# Deep learning:

## *Feed-forward & Convolutional neural networks*

Sebastian N. Deleuran & Yat-Tsai Richie Wan

Immunoinformatics and Machine Learning (IML) group

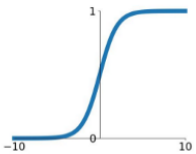
# Recap: Feed-forward Networks (FFNs)

**Keypoints:**

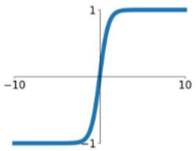
- Weights connecting every node together
- **activation functions** after layers

## Activation Functions

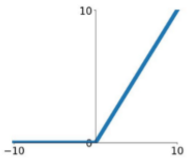
**Sigmoid**
$\sigma(x) = \frac{1}{1+e^{-x}}$

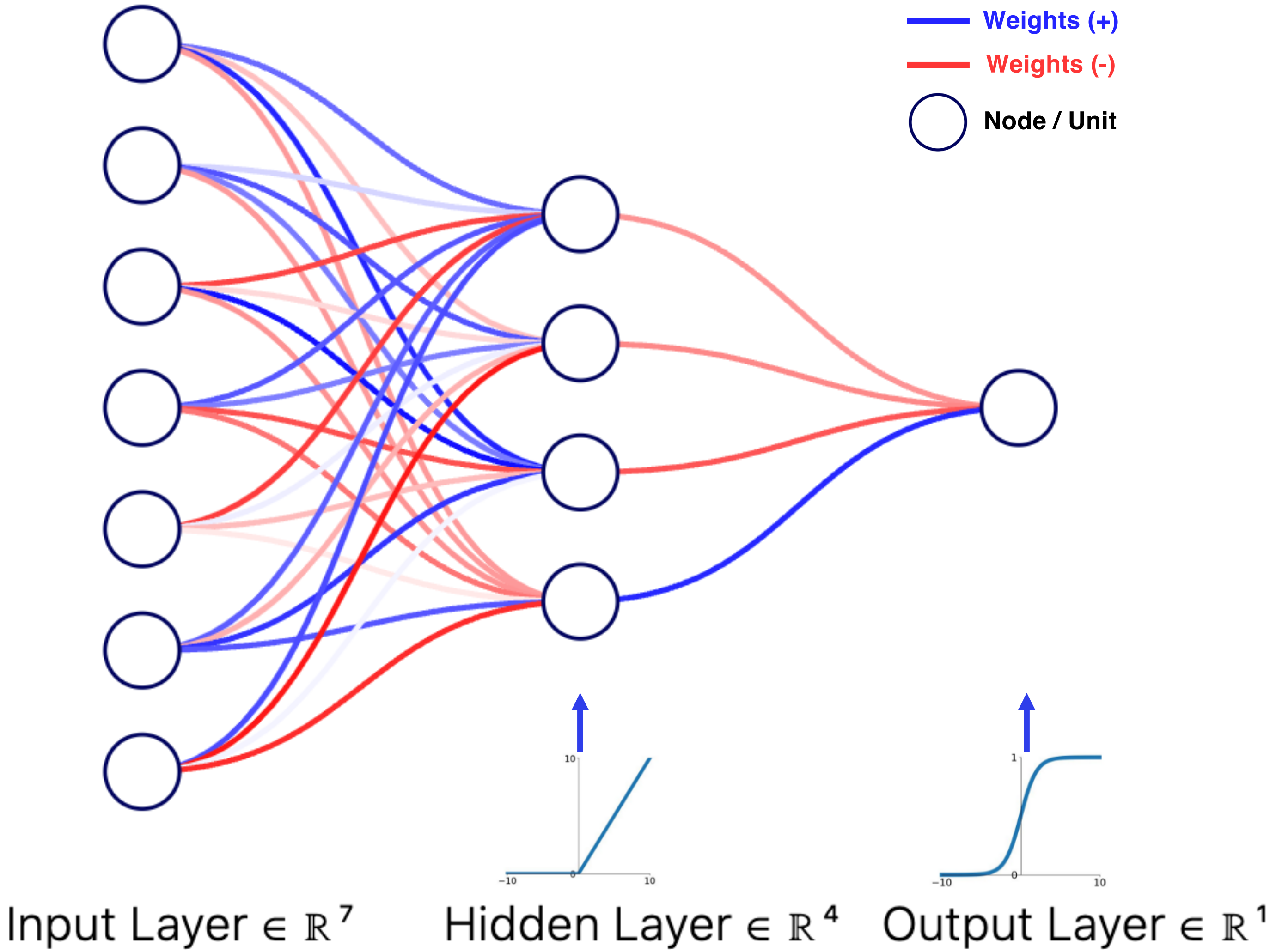


**tanh**
$\tanh(x)$



**ReLU**
$\max(0, x)$





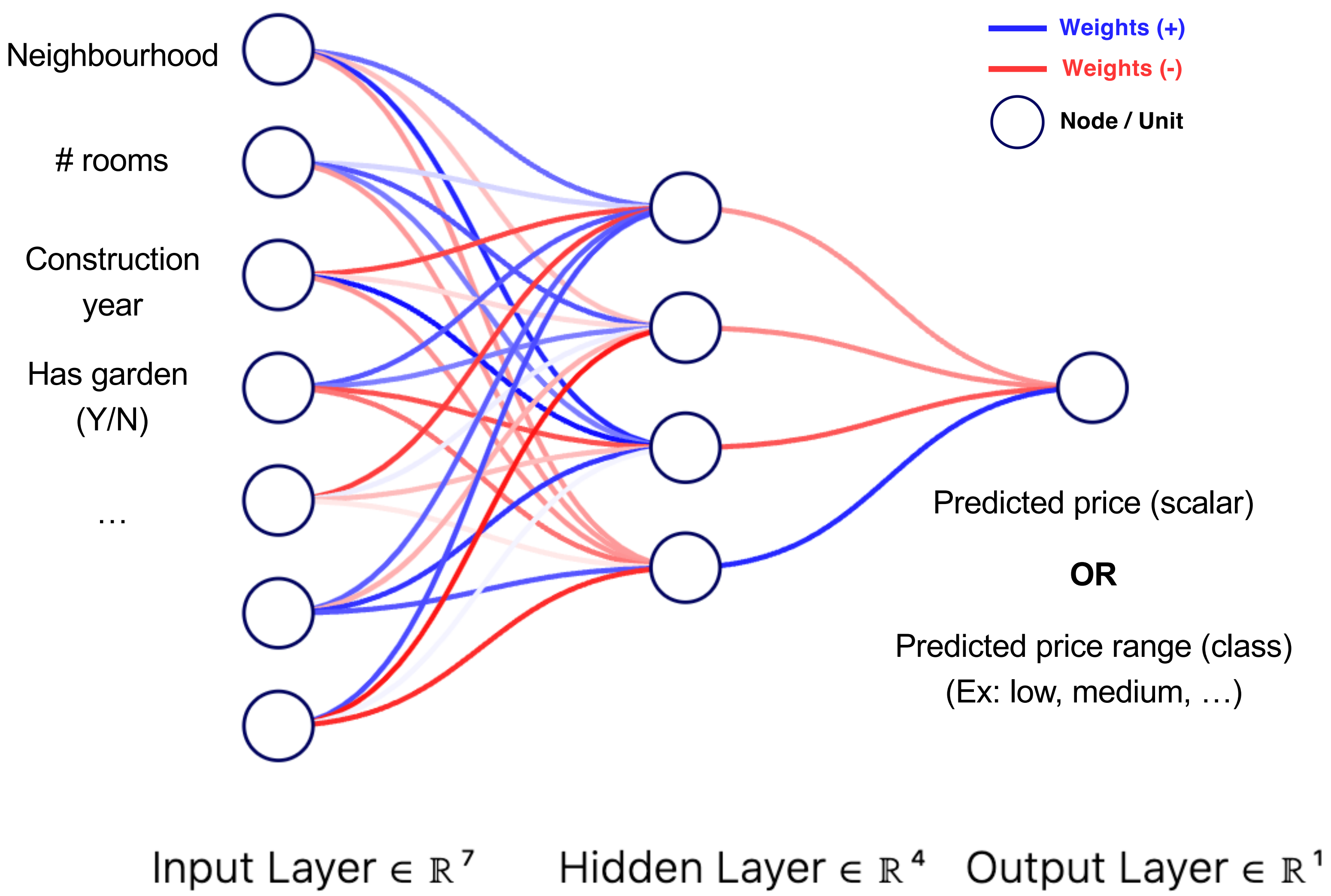
# Recap: Feed-forward Networks (FFNs)

**Use features to :**

- **predict (scalar values)**
- **classify (labels)**

**Ex:**

**Predict the price of a house based on features**



# Accelerating your models

- Yesterday we implemented a neural network using for loops.
- However: Python for loops are extremely slow!
- For larger models we need to use more efficient implementations.

# Accelerating your models

- Python for loops are extremely slow.
- For larger models we need to use more efficient implementations.
- Deep learning libraries with C++ backends provide this. For example:
    - PyTorch
    - TensorFlow
    - JAX + NumPy

# Accelerating your models

- Python for loops are extremely slow.
- For larger models we need to use more efficient implementations.
- Deep learning libraries with C++ backends provide this. For example:
  - PyTorch
  - TensorFlow
  - JAX + NumPy
- Today we will use the vectorization capabilties of NumPy to speed up our models.
- The goal: make few but large matrix operations.

# Using vectorized matrix operations to speed up processing

- We can process an input in terms of matrix operations.

- Yesterday:

```
def forward(X):
      for j in range(hidden_layer_dim):
            z = 0.0
            for i in range(input_layer_dim+1):
