Neural Networks – Part 3

- Training Perceptrons
- Handling Multiclass Problems

CSE 4311 – Neural Networks and Deep Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

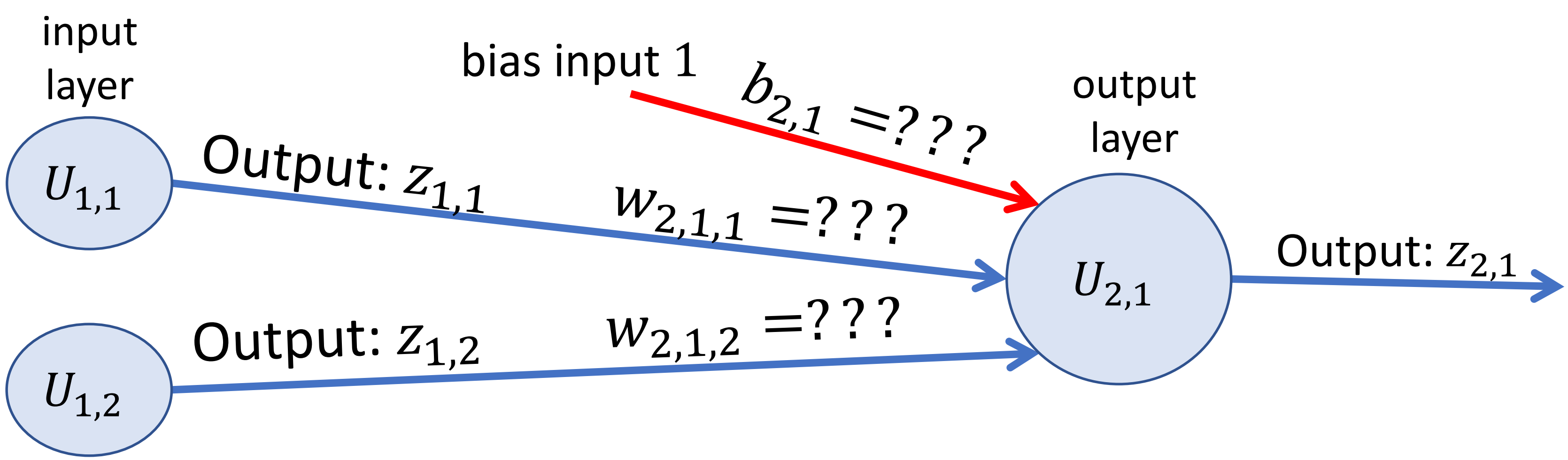
Training as an Optimization Problem

- Training a neural network is an optimization problem.
- In an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
  - Define a quantitative **optimization criterion**.
    - For any choice of parameters, this criterion will measure will tell us how good they are.
    - If we have two different choices, this criterion will tell us which choice one is better.
  - Define an **optimization algorithm** for finding a good set of parameters.

Parameters We Optimize

- In a neural network, what parameters are we optimizing?

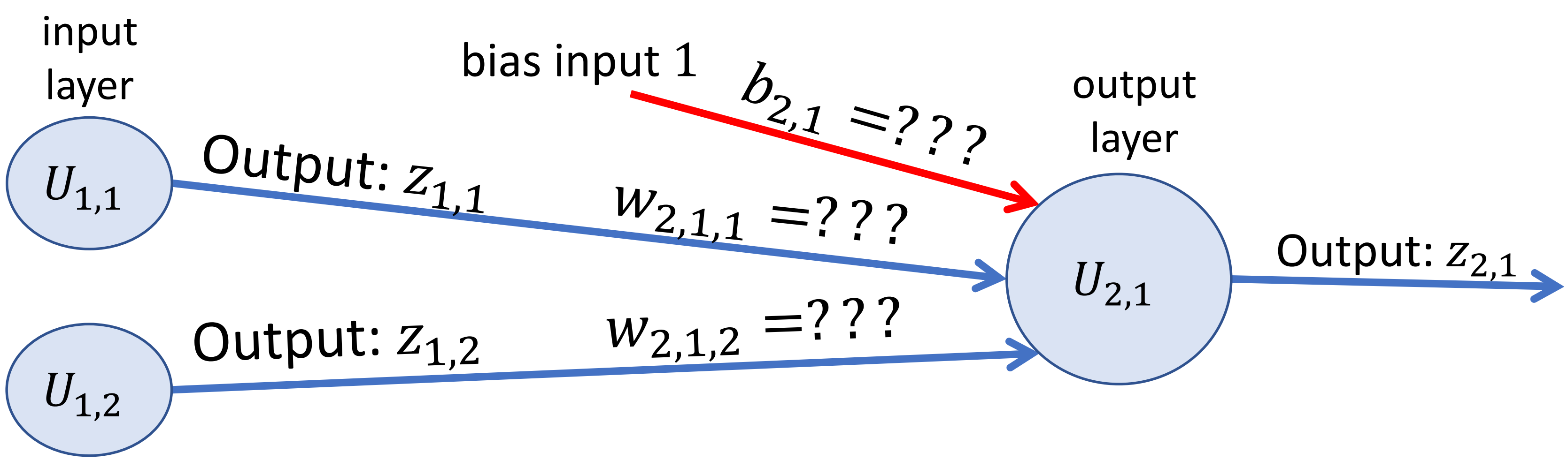
$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 0$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 0$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Parameters We Optimize

- In a neural network, what parameters are we optimizing?
  - Bias weights b and regular weights w.
  - In our toy example, this gives us three values that we have to optimize: $b_{2,1}$, $w_{2,1,1}$, $w_{2,1,2}$.

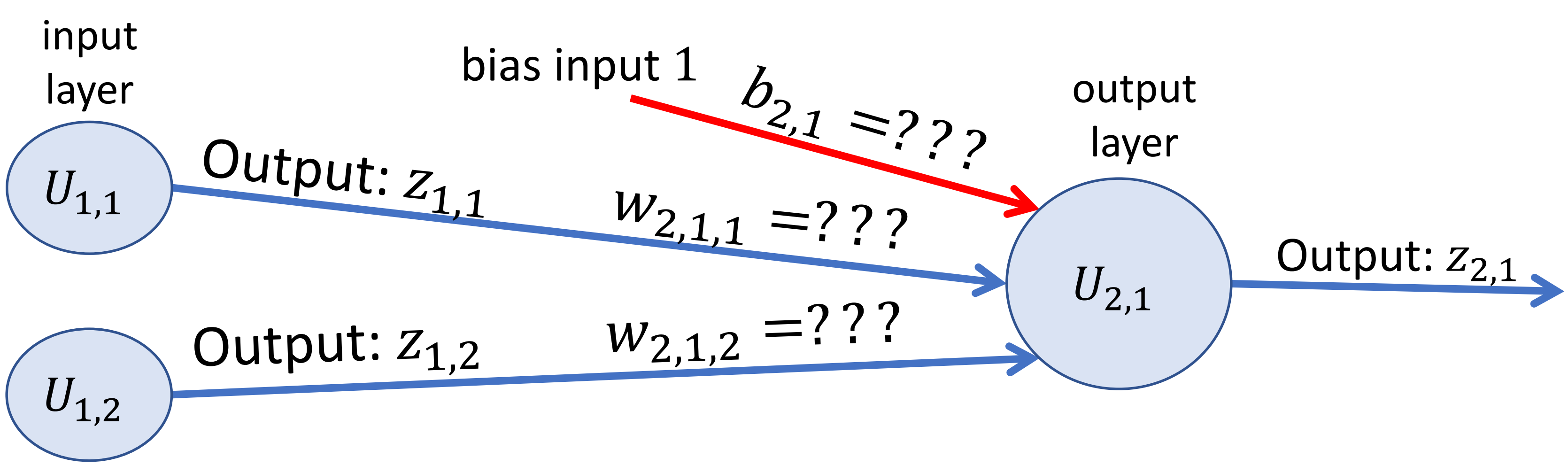
$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 0$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 0$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Optimization Criterion

- Suppose that we are considering some values for $b_{2,1}, w_{2,1,1}, w_{2,1,2}$.
- What quantitative criterion can we use to measure how good (or bad) those values are?
  - One commonly used measure: sum of squared differences.

