

Eighteenth Session

Alireza Moradi



| [Linkedin](#) |



| [k](#)

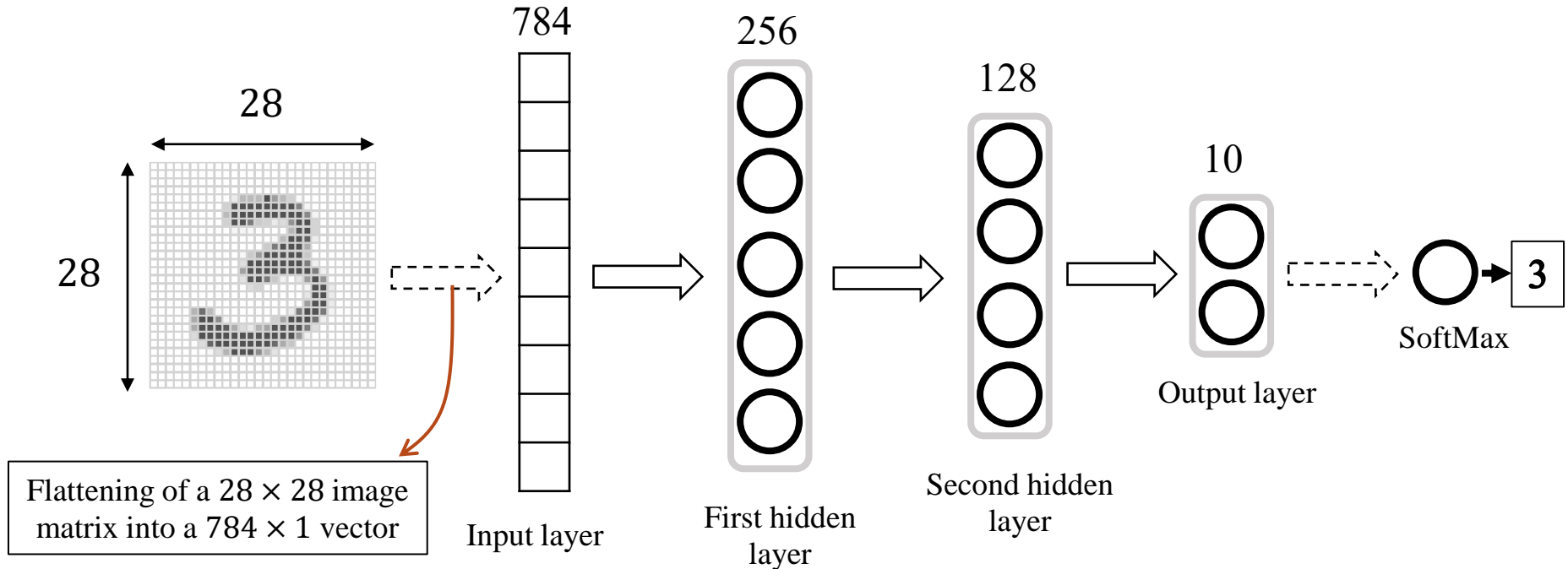
Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- A **convolutional neural network (CNN)** is a type of artificial neural network used primarily for image recognition and processing, due to its **ability to recognize patterns in images**.
- CNNs are employed in a variety of practical scenarios, such as autonomous vehicles, security camera systems, and others.
- A **Convolutional Neural Network (CNN)** is also known as **ConvNet**.
- CNNs are distinguished from classic machine learning algorithms by their ability to autonomously **extract features** at a large scale, bypassing the need for manual feature engineering and thereby enhancing efficiency.

Why CNNs?

- An image cannot be thought of as a simple vector of input pixel values, primarily because **adjacency of pixels** really matters.



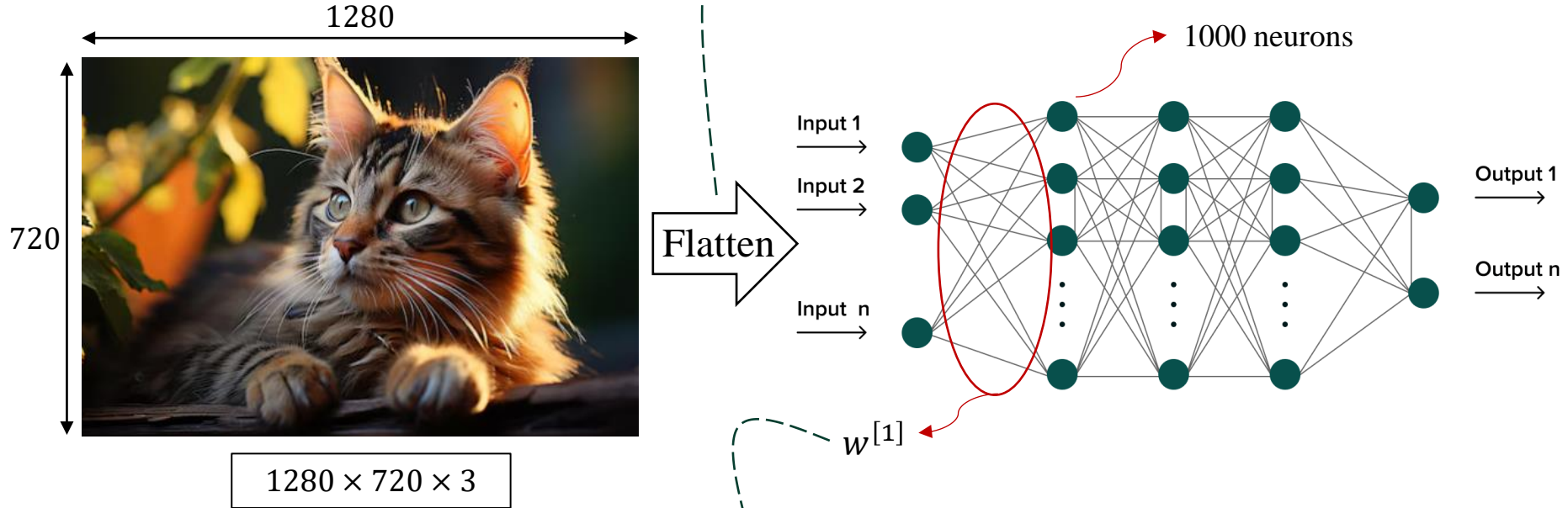
Why CNNs?

- Suppose there are n pixels and n units in the first hidden layer, to which the pixels provide input.
- If the flatten input and the first hidden layer are **fully connected**, that means n^2 weights.
- For a typical megapixel RGB image, that's **9 trillion weights**!
- Such a vast parameter space would require correspondingly vast numbers of training images and a huge computational budget to run the training algorithm.



