Deep Learning Fundamentals

Mihai DOGARIU

www.mdogariu.aimultimedialab.ro

Summary

M3. Terminology

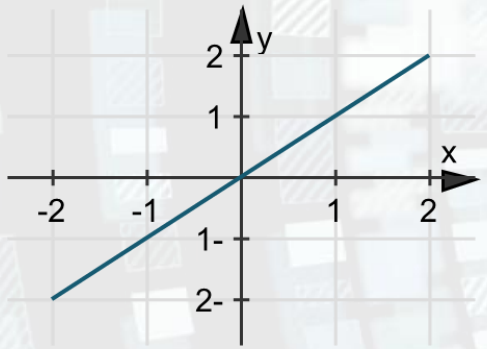
M4. Practical considerations

M3. Terminology

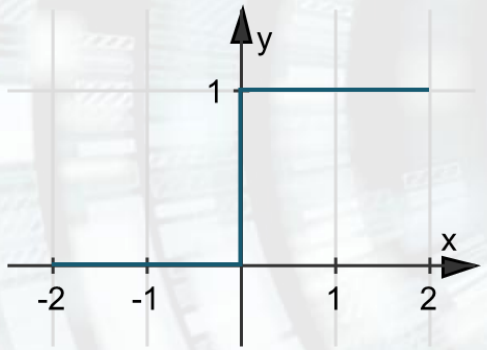
Terminology

1. Activation functions
2. Cost functions
3. Learning rate
4. Parameters vs hyper-parameters
5. Datasets
6. Optimizers
7. Regularizers
8. Popular layers

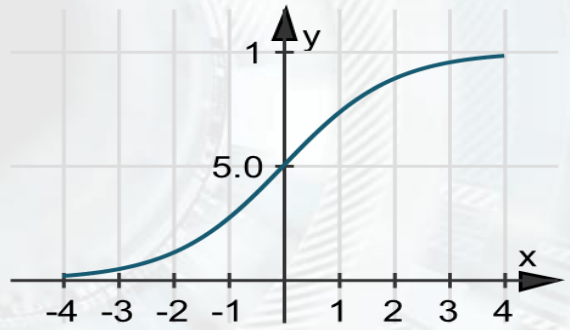
1. Activation functions



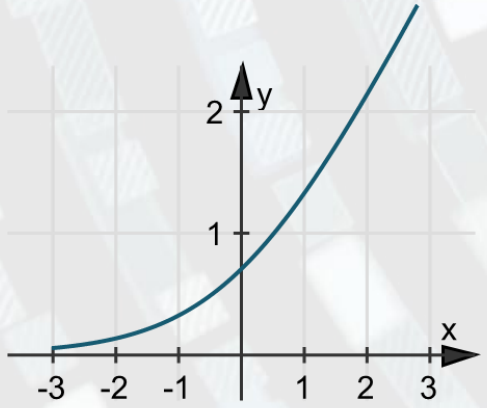
a) linear



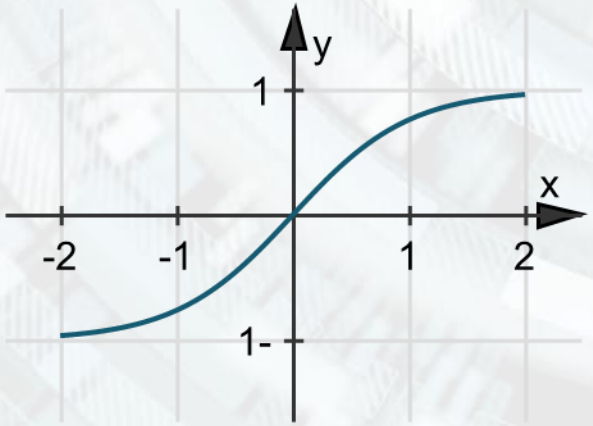
b) Heaviside



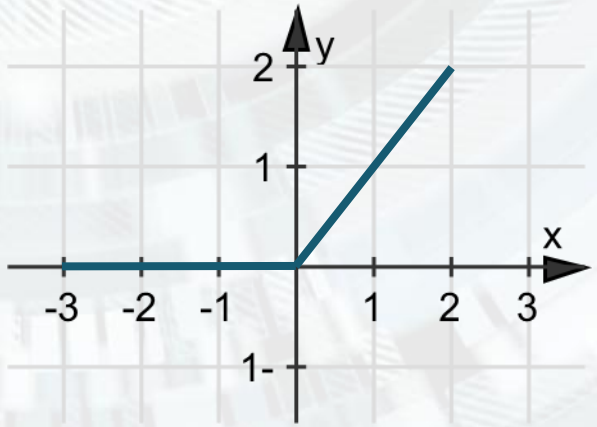
c) sigmoid



d) softplus



e) tanh



f) ReLU

1. Activation functions – what and when?

- Use sigmoid for **logistic regression** and **binary classification**;
- In general, tanh is better than sigmoid, being centered in 0;
- Sigmoid and tanh should be avoided due to **vanishing gradient**;
- Use softmax for **multi-class classification**;
- **Recurrent** nets generally use tanh and sigmoid;
- Conv nets use ReLU in the **hidden** layers;
- If some neurons are never activated, then we replace ReLU with Leaky ReLU. If that doesn't work, we use Parametric ReLU.

2. Cost functions

Loss function = function computed in a **single point** which measures the penalty of taking a decision (distance between predicted and groundtruth value).

Type of

Cost function = loss function computed at the **entire dataset level**. Is more general than the loss function.

Part of

Objective function = function which needs to be optimized, either by minimizing (cost function), or by maximizing.

