LECTURER: Nghia Duong-Trung

DEEP LEARNING

UNIT 2

Network Architectures

STUDY GOALS

– After completing this unit you will be able to ...

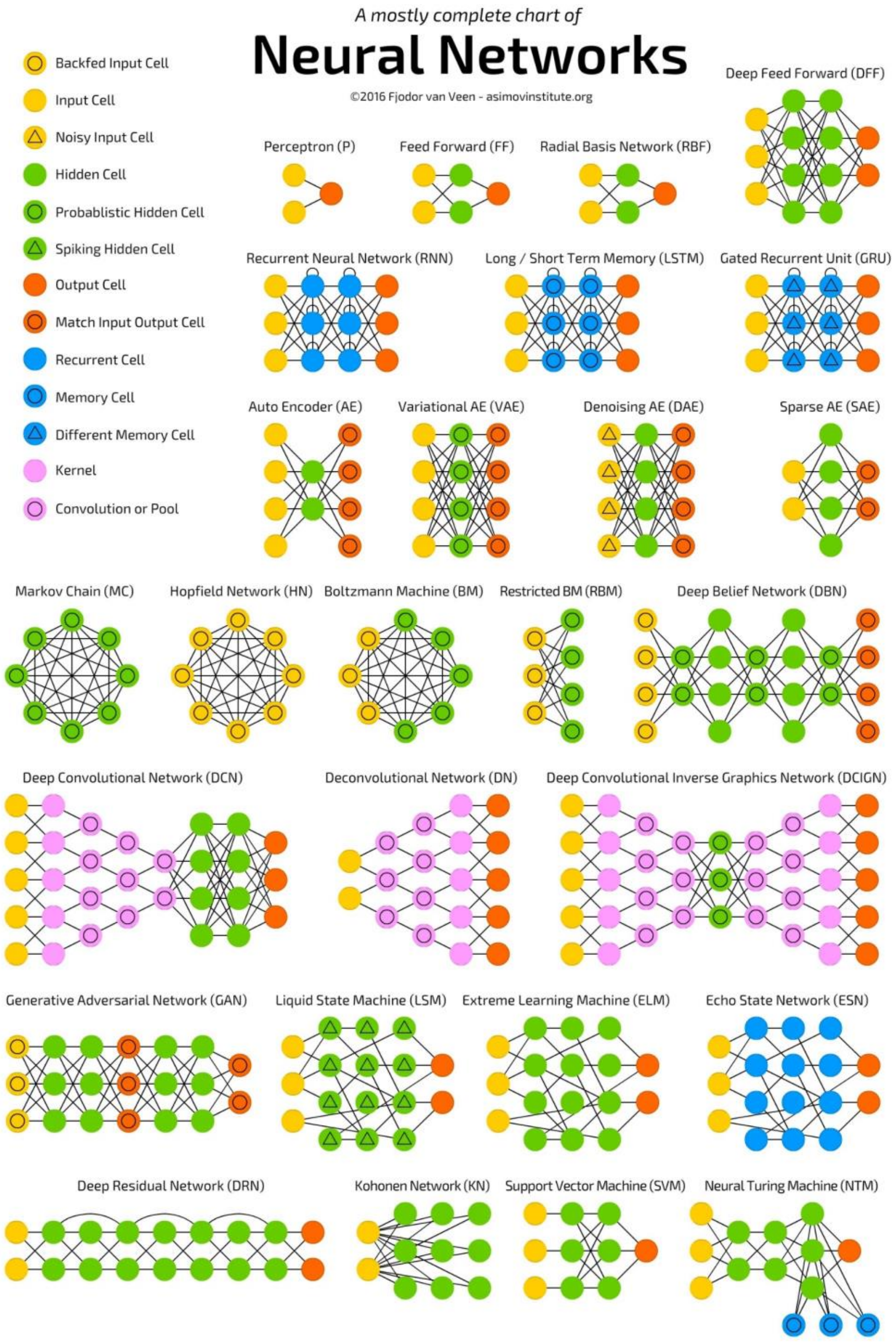
… identify the most important neural network architectures.

… explain what feed-forward, convolutional, and recurrent networks are.

… describe why convolutional neural networks are ideal for image analysis.

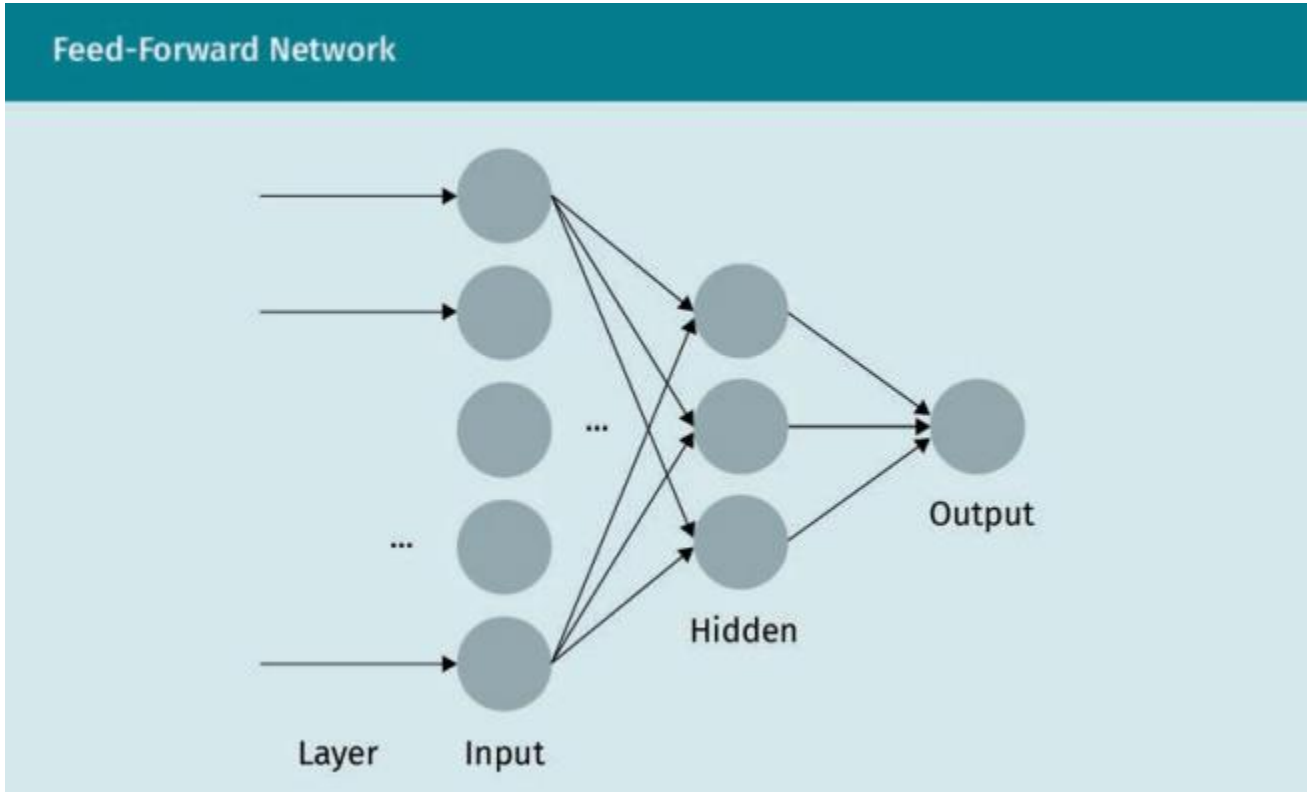
… use recurrent neural networks to encode long sequences of events

