**1. Layers in a Neural Network**

* **Input Layer:**
  + The very first layer.
  + Represents the **data you feed into the network**.
  + Example: if you have two features (like height and weight), your input layer has two neurons.
* **Output Layer:**
  + The final layer.
  + Produces the **predictions**.
  + We compare this layer’s outputs to the **true targets** during training.
* **Hidden Layers:**
  + All layers between the input and output layers.
  + Called “hidden” because:
    - We know the inputs and outputs.
    - But the computations inside are **not directly visible or interpretable**.
  + Each hidden layer consists of **hidden units (neurons/nodes)**.

**2. Hidden Units (Nodes)**

* A **hidden unit** is one neuron in a hidden layer.
* If is the tensor representing a hidden layer, each element of is a **hidden unit**.
* The **number of hidden units in a layer** is called the **width** of the layer.
  + Example: a hidden layer with 5 neurons has a width of 5.
* Usually, layers are stacked with **the same width**, but this is not mandatory.

**3. Depth of a Network**

* **Depth** = the number of **hidden layers** in the network.
* A **deep network** (DNN) is just a network with **multiple hidden layers stacked one after another**.
* Depth is important because more layers allow the network to **learn more complex patterns**.

**4. Width vs Depth**

| **Concept** | **Meaning** | **Example** |
| --- | --- | --- |
| Width | Number of hidden units in a layer | 10 neurons in one hidden layer → width = 10 |
| Depth | Number of hidden layers | 3 hidden layers → depth = 3 |

**Key idea:** Both width and depth control **the capacity of the network** — how complex patterns it can learn.

**5. Parameters vs Hyperparameters**

* **Parameters:**
  + These are **learned during training**.
  + Examples: **weights** and **biases**.
* **Hyperparameters:**
  + These are **set before training**.
  + Examples:
    - Width (number of neurons per layer)
    - Depth (number of hidden layers)
    - Learning rate
    - Batch size
* **Difference:**
  + Parameters → learned via optimization
  + Hyperparameters → chosen by the designer

**✅ Summary**

1. Input → Hidden layers → Output = a deep neural network.
2. Hidden layers are made of hidden units (nodes).
3. Width = number of nodes per hidden layer.
4. Depth = number of hidden layers.
5. Parameters are learned (weights, biases).
6. Hyperparameters are set before training (width, depth, learning rate).

**🎓 Lecture: Deep Nets, Layers, and Non-Linearities**

**1. Revisiting the Picture**

We’ve already seen the classic diagram of a deep neural network. It may look simple, but it’s one of the most fundamental illustrations in deep learning. Every time we discuss deep nets, we’ll come back to this picture.

* The **first layer** is the **input layer**.
  + Each circle is one input feature.
  + Example: in weather forecasting, we might have 8 features:  
    average temperature, highest temperature, lowest temperature, humidity, precipitation, atmospheric pressure, cloud cover, and visibility.

These are the **raw data** we feed into the network.

**2. From Inputs to Hidden Layer**

Now, how do inputs become hidden units?

* Step 1: We combine inputs **linearly** using weights.
  + If we have 8 inputs and 9 hidden neurons, the weight matrix is **8 × 9**.
  + After multiplying, we get a vector of length 9 → this matches the number of hidden units.
* Step 2: We apply a **non-linearity** (like ReLU or sigmoid).
  + This doesn’t change the shape of the data — still 9 numbers.
  + But it changes how the numbers behave (straight lines → curves).

Every **arrow** in the diagram represents one weight.

* With 8 inputs × 9 hidden neurons = **72 arrows/weights**.
* Each weight connects a specific input to a specific hidden neuron.

For example:

* Weight (3,6) connects Input #3 to Hidden Neuron #6.
* Hidden Neuron #6 is built by combining Input1×w(1,6) + Input2×w(2,6) + … + Input8×w(8,6), then applying a non-linearity.

**3. Stacking Layers**

Once we have the first hidden layer, we can do the same process again:

* Take those 9 hidden units.
* Connect them to another 9 hidden units.
* This requires a **9 × 9 weight matrix** (81 arrows).
* Apply non-linearity again → now we have a second hidden layer.

We can keep stacking layers like this. In fact, in theory, you could add 100 hidden layers. That’s how deep neural networks get their **depth**.

Finally, we reach the **output layer**.

* Suppose we want 4 outputs (like predicting tomorrow’s temperature, humidity, precipitation, and pressure).
* From 9 hidden neurons to 4 outputs → weight matrix is **9 × 4 = 36 weights/arrows**.

**4. Why Do We Need Non-Linearities?**

Here’s the crucial point:

Imagine we build a network with multiple layers, but **no non-linearities**.

* Inputs → Linear transform (weights W1) → Hidden → Linear transform (weights W2) → Outputs.

Mathematically:

* Hidden = X × W1
* Output = Hidden × W2 = (X × W1) × W2 = X × (W1 × W2)

But W1 × W2 is just another matrix.  
👉 So the entire deep net collapses into a **single linear model**.

That means:

* Adding hidden layers without non-linearities changes nothing.
* Even with 100 layers, it’s just one big linear model.
* Hidden layers become useless.

Therefore:

* **Non-linearities are essential.**
* They allow stacking layers to actually increase complexity.
* With them, deep nets can represent very complicated, arbitrary functions — not just straight lines.

**✅ Key Takeaway**

* **Layers** = input → hidden(s) → output.
* **Arrows** = weights (connections).
* **Non-linearities** = the magic ingredient that makes deep learning possible.
* Without non-linearities, depth doesn’t matter — everything collapses into a single linear model.

👉 **In one sentence:** To build powerful deep nets that can capture complex patterns, we need to stack layers with non-linearities.