2. Cost functions – examples

Loss functions:

$$\mathcal{L}(y_i, \hat{y_i}) = (y_i - \hat{y_i})^2$$
$$\mathcal{L}(y_i) = \max(0, 1 - t_i y_i), t \in \{-1, 1\}, y = \boldsymbol{wx} + b$$

Cost functions:

$$MSE = \left[\frac{1}{N}\right] L2 = J(y_i, \hat{y_i}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2$$

$$L1 = J(y_i, \hat{y_i}) = \left[\frac{1}{N}\right]\sum_{i=1}^{N}|y_i - \hat{y_i}|$$

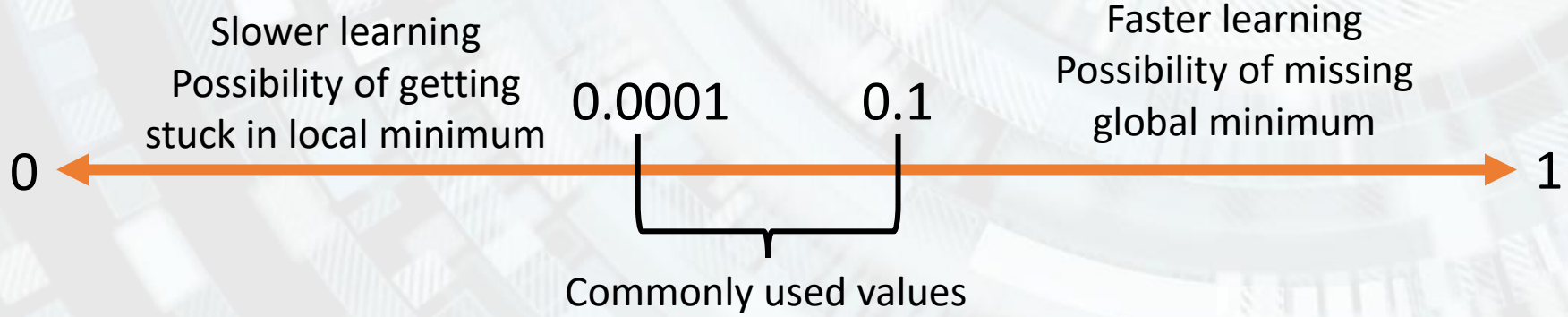
$$CE = -\sum_{c=1}^{M} \mathbb{1}_{[y \equiv \hat{y}]} \log(p(y \equiv \hat{y}))$$

Objective functions:

$$w = \arg\max_{w} \sum_{i=1}^{M} \log p_{model}(\hat{y_i} | x_i, w)$$

3. Learning rate

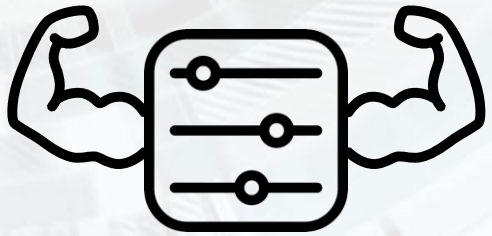
Learning rate = adjustable parameter which determines the step size when advancing towards minimizing the cost function, at each iteration. It represents the „speed” with which an algorithm learns.



4. Parameters vs hiper-parameters

Parameters = adjustable variables representing the internal state of the model. They are deducted from the training data through learning. Example: model weights.

Hyper-parameters = configurable variables, external to the model, influencing the learning process. They are set by the user. Example: learning rate, number of epochs, batch size etc.



5. Datasets

Dataset = a group of elements with common properties. It represents the „experience” that an algorithm makes use of when learning a certain task (according to the definition).

$$D = \{((x_i, y_i)|T), 1 \leq i \leq M\}$$

input output task dimension

LibriSpeech

GLUE

5. Datasets

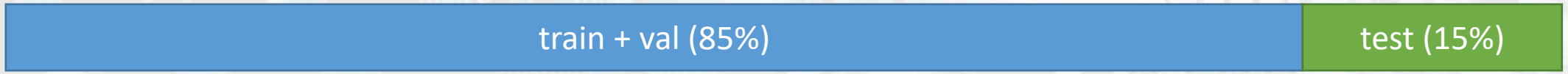
- They usually contain examples of input-output data. Their purpose is to help the model find an association law between input and output.

- A good dataset contains sufficient examples (10 x no. param.) and is uniformely distributed among its classes:
  - for regression: 10 examples/prediction variable
  - for classification: 1000 examples/class

5. Datasets – splits

- During model training, the training is done on a subset of the datasets, called „train”. Hyper-parameter optimization is done by looking at the results obtained when testing the model on another subset, called „val” ⇔ we optimize w.r.t. „val”.

- After finding the best set of hyper-parameters, the model is trained again on the „train+val” subset and its generalizing ability is checked by testing it on the „test” subset, unseen up until that point by the model. The reported performance is the one obtained on „test”.



5. Datasets – browsing

Iteration = the set of processes done between two successive weight updates during training.

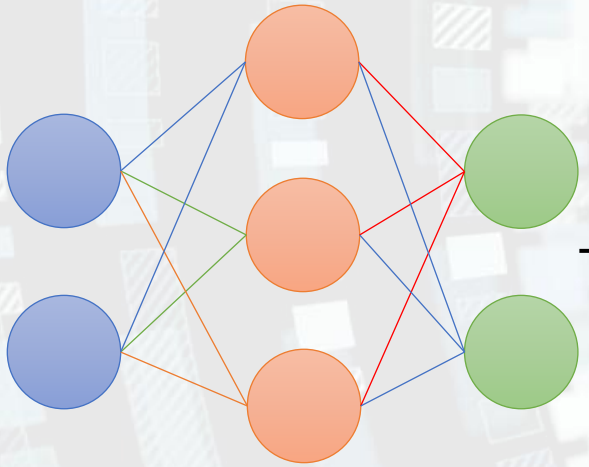
Batch = group of instances from the dataset which can be run by the model at the same time, in parallel.

