4.
  - However, we are given the output $\boldsymbol{z}_4$ of layer 4.
  - Can we compute the output of the network?
  - Can we compute the loss E_n?



Visualizing Function $E_{n,l}$

- In this example, given the output $\boldsymbol{z}_4$ of layer 4, if we know all weights $\boldsymbol{b}$ and $\boldsymbol{w}$, we can compute the final output and the loss.
  - Given $\boldsymbol{z}_4$, we can compute the output $\boldsymbol{z}_5$ of layer 5.
  - Given $\boldsymbol{z}_5$, we can compute the output $\boldsymbol{z}_6$ of layer 6, (the output layer).
  - Given $\boldsymbol{z}_6$ and target output $\boldsymbol{t}_n$, we can compute the loss $E_{n,l}(\mathbf{z}_l, \mathbf{b}, \mathbf{w})$.



Decomposing the Loss Function

- We have three auxiliary functions:
  - $a_{l,i}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})$
  - $\sigma(\alpha)$
  - $E_{n,l}(\mathbf{z}_l, \mathbf{b}, \mathbf{w})$
- Then, E_n is a composition of functions $E_{n,l}$, σ, $a_{l,i}$.

$$E_n(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) = E_{n,l}(\mathbf{z}_l, \mathbf{b}, \mathbf{w})$$

$$= E_{n,l}\left(\underbrace{\left(\sigma\left(a_{l,1}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right), \ldots, \sigma\left(a_{l,J_l}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right)\right)}_{\mathbf{z}_l}, \mathbf{b}, \mathbf{w}\right)$$

Computing the Gradient of E_n

$$E_n(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) = E_{n,l}\left(\left(z_{l,1}, z_{l,2}, \dots, z_{l,J_l}\right), \mathbf{b}, \mathbf{w}\right)$$
$$= E_{n,l}\left(\left(\sigma\left(a_{l,1}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right), \dots, \sigma\left(a_{l,J_l}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right)\right), \mathbf{b}, \mathbf{w}\right)$$

- So, E_n is a composition of function $E_{n,l}$, function σ, and functions $a_{l,i}$.
- This allows us to compute $\frac{\partial E_n}{\partial b_{l,i}}$ and $\frac{\partial E_n}{\partial w_{l,i,j}}$ by applying the chain rule.

Computing the Gradient of E_n

$$E_n(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) = E_{n,l}\left(\left(z_{l,1}, z_{l,2}, \dots, z_{l,J_l}\right), \mathbf{b}, \mathbf{w}\right)$$

$$= E_{n,l}\left(\left(\sigma\left(a_{l,1}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right), \dots, \sigma\left(a_{l,J_l}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w})\right)\right), \mathbf{b}, \mathbf{w}\right)$$

- Applying the chain rule:

$$\frac{\partial E_n}{\partial b_{l,i}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial b_{l,i}}$$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

Computing $\frac{\partial a_{l,i}}{\partial b_{l,i}}$ and $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$

$$\frac{\partial E_n}{\partial b_{l,i}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial b_{l,i}}$$

$$\frac{\partial a_{l,i}}{\partial b_{l,i}} = \frac{\partial\left(b_{l,i} + \sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)\right)}{\partial b_{l,i}}$$

$$=???$$

This is actually very simple. It is of this form:

$$\frac{\partial(x + \text{stuff that is independent of x})}{\partial x} =???$$

Computing $\frac{\partial a_{l,i}}{\partial b_{l,i}}$ and $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$

$$\frac{\partial E_n}{\partial b_{l,i}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial b_{l,i}}$$

$$\frac{\partial a_{l,i}}{\partial b_{l,i}} = \frac{\partial\left(b_{l,i} + \sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)\right)}{\partial b_{l,i}}$$

$$= 1$$

This is actually very simple. It is of this form:

$$\frac{\partial(x + \text{stuff that is independent of x})}{\partial x} = 1$$

Computing $\frac{\partial a_{l,i}}{\partial b_{l,i}}$ and $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$\frac{\partial a_{l,i}}{\partial w_{l,i,j}} = \frac{\partial\left(b_{l,i} + \sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)\right)}{\partial w_{l,i,j}}$$

$$=???$$

How does $w_{l,i,j}$ influence $\sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)$?

