Neural Networks – Part 2

- Training Perceptrons
- Handling Multiclass Problems

CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Training a Neural Network

- In some cases, the training process can find the best solution using a closed-form formula.
  - Example: linear regression, for the sum-of-squares error
- In some cases, the training process can find the best weights using an iterative method.
  - Example: sequential learning for logistic regression.
- In neural networks, we **cannot** find the best weights (unless we have an astronomical amount of luck).
  - We use gradient descent to find **local minima** of the error function.
  - In recent years this approach has produced spectacular results in real-world applications.

Notation for 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.

Perceptron Learning

- Before we discuss how to train an entire neural network, we start with a single perceptron.
- Remember: given input $\boldsymbol{x}_n$, a perceptron computes its output z using this formula: $z(\boldsymbol{x}_n) = h(b + \boldsymbol{w}^T \boldsymbol{x}_n)$
- What are the model parameters that we want to optimize during training?

Perceptron Learning

- Before we discuss how to train an entire neural network, we start with a single perceptron.
- Remember: given input $\boldsymbol{x}_n$, a perceptron computes its output z using this formula: $z(\boldsymbol{x}_n) = h(b + \boldsymbol{w}^T \boldsymbol{x}_n)$
- What are the model parameters that we want to optimize during training?
- The weights:
  - Bias weight b.
  - Weight vector $\boldsymbol{w}$.

Regression or Classification?

- The perceptron produces a continuous value between 0 and 1.
- Thus, perceptrons and neural networks are **<u>regression models</u>**, since they produce continuous outputs.
- However, perceptrons and neural networks can easily be used for classification.
- A perceptron can be treated as a **<u>binary classifier</u>**:
  - One class label is 0.
  - One class label is 1.
- Neural networks can do **<u>multiclass classification</u>** (more details on that later).

Perceptron Learning

- Given input $\boldsymbol{x}_n$, a perceptron computes its output z using this formula: $z(\boldsymbol{x}_n) = h(b + \boldsymbol{w}^T \boldsymbol{x}_n)$
- We use sum-of-squares as our error function.
- $E_n(b, \boldsymbol{w})$ is the contribution of training example $\boldsymbol{x}_n$:

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

- The error E over the entire training set is defined as: $E(b, \boldsymbol{w}) = \sum_{n=1}^{N} E_n(b, \boldsymbol{w})$
- Important: a single perceptron has a single output.
  - Therefore, for perceptrons (but NOT for neural networks in general), we assume that t_n is one-dimensional.

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 $E(b, \boldsymbol{w})$ is not differentiable.
  - Small changes of b or $\boldsymbol{w}$ usually lead to no changes in $h(b + \boldsymbol{w}^T\boldsymbol{x})$.
  - The only exception is when the change in b or $\boldsymbol{w}$ causes $h(b + \boldsymbol{w}^T\boldsymbol{x})$ to switch signs (from positive to negative, or from negative to positive).

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 error $E_n(b, \boldsymbol{w})$ 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 **must be** 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.
- If we do the calculations, the gradients turn out to be:

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

$$\frac{\partial E_n}{\partial \boldsymbol{w}} = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \left(1 - z(\boldsymbol{x}_n)\right) * \boldsymbol{x}_n$$

Computing the Gradient

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

- From the previous slide, the gradients are:

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

$$\frac{\partial E_n}{\partial \boldsymbol{w}} = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \left(1 - z(\boldsymbol{x}_n)\right) * \boldsymbol{x}_n$$

- 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

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

$$\frac{\partial E_n}{\partial \boldsymbol{w}} = \frac{1}{2}\left(t_n - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \left(1 - z(\boldsymbol{x}_n)\right) * \boldsymbol{x}_n$$

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

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

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

Weight Update

- (From previous slide) Update formulas:

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

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

- 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:

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

Perceptron Learning - Summary

- Input: Training inputs $\boldsymbol{x}_1, , \dots, \boldsymbol{x}_N$, target outputs $t_1, \dots, t_N$
  - 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) $b = b - \eta * \left(t_n - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \left(1 - z(\boldsymbol{x}_n)\right)$
   c) For d = 0 to D:
   $$w_d = w_d - \eta * \left(t_n - z(\boldsymbol{x}_n)\right) * z(\boldsymbol{x}_n) * \left(1 - z(\boldsymbol{x}_n)\right) * x_{n,d}$$
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 cumulative squared error E(w) of the perceptron at that point:

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

  - 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.