$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 0$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 0$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Squared Differences

- A neural network defines a mathematical function $f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x})$:
  - $\boldsymbol{b}$, a list that specifies all the bias weights in the network.
  - $\boldsymbol{w}$, a list that specifies all other weights (non-bias weights) in the network.
  - $\boldsymbol{x}$, the vector that is given as input to the network.
- For any training example $\boldsymbol{x}_n$, we define the **loss** $E_n(b, \boldsymbol{w})$ as:

$$E_n(\boldsymbol{b}, \boldsymbol{w}) = \frac{1}{2}\left(f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x}_n) - t_n\right)^2$$

- $E_n(b, \boldsymbol{w})$ is the **squared difference** between the output of the neural network and the target output, multiplied (for reasons of convenience, explained later) by $\frac{1}{2}$.

Sum of Squared Differences

- The **loss** E over the entire training set is defined as:

$$E(\boldsymbol{b},\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{b},\boldsymbol{w}) = \sum_{n=1}^{N} \left[\frac{1}{2}\left(f(\boldsymbol{b},\boldsymbol{w},\boldsymbol{x}_n) - t_n\right)^2\right]$$

- This is called the **sum of squared differences (SSD) loss function**.
  - We simply sum up, over all training examples, the squared difference (squared error) that we get for each example.
- Note that $E(\boldsymbol{b},\boldsymbol{w})$ is a function of network parameters $\boldsymbol{b}$ and $\boldsymbol{w}$.
  - Different choices of $\boldsymbol{b}$ and $\boldsymbol{w}$ give a different loss value $E(\boldsymbol{b},\boldsymbol{w})$.
  - Our training goal is to find values of $\boldsymbol{b}$ and $\boldsymbol{w}$ that **minimize** loss $E(\boldsymbol{b},\boldsymbol{w})$.
  - Finding a **global minimum** is too slow, so we look for a **local minimum**.

Training as an Optimization Problem

- As we said, in an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
    - For neural networks, what is that?
  - Define a quantitative **optimization criterion**.
    - For neural networks, what is that?
  - Define an **optimization algorithm** for finding a good set of parameters.
    - For neural networks, what is that?

Training as an Optimization Problem

- As we said, in an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
    - For neural networks, we search over $\boldsymbol{b}$ and $\boldsymbol{w}$.
  - Define a quantitative **optimization criterion**.
    - For neural networks, we defined the SSD loss function, which we want to minimize. We will also see and use other choices this semester.
  - Define an **optimization algorithm** for finding a good set of parameters.
    - **Gradient descent**. When used specifically for training neural networks, it is called **backpropagation**.

Perceptron Learning

- Suppose that a perceptron is using the step function as its activation function h.

$$h(a) = \begin{cases} 0, \text{if } a < 0 \\ 1, \text{if } a \geq 0 \end{cases}$$

$$z(\boldsymbol{x}) = h(b + \boldsymbol{w}^T \boldsymbol{x}) = \begin{cases} 0, \text{if } b + \boldsymbol{w}^T \boldsymbol{x} < 0 \\ 1, \text{if } b + \boldsymbol{w}^T \boldsymbol{x} \geq 0 \end{cases}$$

- Can we apply gradient descent in that case?

Perceptron Learning

- Suppose that a perceptron is using the step function as its activation function h.

$$h(a) = \begin{cases} 0, \text{if } a < 0 \\ 1, \text{if } a \geq 0 \end{cases}$$

$$z(\boldsymbol{x}) = h(b + \boldsymbol{w}^T \boldsymbol{x}) = \begin{cases} 0, \text{if } \ b + \boldsymbol{w}^T \boldsymbol{x} < 0 \\ 1, \text{if } b + \boldsymbol{w}^T \boldsymbol{x} \geq 0 \end{cases}$$

- Can we apply gradient descent in that case?
- No, because $E(b, \boldsymbol{w})$ gives gradients of 0.
  - This is because the derivative of the step function is zero everywhere, except at 0 where it is not continuous.
- This means that we never update the initial point that we start the gradient descent from.

Perceptron Learning

- A better option is setting h to the sigmoid function:

$$z(\boldsymbol{x}) = h(b + \boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1 + e^{-b-\boldsymbol{w}^T\boldsymbol{x}}}$$

- Then, measured just on a single training object $\boldsymbol{x}_n$, the loss $E_n(b, \boldsymbol{w})$ is defined as:

$$E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1 + e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$$

- Reminder: if our neural network is a single perceptron, then the target output t_n **<u>must be</u>** one-dimensional. These formulas, so far, deal only with training a single perceptron.

Computing the Gradient

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$
- In this form, $E_n(b, \boldsymbol{w})$ is differentiable.
- We want to update b and $\boldsymbol{w}$ using gradient descent.
- Therefore, we have to take these steps:
  - Compute $\frac{\partial E_n}{\partial b}$
  - Compute $\frac{\partial E_n}{\partial \boldsymbol{w}}$
  - Change b and $\boldsymbol{w}$ in the direction opposite to the gradient:

$$b = b - \eta \frac{\partial E_n}{\partial b}, \qquad \boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial E_n}{\partial \boldsymbol{w}}$$

Updates, One Example at a Time

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$
- We update b and $\boldsymbol{w}$ based on the gradients:

  $b = b - \eta \frac{\partial E_n}{\partial b}$, $\boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial E_n}{\partial \boldsymbol{w}}$
- Note: the update formulas are based on E_n (the loss corresponding to the n-th training example).
  - This means that we compute gradients and update b and $\boldsymbol{w}$ separately for each training example.
  - Overall, we loop over all training examples, and for each example we update b and $\boldsymbol{w}$ using those formulas.

Batch Processing Preview

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- We update b and $\boldsymbol{w}$ based on the gradients:

$$b = b - \eta \frac{\partial E_n}{\partial b}, \qquad \boldsymbol{w} = \boldsymbol{w} - \eta \frac{\partial E_n}{\partial \boldsymbol{w}}$$

- A more common approach is to compute the updates to b and $\boldsymbol{w}$ using multiple training examples simultaneously.
  - That is called "batch processing", and those multiple training examples are called a "batch".
  - For now, to keep things simple, each batch is a single example. Later we will see how to generalize this.

Chain Rule for Derivatives

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$
- So, to do gradient descent, we need to compute the gradients $\frac{\partial E_n}{\partial b}$ and $\frac{\partial E_n}{\partial \boldsymbol{w}}$.
- We start with $\frac{\partial E_n}{\partial b}$, which is more simple, since b is a scalar.
- E_n is a complicated formula, its derivative is not obvious.
- In such cases, the chain rule can be used:
- Strategy:
  - Write E_n as a composition of simple functions, whose derivatives is obvious.
  - Use the chain rule to compute the gradients of E_n.

A Note on the Chain Rule

- I have seen the chain rule defined in two different (but equivalent) ways:

1. $(f \circ g)'(x) = f'\big(g(x)\big) * g'(x)$

2. $\frac{\partial (f \circ g)}{\partial x} = \frac{\partial f}{\partial g} * \frac{\partial g}{\partial x}$

- I have always found the second one easier to remember and use.
  - This is the version we use in these slides.

Decomposing E_n

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$
- This is one possible decomposition that works:
  - $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T\boldsymbol{x}_n$
  - $f_2(f_1) = 1 + e^{f_1} = 1 + e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}$
  - $f_3(f_2) = \frac{1}{f_2} = \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}$
  - $f_4(f_3) = t_n - f_3 = t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}$
  - $f_5(f_4) = \frac{1}{2}(f_4)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$
- $E_n(b, \boldsymbol{w}) = f_5\left(f_4\left(f_3(f_2)\left(f_1(b, \boldsymbol{w})\right)\right)\right)$
- Usually we write this as: $E_n = f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T x_n}}\right)^2$
- $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T \boldsymbol{x}_n$
- $f_2(f_1) = 1 + e^{f_1}$
- $f_3(f_2) = \frac{1}{f_2}$
- $f_4(f_3) = t_n - f_3$
- $f_5(f_4) = \frac{1}{2}(f_4)^2$
- $E_n = f_5 \circ f_4 \circ f_3 \circ f_2 \circ f_1$
- Then, according to the chain rule:

$$\frac{\partial E_n}{\partial b} = \frac{\partial f_5}{\partial f_4} * \frac{\partial f_4}{\partial f_3} * \frac{\partial f_3}{\partial f_2} * \frac{\partial f_2}{\partial f_1} * \frac{\partial f_1}{\partial b}$$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T\boldsymbol{x}_n$

- $\frac{\partial f_1}{\partial b} = ???$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T\boldsymbol{x}_n$

- $\frac{\partial f_1}{\partial b} = -1$

- Why? Based on the rule that $\frac{\partial(-x-c)}{\partial x} = -1$
- Note that when we compute $\frac{\partial f_1}{\partial b}$, the term $\boldsymbol{w}^T\boldsymbol{x}_n$ is treated as a constant c, since it does not depend on b.