Why CNNs?

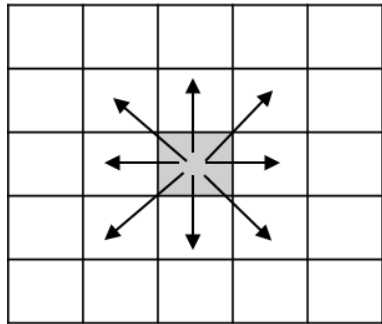
- A fully connected architecture:



Dimension of $w^{[1]} = 2764800 \times 1000 = 2,764,800,000$

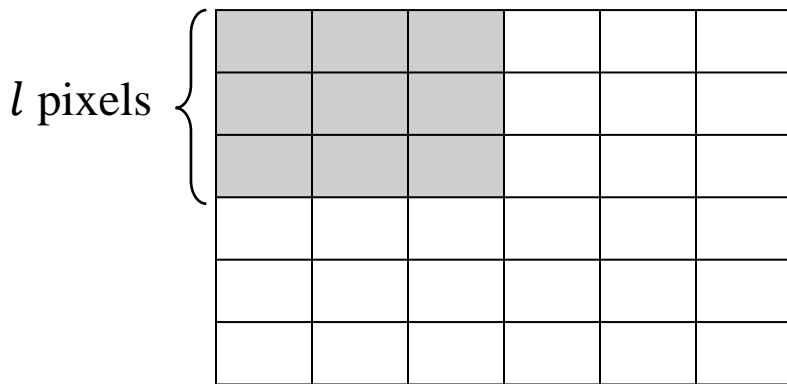
Why CNNs?

- CNNs are designed to work with **grid-structured inputs**, which have **strong spatial dependencies in local regions** of the grid.
- The most obvious example of grid-structured data is a **2-dimensional image**.
- Adjacent spatial locations in an image often have **similar color values**.



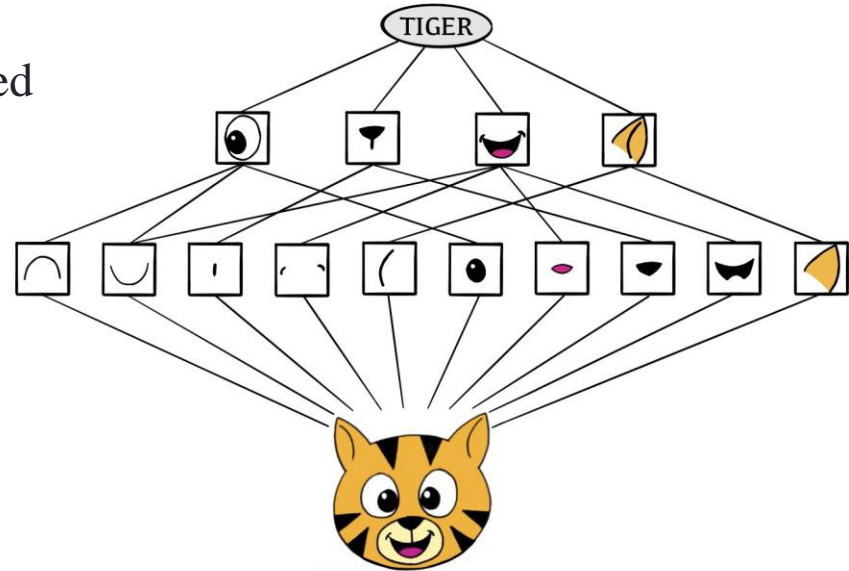
CNN main idea

- These considerations suggest that we should construct the first hidden layer so that each J hidden unit **receives input** from **only a small, local region** of the image.
- This kills two birds with one stone. First, it respects **adjacency**, at least locally. Second, it **cuts down the number of weights**.
- If each local region has $l \ll n$ pixels, then there will be $l \times n \ll n^2$ weights in all.



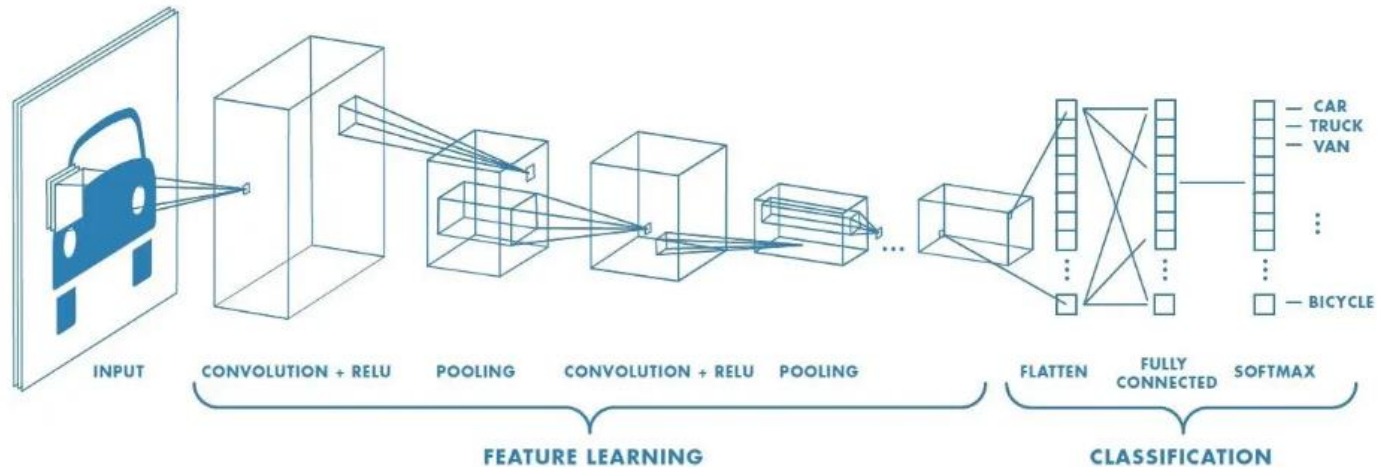
CNN main idea

- The CNN builds up an image in a **hierarchical fashion**.
- Edges and shapes are recognized and **pieced together** to form more complex shapes, eventually assembling the target image.
- This hierarchical construction is achieved using **convolution** and **pooling** layers.