CHART OF NEURAL NETWORKS



https://www.asimovinstitute.org/neural-network-zoo/

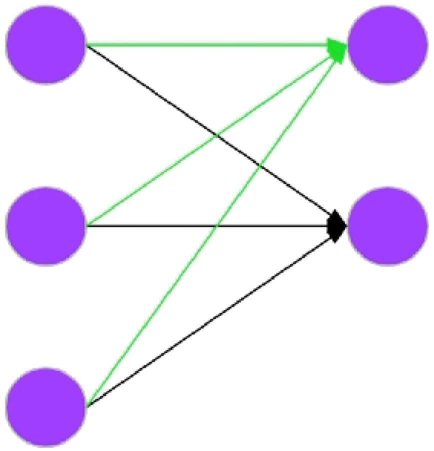
- A model is for making predictions.
- How do we make predictions with a neural network?
- We refer to this as “going in the forward direction”

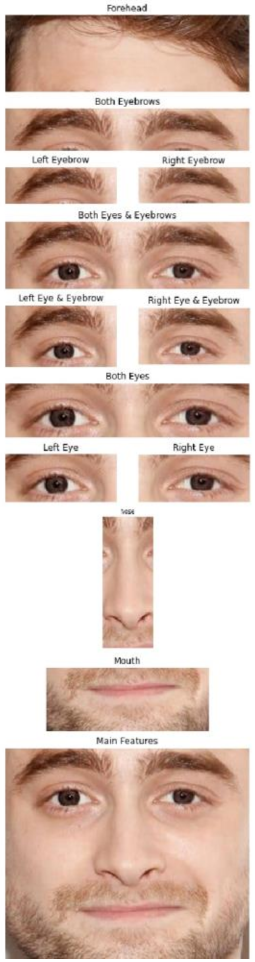


Source of the text: Zöller, 2022.

REPEATING THE SINGLE NEURON

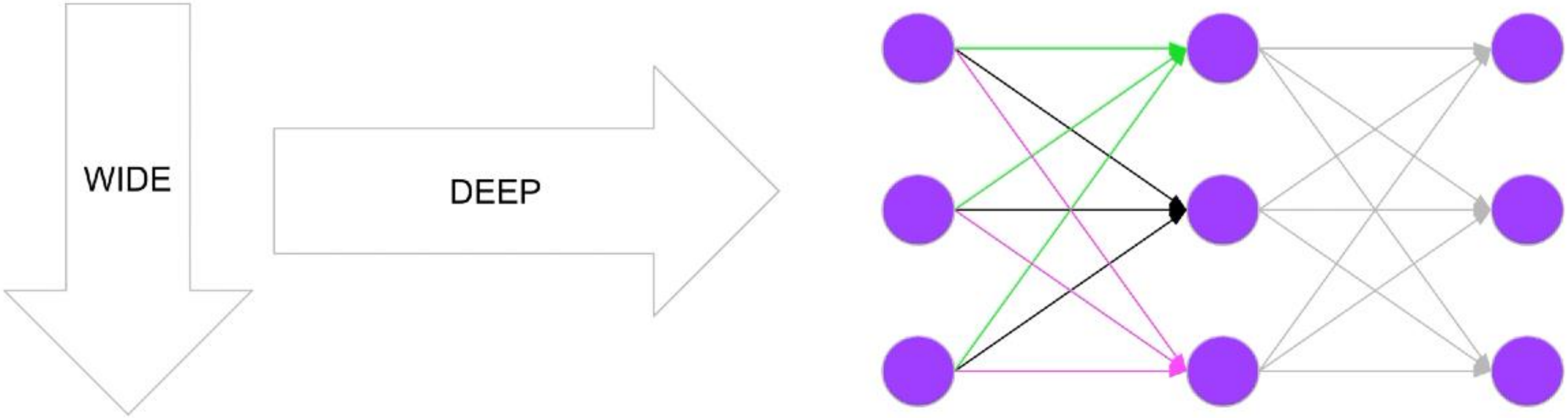
- Each of these neurons may calculate something different
  - Via different weights
- E.g. input is a face
  - One neuron looks for the presence of an eye
  - One neuron looks for the presence of a nose
- They are looking for different features



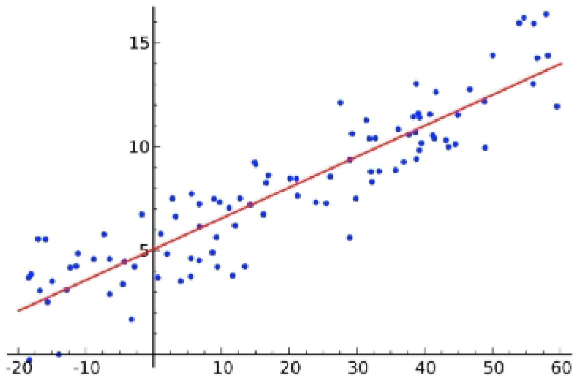


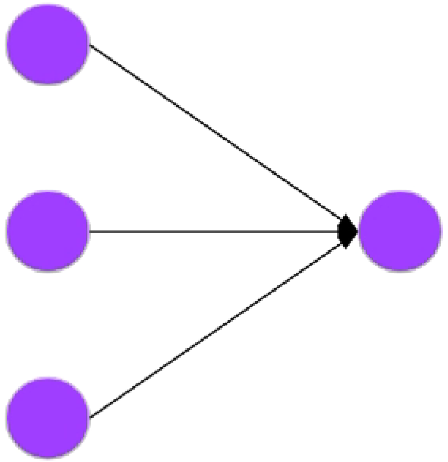
https://github.com/zementalist/Facial-Features-Extraction

- Key insight: we can stack more layers of neurons: a chain of neurons
- We can model the brain as uniform structure
- 2 important ways to extend the single neuron
  - The same inputs can be fed to multiple different neurons, each calculating something different (more neurons per layer)
  - Neurons in one layer can act as inputs to another layer