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

- Suppose we have this training set:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \quad q_1 = \text{dog}, \qquad s_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \quad q_2 = \text{dog}, \qquad s_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \quad q_3 = \text{cat}, \qquad s_3 = 2$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \quad q_4 = \text{fox}, \qquad s_4 = 3$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \quad q_5 = \text{cat}, \qquad s_5 = 2$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \quad q_6 = \text{fox}, \qquad s_6 = 3$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \;\; s_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \;\; s_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \;\; s_3 = 2$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \;\; s_4 = 3$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \;\; s_5 = 2$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \;\; s_6 = 3$
- Step 2: Convert each label s_n to a <u>**one-hot vector**</u> $\boldsymbol{t}_n$.
  - 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T$, $s_1 = 1$ $\quad \boldsymbol{t}_1 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T$, $s_2 = 1$ $\quad \boldsymbol{t}_2 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T$, $s_3 = 2$ $\quad \boldsymbol{t}_3 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T$, $s_4 = 3$ $\quad \boldsymbol{t}_4 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T$, $s_5 = 2$ $\quad \boldsymbol{t}_5 = (?\,,?\,,?\,)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T$, $s_6 = 3$ $\quad \boldsymbol{t}_6 = (?\,,?\,,?\,)^T$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \ \ s_1 = 1 \qquad \boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \ \ s_2 = 1 \qquad \boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \ \ s_3 = 2 \qquad \boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \ \ s_4 = 3 \qquad \boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \ \ s_5 = 2 \qquad \boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \ \ s_6 = 3 \qquad \boldsymbol{t}_6 = (0, 0, 1)^T$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \ \ s_1 = 1$ $\quad\quad$ $\boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \ \ s_2 = 1$ $\quad\quad$ $\boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \ \ s_3 = 2$ $\quad\quad$ $\boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \ \ s_4 = 3$ $\quad\quad$ $\boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \ \ s_5 = 2$ $\quad\quad$ $\boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \ \ s_6 = 3$ $\quad\quad$ $\boldsymbol{t}_6 = (0, 0, 1)^T$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \;\; t_1 = 1$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \;\; t_2 = 1$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \;\; t_3 = 0$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \;\; t_4 = 0$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \;\; t_5 = 0$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \;\; t_6 = 0$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \;\; s_1 = 1 \qquad \boldsymbol{t}_1 = (1, 0, 0)^T$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \;\; s_2 = 1 \qquad \boldsymbol{t}_2 = (1, 0, 0)^T$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \;\; s_3 = 2 \qquad \boldsymbol{t}_3 = (0, 1, 0)^T$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \;\; s_4 = 3 \qquad \boldsymbol{t}_4 = (0, 0, 1)^T$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \;\; s_5 = 2 \qquad \boldsymbol{t}_5 = (0, 1, 0)^T$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \;\; s_6 = 3 \qquad \boldsymbol{t}_6 = (0, 0, 1)^T$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \;\; t_1 = 0$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \;\; t_2 = 0$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \;\; t_3 = 1$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \;\; t_4 = 0$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \;\; t_5 = 1$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \;\; t_6 = 0$
- 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:
  - $\boldsymbol{x}_1 = (0.5, 2.4, 8.3, 1.2, 4.5)^T, \;\; t_1 = 0$
  - $\boldsymbol{x}_2 = (3.4, 0.6, 4.4, 6.2, 1.0)^T, \;\; t_2 = 0$
  - $\boldsymbol{x}_3 = (4.7, 1.9, 6.7, 1.2, 3.9)^T, \;\; t_3 = 0$
  - $\boldsymbol{x}_4 = (2.6, 1.3, 9.4, 0.7, 5.1)^T, \;\; t_4 = 1$
  - $\boldsymbol{x}_5 = (8.5, 4.6, 3.6, 2.0, 6.2)^T, \;\; t_5 = 0$
  - $\boldsymbol{x}_6 = (5.2, 8.1, 7.3, 4.2, 1.6)^T, \;\; t_6 = 1$
- 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 $C_1, \dots, C_K$, where $K > 2$.
- We have training inputs $\boldsymbol{x}_1, , \dots, \boldsymbol{x}_N$, and target values $\boldsymbol{t}_1, \dots, \boldsymbol{t}_N$.
- Each target value $\boldsymbol{t}_n$ is a K-dimensional vector:
  - $\boldsymbol{t}_n = (t_{n,1}, t_{n,2}, \dots, t_{n,K})$
  - $t_{n,k} =$ 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.