CNN main constituents

1. Convolutional layers
2. Activation functions
3. Pooling layers
4. Fully connected layer

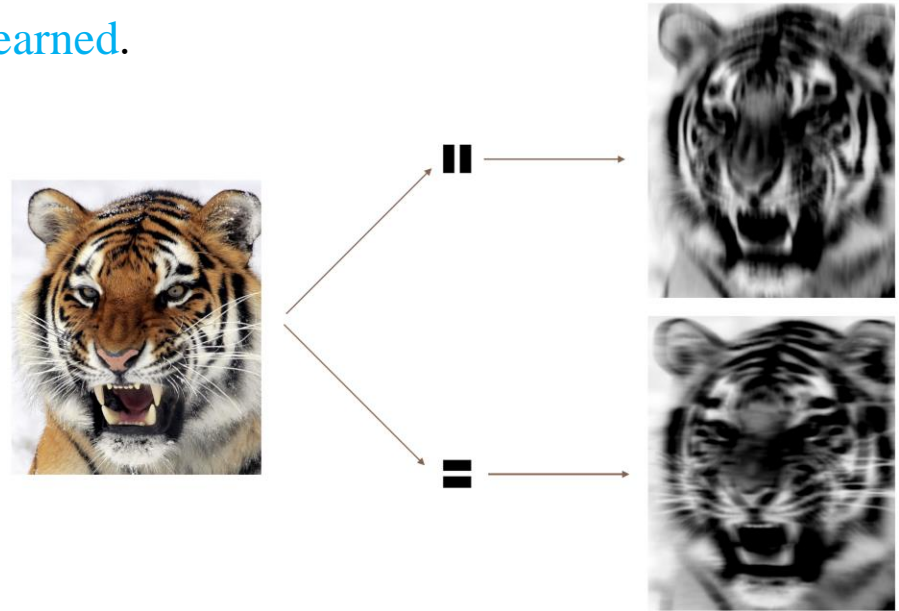


Convolution Layer

- As the name suggests, the main mathematical task performed is called convolution, which is the application of a **sliding window function to a matrix of pixels** representing an image.
- The **sliding function** applied to the matrix is called **kernel** or **filter**.
- In the convolution layer, **several filters of equal size are applied**.
- For example, one filter might be good at finding straight lines, another might find curves, and so on.
- By **using several different filters**, the CNN can get a good idea of all the different patterns that make up the image.

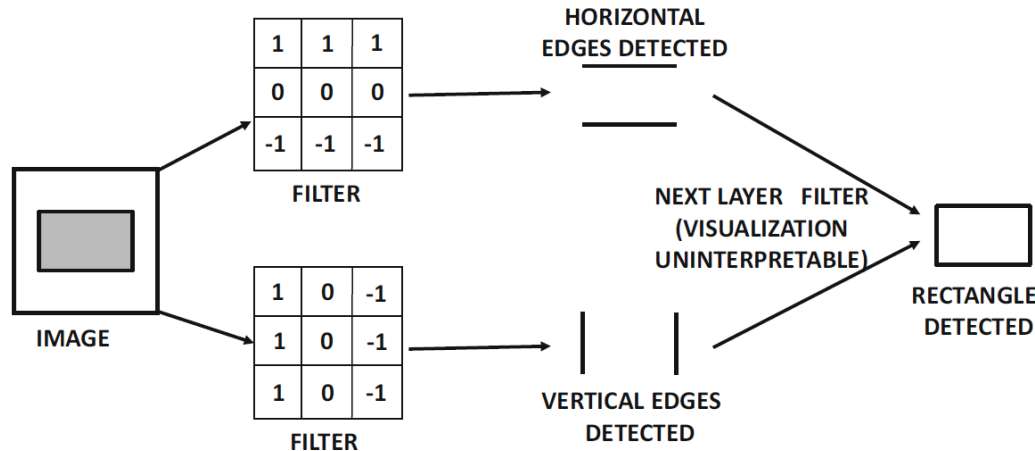
Convolution filter

- The idea of convolution with a filter is to **find common patterns** that occur in different parts of the image.
- **Convolution** are **small images that is learned**.
- The two filters shown here highlight vertical and horizontal stripes.
- The result of the convolution is a new feature map.



Convolution filter

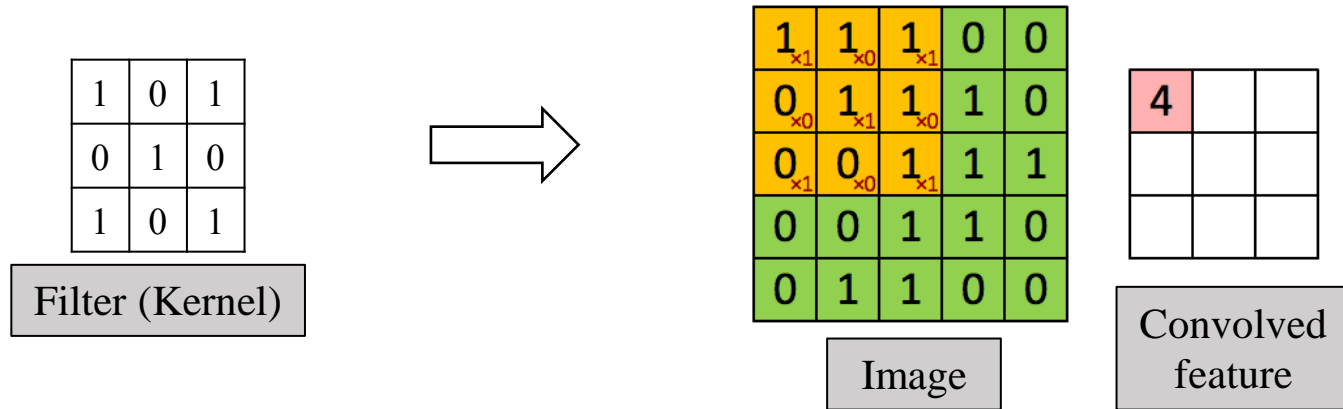
- In the next layer, the detected edges from the last layer are **combined** to form rectangles.



- The filters are learned during training. Specifically, the **weights in the filters are learned** by the network.

Convolution filter

- we can perform the convolution operation by applying the dot product:



Convolution filter

- Since images have three colors channels, the filters does as well: one filter per channel, and then dot-products are summed

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...

Input Channel #1 (Red)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...

Input Channel #2 (Green)

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...