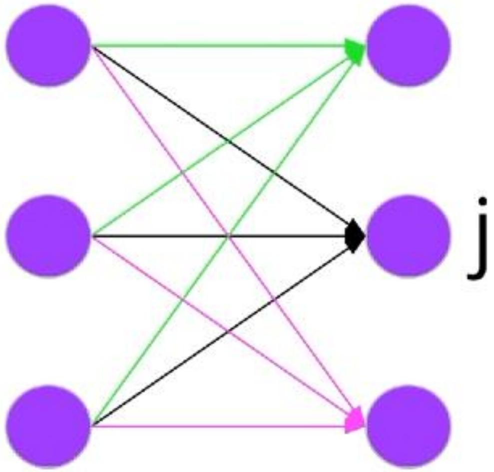


LINES TO NEURONS

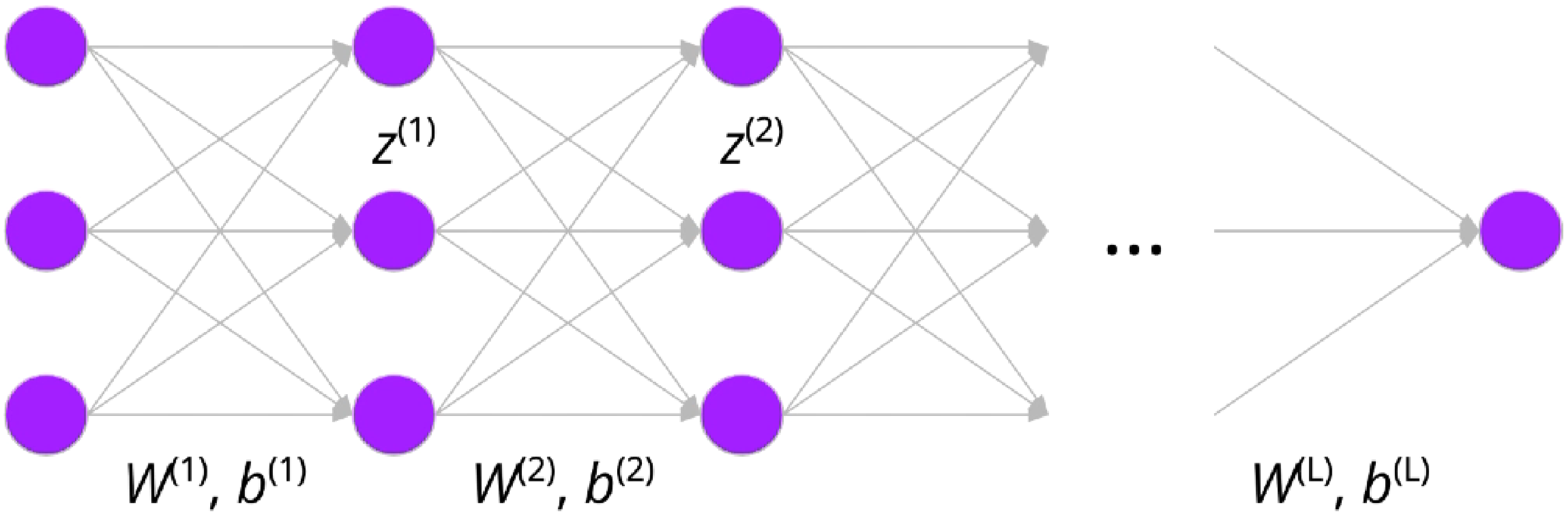




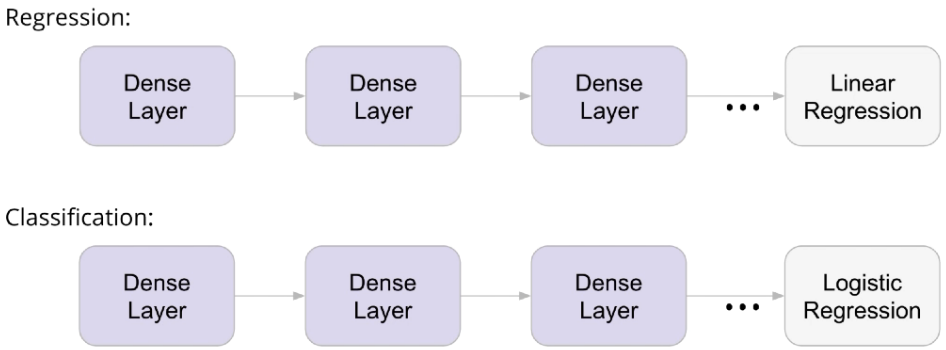
- A line: $ax + b$
- A neuron: $\sigma(w^T x + b)$
- Multiple neurons per layer
  - Call the output of the jth neuron zj: $z_j = \sigma(w^T x + b)$, for I = 1…M
  - Vectorize the neuron: $z = \sigma(W^T x + b)$
  - Shapes:
    - z is a vector of size M
    - x is a vector of size D
    - W is a matrix of size D x M
    - b is a vector of size M
    - $\sigma()$ is an element-wise operation



INPUT TO OUTPUT FOR AN L-LAYER NEURAL NETWORK



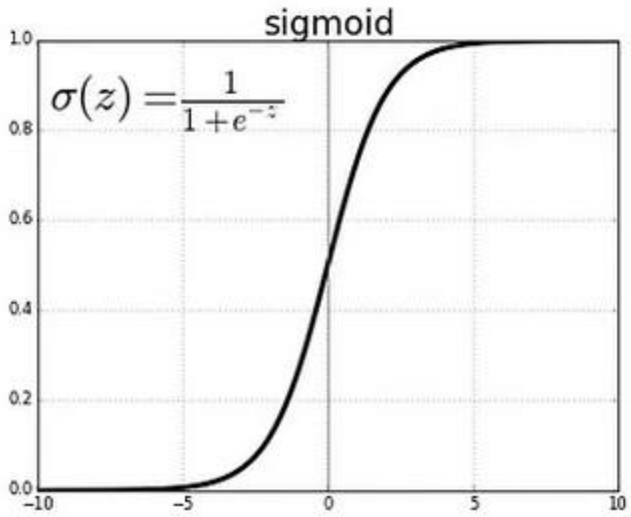
$$z^{(1)} = \sigma(W^{(1)T} x + b^{(1)})$$
$$z^{(2)} = \sigma(W^{(2)T} z^{(1)} + b^{(2)})$$
$$z^{(3)} = \sigma(W^{(3)T} z^{(2)} + b^{(3)})$$
$$\dots$$
$$p(y = 1 \mid x) = \sigma(W^{(L)T} z^{(L-1)} + b^{(L)})$$