                  z += X[0][i]* w1[i,j]
            X[1][j] = sigmoid(z)
```

# Using vectorized matrix operations to speed up processing

- We can process an input in terms of matrix operations.

- Yesterday:

```
def forward(X):
      for j in range(hidden_layer_dim):
            z = 0.0
            for i in range(input_layer_dim+1):
                  z += X[0][i]* w1[i,j]
            X[1][j] = sigmoid(z)
```

# Using vectorized matrix operations to speed up processing

- We can process an input in terms of matrix operations.

- Today:

```
def forward(x):
        # First layer
        z1 = np.dot(x, W1)
        a1 = activation(z1)

        # Output layer
        z2 = np.dot(a1, W2)
        a2 = activation(z2)
```

# Batching data to speed up processing

- We can process batches of data at the same time using NumPy broadcasting.

- Yesterday:

```
predictions = []
for peptide_sequence in dataset:
    y_pred = forward(peptide_sequence)
    predictions.append(y_pred)
```

# Batching data to speed up processing

- We can process batches of data at the same time using NumPy broadcasting.

- Yesterday:

```
predictions = []
for peptide_sequence in dataset:
    y_pred = forward(peptide_sequence)
    predictions.append(y_pred)
```

# Batching data to speed up processing

- We can process batches of data at the same time.

- Today:

```
predictions = forward(dataset)
```

Encoding of input data (peptides)

ALAKAAAAM
ALAKAAAAN
ALAKAAAAR
ALAKAAAAT
ALAKAAAAV
GMNERPILT
GILGFVFTM
TLNAWVKVV
KLNEPVLLL
AVVPFIVSV

0.9 0.05 0.05 …
0.9 0.05 0.05 ..

N_input

| | | | | |
|---|---|---|---|---|
| x1_1 | x1_2 | x1_3 | x1_4 | x1_5 |
| x2_1 | x2_2 | x2_3 | x2_4 | x2_5 |
| … | | | | |
| xn_1 | xn_2 | xn_3 | xn_4 | xn_5 |

N_datasize/batch

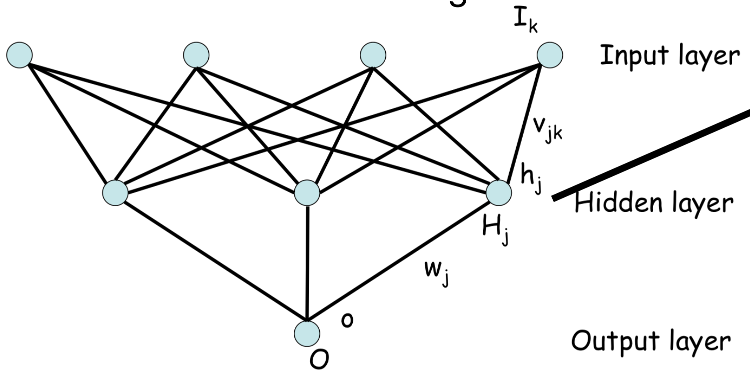
# Batching data to speed up processing

- We can process batches of data at the same time.