Input Channel #3 (Blue)

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

+

+

+ 1 = -25



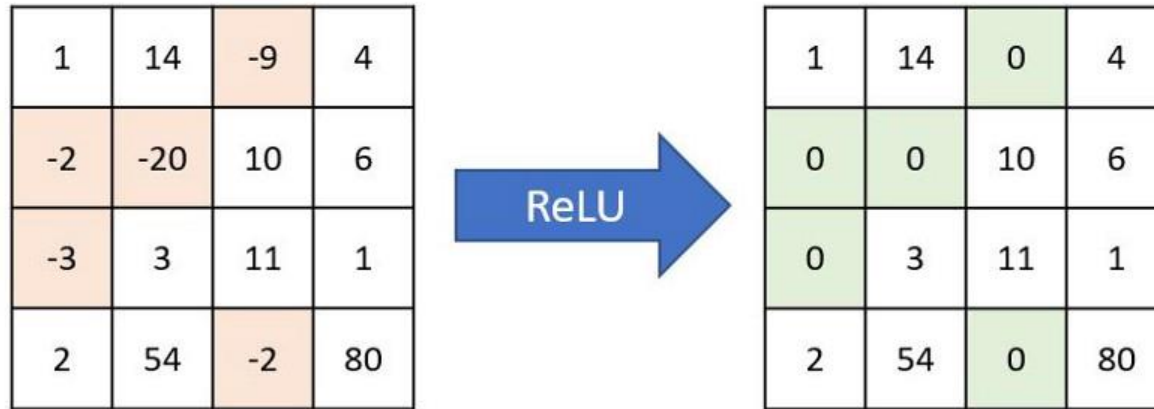
Bias = 1

Output

-25				...
				...
				...
				...
...

CNN; Activation function

- An activation function is applied after each convolution operation.
- This function helps the network **learn non-linear relationships** between the features in the image.
- For example:



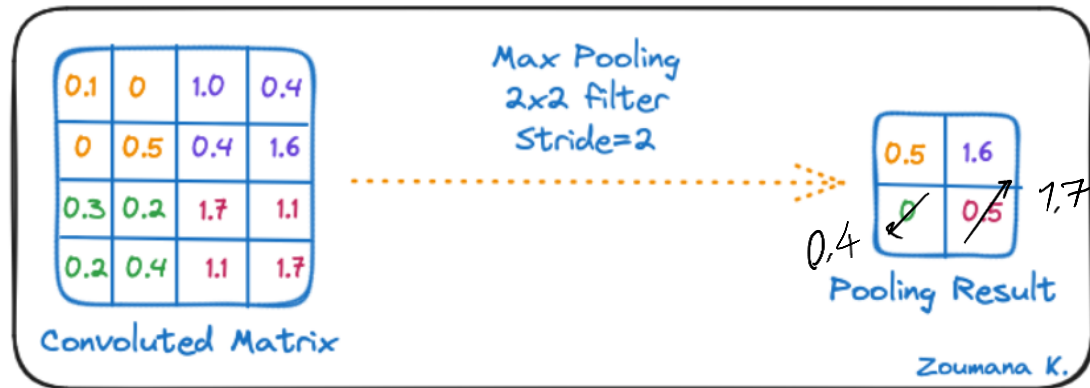
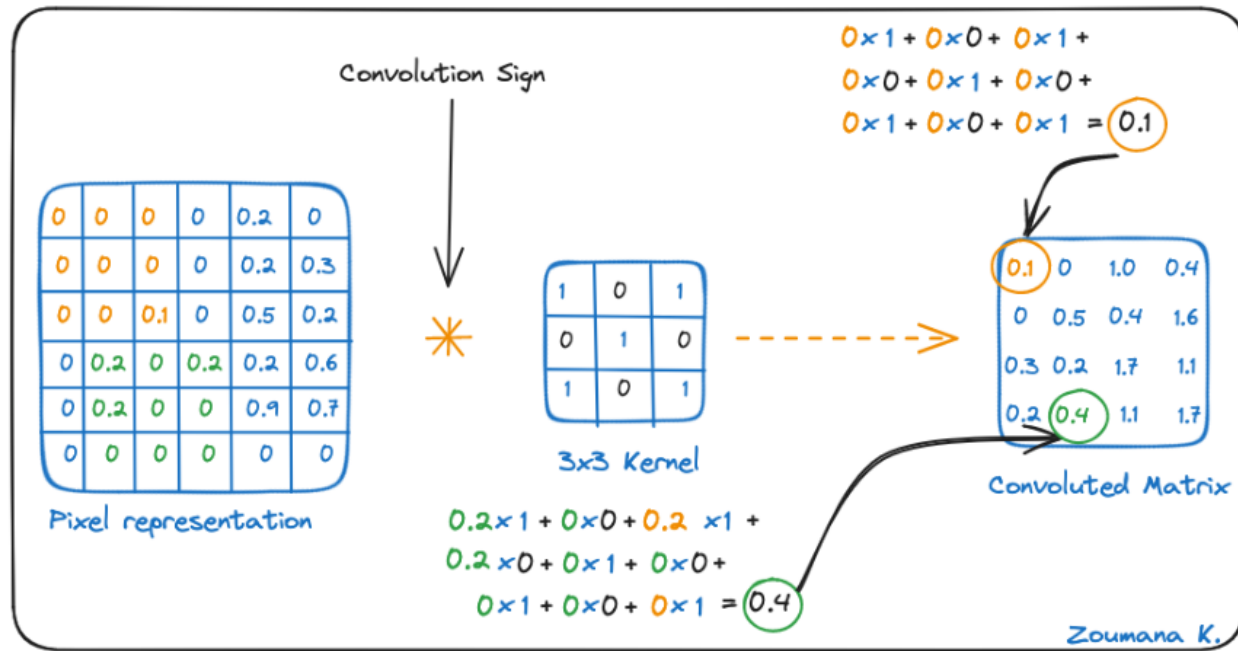
CNN; Pooling layers

- The goal of the pooling layer is to **pull the most significant features** from the convoluted matrix.
- This is done by applying some **aggregation operations**, which reduce the dimension of the feature map (convoluted matrix), hence reducing the memory used while training the network.
- The most common aggregation functions that can be applied are:

1. Max pooling
2. Sum pooling
3. Average pooling

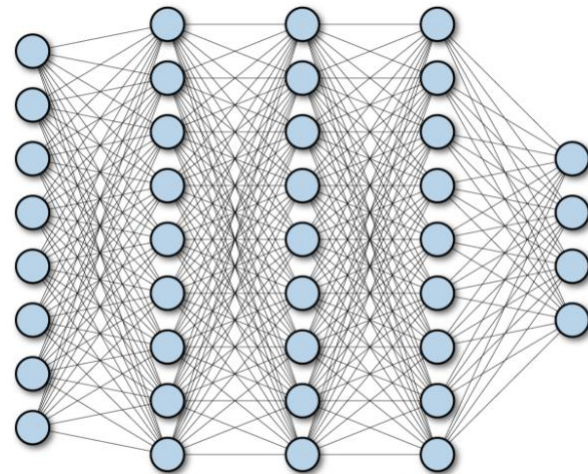
Max pool $\begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$