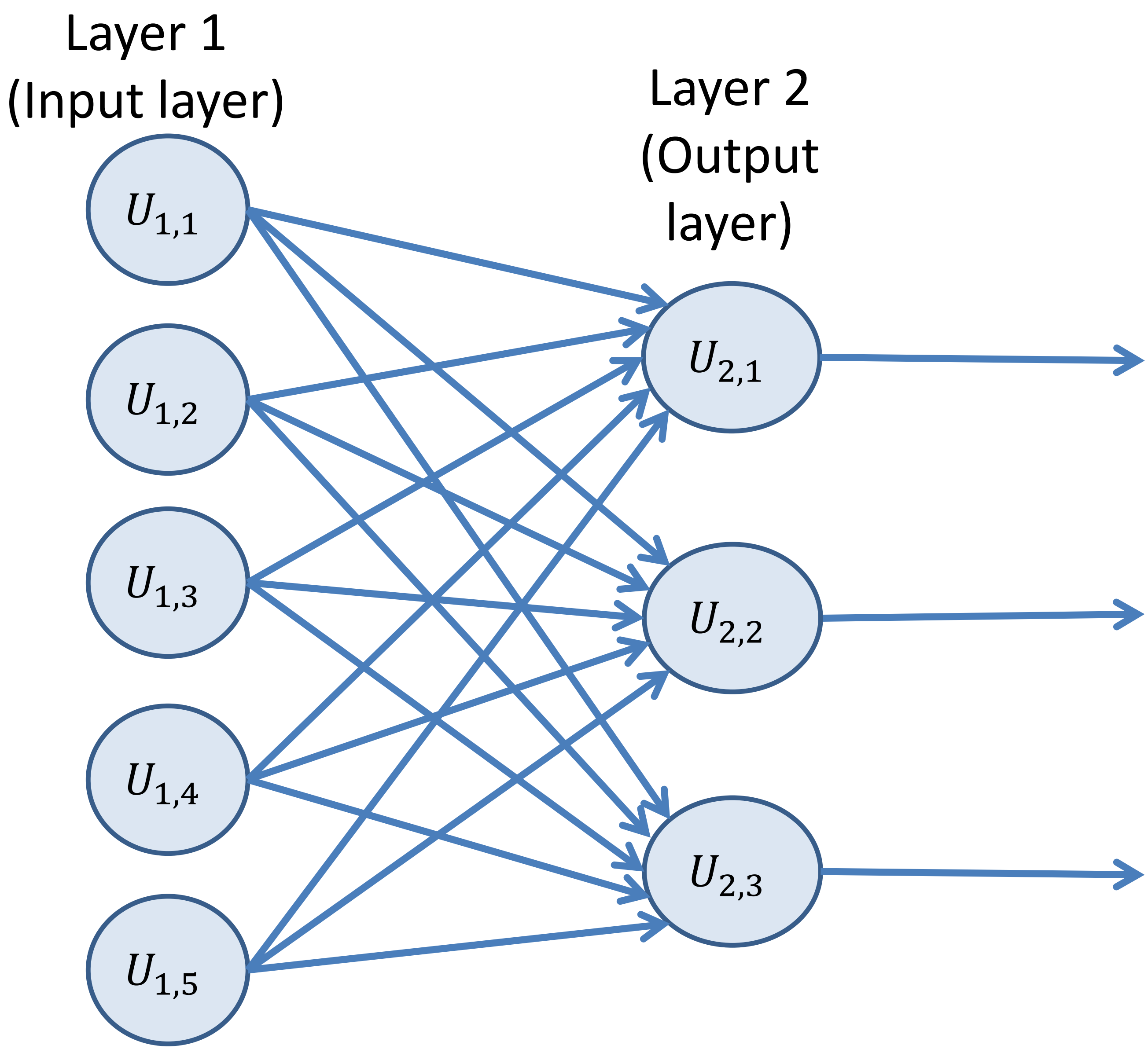
One-Versus-All Perceptrons

- To classify a test 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**.