- Today:

```
predictions = forward(dataset)
```

Network weights




N_input

| Neuron_1 | N2. | N3. | ….. |
|---|---|---|---|
| w1_1 | w1_2 | w1_3 | |
| w2_1 | w2_2 | w2_3 | |
| w3_1 | w3_2 | w3_3 | |
| w4_1 | w4_2 | w4_3 | |
| w5_1 | w5_2 | w5_3 | |
| … | | | |
| … | | | |
| … | | | |
| … | | | |
| … | | | |

# Batching data to speed up processing

- We can process batches of data at the same time.
- We can apply the weights of all hidden neurons to the input in a single operation.

n

Inp_1. Inp_2 Inp 5

| | | | | | |
|---|---|---|---|---|---|
| x1 | x1_1 | x1_2 | x1_3 | x1_4 | x1_5 |
| x2 | x2_1 | x2_2 | x2_3 | x2_4 | x2_5 |
| …. …. | … | | | | |
| xn | xn_1 | xn_2 | xn_3 | xn_4 | xn_5 |

x

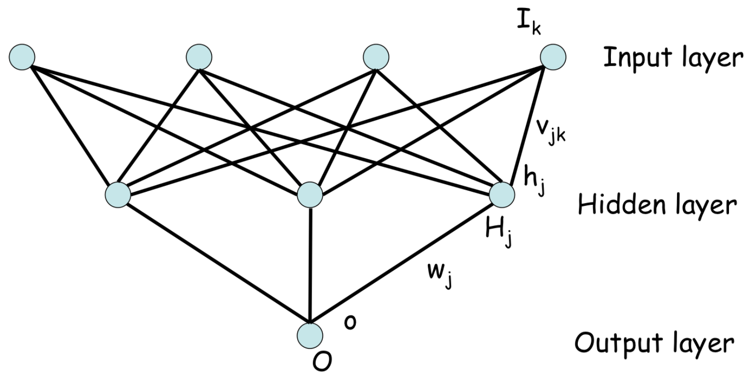
Neuron_1 N2. N3. …..

| | | | |
|---|---|---|---|
| I1 | w1_1 | w1_2 | w1_3 |
| I2 | w2_1 | w2_2 | w2_3 |
| …. | w3_1 | w3_2 | w3_3 |
| …. | w4_1 | w4_2 | w4_3 |
| I5 | w5_1 | w5_2 | w5_3 |

=

Neu1 N2 N3 …..

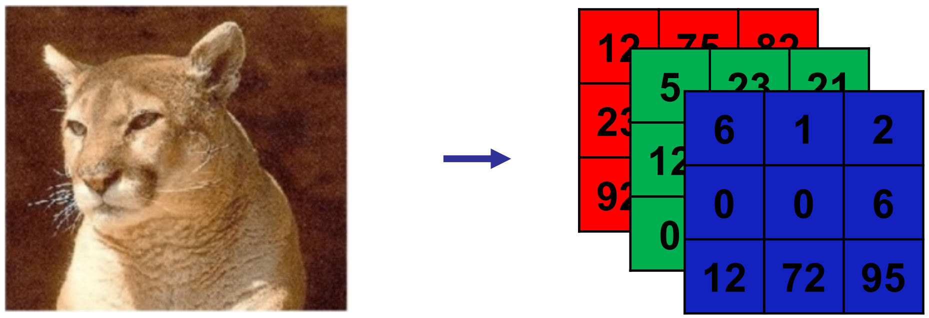
| | |
|---|---|
| x1 | H[1][1] ….. |
| x2 | |
| …. …. | |
| xn | |