- Example



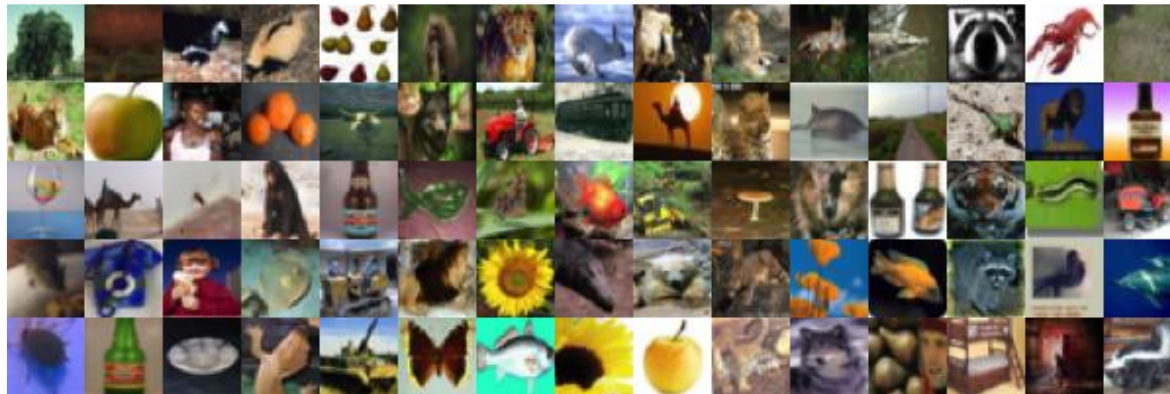
CNN; Fully connected

- Adding a **Fully-Connected layer** is a (usually) cheap way of **learning non-linear combinations of the high-level features** as represented by the output of the convolutional layer.
- To feed the output of the convolutional layers into a fully connected layer, we need to **flatten the image into a single-column vector**.
- The flattened output is fed to a feed-forward neural network and **backpropagation** is applied to every iteration of training.



CNN; Example

- The [CIFAR-100 dataset](#) (Canadian Institute for Advanced Research, 100 classes) is a subset of the tiny Images dataset.

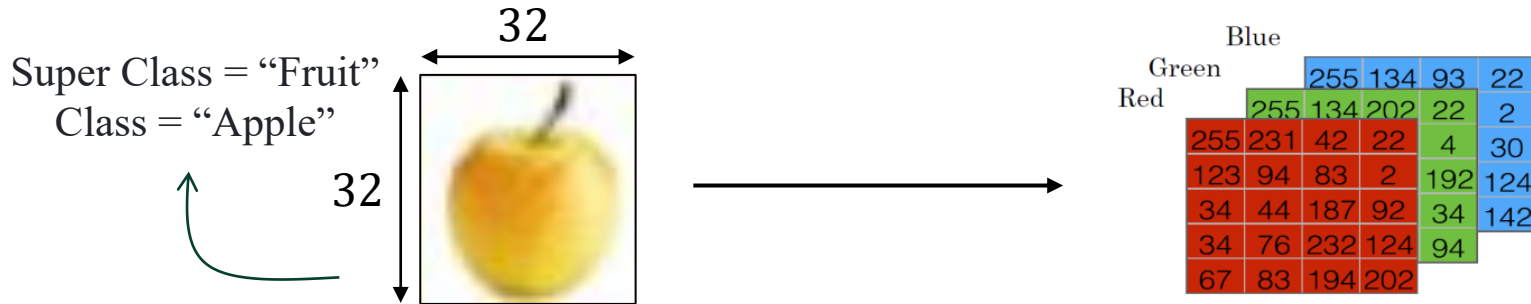


- The **100 classes** in the CIFAR-100 are grouped into **20 super classes**. There are 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). There are 500 training images and 100 testing images per class.

CNN; Example

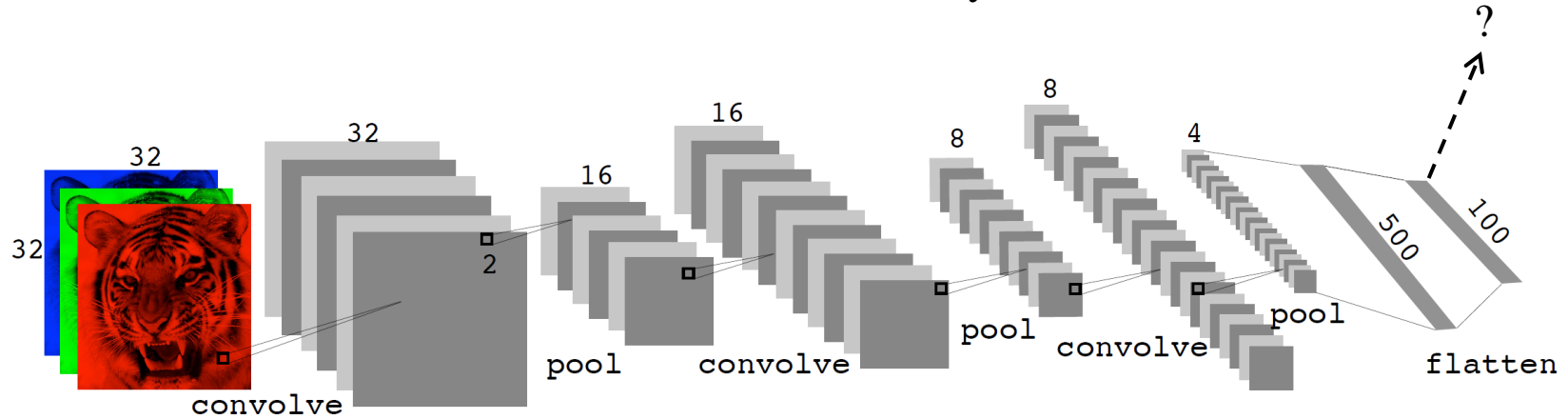
- Some super class and classes:
- 32×32 color pictures
- 50K training images
- 10K test images.
- Each image is a three-dimensional array or feature map:
 $32 \times 32 \times 3$ array of 8-bit numbers
- The last dimension represents the three-color channels for red, green and blue.

Super Class	Classes
fish	aquarium fish, flatfish, ray, shark, trout
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
people	baby, boy, girl, man, woman



CNN Architecture

- Filters are typically small, e.g., each channel 3 by 3.
- Each filter creates a new channel in convolution layer.