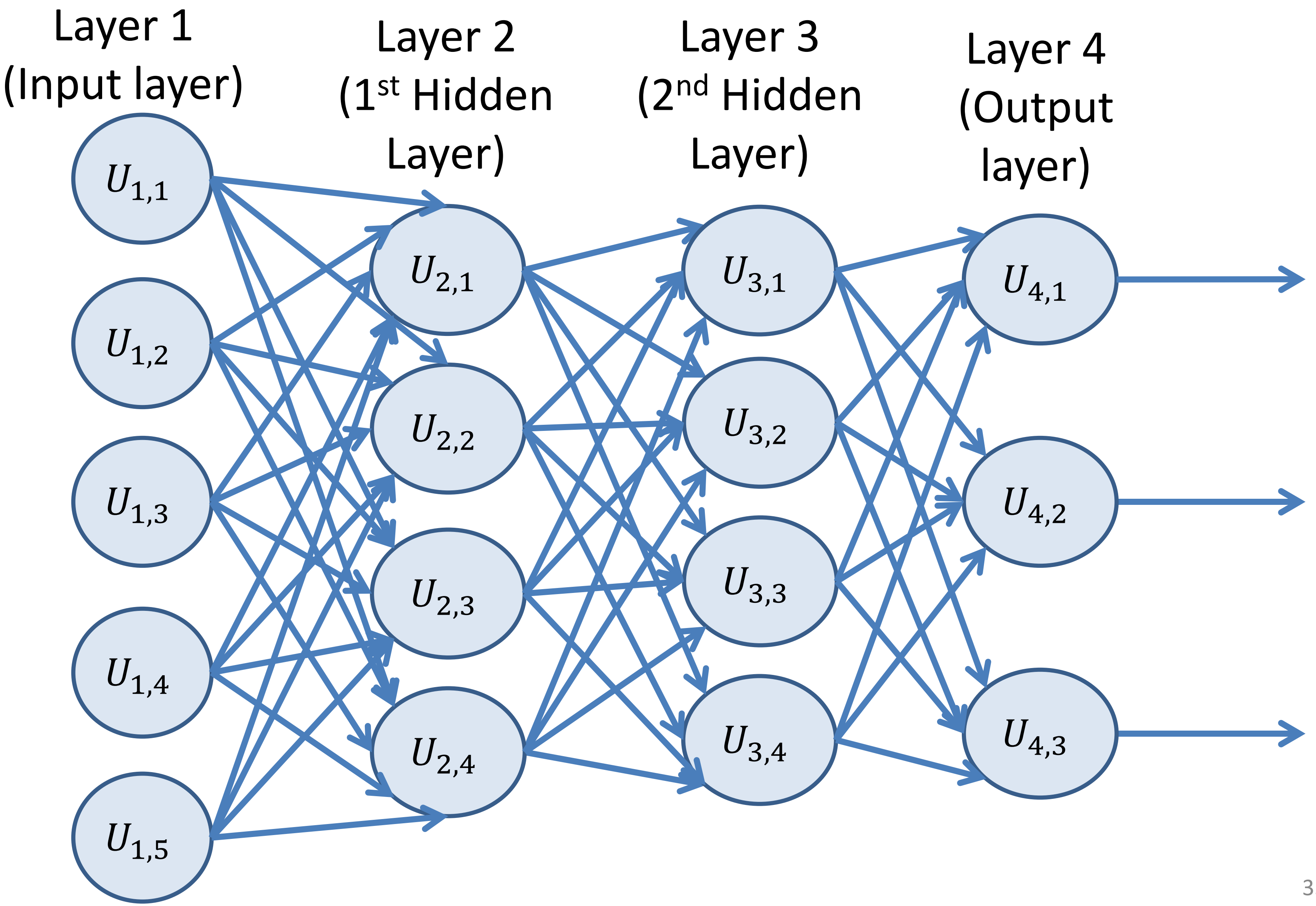
OVA Perceptrons as a Single Network



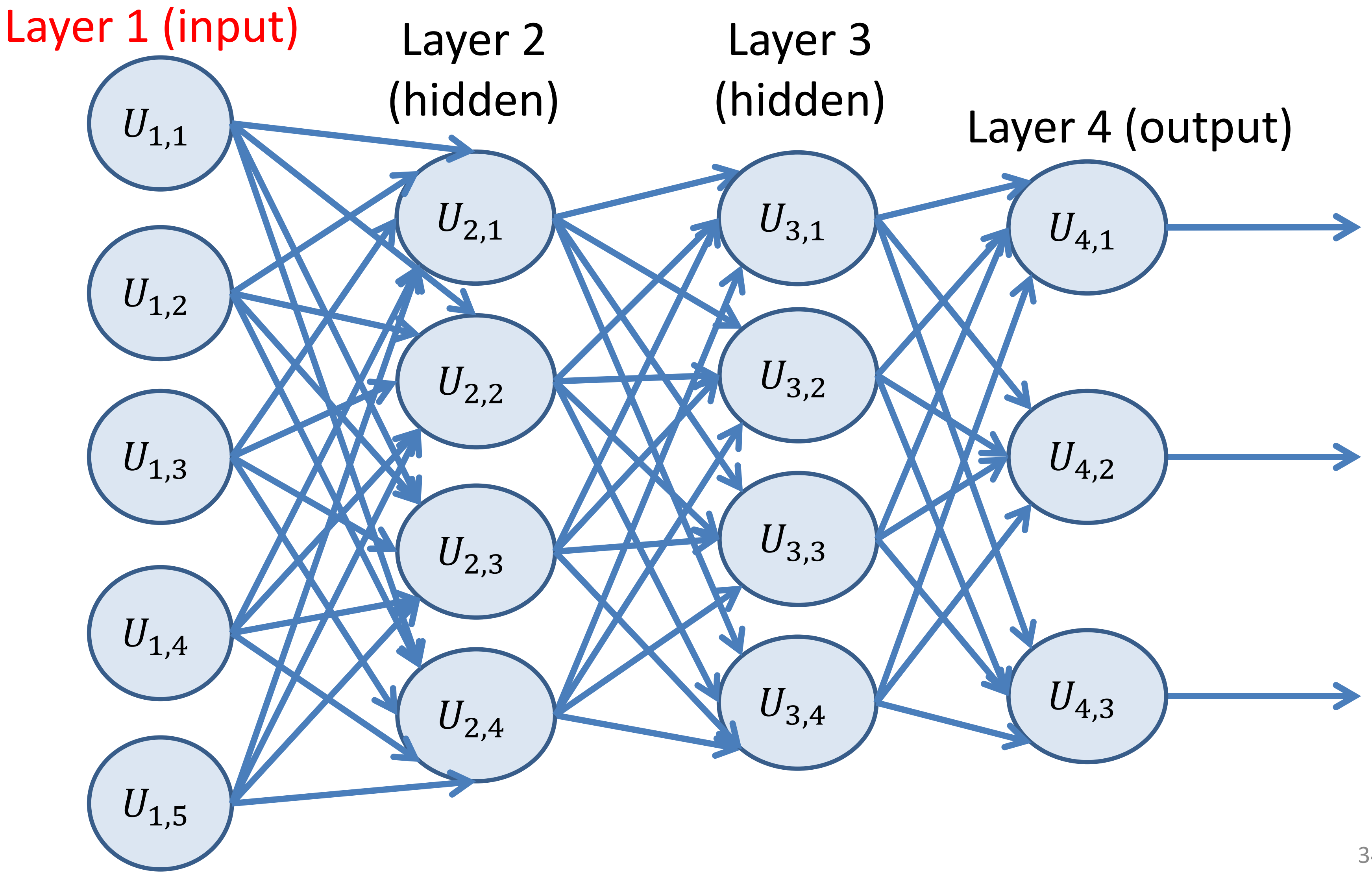
Multiclass Neural Networks

- 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.