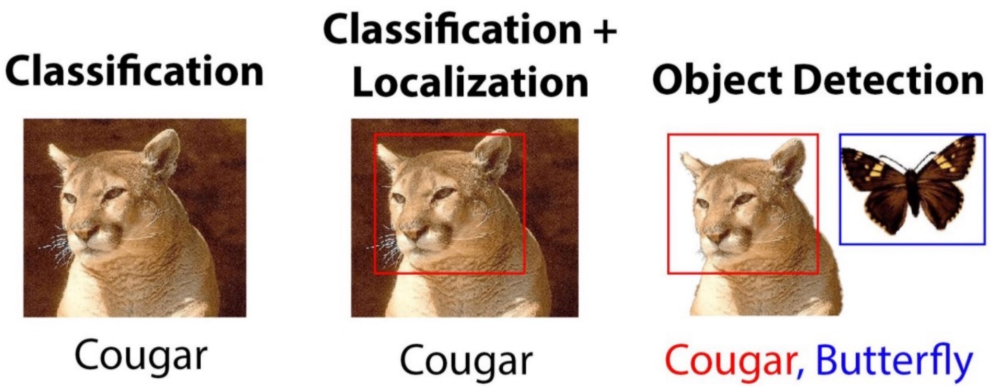


# Different inputs?

## FFN do not perform well on more complex inputs and tasks:

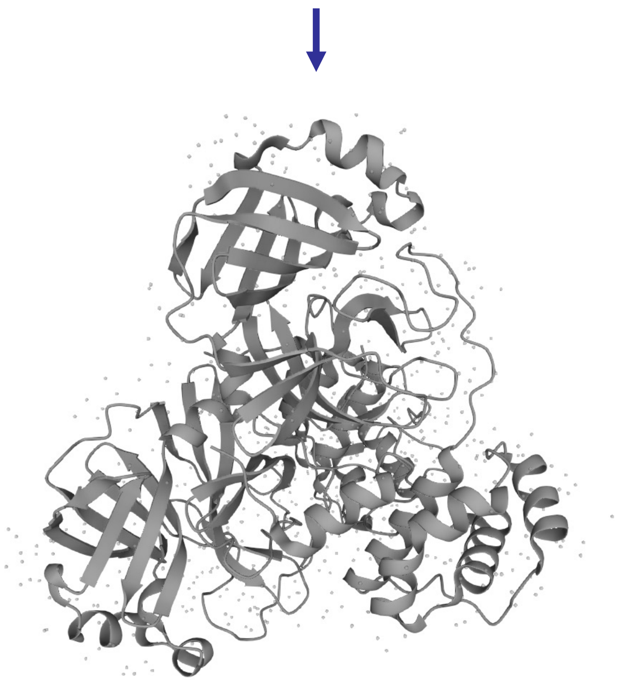
### Computer vision



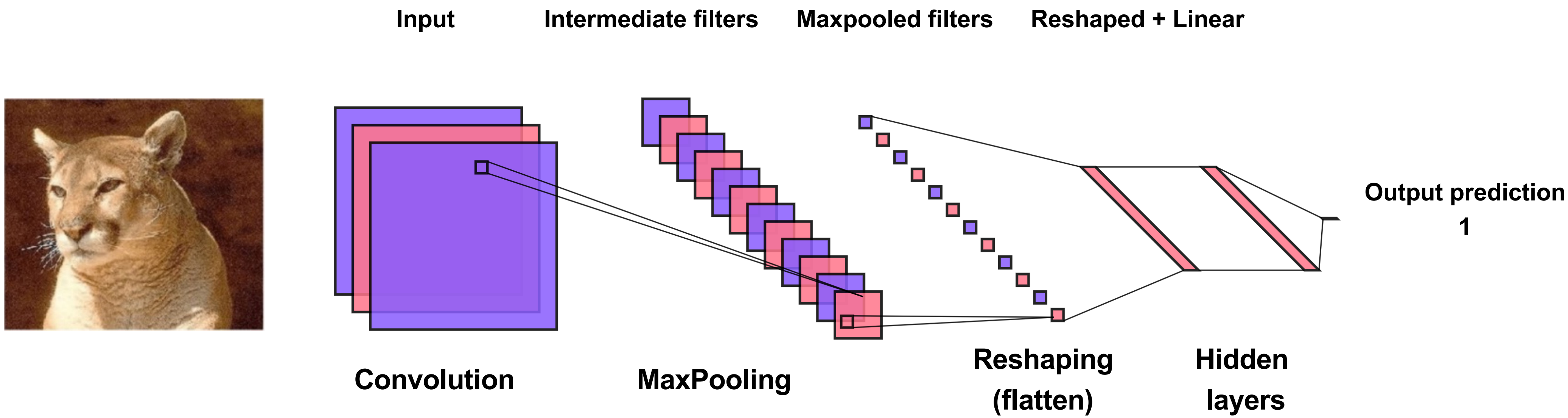

### Sequential data

**MESLVPGFNEKTHVQLSLPVLQVRDV...**



# Convolutional Neural Networks

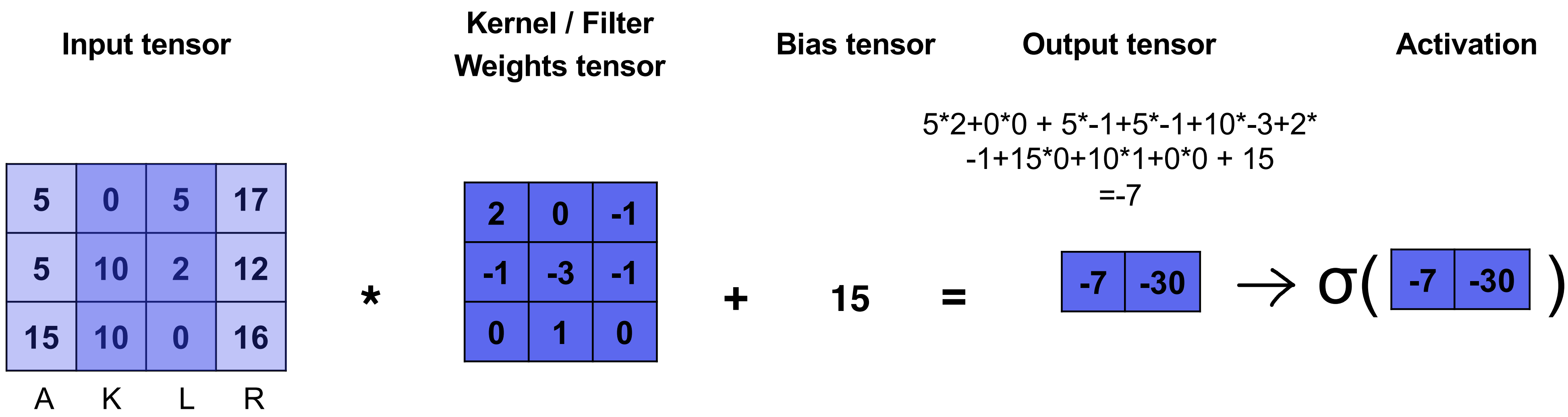
- Based on filters to extract features from the input
- Can be used on 1D, 2D, 3D inputs (sequences, Images, etc.)
- Handles inputs of different size (ex: different sequence lengths)
- Preserves signal structure & local information motifs



# Convolutional Neural Networks

- Based on filters to extract features from the input
- Can be used on 1D, 2D, 3D inputs (sequences, Images, etc.)
- Handles inputs of different size (ex: different sequence lengths)
- Preserves signal structure & local information motifs

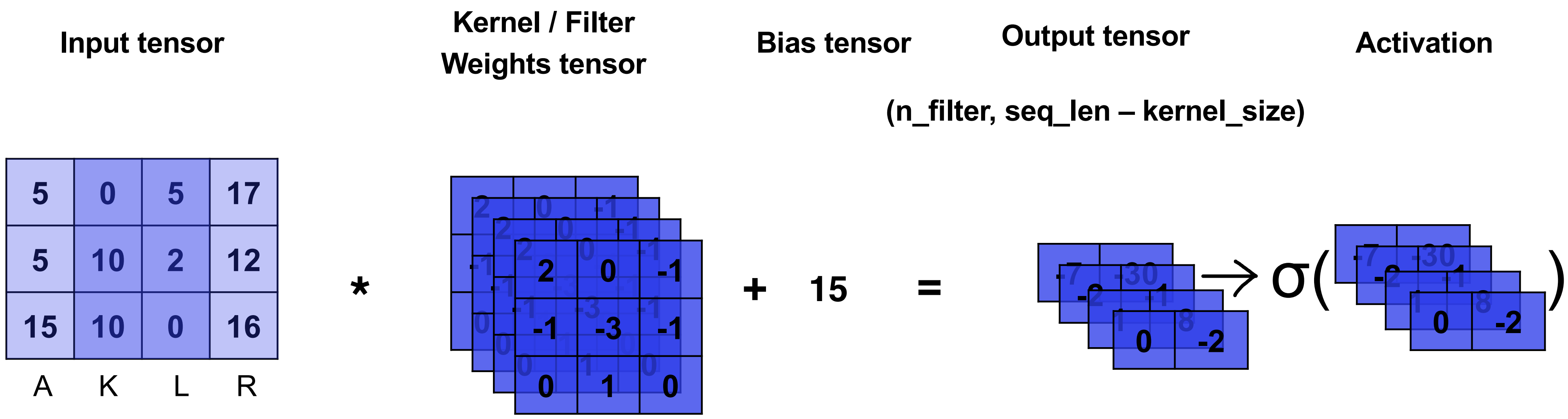
**Basic convolution operation:**



# Convolutional Neural Networks

- Based on filters to extract features from the input
- Can be used on 1D, 2D, 3D inputs (sequences, Images, etc.)
- Handles inputs of different size (ex: different sequence lengths)
- Preserves signal structure & local information motifs