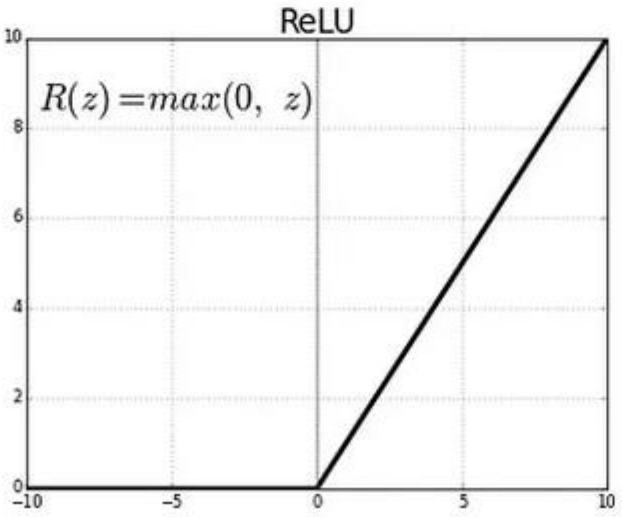


- It’s just a thing function that you use to get the output of node. It is also known as Transfer Function.
- It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).
- The Activation Functions can be basically divided into 2 types:
  - Linear activation function
  - Non-linear activation fuction
    - The Nonlinear Activation Functions are mainly divided on the basis of their range or curves
    - Sigmoid or Logistic Activation Function
    - Tanh or hyperbolic tangent Activation Function
    - ReLU (Rectified Linear Unit) Activation Function
    - Leaky ReLU

Source of the text: Zöller, 2022.

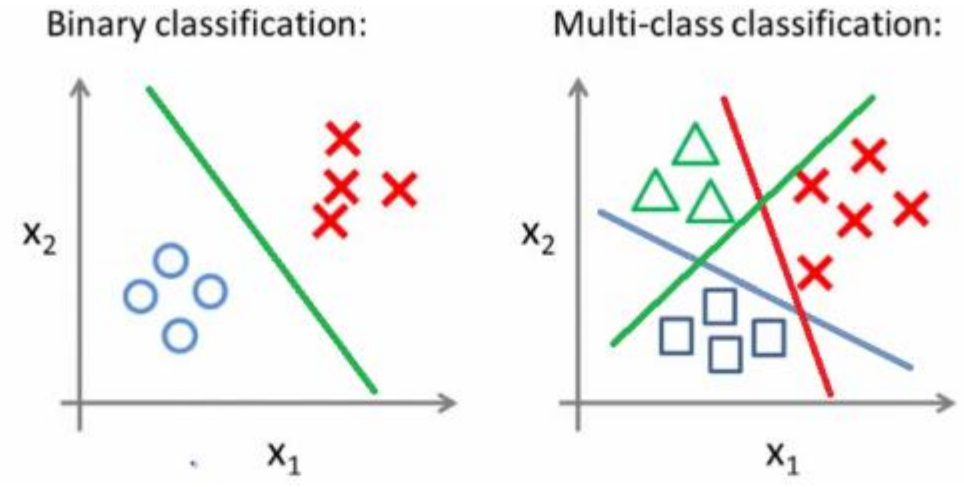
- Most people use ReLU as a reasonable default.
- Sometimes, you’ll find LReLU and ELU benefit.
- ML is experimentation, not philosophy.
- Never use your mind to guess the output of a computer program.
-





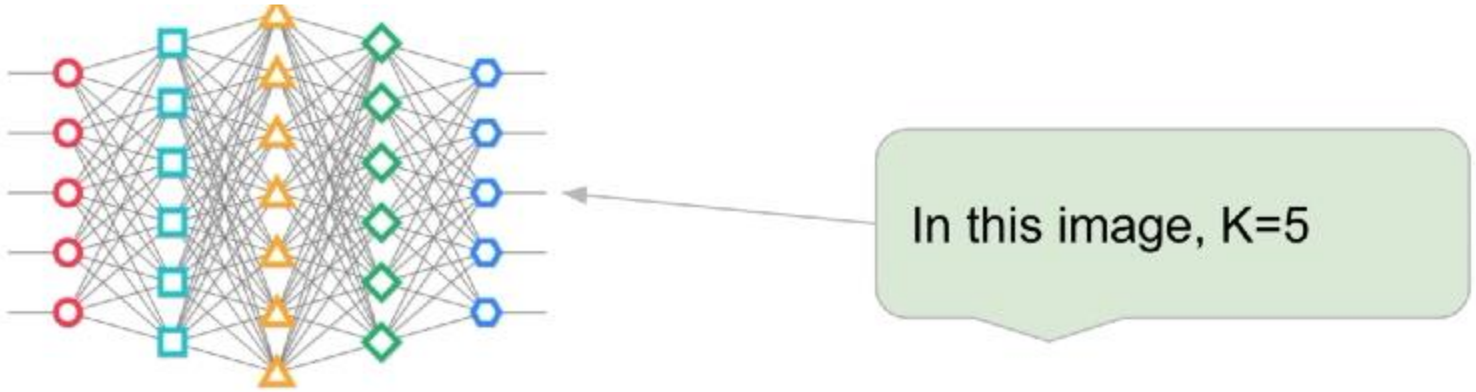
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

- For binary classification, we use a sigmoid at the output.
- We replaced sigmoids with ReLUs in the hidden layers.
- For output, sigmoid is still the right choice for binary classification.



https://utkuufuk.com/2018/06/03/one-vs-all-classification/

- Suppose we calculate the value right before applying the final activation function.
- $a^{(L)}$ is a vector of size K (for a K-class classifier)
- How do we turn this vector into a set of probabilities for each of the K classes?
  - $a^{(L)} = W^{(L)T} z^{(L-1)} + b^{(L)}$
- We need a probability distribution over K distinct values
  - Requirement 1: $p(y = k \mid x) \geq 0$
  - Requirement 2: $\sum_{k=1}^{K} p(y = k \mid x) = 1$