Batch size = number of instances inside a batch.

Let's test it out – unit #5

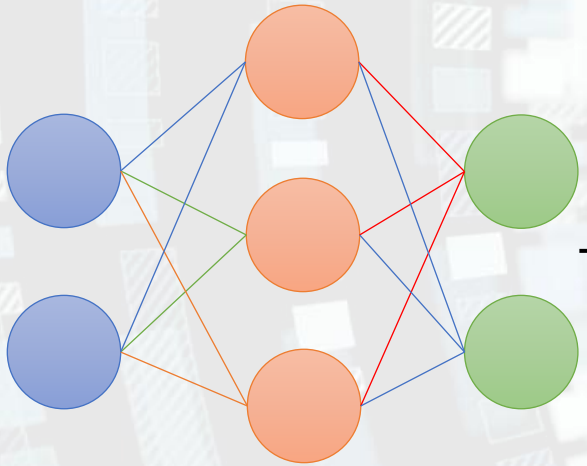
Epoch = a term which signals a complete pass of the entire dataset through the learning algorithm.

5. Datasets – batching (e.g. batch_size = 4)

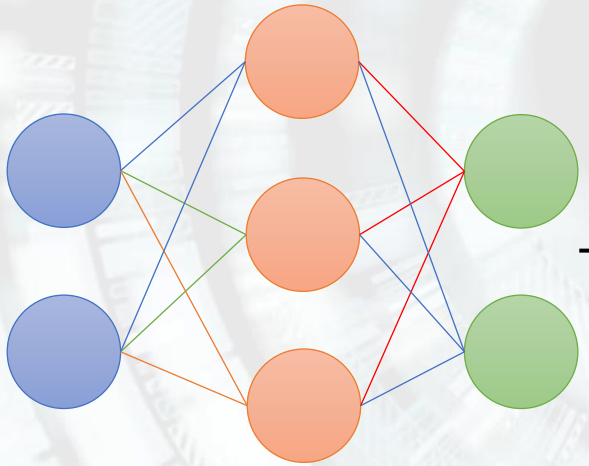


$\longrightarrow \mathcal{L}_1(y_{i_1}, \widehat{y_{i_1}})$

5. Datasets – batching (e.g. batch_size = 4)

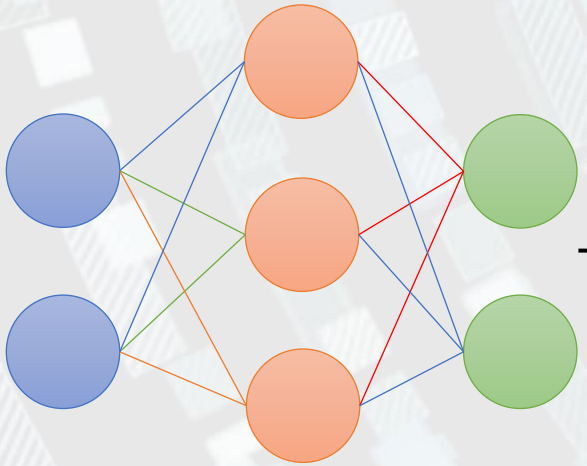


$\rightarrow \mathcal{L}_1(y_{i_1}, \widehat{y_{i_1}})$

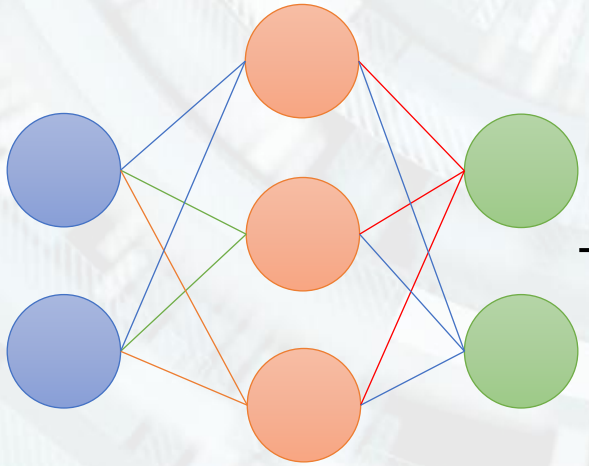


$\rightarrow \mathcal{L}_3(y_{i_3}, \widehat{y_{i_3}})$

$$J_{batch} = \frac{\sum_{i=1}^{batch\_size} \mathcal{L}_i}{batch\_size}$$



$\rightarrow \mathcal{L}_2(y_{i_2}, \widehat{y_{i_2}})$



$\rightarrow \mathcal{L}_4(y_{i_4}, \widehat{y_{i_4}})$

6. Optimizers

Optimizer = algorithm used for optimizing (minimizing or maximizing) an objective function.