**Basic convolution operation:**



# Batching data to speed up processing

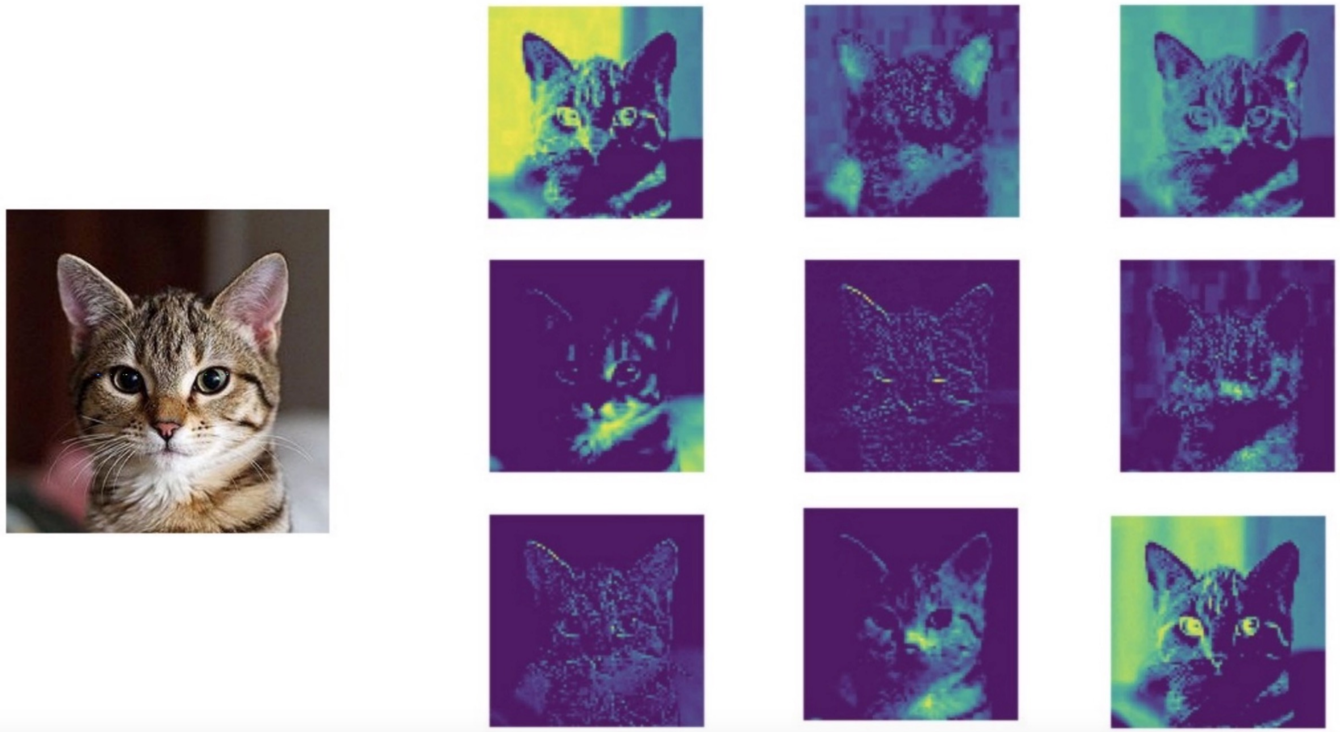
- We can process batches of data at the same time.
- If we want to apply different matrices of weights at the same time (e.g. different filters), we can fuse those weights into a single matrix and use only a single matrix multiplication call.

| | | | | |
|---|---|---|---|---|
| x1_1 | x1_2 | x1_3 | x1_4 | x1_5 |
| x2_1 | x2_2 | x2_3 | x2_4 | x2_5 |
| … | | | | |
| xn_1 | xn_2 | xn_3 | xn_4 | xn_5 |

x

| | | |
|---|---|---|
| F1_1_1 | F1_1_2 | F1_1_3 |
| F1_2_1 | F1_2_2 | F1_2_3 |
| F1_3_1 | F1_3_2 | F1_3_3 |
| F1_4_1 | F1_4_2 | F1_4_3 |
| F1_5_1 | F1_5_2 | F1_5_3 |
| F2_1_1 | F2_1_2 | F2_1_3 |
| F2_2_1 | F2_2_2 | F2_2_3 |
| F2_3_1 | F2_3_2 | F2_3_3 |
| F2_4_1 | F2_4_2 | F2_4_3 |
| F2_5_1 | F2_5_2 | F2_5_3 |
| **…** | | |
| … | | |
| … | | |
| … | | |
| … | | |
| Fn_1_1 | Fn_1_2 | Fn_1_3 |
| Fn_2_1 | Fn_2_2 | Fn_2_3 |
| Fn_3_1 | Fn_3_2 | Fn_3_3 |
| Fn_4_1 | Fn_4_2 | Fn_4_3 |
| Fn_5_1 | Fn_5_2 | Fn_5_3 |

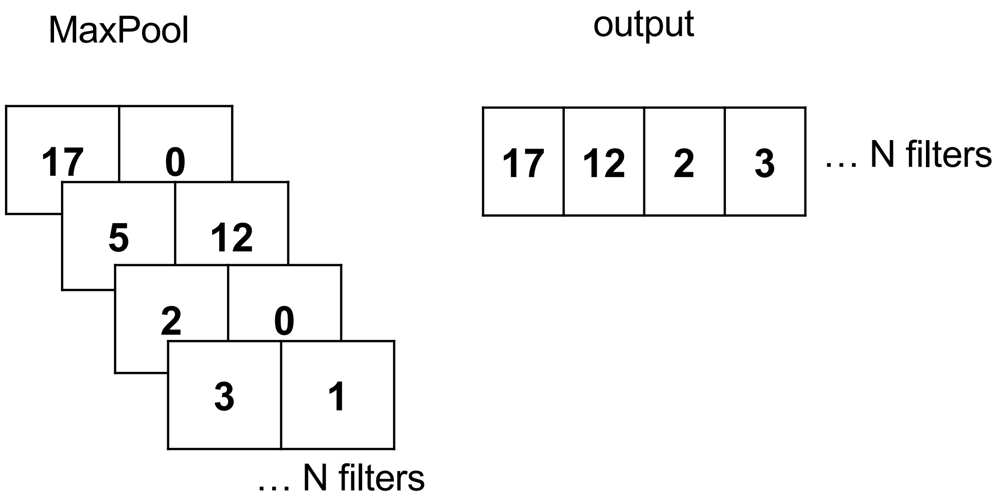
# Example: Visualising learned filters



# Convolutional Neural Networks

- Based on filters to extract features from the input
- Can be used on 1D, 2D, 3D inputs (sequences, Images, etc.)
- Handles inputs of different size (ex: different sequence lengths)
- Preserves signal structure & local information motifs