Computing $\frac{\partial a_{l,i}}{\partial b_{l,i}}$ and $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$\frac{\partial a_{l,i}}{\partial w_{l,i,j}} = \frac{\partial\left(b_{l,i} + \sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)\right)}{\partial w_{l,i,j}}$$

$$= ???$$

In $\sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)$, at some point $k = j$. Then, $w_{l,i,k}$ is multipled by $z_{l-1,j}$. Based on that, $\frac{\partial a_{l,i}}{\partial w_{l,i,j}} = ???$

Computing $\frac{\partial a_{l,i}}{\partial b_{l,i}}$ and $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$\frac{\partial a_{l,i}}{\partial w_{l,i,j}} = \frac{\partial\left(b_{l,i} + \sum_{k=1}^{J_{l-1}}\left(w_{l,i,k} * z_{l-1,k}\right)\right)}{\partial w_{l,i,j}}$$

$$= z_{l-1,j}$$

Computing $\frac{\partial z_{l,i}}{\partial a_{l,i}}$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$\frac{\partial z_{l,i}}{\partial a_{l,i}} = \frac{\partial \left(\sigma(a_{l,i})\right)}{\partial a_{l,i}} = \sigma(a_{l,i}) * \left(1 - \sigma(a_{l,i})\right) = z_{l,i} * (1 - z_{l,i})$$

- We just use the known formula for the derivative of σ.
  - One of the reasons we like using the sigmoidal function for activation is that its derivative has such a simple form.

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 1: If Unit $U_{l,i}$ Is an Output Unit

- If $U_{l,i}$ is an output unit, then $z_{l,i}$ is an output of the entire network.
- $z_{l,i}$ contributes to the loss the term $\frac{1}{2}\left(t_{n,i} - z_{l,i}\right)^2$.
- Therefore:

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \frac{\partial \frac{1}{2}\left(t_{n,i} - z_{l,i}\right)^2}{\partial z_{l,i}} = z_{l,i} - t_{n,i}$$

Updating Weights of Output Units

- If $U_{l,i}$ is an output unit, then we have computed all the terms we need for $\frac{\partial E_n}{\partial w_{l,i,j}}$.

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \overbrace{\left(z_{l,i} - t_{n,i}\right)} * \overbrace{z_{l,i} * \left(1 - z_{l,i}\right)} * \overbrace{z_{l-1,j}}$$

- So, if $U_{l,i}$ is an output unit, we update $w_{l,i,j}$ as:

$$w_{l,i,j} = w_{l,i,j} - \eta\left(z_{l,i} - t_{n,i}\right) * z_{l,i} * \left(1 - z_{l,i}\right) * z_{l-1,j}$$

Updating Weights of Output Units

- Similarly, if $U_{l,i}$ is an output unit, we can compute $\frac{\partial E_n}{\partial b_{l,i}}$.

$$\frac{\partial E_n}{\partial b_{l,i}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial b_{l,i}}$$

$$\frac{\partial E_n}{\partial b_{l,i}} = \overbrace{(z_{l,i} - t_{n,i})} * \overbrace{z_{l,i} * (1 - z_{l,i})} * \overbrace{1}$$

- So, if $U_{l,i}$ is an output unit, we update $b_{l,i}$ using:

$$b_{l,i} = b_{l,i} - \eta(z_{l,i} - t_{n,i}) * z_{l,i} * (1 - z_{l,i})$$

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 2: If Unit $U_{l,i}$ Is a Hidden Unit

- We want to compute $\frac{\partial E_{n,l}}{\partial z_{l,i}}$ when $U_{l,i}$ is a hidden unit.
- We use the chain rule, to relate $E_{n,l}$ to $E_{n,l+1}$.