- Number of layers can be very large. E.g., [resnet50](#) trained on [ImageNet 1000-class image data base](#) has 50 layers.

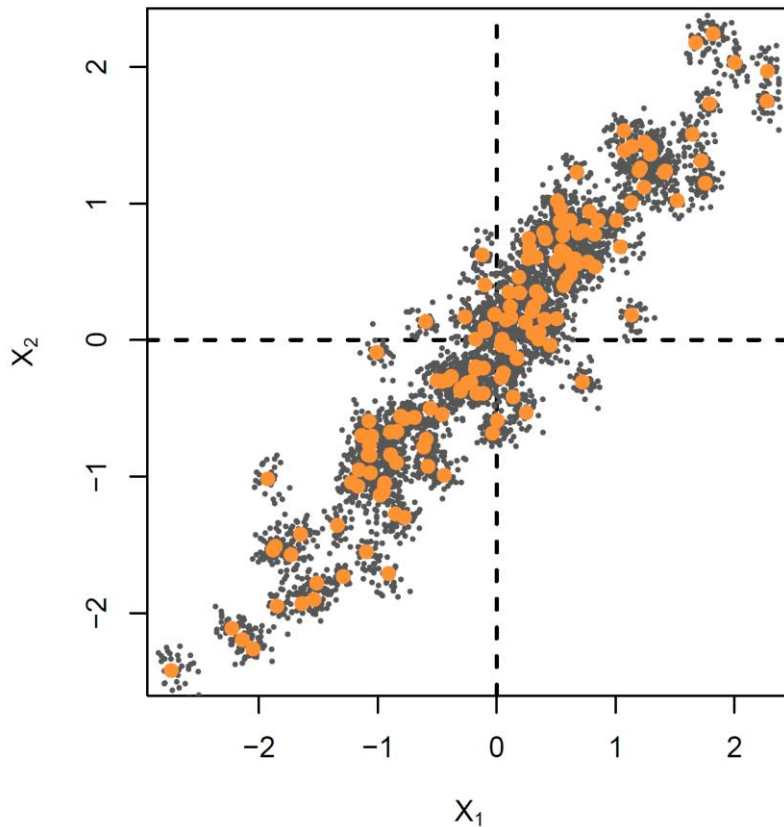
Data augmentation

- **Data augmentation** is the process of artificially **generating new data from existing data**.

{ Make many copies of each data point (x_i, y_i) and add a small amount of Gaussian noise to the x_i .

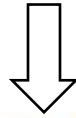
{ Leave the copies of y_i alone!

- This makes the model **robust** to small perturbations in x_i .



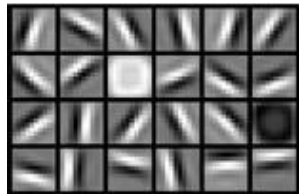
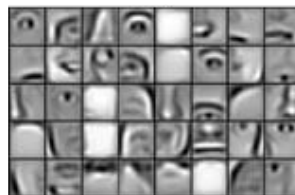
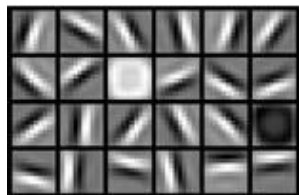
Data augmentation in CNN

- The **label** is left unchanged. All these pictures are still a tiger.



Transfer learning

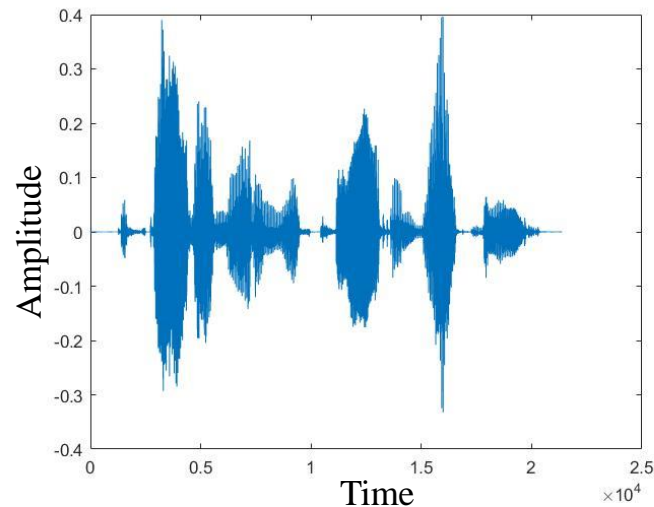
- **Transfer learning** is the **reuse of a pre-trained model** on a new problem.
- In **transfer learning**, a machine exploits the **knowledge gained from a previous task** to improve generalization about another.



Recurrent Neural Networks

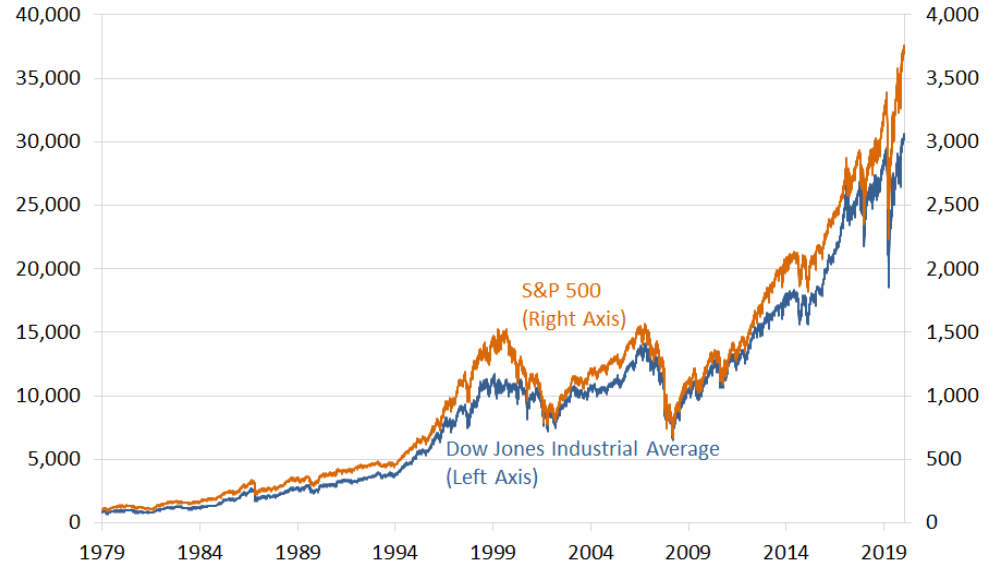
Sequential data

- **Sequential data** refers to a type of data where the order and sequence of the elements are significant.
- It is a collection of observations or events that are arranged in a specific order, and the relationship between the elements depends on their position in the sequence.
- Some examples of sequential data:
 - Documents are sequences of words
 - Time-series such as weather data
 - Financial time-series such as stock prices
 - Recorded speech or music
 - Handwriting, such as doctor's notes
 - DNA sequences in genomics
 - User clickstream data on a website