- (Batch, Stochastic, Mini-batch) Gradient Descent
- Momentum
- Learning rate scheduling
- Adagrad
- RMSProp
- Adam

6. Optimizers

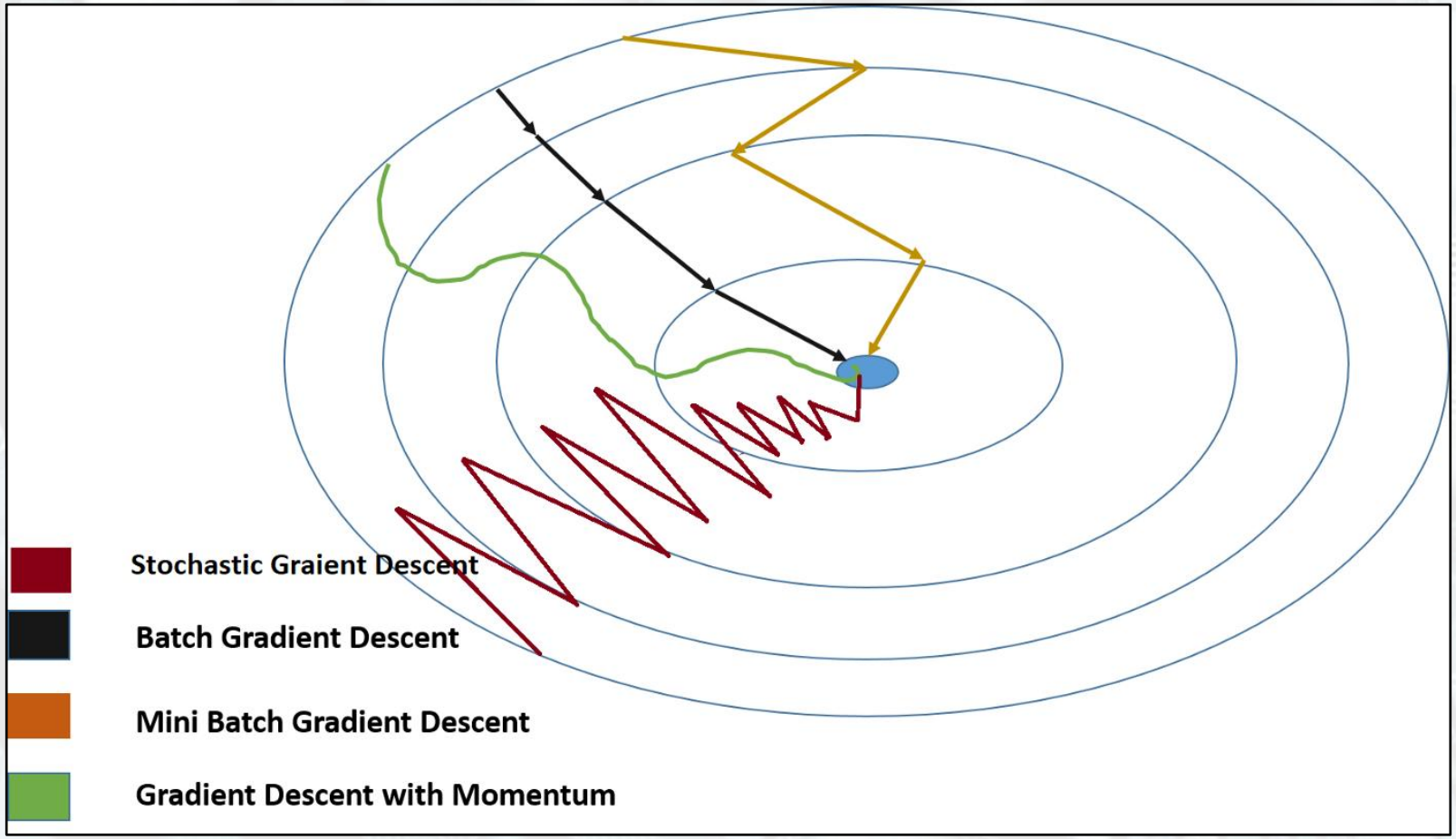
- Batch gradient descent – loads the entire dataset, runs forward propagation and updates the weights accordingly => best model approximation, requires storing the dataset in memory.

- Stochastic gradient descent – loads one sample, runs forward propagation, updates the weights, repeats for next sample => best sample approximation, poor model approximation, loads a single sample at a time.

- Mini-batch gradient descent – loads a batch of samples, runs forward propagation, updates the weights, repeats for next batch => best compromise between memory and model approximation.