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * \frac{\partial z_{l+1,k}}{\partial a_{l+1,k}} * \frac{\partial a_{l+1,k}}{\partial z_{l,i}} \right)$$

- We need to compute these three terms.

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 2: If Unit $U_{l,i}$ Is a Hidden Unit

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * \frac{\partial z_{l+1,k}}{\partial a_{l+1,k}} * \frac{\partial a_{l+1,k}}{\partial z_{l,i}} \right)$$

$$\frac{\partial a_{l+1,k}}{\partial z_{l,i}} = \frac{\partial \left(b_{l+1,k} + \sum_{j=1}^{J_l} \left(w_{l+1,k,j} * z_{l,j} \left(a_{l,j}(\boldsymbol{x}_n, \mathbf{b}, \mathbf{w}) \right) \right) \right)}{\partial z_{l,i}}$$

$$= w_{l+1,k,i}$$

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 2: If Unit $U_{l,i}$ Is a Hidden Unit

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * \frac{\partial z_{l+1,k}}{\partial a_{l+1,k}} * \frac{\partial a_{l+1,k}}{\partial z_{l,i}} \right)$$

$$\frac{\partial z_{l+1,k}}{\partial a_{l+1,k}} = \frac{\partial \left(\sigma(a_{l+1,k})\right)}{\partial a_{l+1,k}} = \sigma(a_{l+1,k}) * \left(1 - \sigma(a_{l+1,k})\right)$$

$$= z_{l,i} * \left(1 - z_{l,i}\right)$$

Derivative of the sigmoid function

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 2: If Unit $U_{l,i}$ Is a Hidden Unit

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * \frac{\partial z_{l+1,k}}{\partial a_{l+1,k}} * \frac{\partial a_{l+1,k}}{\partial z_{l,i}} \right)$$

- We can plug in our results for $\frac{\partial z_{l+1,k}}{\partial a_{l+1,k}}$ and $\frac{\partial a_{l+1,k}}{\partial z_{l,i}}$.
- So, the formula becomes:

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * z_{l+1,k} * \left(1 - z_{l+1,k}\right) * w_{l+1,k,i} \right)$$

Computing $\frac{\partial E_{n,l}}{\partial z_{l,i}}$, Case 2: If Unit $U_{l,i}$ Is a Hidden Unit

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * z_{l+1,k}\left(1 - z_{l+1,k}\right) * w_{l+1,k,i} \right)$$

- Notice that $\frac{\partial E_{n,l}}{\partial z_{l,i}}$ is defined using $\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}}$.
  - This is a **recursive** definition. To compute the values for layer l, we use the values from the **next** layer (i.e., layer $l + 1$).
  - This is why the whole algorithm is called **backpropagation**.
    - We propagate computations from the output layer backwards towards the input layer.

Computing $\frac{\partial E_n}{\partial w_{l,i,j}}$ for Hidden Units

- From the previous slides, we have these formulas:

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial w_{l,i,j}}$$

$$= \frac{\partial E_{n,l}}{\partial z_{l,i}} * \left(z_{l,i} - t_{n,i}\right) * z_{l,i} * \left(1 - z_{l,i}\right) * z_{l-1,j}$$

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * z_{l+1,k} * \left(1 - z_{l+1,k}\right) * w_{l+1,k,i} \right)$$

- We can combine these formulas, to compute $\frac{\partial E_n}{\partial w_{l,i,j}}$ for any weight of any hidden unit.