THE SOFTMAX FUNCTION

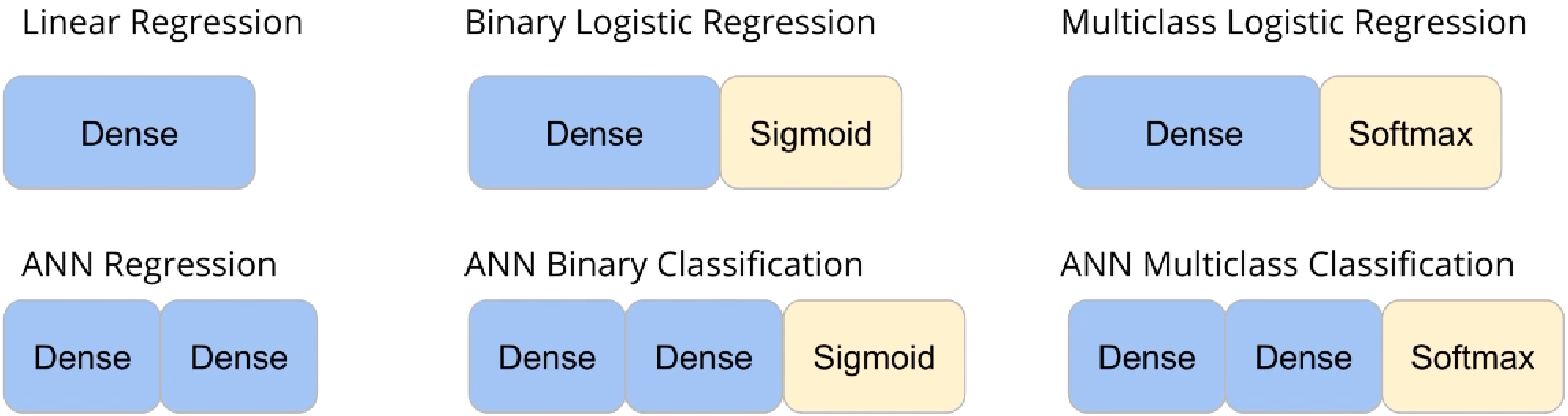
- Drop the superscript (L) for convenience.
- This function meets both of previous mentioned requirements.
  - Exp(any number) is positive
  - The denominator is the sum of all possible values of the numerator
    - $p(y = k \mid x) = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$
    - $\sum_{k=1}^{K} p(y = k \mid x) = \sum_{k=1}^{K} \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)} = 1$
  - The softmax is technically an activation function
  - CrossEntropyLoss already combines softmax + cross entropy

Task	Activation Function
Regression	None / Identity
Binary classification	Sigmoid
Multiclass classification	Softmax

```
model = nn.Linear(D, K)
criterion = nn.CrossEntropyLoss()
```

```
nn.Sequential(
  nn.Linear(D, M),
  nn.ReLU(),
  nn.Linear(M, K),
  nn.Softmax()
)
```

- Same pattern applies to CNNs, RNNs – the type of task corresponds only to the final activation function



HOW TO REPRESENT IMAGES

- We need 3 dimensions: height, width, color
- Color dimension always has size 3, corresponding to the RGB channels
- A(I,j,k) stores the value of the i'th, j'th column, k'th color
  - k = red, green, blue
- Physically, color is light, measured by light intensity
- It's a continuous value
- More precision -> Image takes up more space
- We've figured out that 8 bits (1 byte) is good enough
- $2^8 = 256$ possible values (0,1,2,…255)
  - This gives us 2^8 2^8 2^8 = 16.8 million possible colors
- How much space does a 500x500 image take up?
  - 500 x 500 x 3 x 8 = 6 million bits -> 750,000 bytes -> 732KB JPEG

GRAYSCALE IMAGES

- Images that do not have color can be simplified.
- We call them grayscale because each pixel value can only be black, white, or some shade of gray
- Black = 0, White = 255
- Only requires a 2-D array (height, width)

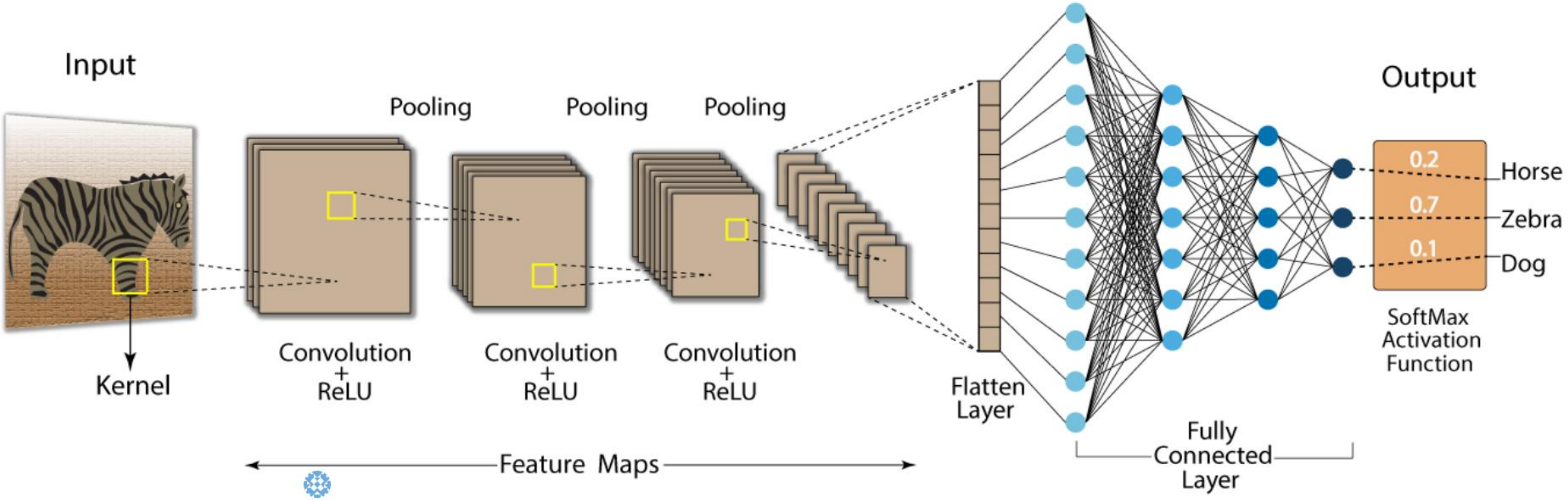
ANN CODE PREPARATION

- Recap of the steps
  - #1 Load in the data
  - #2 Build the model
  - #3 Train the model
  - #4 Evaluate the model
  - #5 Make predictions