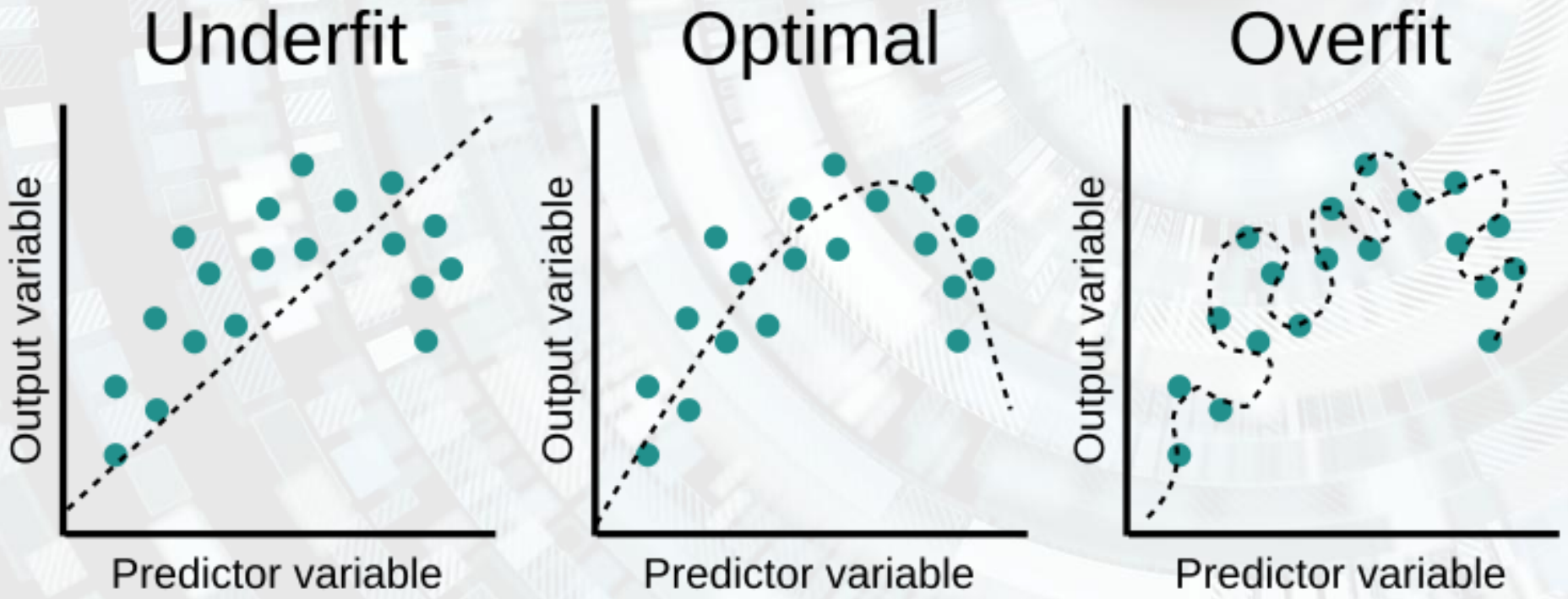
Image: Bouzouitina Hamdi

6. Optimizers – comparison



Sursă imagine: educative.io

7. Regularization

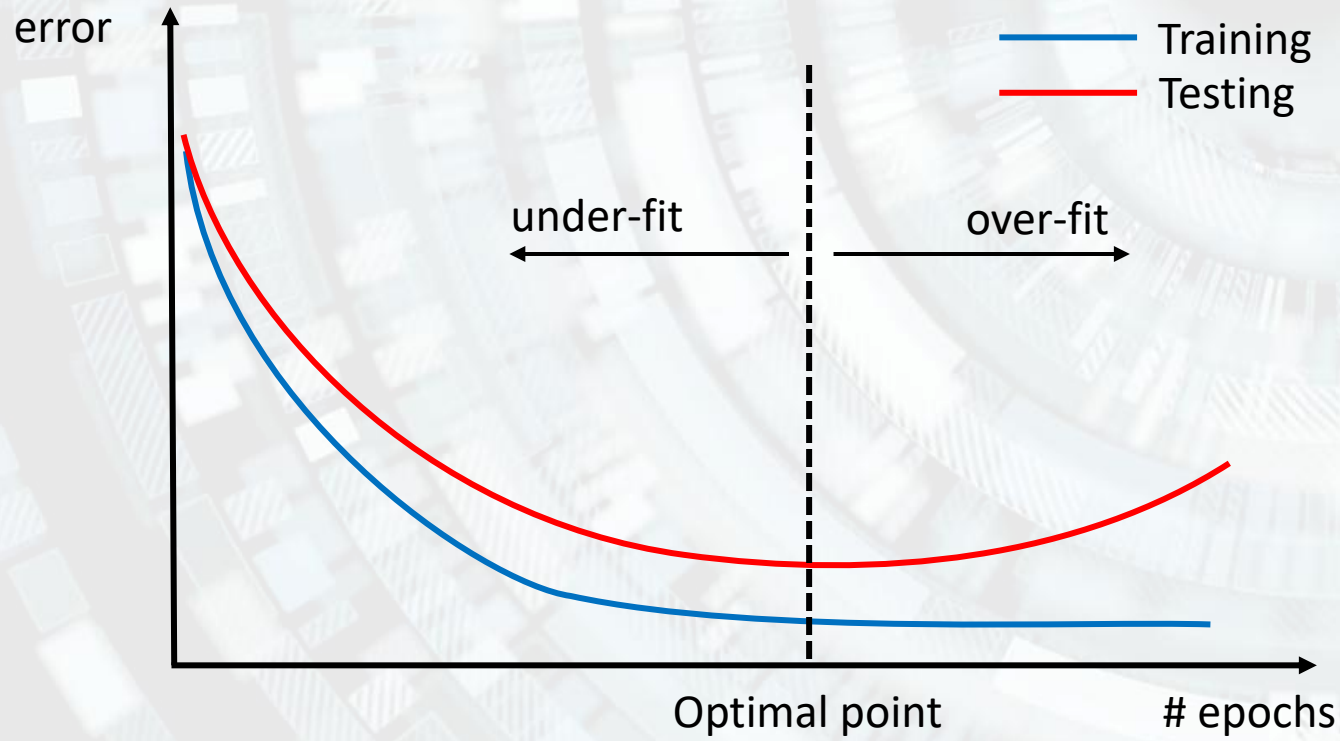


7. Regularization

- Model learning has 2 phases:
  1. Training the model on the training dataset – optimization;
  2. Testing the predictive capacity on the (unseen) test dataset - generalization.

- A high performant model is characterized by 2 aspects:
  1. Low training error
  2. Small difference between training and testing errors.