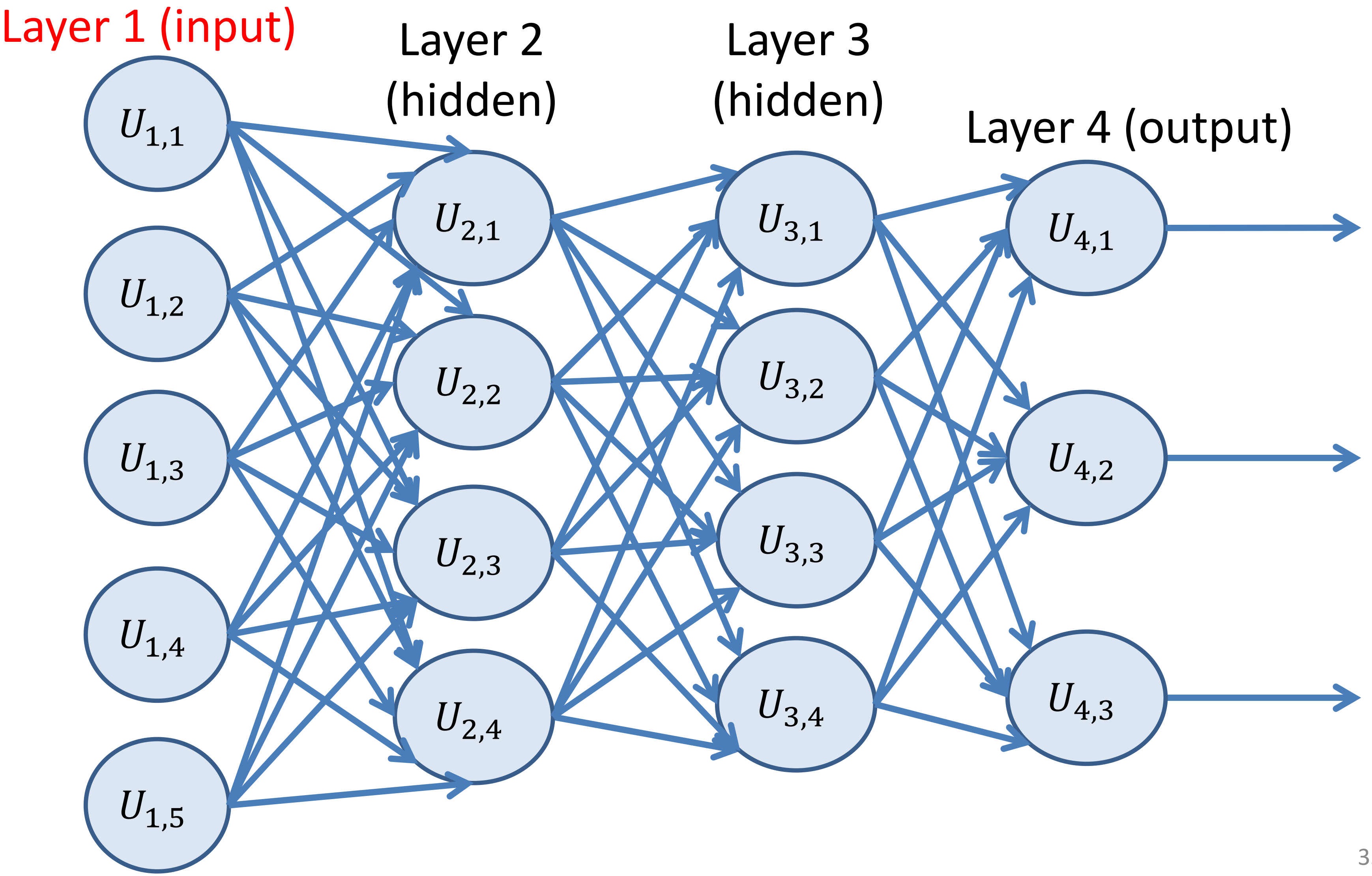
A Network for Our Example



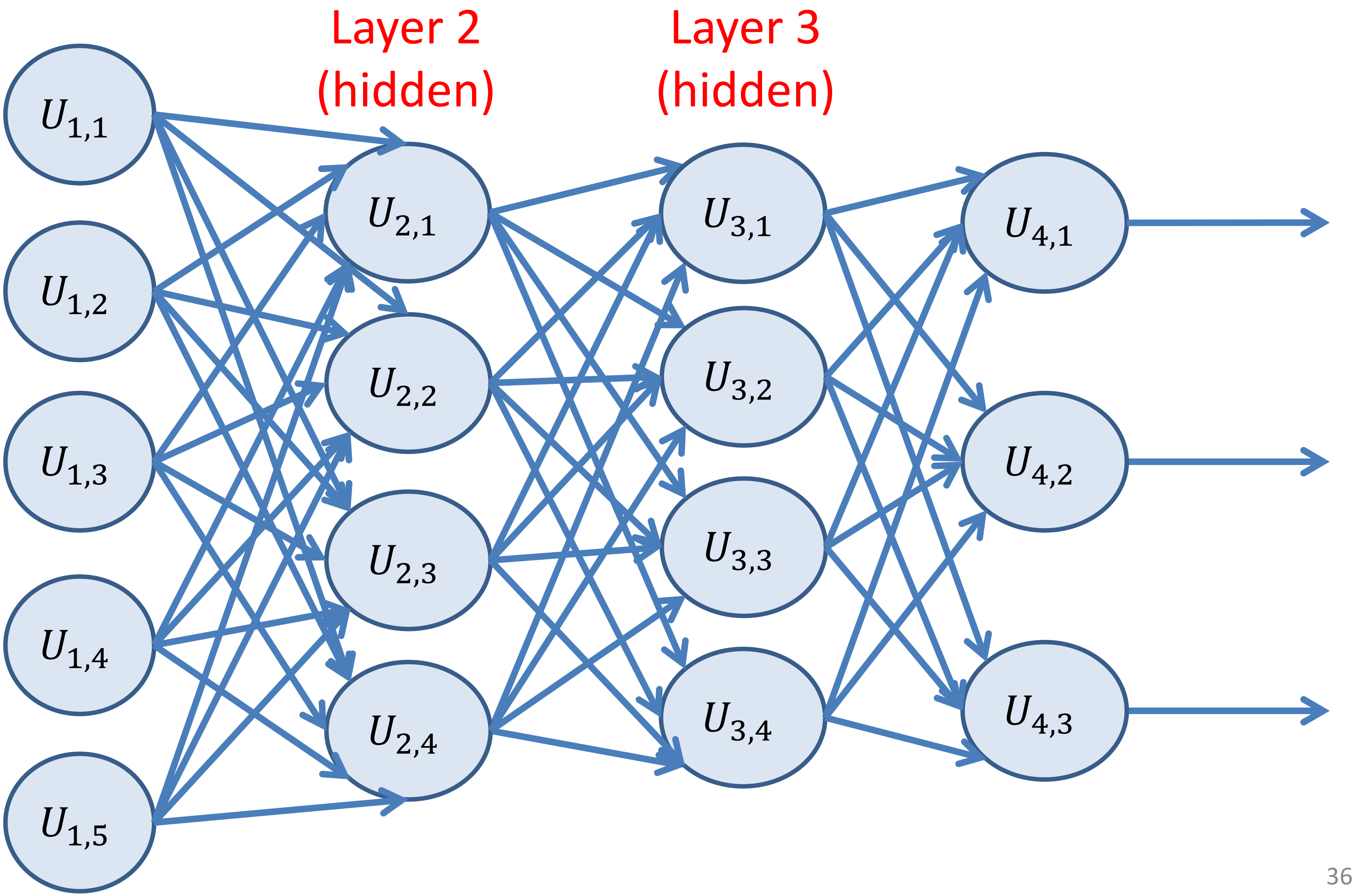
Input Layer: How many units does it have? Could we have a different number? Is the number of input units a hyperparameter?