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right)^2$

- $f_2(f_1) = 1 + e^{f_1}$

- $\frac{\partial f_2}{\partial f_1} = ???$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right)^2$

- $f_2(f_1) = 1 + e^{f_1}$

- $\frac{\partial f_2}{\partial f_1} = e^{f_1}$

- Why? Based on the rule that $\frac{\partial(e^x)}{\partial x} = e^x$.

- Again, note that $\frac{\partial(1+e^x)}{\partial x} = \frac{\partial(e^x)}{\partial x}$, since 1 is a constant.

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_3(f_2) = \frac{1}{f_2}$

- $\frac{\partial f_3}{\partial f_2} = ???$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right)^2$

- $f_3(f_2) = \frac{1}{f_2}$

- $\frac{\partial f_3}{\partial f_2} = -\frac{1}{(f_2)^2}$

- Why? $\frac{1}{f_2} = (f_2)^{-1}$, and $\frac{\partial (x^n)}{\partial x} = nx^{n-1}$.

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_4(f_3) = t_n - f_3$

- $\frac{\partial f_4}{\partial f_3} = ???$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_4(f_3) = t_n - f_3$

- $\frac{\partial f_4}{\partial f_3} = -1$

- Why? $\frac{\partial(c-x)}{\partial x} = -1.$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right)^2$

- $f_5(f_4) = \frac{1}{2}(f_4)^2$

- $\frac{\partial f_5}{\partial f_4} = ???$

Using the Chain Rule

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T\boldsymbol{x}_n}}\right)^2$

- $f_5(f_4) = \frac{1}{2}(f_4)^2$

- $\frac{\partial f_5}{\partial f_4} = f_4$

- Why? $\frac{\partial(\frac{1}{2}x^2)}{\partial x} = \frac{1}{2} * 2x = x$

Combining the Results

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-w^T x_n}}\right)^2$

- $\frac{\partial E_n}{\partial b} = \frac{\partial f_5}{\partial f_4} * \frac{\partial f_4}{\partial f_3} * \frac{\partial f_3}{\partial f_2} * \frac{\partial f_2}{\partial f_1} * \frac{\partial f_1}{\partial b}$

- $\frac{\partial E_n}{\partial b} = f_4 * (-1) * \left(-\frac{1}{(f_2)^2}\right) * e^{f_1} * (-1)$

- Simplifying: $\frac{\partial E_n}{\partial b} = -\frac{f_4 * e^{f_1}}{(f_2)^2}$

Combining the Results

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-w^T x_n}}\right)^2$

- $\frac{\partial E_n}{\partial b} = \frac{\partial f_5}{\partial f_4} * \frac{\partial f_4}{\partial f_3} * \frac{\partial f_3}{\partial f_2} * \frac{\partial f_2}{\partial f_1} * \frac{\partial f_1}{\partial b} = -\frac{f_4 * e^{f_1}}{(f_2)^2}$

- We want a formula in terms of our original variables: $b, \boldsymbol{w}, t_n, \boldsymbol{x}_n$
- So, we need to write out f_1, f_2, f_4 as functions of those variables.

Combining the Results

- $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T \boldsymbol{x}_n$
- $f_2(f_1) = 1 + e^{f_1} = 1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}$
- $f_4(f_3) = t_n - f_3 = t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}$

$$\frac{\partial E_n}{\partial b} = -\frac{f_4 * e^{f_1}}{(f_2)^2}$$

$$= -\left(t_n - \frac{1}{1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \frac{1}{\left(1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right)^2}$$

We can write this in a more simple way, because the red part is just the perceptron output $z(\boldsymbol{x}_n)$.

Combining the Results

- $f_1(b, \boldsymbol{w}) = -b - \boldsymbol{w}^T \boldsymbol{x}_n$
- $f_2(f_1) = 1 + e^{f_1} = 1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}$
- $f_4(f_3) = t_n - f_3 = t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}$

$$\frac{\partial E_n}{\partial b} = -\frac{f_4 * e^{f_1}}{(f_2)^2}$$

$$= -\left(t_n - \frac{1}{1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \frac{1}{\left(1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right)^2}$$

We can write this in a more simple way, because the red part is just the perceptron output $z(\boldsymbol{x}_n)$.

$$= -(t_n - z(\boldsymbol{x}_n)) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \left(z(\boldsymbol{x}_n)\right)^2$$

$$= (z(\boldsymbol{x}_n) - t_n) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \left(z(\boldsymbol{x}_n)\right)^2$$

Combining the Results

- What we have so far:
- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}}\right)^2$
- $\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \left(z(\boldsymbol{x}_n)\right)^2$
- This formula for $\frac{\partial E_n}{\partial b}$ is good enough, we know everything we need to know $(b, \boldsymbol{w}, t_n, \boldsymbol{x}_n, z(\boldsymbol{x}_n))$, to compute it.
- We simplify the part in red a bit more, by noting that:

$1 + e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n} = \frac{1}{z(\boldsymbol{x}_n)}$, and therefore:

$$e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n} = \frac{1}{z(\boldsymbol{x}_n)} - 1 = \frac{1-z(\boldsymbol{x}_n)}{z(\boldsymbol{x}_n)}$$

Final Formula for $\frac{\partial E_n}{\partial b}$

- What we have so far:
- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-\boldsymbol{w}^T x_n}}\right)^2$
- $\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * \left(e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}\right) * \left(z(\boldsymbol{x}_n)\right)^2$
- Substituting $\frac{1-z(\boldsymbol{x}_n)}{z(\boldsymbol{x}_n)}$ for $e^{-b-\boldsymbol{w}^T \boldsymbol{x}_n}$ we get:

$\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * \left(\frac{1-z(\boldsymbol{x}_n)}{z(\boldsymbol{x}_n)}\right) * \left(z(\boldsymbol{x}_n)\right)^2$, and finally:

$$\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * \left(1 - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n)$$

Gradients

- $E_n(b, \boldsymbol{w}) = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\left(t_n - \frac{1}{1+e^{-b-w^T x_n}}\right)^2$

- As we have seen: $\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * \left(1 - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n)$

- With a similar derivation (which we skip), we can show that:

$$\frac{\partial E_n}{\partial \boldsymbol{w}} = (z(\boldsymbol{x}_n) - t_n) * \left(1 - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \boldsymbol{x}_n$$

- Note that $\frac{\partial E_n}{\partial \boldsymbol{w}}$ is a D-dimensional vector. It is a scalar (shown in red) multiplied by vector $\boldsymbol{x}_n$.

Weight Update

$$\frac{\partial E_n}{\partial b} = (z(\boldsymbol{x}_n) - t_n) * z(\boldsymbol{x}_n) * \big(1 - z(\boldsymbol{x}_n)\big)$$

$$\frac{\partial E_n}{\partial \boldsymbol{w}} = (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n) * \boldsymbol{x}_n$$

- So, we update the bias weight b and weight vector $\boldsymbol{w}$ as follows:

$$b = b - \eta * (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n)$$

$$\boldsymbol{w} = \boldsymbol{w} - \eta * (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n) * \boldsymbol{x}_n$$

Weight Update

- (From previous slide) Update formulas:

$$b = b - \eta * (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n)$$

$$\boldsymbol{w} = \boldsymbol{w} - \eta * (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n) * \boldsymbol{x}_n$$

- As before, η is the learning rate parameter.
  - It is a positive real number that should be chosen carefully, so as not to be too big or too small.
- In terms of individual weights w_d, the update rule is:

$$w_d = w_d - \eta * (z(\boldsymbol{x}_n) - t_n) * \big(1 - z(\boldsymbol{x}_n)\big) * z(\boldsymbol{x}_n) * x_{n,d}$$

Perceptron Learning - Summary

- Input: Training inputs $\boldsymbol{x}_1, , \ldots , \boldsymbol{x}_N$, target outputs $t_1, \ldots , t_N$
  - For a binary classification problem, each t_n is set to 0 or 1.