7. Regularization



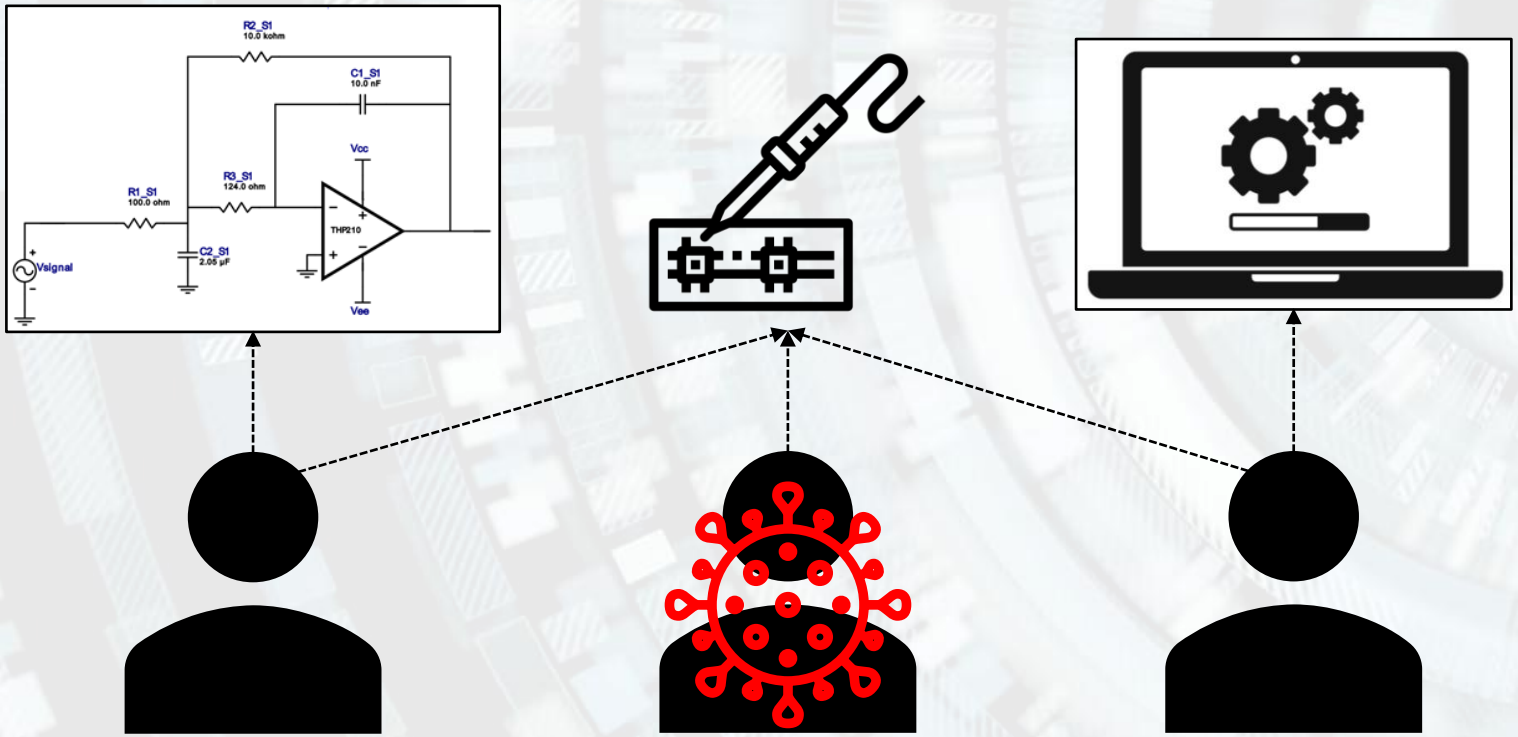
7. Regularization

Regularization = modifying the learning algorithm such that it increases the generalization capacity and prevents overfitting.

- Types of regularization:
  1. L1, L2;
  2. Dropout;
  3. Early stopping;
  4. Dataset augmentation.

Images:hardwarebee, flaticon, vectorstock

7. Regularization – dropout



Scenario – building a robot:

- Team of 3 students;
- Each students specializes in a different field;
- One of the students cannot work at the project anymore;
- The other 2 students must take over his task;
- The knowledge of the 2 remaining students increases.

8. Popular layers

Layer = fundamental topological component composed of nodes, which is part of a neural network. Layers take information from previous layers and transmit it to the next ones. Chaining several layers forms a neural network.

- Input
- Dense
- Convolutional
- Pooling
- Recurrent
- Activations
- Dropout
- Batch normalization
- Output

8. Popular layers

- Input layer:
  - First layer of any network;
  - We load the input data in this layer;
  - Can have more dimensions: $num \times width \times height \times channel$.

- Dense / fully-connected layer:
  - Each input neuron is connected to each output neuron;
  - One of the most commonly used layers, especially at the end of a network;
  - Described in the multi-layer network.

8. Popular layers

- Convolutional layer:
  - Performs „convolution” between an input signal and a set of features, named kernel/filter;
  - Traditionally, it is applied on 2D data, but it also has support for 1D and 3D;
  - It is the main layer for most computer vision applications;
  - Its purpose is to extract relevan features and reduce input dimension;
  - Special cases: dilated convolutions, 1×1 convolutions.
  - Convolutions are invariant to translations

8. Popular layers – convolutional layer

padding=1

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	1	1	0	2	0	0
0	0	0	1	1	1	0
0	1	2	2	0	2	0
0	2	0	1	2	1	0
0	0	0	0	0	0	0