TRANSFER TASK

- Try the notebook 02. PyTorch_ANN_MNIST.ipynb

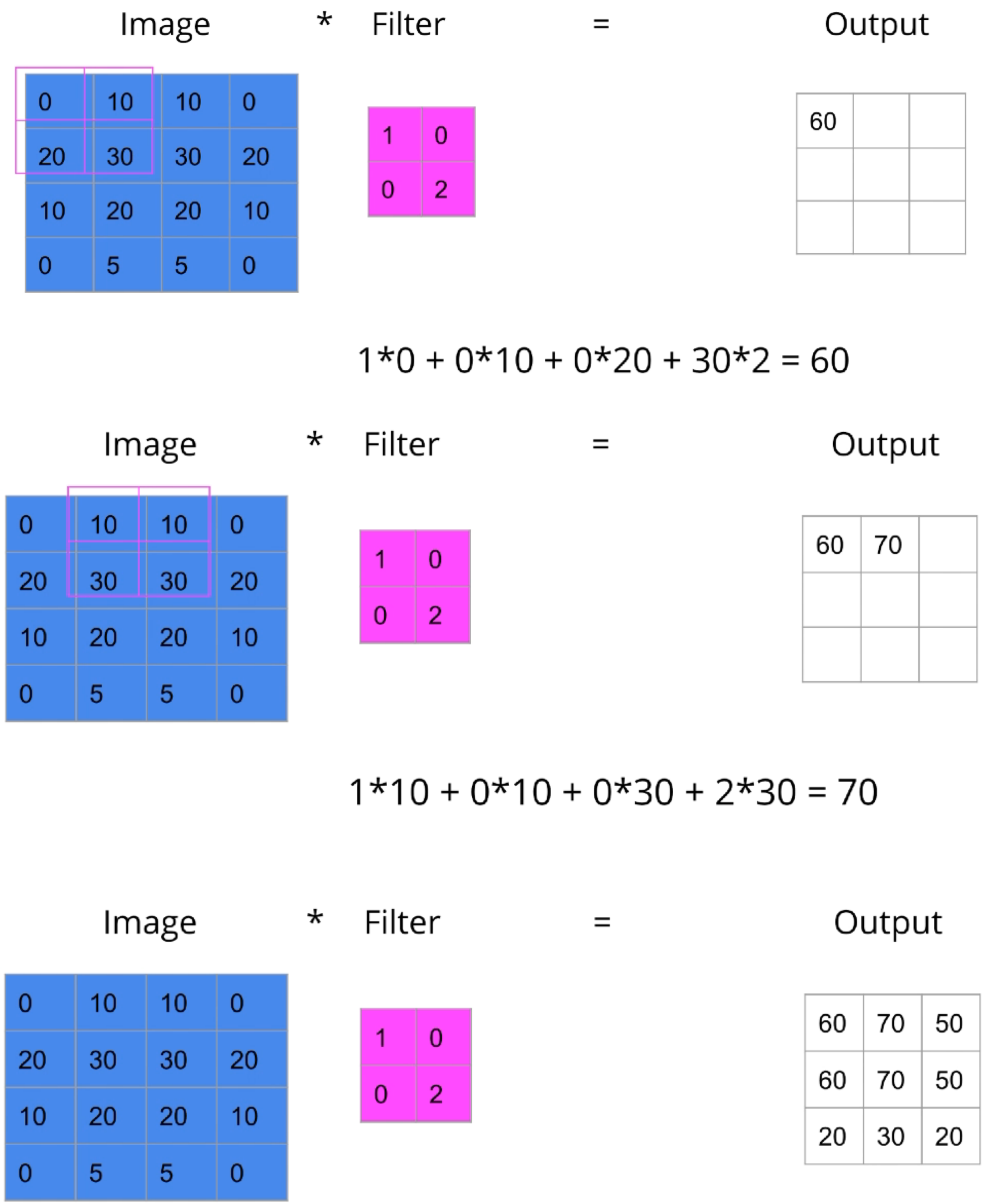
- A CNN is a “neural network with convolution”
  - There are only 2 requirements: add and multiply
- Convolution = image modifier = pattern finding
  - The filter



https://developersbreach.com/convolution-neural-network-deep-learning/

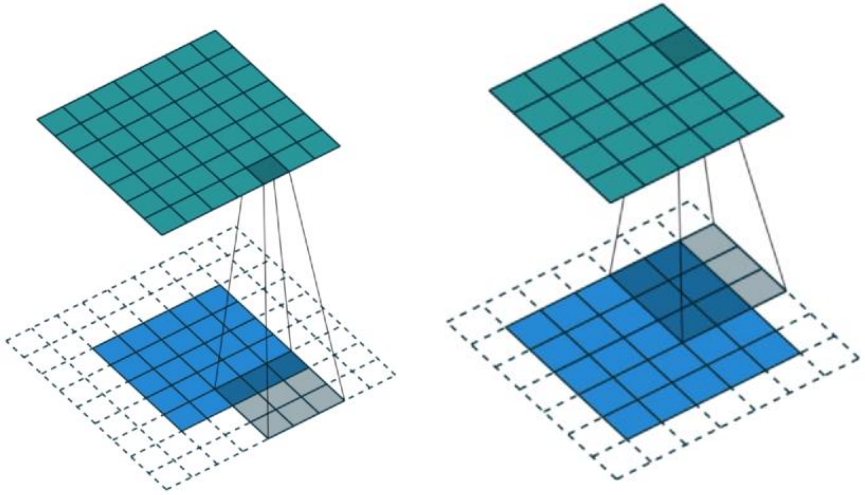
THE MECHANICS OF CONVOLUTION

- Given: input_image, kernel
- Output_height = input_height – kernel_height + 1
- Output_width = input_width – kernel_width + 1
- Images themselves are not square – that’s just the nature of cameras and screens.
- Some neural nets use square images for convenience
- Kernels/filters are almost always square
- Translational invariance
  - We want the same filter to look at all locations in the image



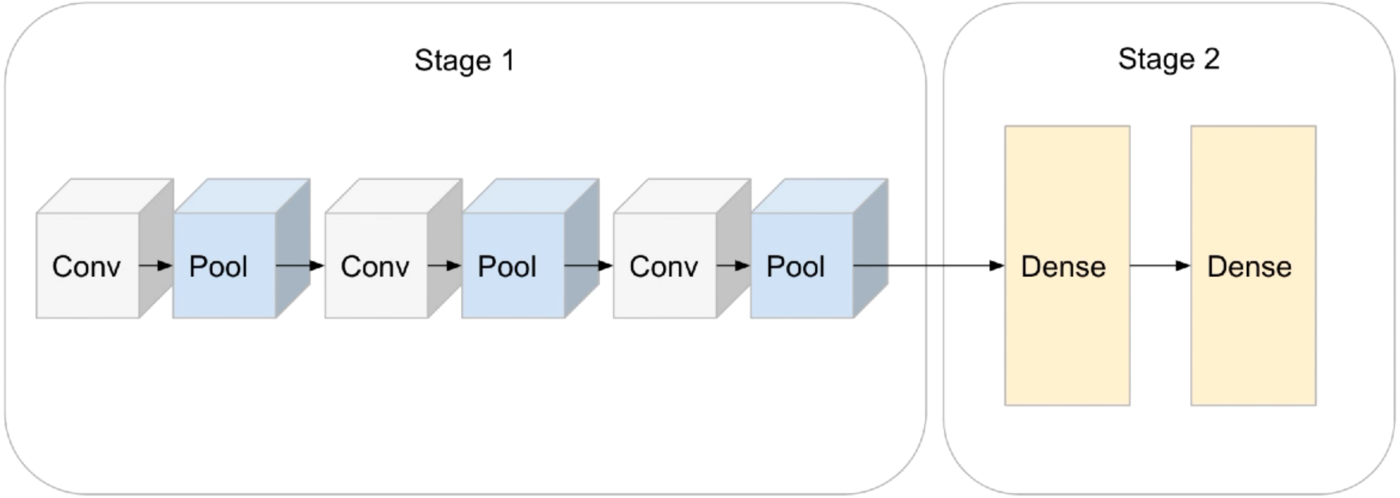
PADDING

- What if we want the output to be the same size as the input?
- Then we can use padding (add imaginary zeros around the input)
- We could extend the filter further and still get non-zero outputs
- Full padding:
  - Input length = N
  - Kernel length = K
  - Output length = N + - 1