1. Initialize b and each w_d to small random numbers.
   - For example, set b and each w_d to a random value between -0.1 and 0.1
2. For n = 1 to N:
   a) Compute $z(x_n)$.
   b) $b = b - \eta * (z(\boldsymbol{x}_n) - t_n) * \left(1 - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n)$
   c) For d = 0 to D:
   $$w_d = w_d - \eta * (z(\boldsymbol{x}_n) - t_n) * \left(1 - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * x_{n,d}$$
3. If some stopping criterion has been met, **exit**.
4. Else, go to step 2.

Stopping Criterion

- At step 3 of the perceptron learning algorithm, we need to decide whether to stop or not.
- One thing we can do is:
  - Compute the SSD (sum of squared differences) loss $E(b, \boldsymbol{w})$ of the perceptron at that point over the entire training set.

$$E(b, \boldsymbol{w}) = \sum_{n=1}^{N} E_n(b, \boldsymbol{w}) = \sum_{n=1}^{N} \left\{ \frac{1}{2} \left(t_n - z(\boldsymbol{x}_n) \right)^2 \right\}$$

  - Compare the current value of $E(b, \boldsymbol{w})$ with the value of $E(b, \boldsymbol{w})$ computed at the previous iteration.
  - If the difference is too small (e.g., smaller than 0.00001) we stop.

Notation for Multiclass Training Set

- We have a set X of N training examples.
  - $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$
- Each $\boldsymbol{x}_n$ is a D-dimensional column vector.
  - $\boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \dots, x_{n,D})'$
- We also have a set T of N target outputs.
  - $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \dots, \boldsymbol{t}_N\}$
  - $\boldsymbol{t}_n$ is the target output for training example $\boldsymbol{x}_n$.
- Each $\boldsymbol{t}_n$ is a K-dimensional column vector:
  - $\boldsymbol{t}_n = (t_{n,1}, t_{n,2}, \dots, t_{n,K})'$
- Note: K typically is not equal to D.
  - In your assignment, K is equal to the number of classes.
  - K is also equal to the number of units in the output layer.

Using Perceptrons for Multiclass Problems

- “Multiclass” means that we have more than two classes.
- A perceptron outputs a number between 0 and 1.
- This is sufficient only for binary classification problems.
- For more than two classes, there are many different options.
- We will follow a general approach called **one-versus-all classification** (also known as OVA classification).
  - This approach is a general method, that can be combined with various binary classification methods, so as to solve multiclass problems. Here we see the method applied to perceptrons.

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 $\boldsymbol{x}_n$ is a five-dimensional vector.
  - The class labels q_n are strings.

Converting to One-Versus-All

- Training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T$, $q_1 = \text{dog}$, $s_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T$, $q_2 = \text{dog}$, $s_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T$, $q_3 = \text{cat}$, $s_3 = 2$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T$, $q_4 = \text{fox}$, $s_4 = 3$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T$, $q_5 = \text{cat}$, $s_5 = 2$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T$, $q_6 = \text{fox}$, $s_6 = 3$
- Step 1:
  - Generate new class labels s_n, where classes are numbered sequentially starting from 1.
    - Thus, in our example, the class labels become 1, 2, 3.

Converting to One-Versus-All

- Training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad s_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad s_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad s_3 = 2$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad s_4 = 3$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad s_5 = 2$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad s_6 = 3$
- Step 2: Convert each label s_n to a **one-hot vector** $\boldsymbol{t}_n$.
  - Vector $\boldsymbol{t}_n$ has as many dimensions as the number of classes.
    - How many dimensions should we use in our example?

Converting to One-Versus-All

- Training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T$, $s_1 = 1$ $\boldsymbol{t}_1 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T$, $s_2 = 1$ $\boldsymbol{t}_2 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T$, $s_3 = 2$ $\boldsymbol{t}_3 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T$, $s_4 = 3$ $\boldsymbol{t}_4 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T$, $s_5 = 2$ $\boldsymbol{t}_5 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T$, $s_6 = 3$ $\boldsymbol{t}_6 = (?\,,?\,,?\,)^T$
- Step 2: Convert each label s_n to a **one-hot vector** $\boldsymbol{t}_n$.
  - Vector $\boldsymbol{t}_n$ has as many dimensions as the number of classes.
    - In our example we have three classes, so each $\boldsymbol{t}_n$ is 3-dimensional.
  - If $s_n = i$, then set the i-th dimension of t_n to 1.
  - Otherwise, set the i-th dimension of t_n to 0.

Converting to One-Versus-All

- Training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad s_1 = 1 \qquad \boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad s_2 = 1 \qquad \boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad s_3 = 2 \qquad \boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad s_4 = 3 \qquad \boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad s_5 = 2 \qquad \boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad s_6 = 3 \qquad \boldsymbol{t}_6 = (0, 0, 1)^T$
- Step 2: Convert each label s_n to a **one-hot vector** $\boldsymbol{t}_n$.
  - Vector $\boldsymbol{t}_n$ has as many dimensions as the number of classes.
    - In our example we have three classes, so each $\boldsymbol{t}_n$ is 3-dimensional.
  - If $s_n = i$, then set the i-th dimension of t_n to 1.
  - Otherwise, set the i-th dimension of t_n to 0.

Converting to One-Versus-All

- Training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad s_1 = 1 \qquad \boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad s_2 = 1 \qquad \boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad s_3 = 2 \qquad \boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad s_4 = 3 \qquad \boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad s_5 = 2 \qquad \boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad s_6 = 3 \qquad \boldsymbol{t}_6 = (0, 0, 1)^T$
- Step 3: Train three separate perceptrons (as many as the number of classes).
- For training the **first** perceptron, use the **first** dimension of each $\boldsymbol{t}_n$ as target output for $\boldsymbol{x}_n$.

Training Set for the First Perceptron

- Training set used to train the first perceptron:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad t_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad t_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad t_3 = 0$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad t_4 = 0$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad t_5 = 0$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad t_6 = 0$
- Essentially, the first perceptron is trained to output "1" when:
  - The original class label q_n is "dog".
  - The sequentially numbered class label s_n is 1.

Converting to One-Versus-All

- Training set for the multiclass problem:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad s_1 = 1 \qquad \boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad s_2 = 1 \qquad \boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad s_3 = 2 \qquad \boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad s_4 = 3 \qquad \boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad s_5 = 2 \qquad \boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad s_6 = 3 \qquad \boldsymbol{t}_6 = (0, 0, 1)^T$
- Step 3: Train three separate perceptrons (as many as the number of classes).
- For training the **second** perceptron, use the **second** dimension of each t_n as target output for x_n.

Training Set for the Second Perceptron

- Training set used to train the second perceptron:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad t_1 = 0$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad t_2 = 0$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad t_3 = 1$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad t_4 = 0$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad t_5 = 1$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad t_6 = 0$
- Essentially, the second perceptron is trained to output "1" when:
  - The original class label q_n is "cat".
  - The sequentially numbered class label s_n is 2.

Converting to One-Versus-All

- Training set for the multiclass problem:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T$, $s_1 = 1$ $\boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T$, $s_2 = 1$ $\boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T$, $s_3 = 2$ $\boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T$, $s_4 = 3$ $\boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T$, $s_5 = 2$ $\boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T$, $s_6 = 3$ $\boldsymbol{t}_6 = (0, 0, 1)^T$
- Step 3: Train three separate perceptrons (as many as the number of classes).
- For training the **third** perceptron, use the **third** dimension of each t_n as target output for x_n.

Training Set for the Third Perceptron

- Training set used to train the third perceptron:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad t_1 = 0$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad t_2 = 0$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad t_3 = 0$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad t_4 = 1$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad t_5 = 0$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad t_6 = 1$
- Essentially, the third perceptron is trained to output “1” when:
  - The original class label q_n is “fox”.
  - The sequentially numbered class label s_n is 3.

One-Versus-All Perceptrons: Recap

- Suppose we have K classes $C_1, \ldots, C_K$, where $K > 2$.
- We have training inputs $\boldsymbol{x}_1, , \ldots, \boldsymbol{x}_N$, and target values $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_N$.
- Each target value $\boldsymbol{t}_n$ is a K-dimensional vector:
  - $\boldsymbol{t}_n = (t_{n,1}, t_{n,2}, \ldots, t_{n,K})$
  - $t_{n,k} = 0$ if the class of $\boldsymbol{x}_n$ is not C_k.
  - $t_{n,k} = 1$ if the class of $\boldsymbol{x}_n$ is C_k.
- For each class C_k, train a perceptron z_k by using $t_{n,k}$ as the target value for $\boldsymbol{x}_n$.
  - So, perceptron z_k is trained to recognize if an object belongs to class C_k or not.
  - In total, we train K perceptrons, one for each class.