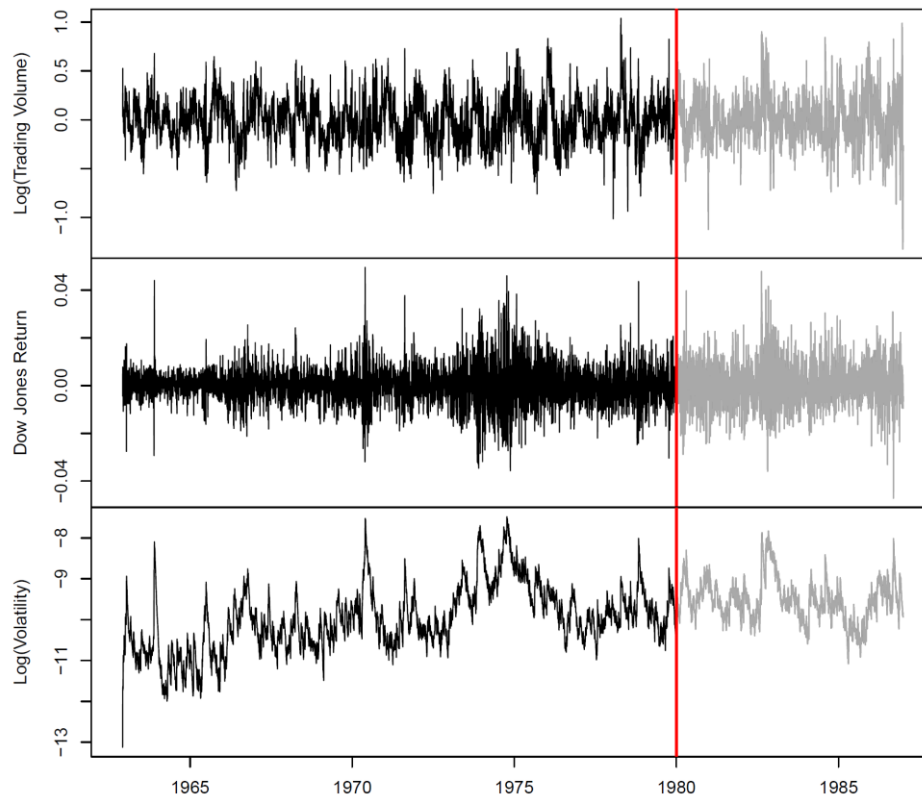
Sequential data; Time-series

- **Time-series data** refers to a **sequence of data points collected over successive time intervals**.
- Each data point is associated with a **specific timestamp** or time index, indicating when the measurement or observation was made.
- ✓ What are other examples?



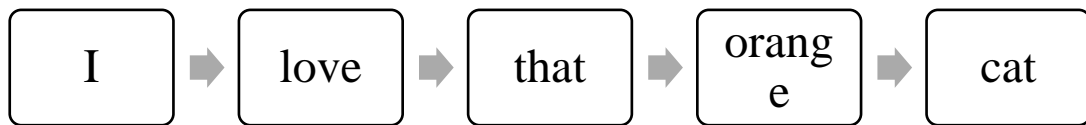
Train test split in time-series

- Splitting your time-series data for training and testing requires a different approach than a regular random split.
- It's crucial to ensure that the time-series model is **only exposed to past data points** during training.
- This reflects the real-world scenario where we don't have access to future information.



Sequential data; Text data

- **Textual data** refers to data that is **represented in the form of text**, such as sentences, paragraphs, or documents.
- Textual data is considered sequential data because the order of the words or characters in the text carries important meaning and context.
- The sequential nature of textual data implies that the meaning of a word or phrase can **depend on the words that precede or follow it**.



Text data; Example

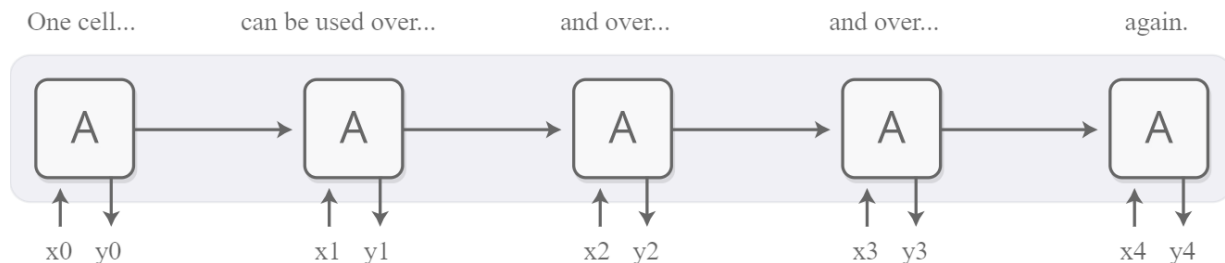
- The [IMDB corpus](#) consists of user-supplied movie ratings for a large collection of movies.
- Each has been labeled for sentiment as **positive** or **negative**. Here is the beginning of a **negative** review:

This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn ...

- We wish to build a classifier to predict the sentiment of a review.

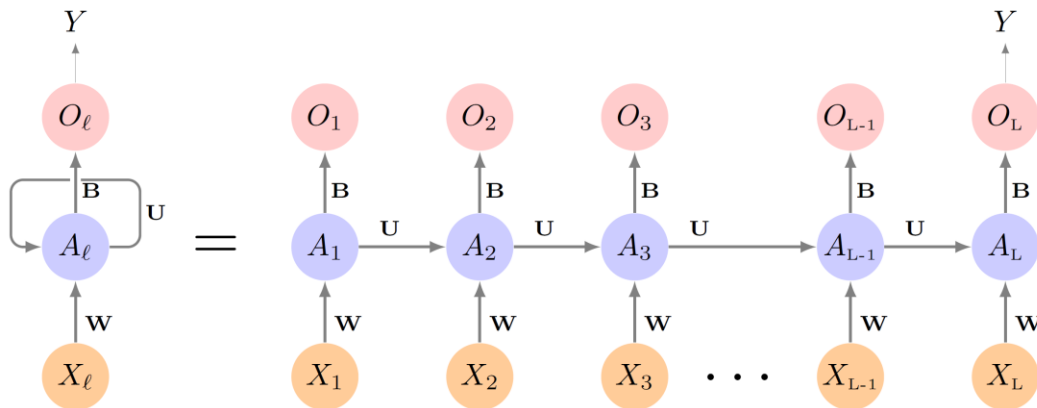
Why do we need another architecture?