Computing $\frac{\partial E_n}{\partial b_{l,i}}$ for Hidden Units

- The formula for $\frac{\partial E_n}{\partial b_{l,i}}$ is similar, we just replace $\frac{\partial a_{l,i}}{\partial w_{l,i,j}}$ with $\frac{\partial a_{l,i}}{\partial b_{l,i}}$.

$$\frac{\partial E_n}{\partial b_{l,i}} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}} * \frac{\partial a_{l,i}}{\partial b_{l,i}}$$

$$= \frac{\partial E_{n,l}}{\partial z_{l,i}} * \left(z_{l,i} - t_{n,i}\right) * z_{l,i} * \left(1 - z_{l,i}\right) * 1$$

$$\frac{\partial E_{n,l}}{\partial z_{l,i}} = \sum_{k=1}^{J_{l+1}} \left(\frac{\partial E_{n,l+1}}{\partial z_{l+1,k}} * z_{l+1,k} * \left(1 - z_{l+1,k}\right) * w_{l+1,k,i} \right)$$

Simplifying Notation

- The previous formulas are sufficient and will work, but look complicated.
- We can simplify the formulas considerably, by defining:

$$\delta_{l,i} = \frac{\partial E_{n,l}}{\partial z_{l,i}} * \frac{\partial z_{l,i}}{\partial a_{l,i}}$$

- Then, if we combine calculations we already did:
  - If $U_{l,i}$ is an output unit, then:

$$\delta_{l,i} = (z_{l,i} - t_{n,i}) * z_{l,i} * (1 - z_{l,i})$$

  - If $U_{l,i}$ is a hidden unit, then:

$$\delta_{l,i} = \left(\sum_{k=1}^{J_{l+1}} (\delta_{l+1,k} * w_{l+1,k,i}) \right) * z_{l,i} * (1 - z_{l,i})$$

Final Backpropagation Formulas

- Using the definition of $\delta_{l,i}$ from the previous slide, we finally get very simple formulas:

$$\frac{\partial E_n}{\partial w_{l,i,j}} = \delta_{l,i} * z_{l-1,j} \qquad \frac{\partial E_n}{\partial b_{l,i}} = \delta_{l,i}$$

- Therefore, given a training input $\boldsymbol{x}_n$, and given a positive learning rate η, weights $w_{l,i,j}$ and $b_{l,i}$ are updated as follows:

$$w_{l,i,j} = w_{l,i,j} - \eta * \delta_{l,i} * z_{l-1,j} \qquad b_{l,i} = b_{l,i} - \eta * \delta_{l,i}$$

Backpropagation for One Object Step 1: Initialize Input Layer

- We will now see how to apply backpropagation, step by step, in pseudocode style, for a single training object.
- **NOTE: IN THE PSEUDOCODE, ARRAY INDICES START AT 1, NOT 0.**
- First, given a training example $\boldsymbol{x}_n$, and its target output $\boldsymbol{t}_n$, we must initialize the input units:

// 2D array z will store, for every unit $U_{l,i}$, its output

- double $**\, Z$ = new double*$[L]$ // L is the number of layers
- Z[1] = new double[D] // D is the dimensionality of $\boldsymbol{x}_n$

// Update the input layer, set inputs equal to $\boldsymbol{x}_n$.

- For $I \; = 1$ to D:
  - $z[1][I] = x_{n,i}$ // $x_{n,i}$ is the i-th dimension of training input $\boldsymbol{x}_n$.

Backpropagation for One Object Step 2: Compute Outputs

// we create a 2D array a, which will store, for every
// unit $U_{l,i}$, the weighted sum of the inputs of $U_{l,i}$.

- double $**$ a = new double*[L]

// Update the rest of the layers:

- For $l = 2$ to L: // L is the number of layers
  - $a[l]$= new double[J_l] // J_l is the number of units in layer l
  - $z[l]$= new double[J_l]
  - For each unit $U_{l,i}$ in layer l:
    - $a[l][i] = b_{l,i} + \sum_{j=1}^{J_{l-1}} \left(w_{l,i,j} z[l-1][j] \right)$ // weighted sum
    - $z[l][i] = \sigma(a[l][i]) = \frac{1}{1 + e^{-a[l][i]}}$ // output of unit $U_{l,i}$

Backpropagation for One Object Step 3: Compute New δ Values

// array δ will store, for every unit $U_{l,i}$, value $\delta_{l,i}$.