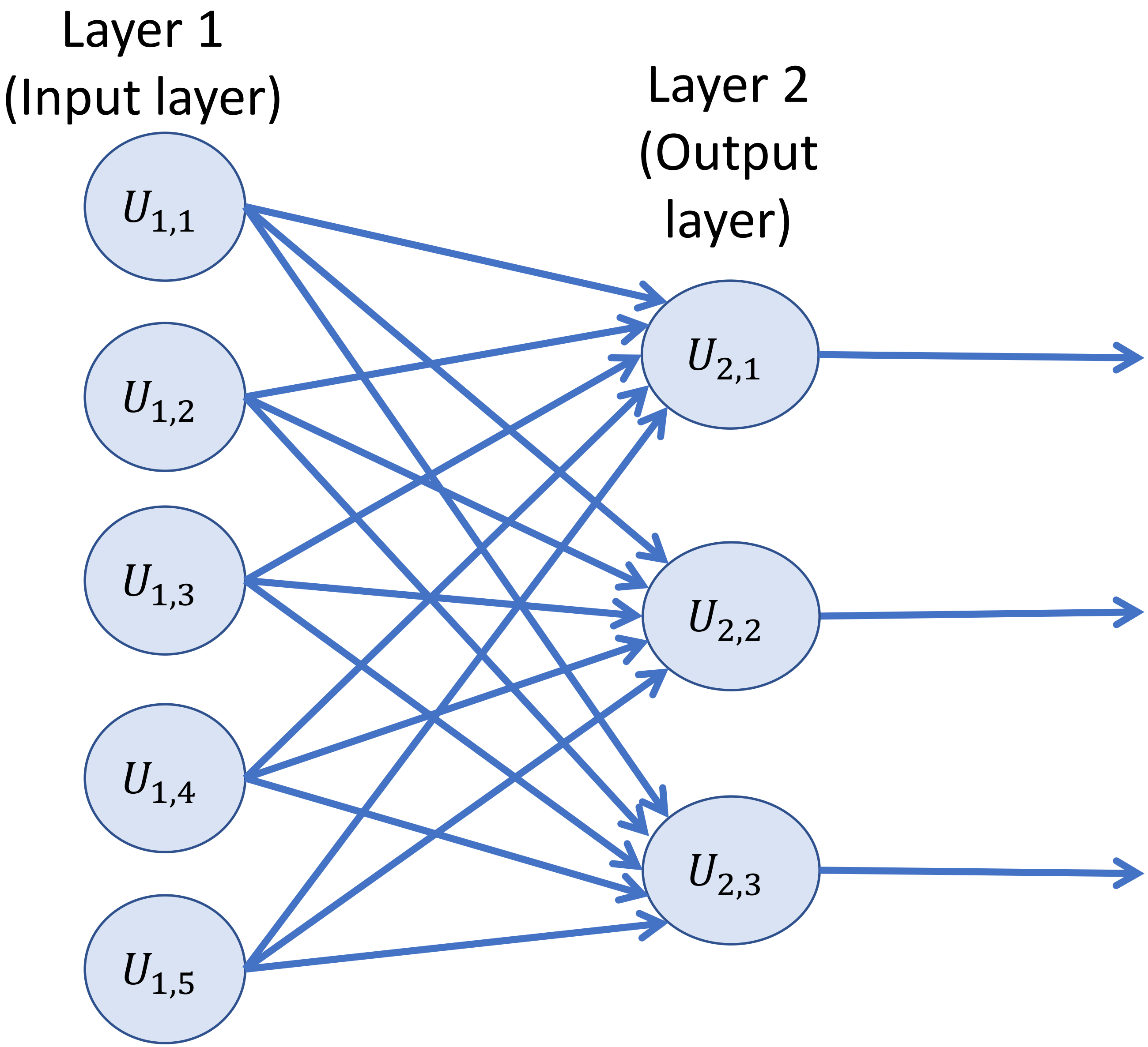
One-Versus-All Perceptrons

- At inference time, to classify an input pattern $\boldsymbol{x}$:
  - Compute the responses $z_k(\boldsymbol{x})$ for all K perceptrons.
  - Find the perceptron z_{k*} such that the value $z_{k*}(\boldsymbol{x})$ is higher than all other responses.
  - Output that the class of x is C_{k*}.
- In summary: we assign $\boldsymbol{x}$ to the class whose perceptron produced the highest output value for $\boldsymbol{x}$.

Multiclass Neural Networks

- For perceptrons, we saw that we can perform multiclass (i.e., for more than two classes) classification using the one-versus-all (OVA) approach:
  - We train one perceptron for each class.
- These multiple perceptrons can also be thought of as a **single neural network**.