- Feedforward neural networks require inputs of **fixed dimensions**. We would need to fix the length of the input text by either **truncating** or **padding** it.
- This approach discards or artificially extends the text, which can lead to information loss or introduce irrelevant information.
- Feedforward neural networks do **not consider the order of words** in the input.
- **Recurrent Neural Networks (RNNs)** build models that consider this **sequential nature of the data** and build a memory of the past.



RNN Architecture

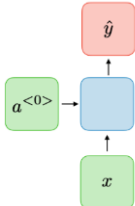
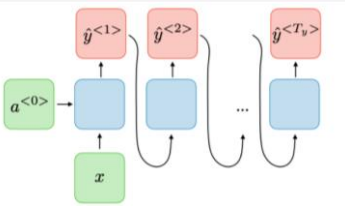
- RNNs use the same weights for each element of the sequence, decreasing the number of parameters and allowing the model to generalize to sequences of varying lengths.



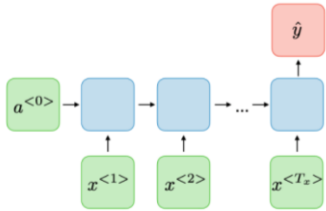
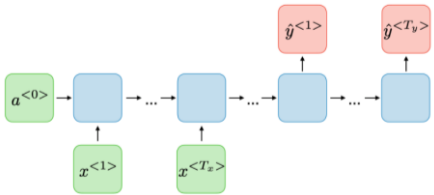
- The same weights W, U , and B are used at each step in the sequence, leading to the term “**recurrent**”.
- The downside is that the **computation process is slow**.

Different Types of RNNs

- **RNN models** are mostly used in the fields of natural language processing (NLP) and speech recognition.

Type of RNN	Illustration	Example
One-to-one		Traditional Neural Networks
One-to-many		Text Generation

Different Types of RNNs

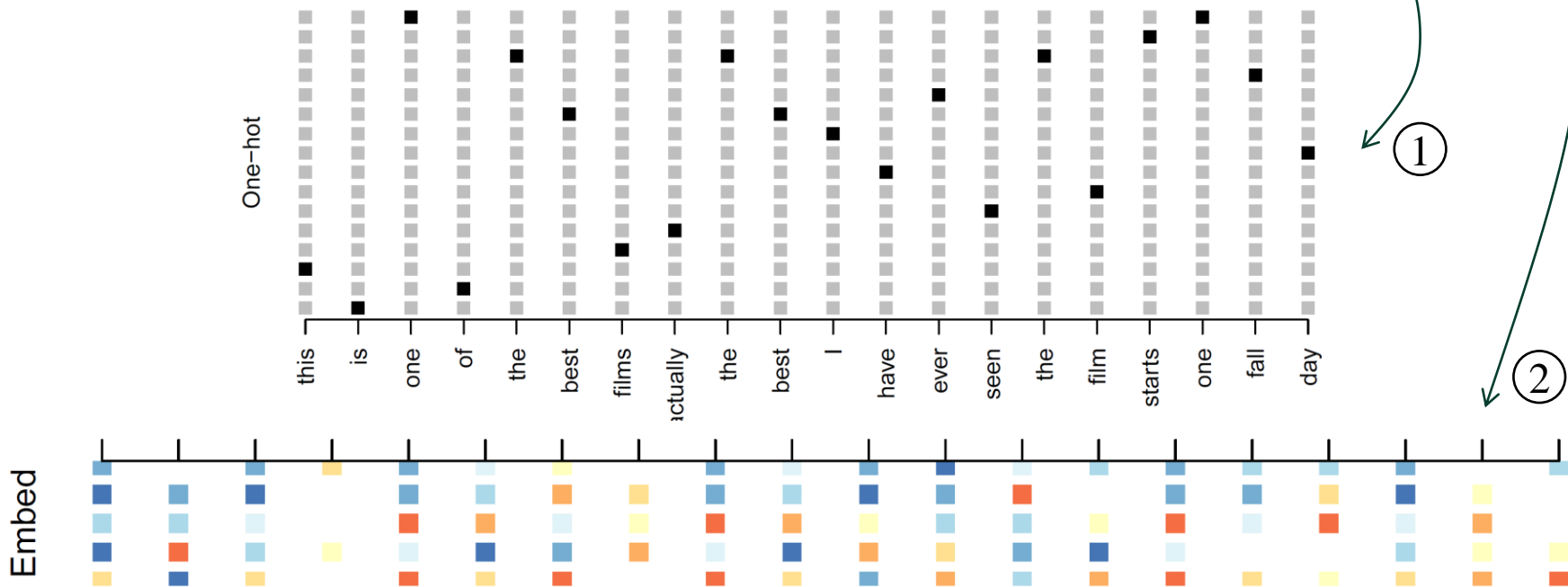
Type of RNN	Illustration	Example
Many-to-one		Text Classification
Many-to-many		Machine Translation

Word Embedding

- In our sentiment prediction problem, we can represent each word as a **one-hot encoded binary vector** (dummy variable) of **length 10K**, with all zeros and a single one in the position for that word in the dictionary.
- This results in an extremely **sparse feature representation** and would not work well.
- Instead, we use a lower-dimensional pretrained word embedding matrix E .
- This reduces the binary feature vector of length $10k$ to a real feature vector of dimension $m \ll 10k$.
- **Embeddings** are **pretrained on very large corpora of documents**, using methods like principal components. [word2vec](#) and [GloVe](#) are popular.

Word Embedding

“this is one of the best films actually the best I have ever seen. The film starts one fall day.”



long short-term memory(LSTM)

- The basic RNN design **struggles with longer sequences**, but a special variant -“**long short-term memory**”- networks can even work with these.
- Such models have been found to be very powerful, achieving remarkable results in many tasks including translation, voice recognition, and image captioning.
- Training an **LSTM** model on the IMDB dataset can yield high accuracy (around **85-90%**) in sentiment prediction.



- ✓ Which type of RNN is applicable here?

Image captioning; Example

- First, a **ConvNet** processes the image, **extracting high-level features**.
- Then an **RNN** runs, **generating a description** of the image.
- As it generates each word in the description, the RNN focuses on the ConvNet's interpretation of the relevant parts of the image.



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.