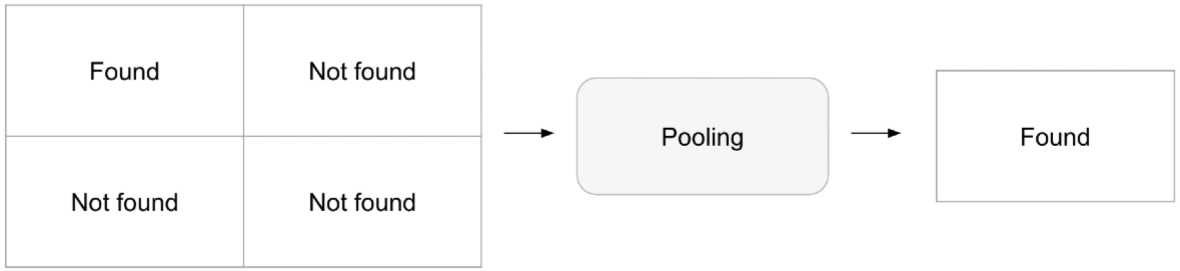
Mode	Output Size	Usage
Valid	N - K + 1	Typical
Same	N	Typical
Full	N + K - 1	Atypical

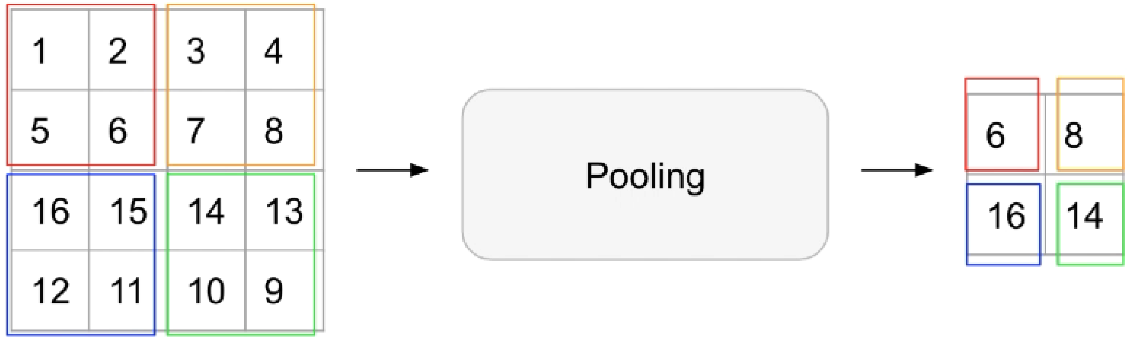
- Hyperparameters
  - Learning rate, # hidden layers, # hidden units per layer
  - With CNNs, the conventions are pretty standard
    - Small filters relative to image, e.g., 3x3, 5x5, 7x7
    - Repeat: convolution -> polling -> …
    - Increase #feature maps: e.g., 32 -> 64 -> 128

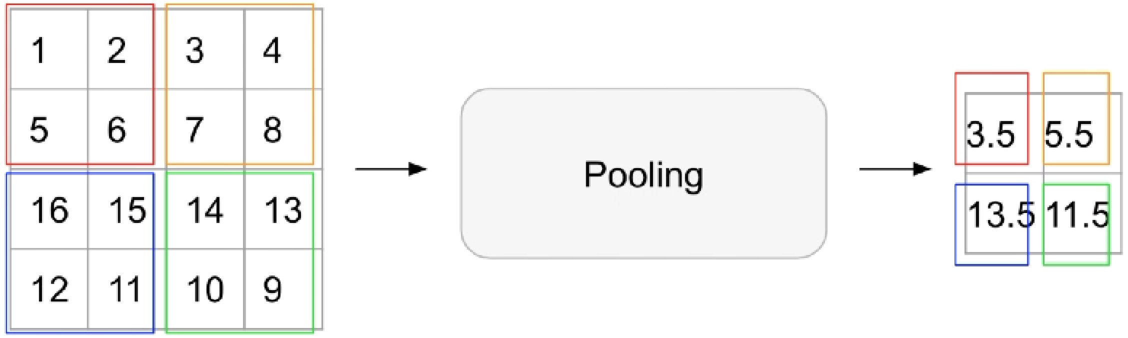


POOLING

- There are 2 kinds of pooling: max and average
- Practical: if we shrink the image, we have less data to process
- Translational invariance: I don’t care where in the image the feature occurred, I just care that it did