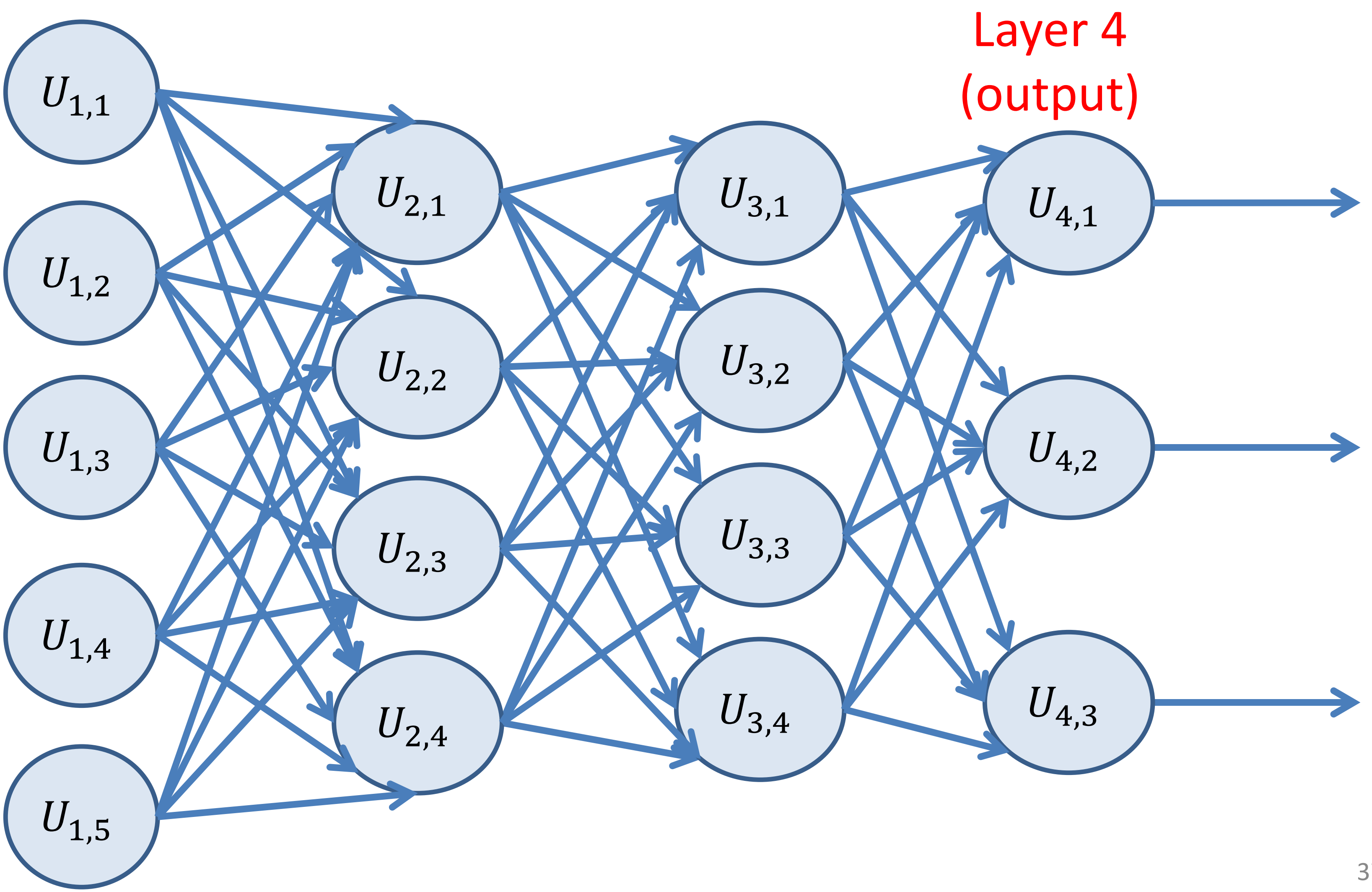
In our example, the input layer it **must** have five units, because each input is five-dimensional. We don't have a choice.



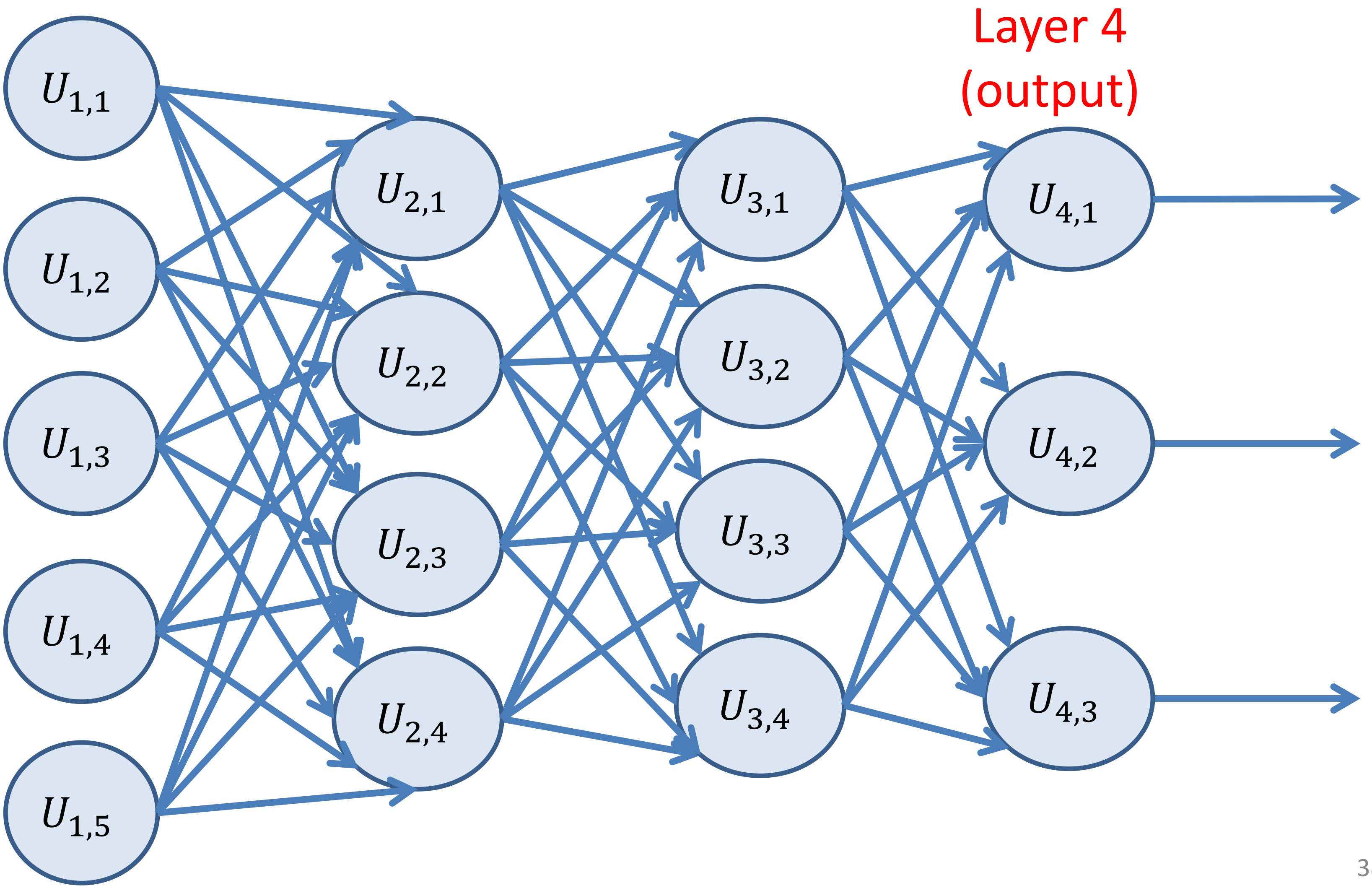
- This network has two hidden layers, with four units per layer.
- The number of hidden layers and the number of units per layer are hyperparameters, they can take different values.