Input features

0	1	-1
-1	1	0
0	1	1

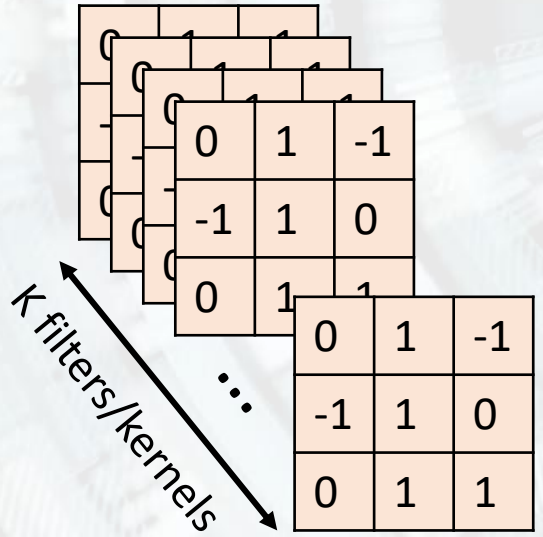
Filter/kernel

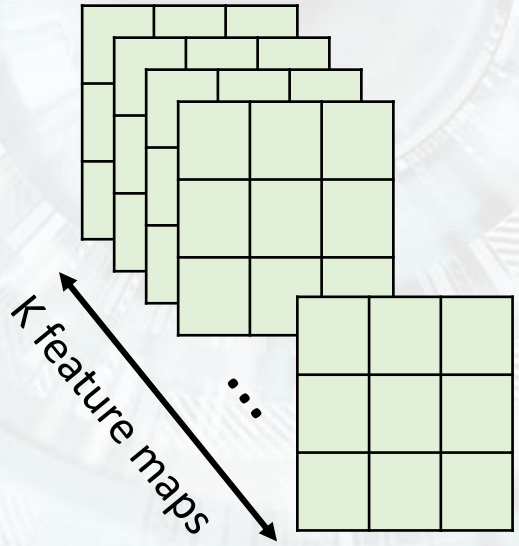
Output features

8. Popular layers – convolutional layer

padding=1

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	1	1	0	2	0	0
0	0	0	1	1	1	0
0	1	2	2	0	2	0
0	2	0	1	2	1	0
0	0	0	0	0	0	0





Input features

Filter/kernel

Output features

8. Popular layers – convolutional layer

0	0	0	0	0	0	0
0	2	1	0	1	1	0
0	1	1	0	2	0	0
0	0	0	1	1	1	0
0	1	2	2	0	2	0
0	2	0	1	2	1	0
0	0	0	0	0	0	0