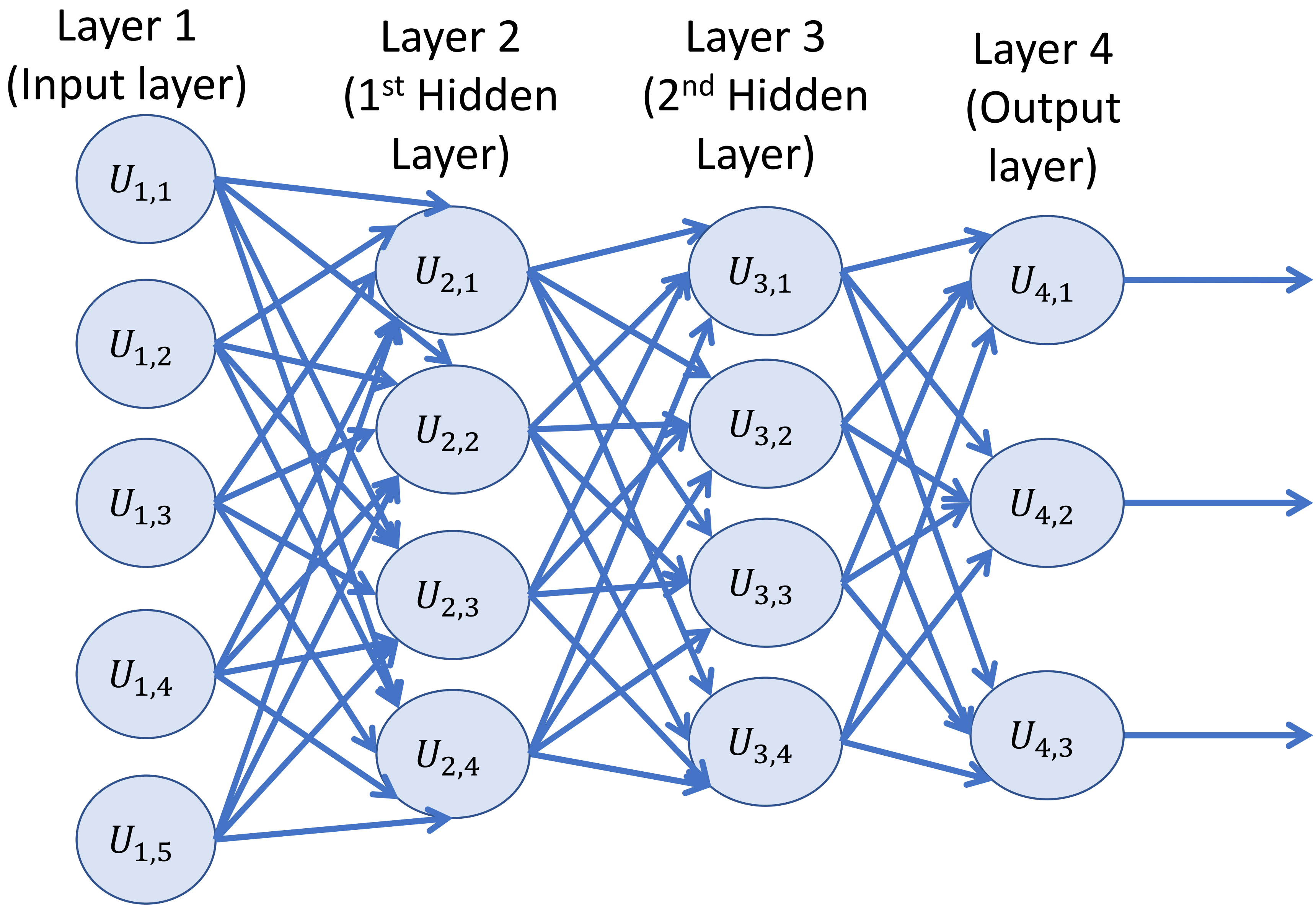
OVA Perceptrons as a Single Network



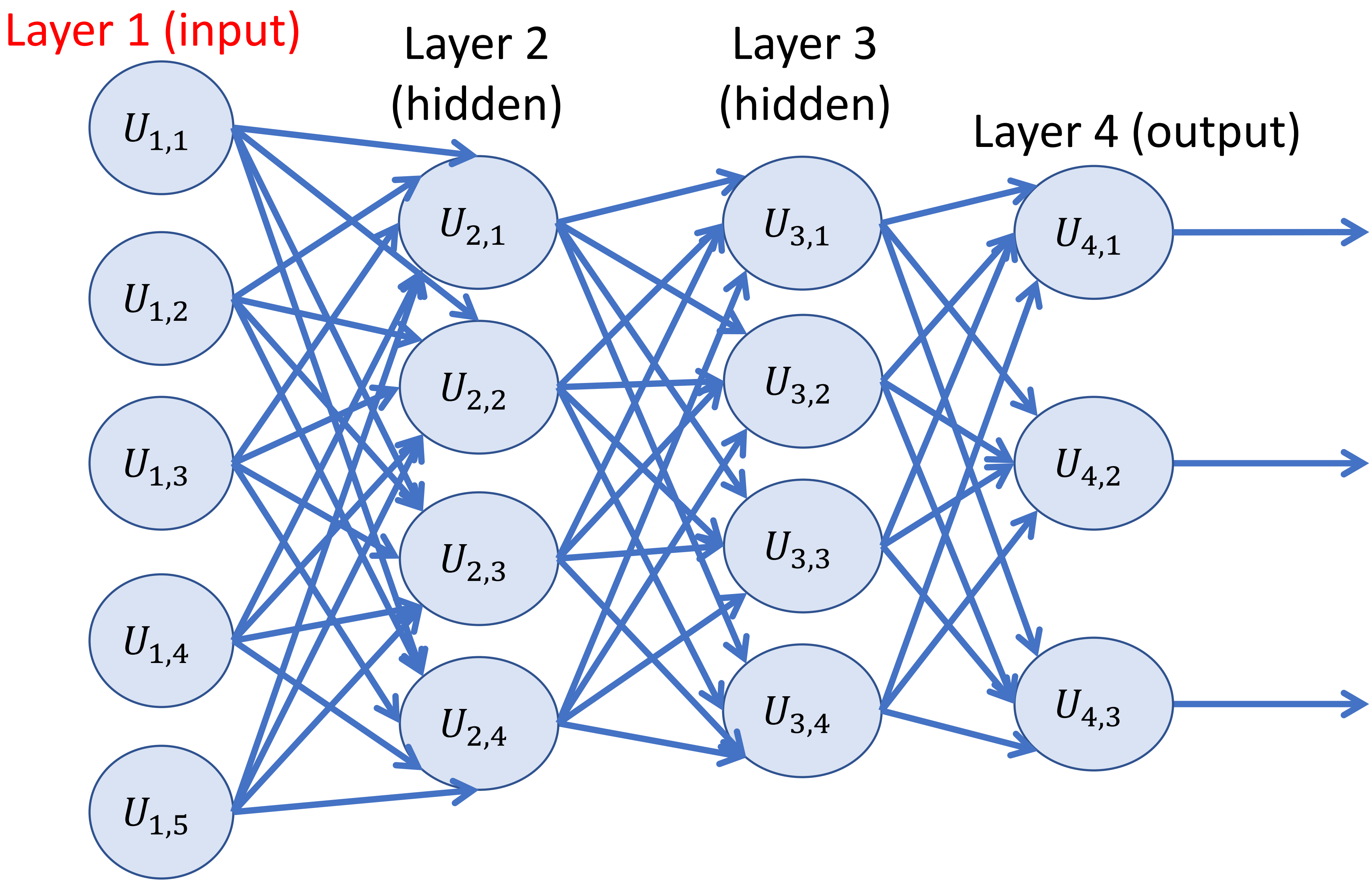
Multiclass Neural Networks

- For perceptrons, we saw that we can perform multiclass (i.e., for more than two classes) classification using the one-versus-all (OVA) approach:
  - We train one perceptron for each class.
- These multiple perceptrons can also be thought of as a **single neural network**.
- In the simplest case, a neural network designed to recognize multiple classes looks like the previous example.
- In the general case, there are also hidden layers.