**MaxPool operation for multiple filters:**



# Backpropagation intuition in Convolutional Neural Networks

Forward pass:

**Basic convolution operation:**

**Input tensor**

| | | | | |
|---|---|---|---|---|
| 5 | 0 | 5 | 17 | ... |
| 5 | 10 | 2 | 12 | ... |
| 15 | 10 | 0 | 16 | ... |
| A | K | L | R | |

*

**Kernel / Filter**
**Weights tensor**

| | | |
|---|---|---|
| 2 | 0 | -1 |
| -1 | -3 | -1 |
| 0 | 1 | 0 |

=

**Output tensor**

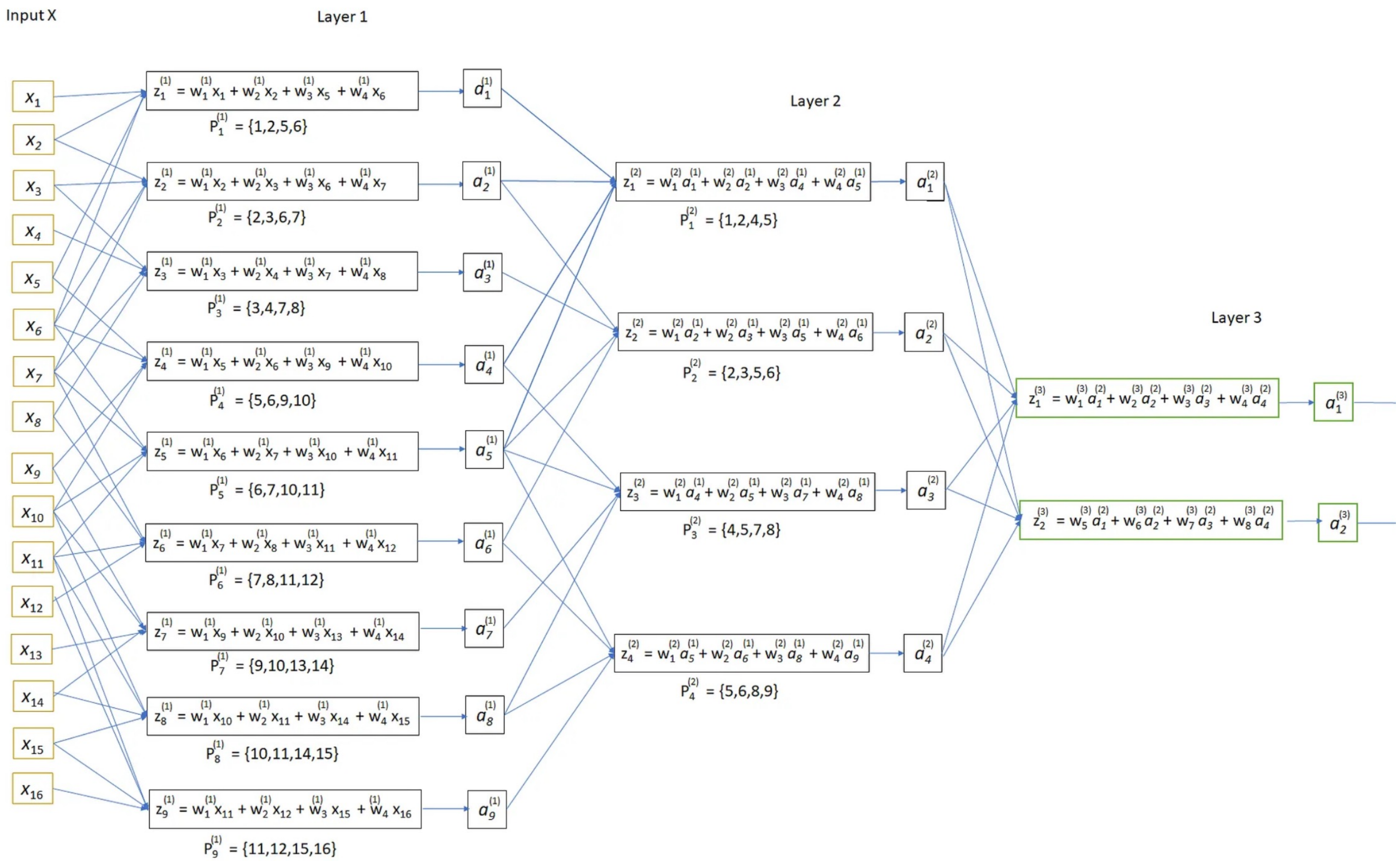
| | | |
|---|---|---|
| z_1 | z_2 | z_n |

$z\_1 = x_{11} w_{11} + x_{12} w_{12} + x_{13} w_{13} + x_{21} w_{21} + x_{22} w_{22} + x_{23} w_{23} + x_{31} w_{31} + x_{32} w_{32} + x_{33} w_{33}$

$z\_2 = x_{12} w_{11} + x_{13} w_{12} + x_{14} w_{13} + x_{22} w_{21} + x_{23} w_{22} + x_{24} w_{23} + x_{32} w_{31} + x_{33} w_{32} + x_{34} w_{33}$

z_n = ...

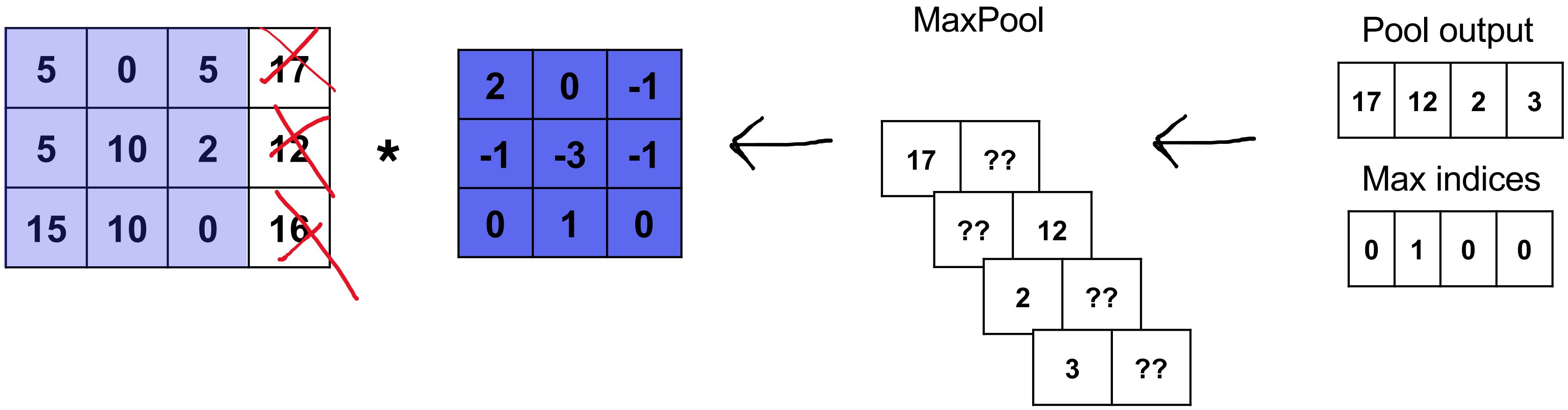
# Backpropagation intuition in Convolutional Neural Networks