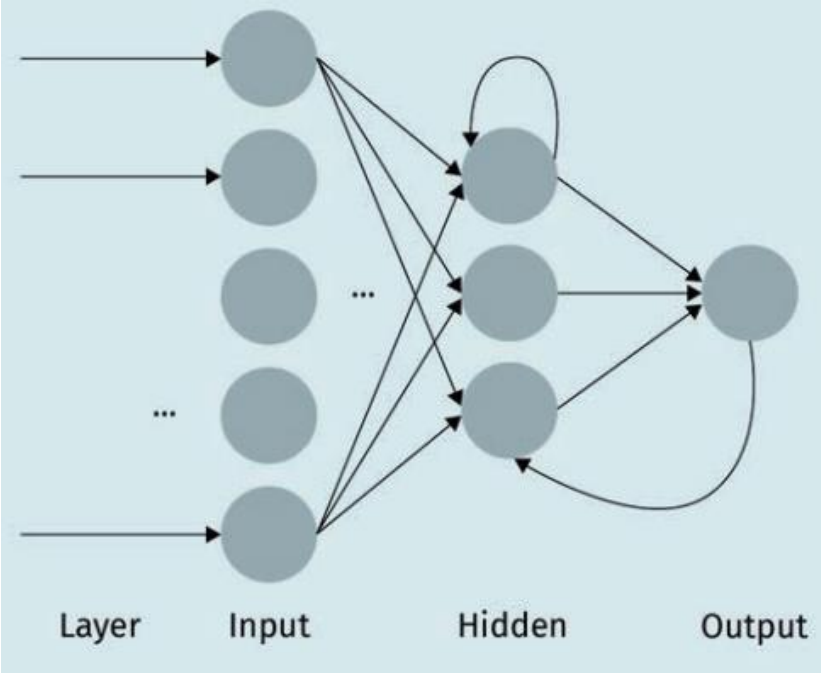




TRANSFER TASK

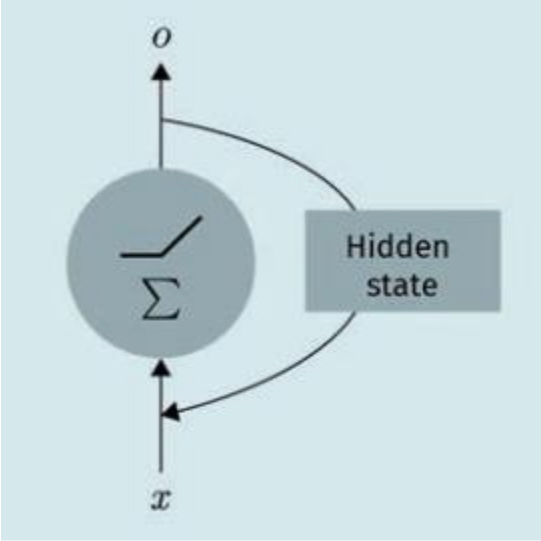
- Run the notebook CNN\01. PyTorch_CNN_Fashion_MNIST.ipynb

- RNN allows connections not only to the next layer but also to previous layers, the current layer, or even onto the neurons themselves
- This allows RNNs to develop a memory that can, for example, be used for language generation
- This means that the output of the neuron now depends on both the previous inputs to the neuron and the output it has calculated



MEMORY CELLS

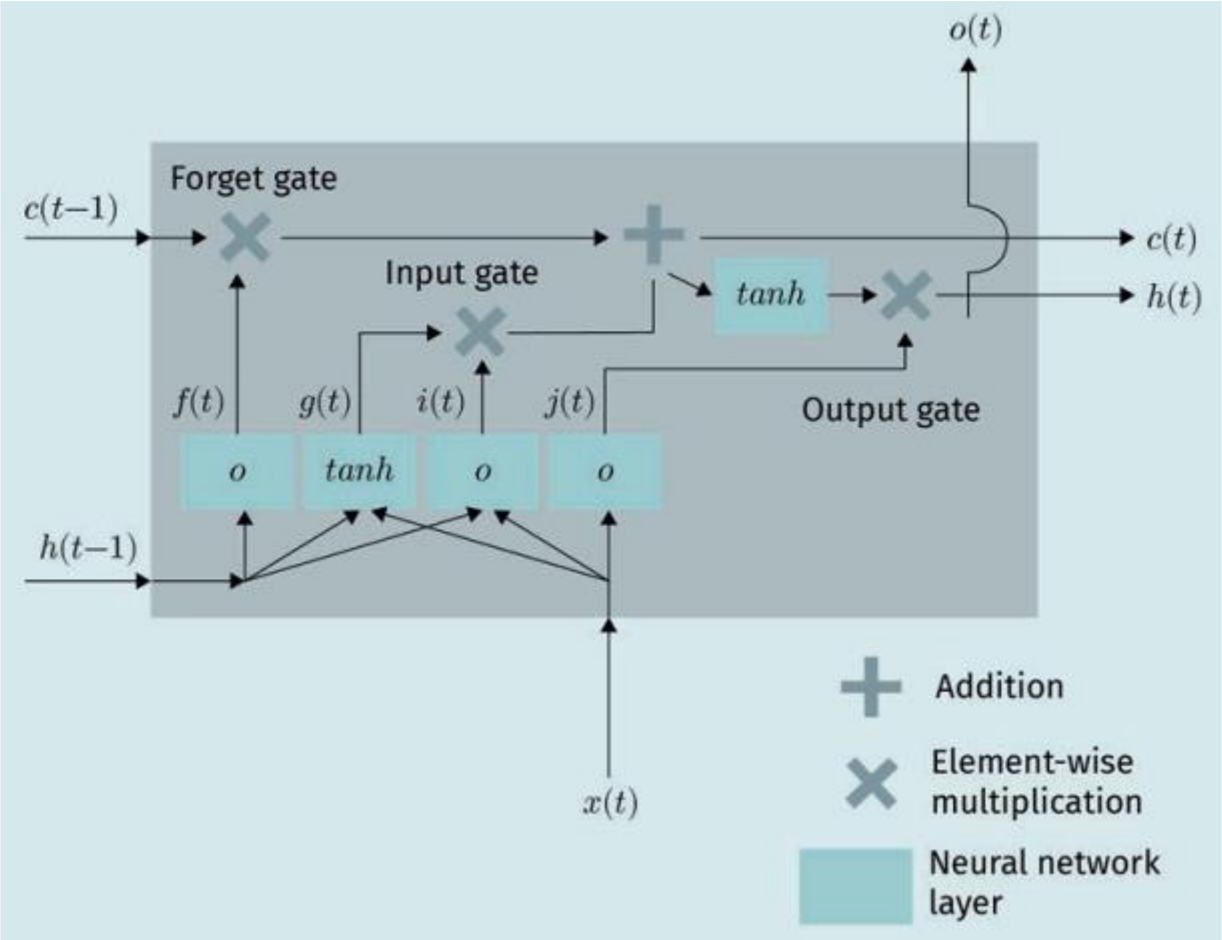
- In the simple recurrent neuron or layer of recurrent neurons, we sent the output computed by the neuron(s) back as additional input.
- However, we can make the recurrent neuron more powerful by extending this approach and including a hidden state into the loop, sending the output back and creating a memory cell.
- This hidden state is characterized by some function h(t), which depends on the previous inputs, earlier outputs of the neuron, and the hidden state at earlier steps



- One of the main challenges of using recurrent neural networks is learning long sequences, such as a very long sentence or a poem.
- Learning such long sequences requires that we take more than one previous output into account, i.e., the output does not only depend on time steps t and t - 1, but also steps t - 2, t - 3, …, t - n. This makes the training of RNNs very difficult and unstable.
- By creating a sophisticated hidden state, the RNN can perform much better.
- The general idea is to create a memory structure (M) and decide during the training process which parts of the previously-seen input data should be remembered and which parts of the memory should be forgotten, as they are no longer relevant.
  - This then forms a candidate for the new memory (M')
- We can express this as M(t) = remember(t) * M(t – 1) + add(t) * M’(t).

LSTM CELL

- Compared to the simple recurrent cell, the hidden state is split into two functions: c(t) and h(t).
  - c(t): the long-term hidden state
  - h(t): the short-term hidden state
- This is done via a series of *gates*



- After completing this unit you will be able to ...

  … identify the most important neural network architectures.

  … explain what feed-forward, convolutional, and recurrent networks are.

  … describe why convolutional neural networks are ideal for image analysis.

  … use recurrent neural networks to encode long sequences of events

SESSION 1

Network Architectures