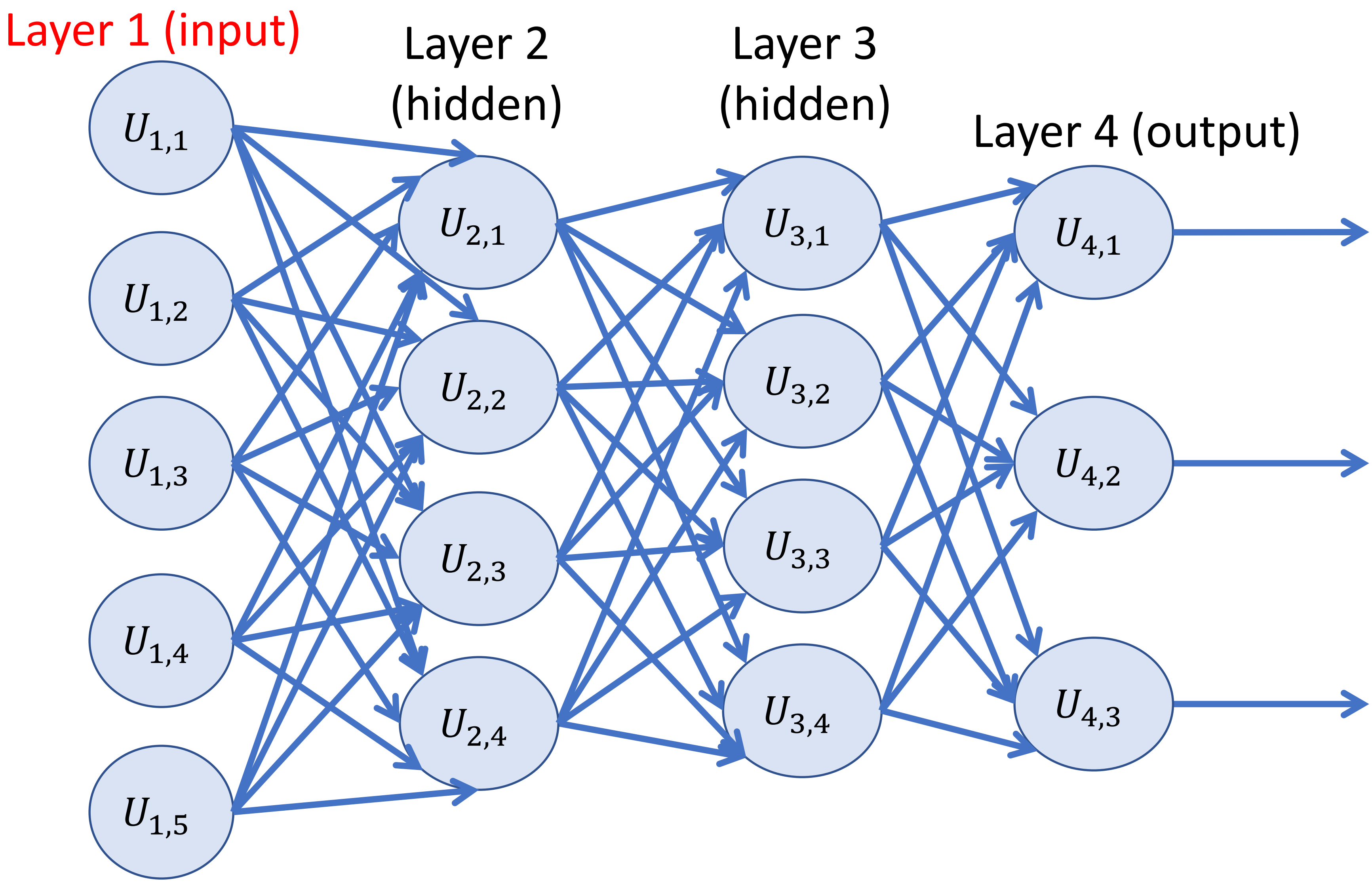
A Network for Our Example



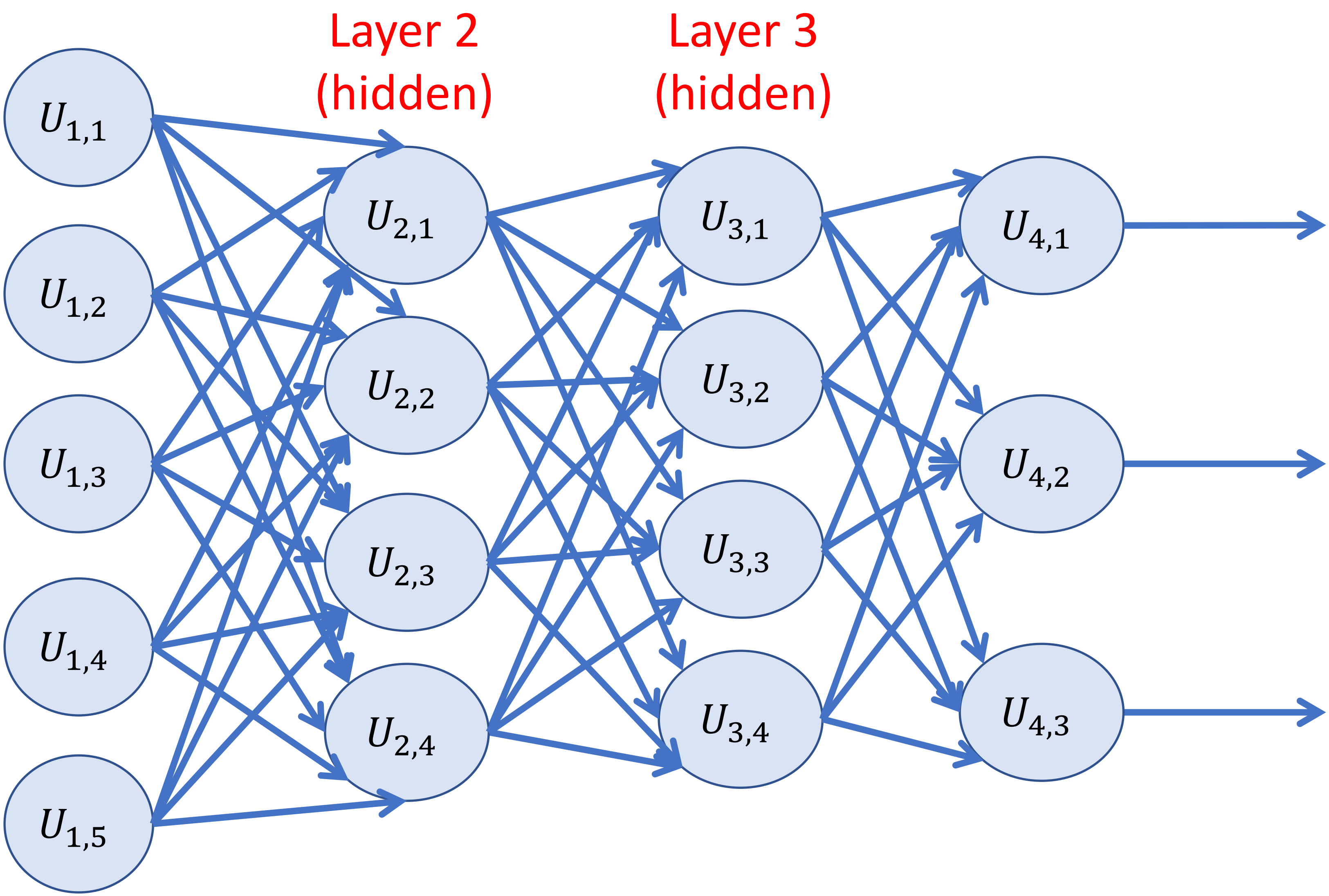
Input Layer: How many units does it have? Could we have a different number? Is the number of input units a hyperparameter?



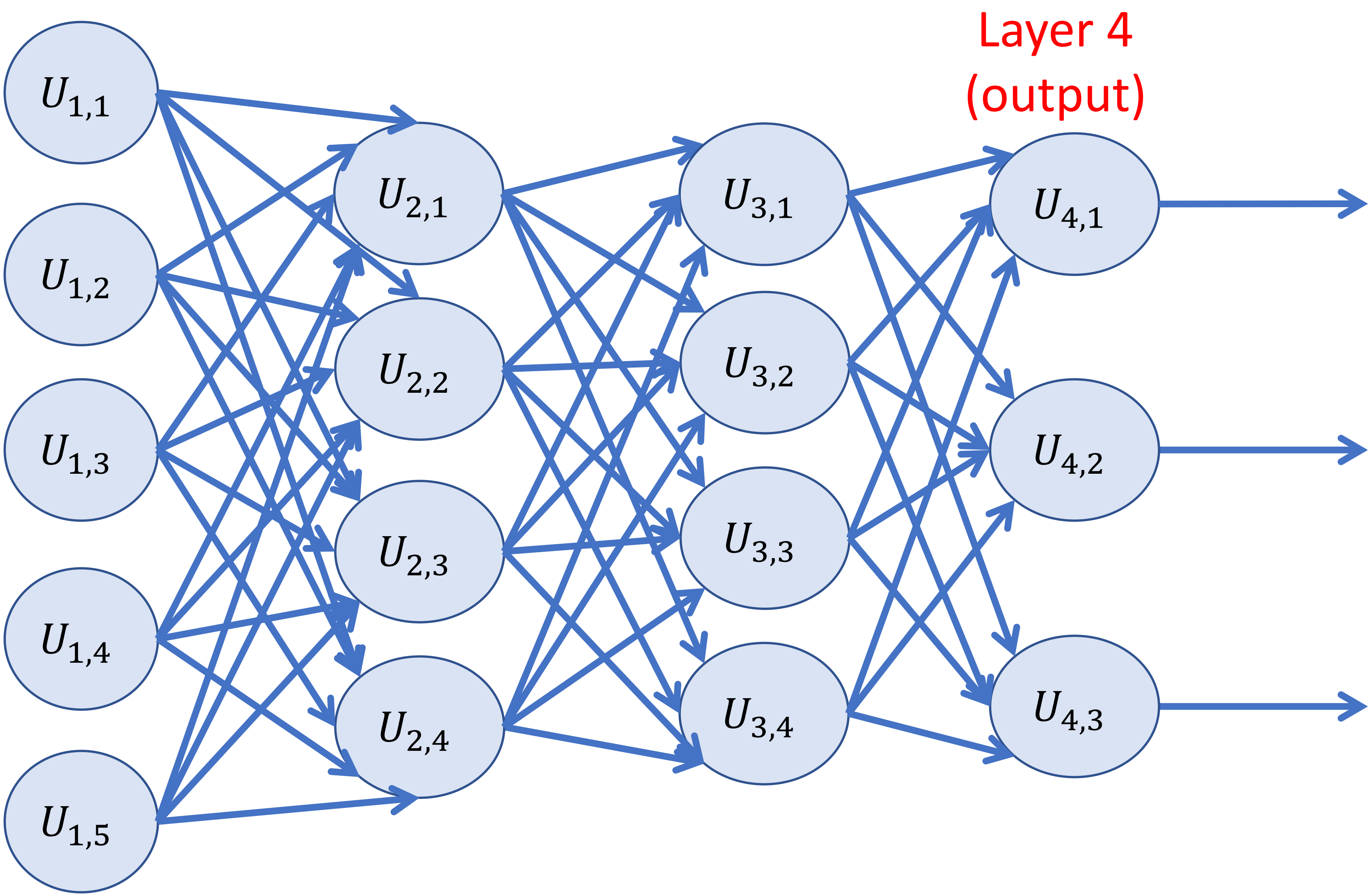
In our example, the input layer it **must** have five units, because each input is five-dimensional. We don't have a choice.