- double ** δ = new double*$[L]$
- $\delta[L]$ = new double*$[K]$ // K is the number of classes

- For each output unit $U_{L,i}$:
  - $\delta[L][i] = \left(z[L][i] - t_{n,i}\right) * z[L][i] * (1 - z[L][i])$
- For $l = L - 1$ to 2: // MUST be in decreasing order of l
  - $\delta[l]$ = new double$[J_l]$ // J_l is the number of units in layer l
  - For each unit $U_{l,i}$ in layer l:
    - $\delta[l][i] = \left(\sum_{k=1}^{J_{l+1}} \left(\delta[l+1][k] * w_{l+1,k,i}\right)\right) * z[l][i] * (1 - z[l][i])$

Backpropagation for One Object Step 4: Update Weights

- For $l = 2$ to L: // Order does not matter here, we can go
  // from 2 to L or from L to 2.
  - For $i = 1$ to J_l:
    - $b_{l,i} = b_{l,i} - \eta * \delta[l][\mathrm{i}]$
    - For $j = 1$ to J_{l-1}:
      - $w_{l,i,j} = w_{l,i,j} - \eta * \delta[l][\mathrm{i}] * z[l-1][j]$

IMPORTANT: Do Step 3 before Step 4. Do NOT do steps 3 and 4 as a single loop.

- All δ values must be computed using the old values of weights.
- Then, all weights must be updated using the new δ values .

Backpropagation Summary

- Inputs:
  - N D-dimensional training objects $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$.
  - The associated target values $\boldsymbol{t}_1, \dots, \boldsymbol{t}_N$, which are K-dimensional vectors.

1. Initialize weights $b_{l,i}$ and $w_{l,i,j}$ to small random numbers.
   - For example, set each $b_{l,i}$ and $w_{l,i,j}$ to a value between -0.1 and 0.1.
2. last_loss = $E(\boldsymbol{b}, \boldsymbol{w})$ // sum over all training examples
3. For $n = 1$ to N:
   - Given $\boldsymbol{x}_n$, update weights $b_{l,i}$ and $w_{l,i,j}$ as described in the previous slides.
4. loss = $E(\boldsymbol{b}, \boldsymbol{w})$ // sum over all training examples
5. If |loss – last_loss| < threshold, **exit**. // threshold can be 0.00001.
6. Else:
   7. last_loss = loss
   8. go to step 3.

Classification with Neural Networks

- Suppose we have K classes $C_1, \ldots, C_K$, where $K > 2$.
- Each class C_k corresponds to an output unit $U_{L,k}$.
- As we said before, for training we need to convert each class label to the appropriate one-hot vector.
- At inference time, given a pattern $\boldsymbol{x}$ to classify:
  – Compute outputs for all units of the network, working from the input layer towards the output layer.
  – Find the output unit $U_{L,k}$ with the highest output $z_{L,k}$.
  – Return class C_k.

Structure of Neural Networks

- Backpropagation describes how to learn weights.
- However, it does not describe how to learn the structure:
  - How many layers?
  - How many units at each layer?
- These are parameters that we have to choose somehow.
- A good way to choose such parameters is by using a validation set, containing examples and their class labels.
  - The validation set should be separate (disjoint) from the training set.

Structure of Neural Networks

- To choose the best structure for a neural network using a validation set, we try many different parameters (number of layers, number of units per layer).
- For each choice of parameters:
  - We train several neural networks using backpropagation.
  - We measure how well each neural network classifies the validation examples.
  - Why not train just one neural network?

Structure of Neural Networks

- To choose the best structure for a neural network using a validation set, we try many different parameters (number of layers, number of units per layer).
- For each choice of parameters:
  - We train several neural networks using backpropagation.
  - We measure how well each neural network classifies the validation examples.
  - Why not train just one neural network?
  - Each network is randomly initialized, so after backpropagation it can end up being different from the other networks.
- At the end, we select the neural network that did best on the validation set.