Hi everyone,
My name is Li Li, and I am a PhD student in Computer Science at the University of Southern California.
Today, I will be presenting a lecture for CSCI 566 — *Deep Learning and Its Applications*.
We will take a brief look at some important neural network architectures that serve as the building blocks for modern deep learning systems.

Let us start with the motivation.
Deep learning models, especially neural networks, are designed to process complex, high-dimensional data.
The term “deep” comes from the presence of multiple layers in the network.
These layers are not all the same — they can serve different roles, such as filtering, feature extraction, or dimensionality reduction.
The idea is to progressively transform raw input data into representations that are easier for a simpler network to learn.
Today, we will focus on a few key techniques that are commonly used to build more advanced models.

Here is an overview of some classic architectures.
First, **Long Short-Term Memory networks**, or LSTMs, which are useful for modeling sequential data.
Second, **Autoencoders**, which are often used for unsupervised representation learning.
Third, **Generative Adversarial Networks**, or GANs, which have gained attention for their ability to generate realistic data.
And finally, **Convolutional Neural Networks**, or CNNs, which are the standard in image processing tasks.
Let us now start with Recurrent Neural Networks, which are the foundation of LSTMs.

A key issue with traditional feedforward neural networks, such as multilayer perceptrons, is that they assume input samples are independent.
This assumption does not hold in many real-world applications.
For example, in natural language, the word “often” tends to follow “is” or “are.” These dependencies matter.
Recurrent Neural Networks, or RNNs, address this by incorporating past information when processing current input.
In an RNN, the output of a hidden layer is fed back into itself, allowing it to maintain memory of previous inputs.

Training RNNs is done using standard gradient descent.
To visualize this, we can unroll the RNN in time.
This unrolled version shows how each time step is connected to the next through hidden state transitions.

As a result, learning at each time step is influenced by the entire history of previous inputs. This makes RNNs suitable for tasks like speech recognition or time series prediction, where context matters.

This slide shows what happens when we unroll an RNN across time steps.
Each input at time step *t* produces an output, but also passes a hidden state to the next time step.
This recurrent connection allows the network to “remember” previous inputs.
For example, in natural language processing, the meaning of a current word can depend heavily on the words before it.
By unrolling the network, we can see how dependencies are captured over multiple time steps.
However, this also increases the complexity of training, since errors have to be backpropagated through many steps — a process known as backpropagation through time.

Here, we have a more visual representation of the RNN in action.
Each circle represents a repeated instance of the same RNN cell. The parameters are shared across time, which is a key design choice.
The diagram highlights how each output depends not just on the current input, but also on the accumulated history.
This is what gives RNNs their strength in sequential modeling.
But again, as the sequence gets longer, it becomes harder for the network to maintain long-term dependencies — leading to the vanishing gradient problem.
In the next part, we will look at how LSTMs are designed specifically to solve this.