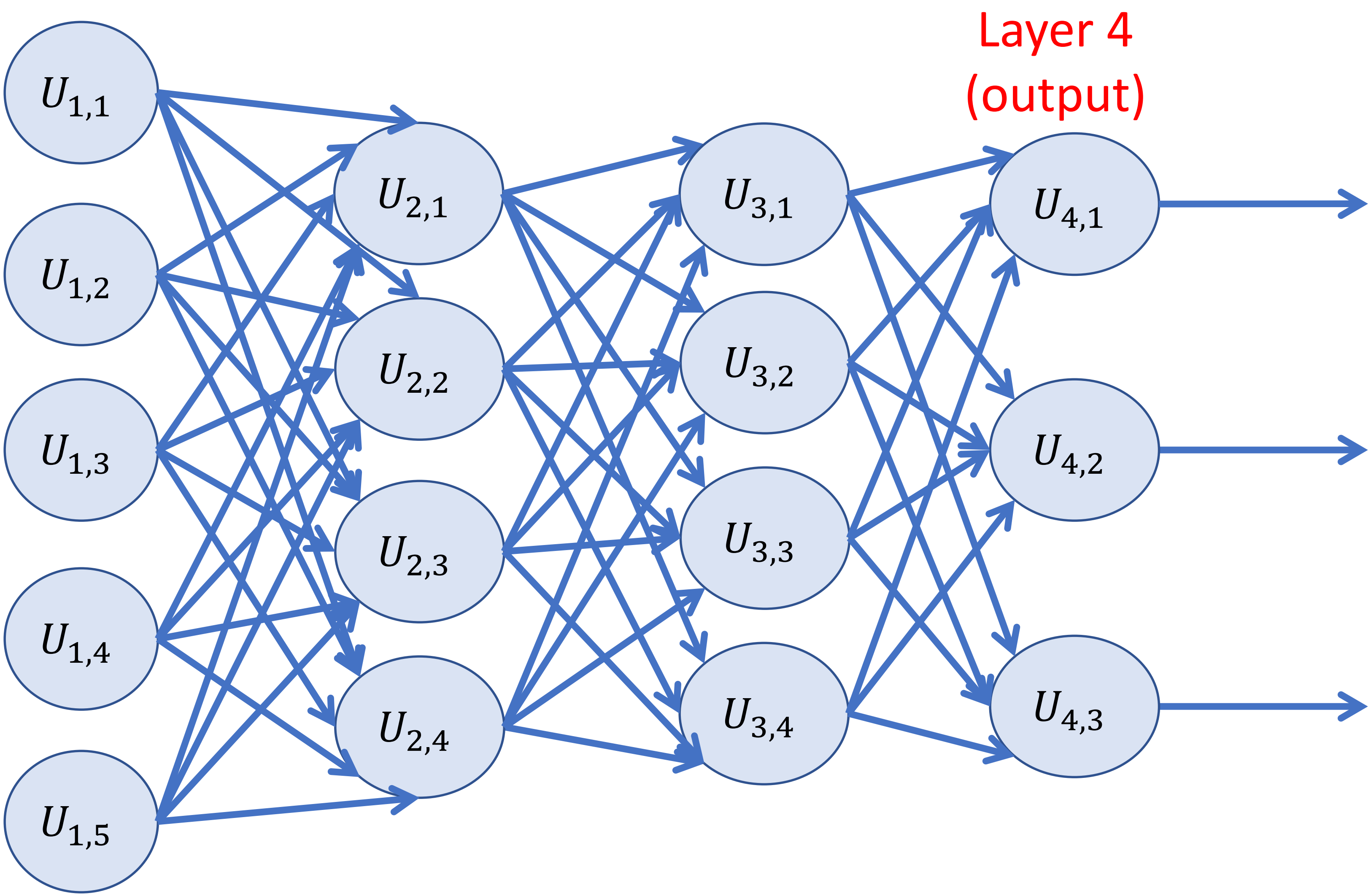
- This network has two hidden layers, with four units per layer.
- The number of hidden layers and the number of units per layer are hyperparameters, they can take different values.



Output Layer: How many units does it have? Could we have a different number? Is the number of output units a hyperparameter?

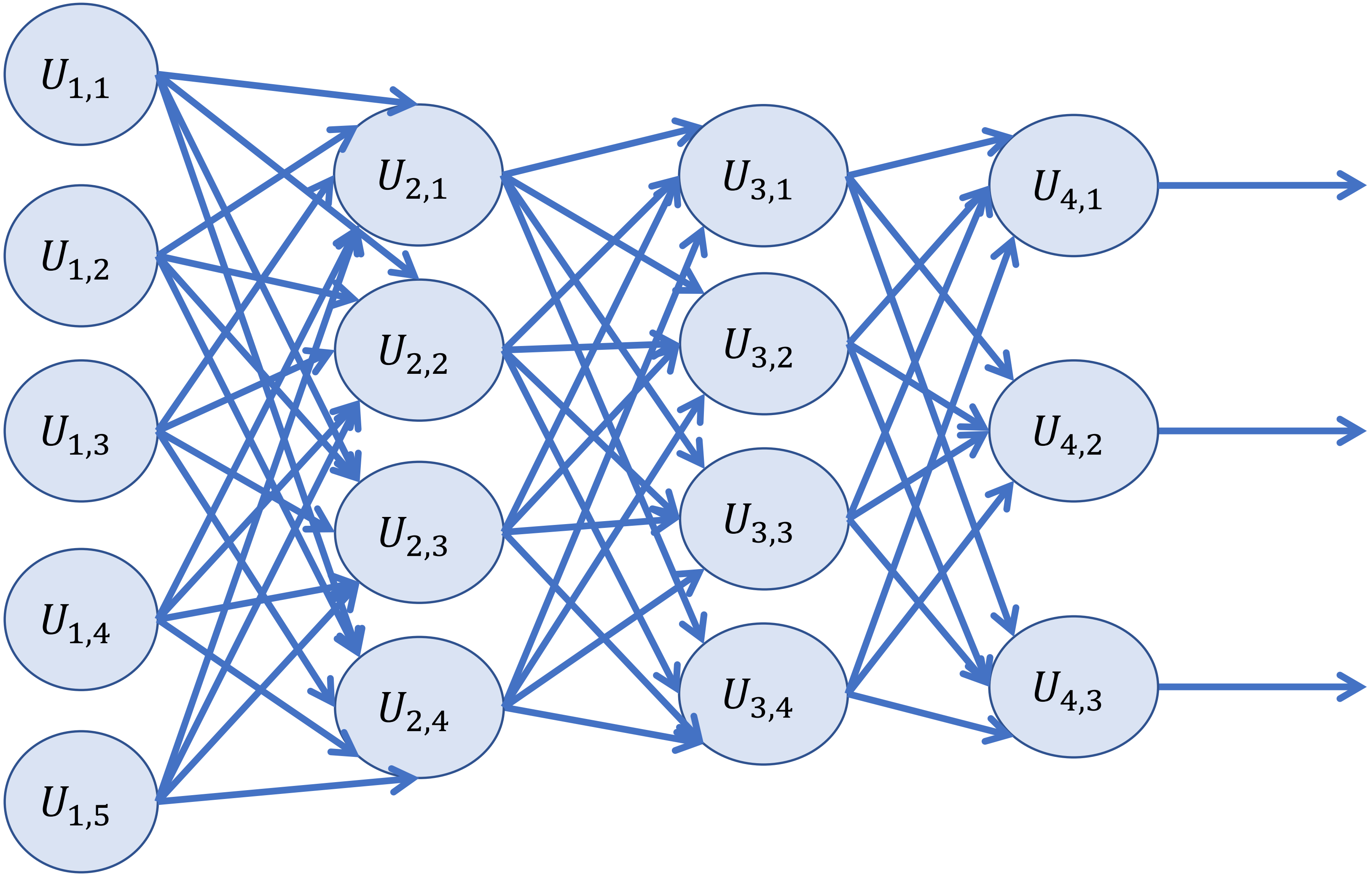


- In our example, the output layer **must** have three units, because we want to recognize three different classes (dog, cat, fox). We have no choice.



Network connectivity:

- In this neural network, at layers 2, 3, 4, every unit receives as input the output of ALL units in the previous layer.
- This is also a hyperparameter, it doesn't have to be like that.



Next: Training a Multi-Layer Network

- The next set of slides will describe how to train such a network.
- Training a neural network is done using gradient descent.
- The specific method is called **backpropagation**, but it really is just a straightforward application of gradient descent for neural networks.