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

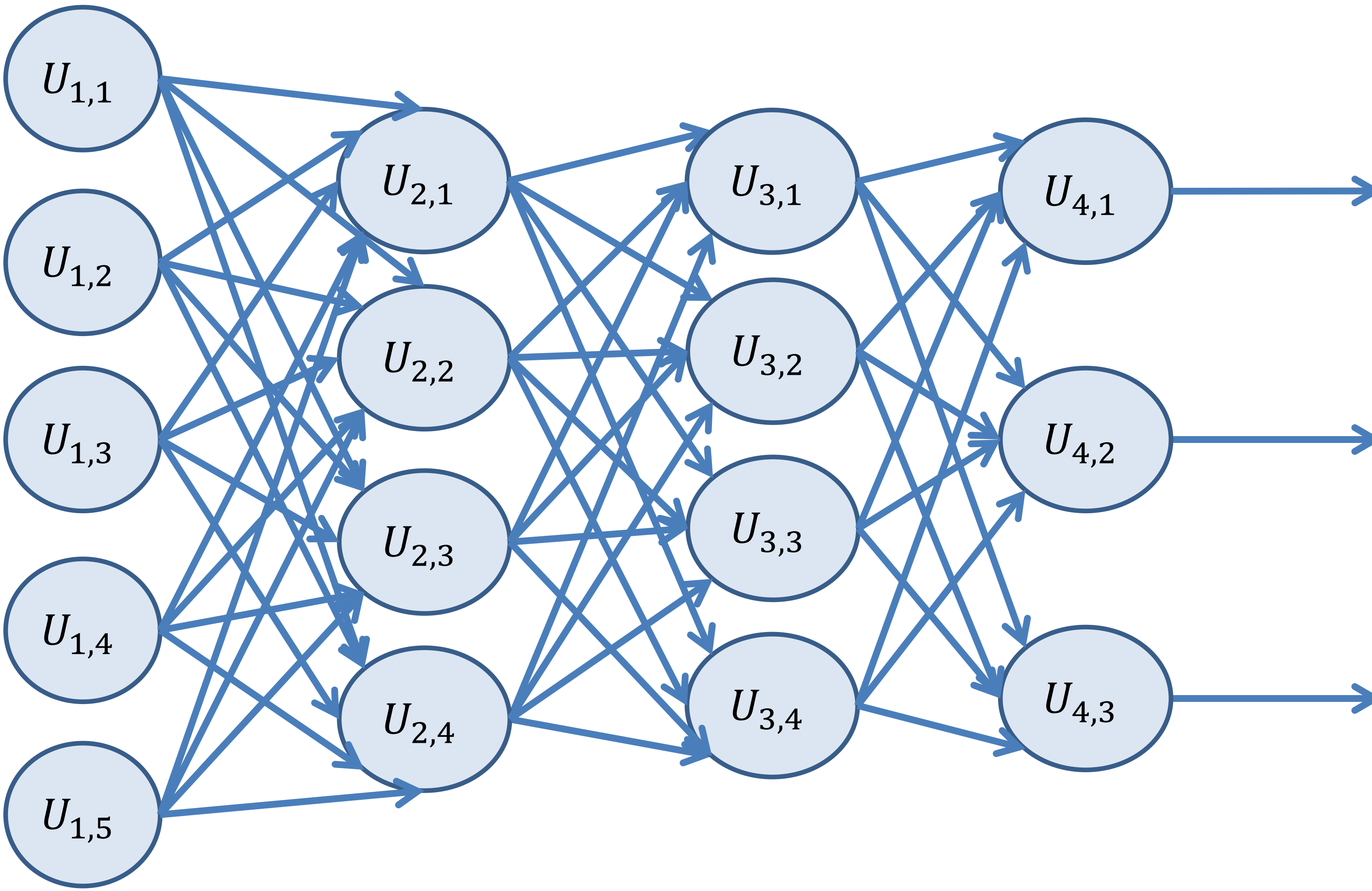


- In our example, the output layer **must** have three units, because we want to recognize three different classes (dog, cat, fox). We have no choice.



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

- 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.