Source: https://towardsdatascience.com/backpropagation-in-fully-convolutional-networks-fcns-1a13b75fb56a

# Backpropagation intuition in Convolutional Neural Networks

- Max pooling removes information.
- The gradient can only flow through max value indices.
- We keep track of the max indices during the forward pass.




Source: https://towardsdatascience.com/backpropagation-in-fully-convolutional-networks-fcns-1a13b75fb56a

# Today's exercises: peptide-MHC binding affinity prediction

**Train a model to predict the binding afffinity of a peptide sequence for a given HLA allele**

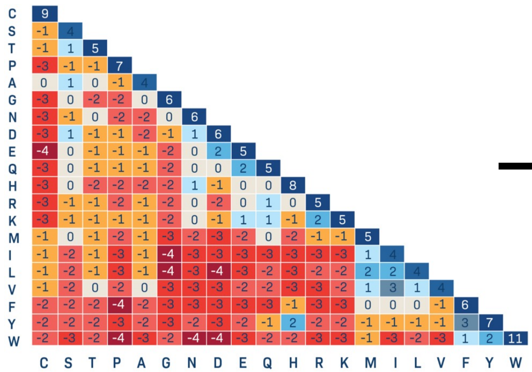
**Sequence representation: encode sequence using BLOSUM matrix**

**Input sequence**

**BL62 (or 50)**

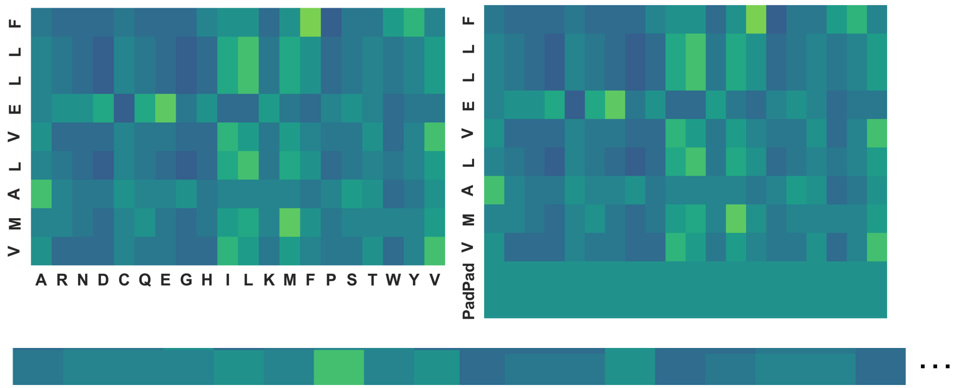
**BLOSUM-encoded sequence**

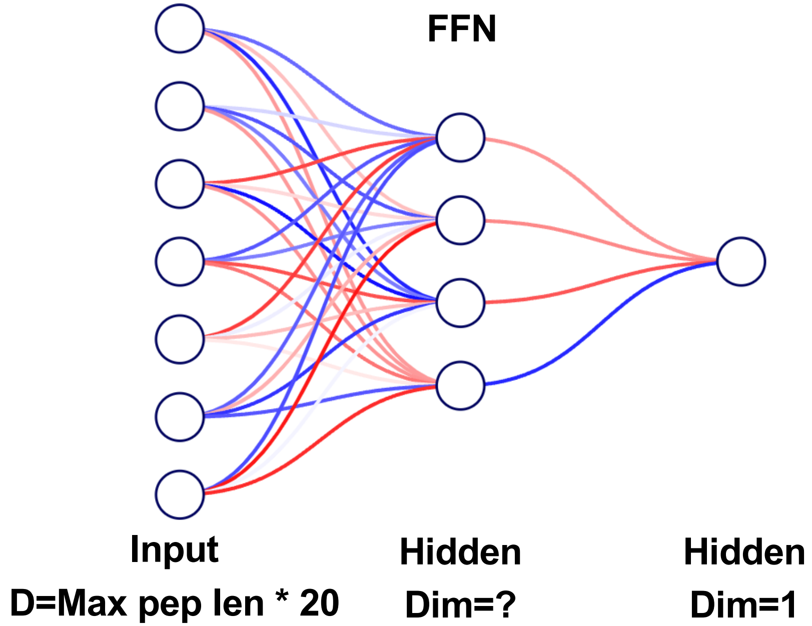
FLLEVLAMV *




# Today's exercises: Part 1

- Implement an FFNN using efficient matrix multiplication operations.
- Peptides may have different lengths. Shorter sequences must be padded. Why?
- Use a flattened array as input.
- Try to change the learning rate or number of neurons and see if it impacts performance.

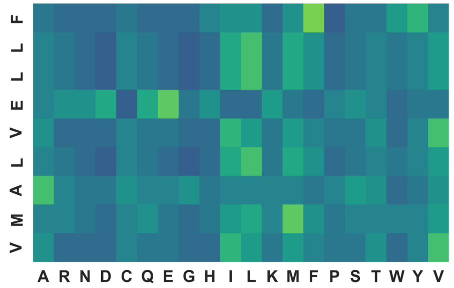


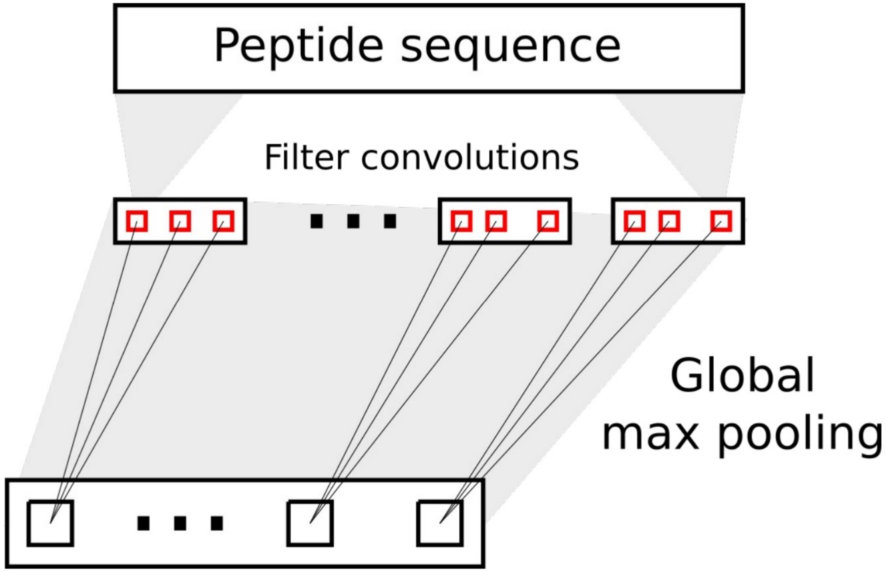



# Today's exercises: Part 2

- **Implement an CNN using efficient matrix multiplication operations.**
- **Peptides may have different lengths. Shorter sequences must be padded.**
- **The CNN operates on a sequence of vectors.**
- **A CNN can take inputs of variable length, so why must we still pad?**

**2d Tensor input**

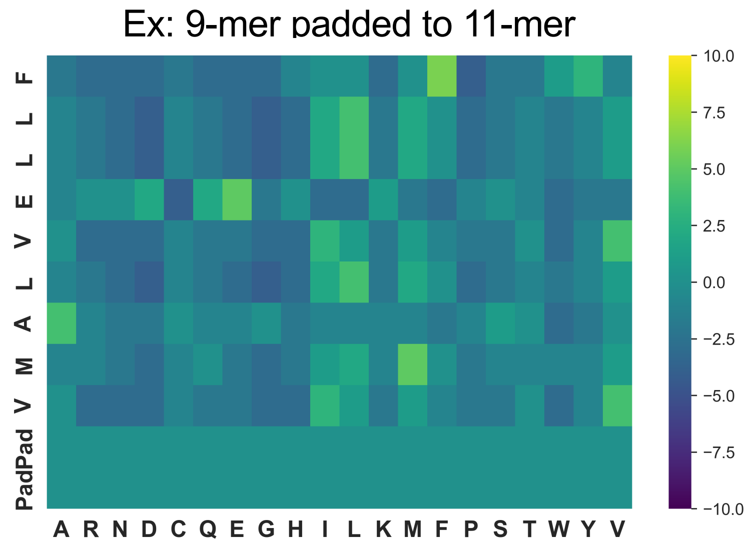


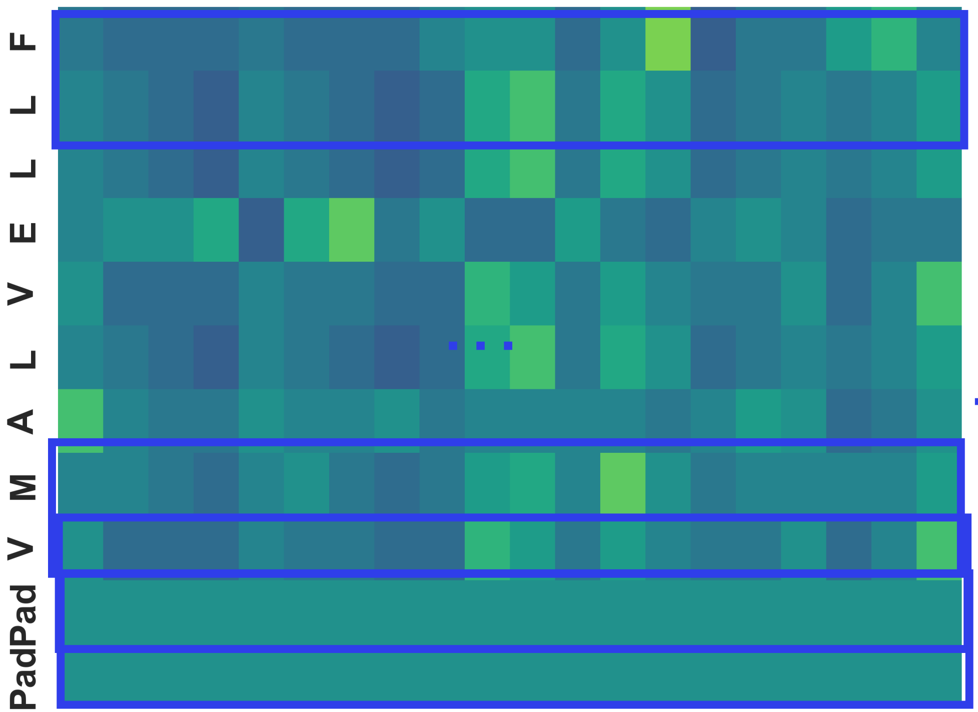
# Today's exercises: Part 2 in practice

Pad all inputs to the max length (11)

The CNN should learn to ignore padded values



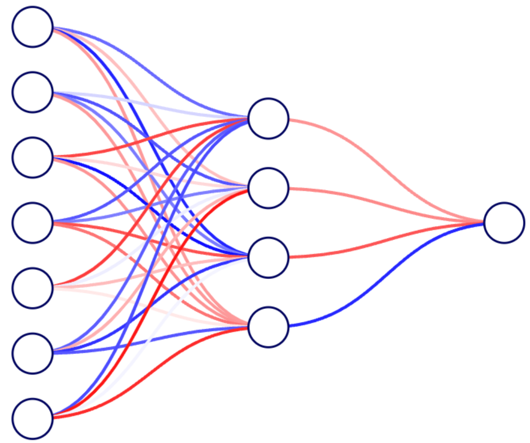

**Example : K=3**




**Activation**

**MaxPool**

**Reshape**



# Today's exercises Part 2: Hyperparameter tuning and cross-validation

- Train and test your models using different datasets. Try using the cross-validation data from the SMM exercise.

- The same scripts can be used for hyperparameter tuning. Set up a hyperparameter tuning experiment.

- Model ensembling almost always improves performance. Train and evaluate an ensemble.

**We are here to take your questions.**

(when in doubt: Print output of a given operation and their dimensions of a tensor x using x.shape or x.size)