0	1	-1
-1	1	0
0	1	1

4		

0	0	0
0	2	1
0	1	1

×

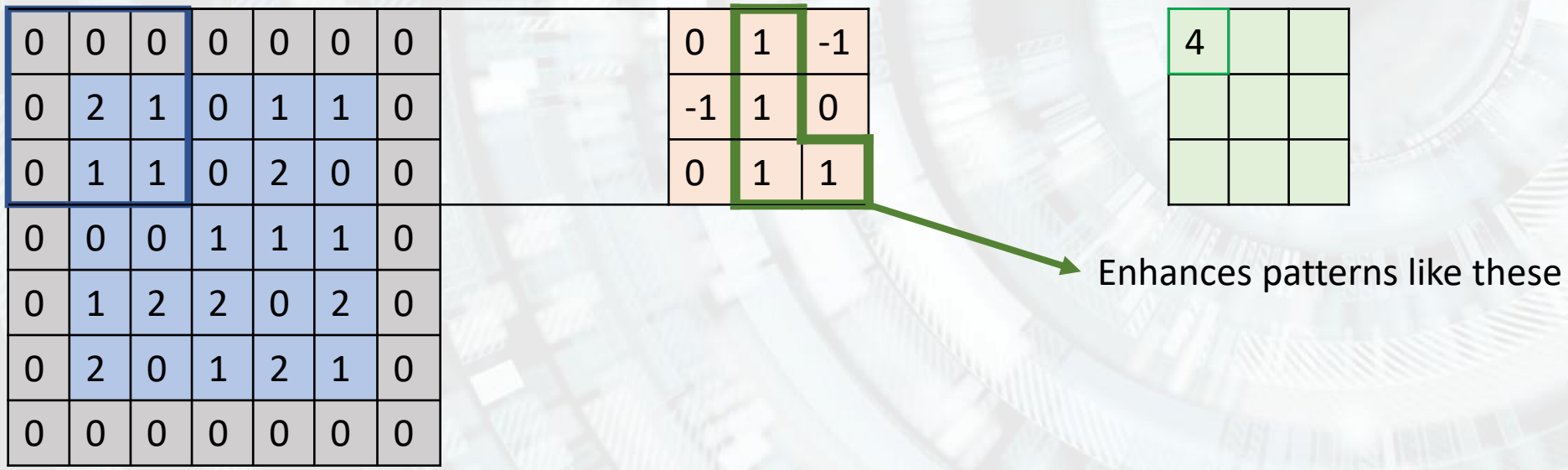
0	1	-1
-1	1	0
0	1	1

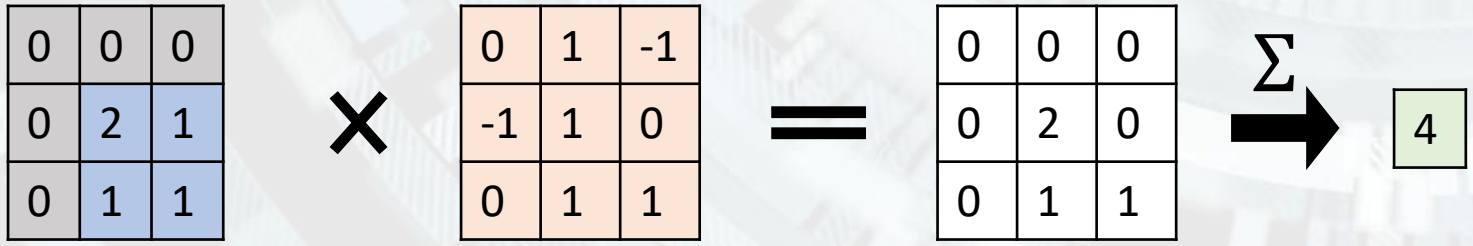
=

0	0	0
0	2	0
0	1	1

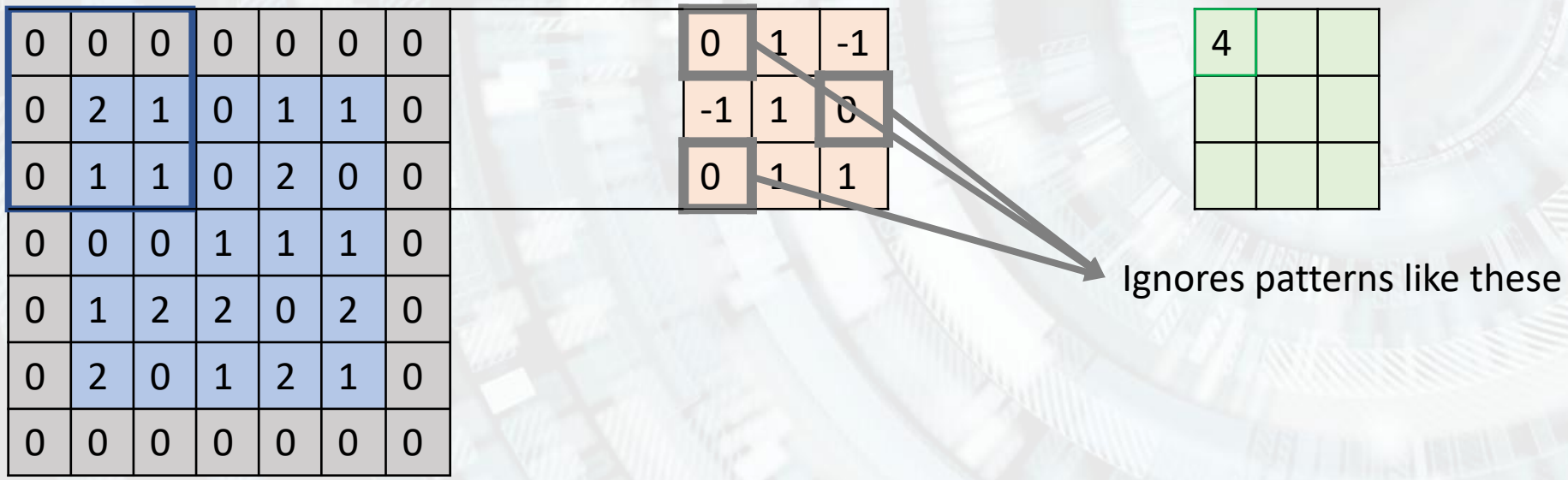
Σ → 4

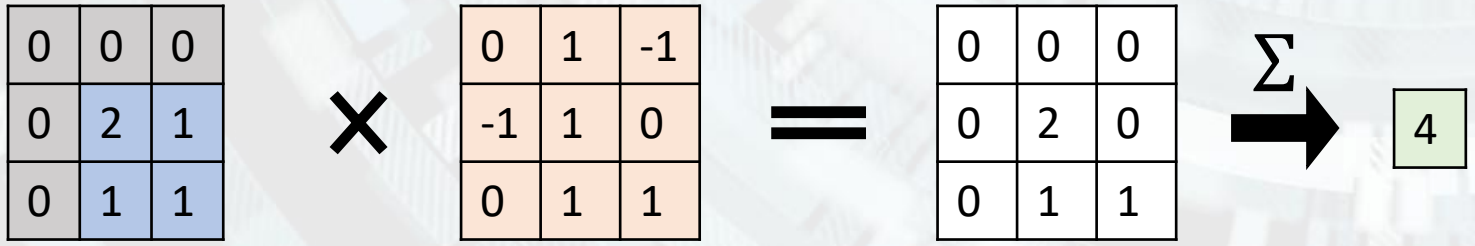
8. Popular layers – convolutional layer



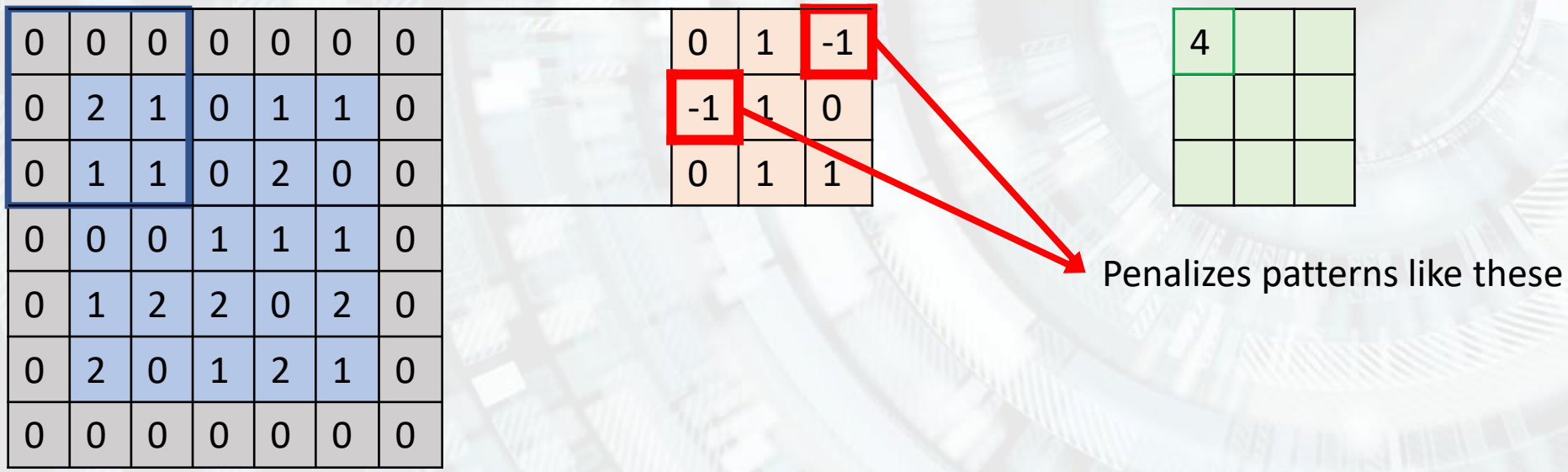


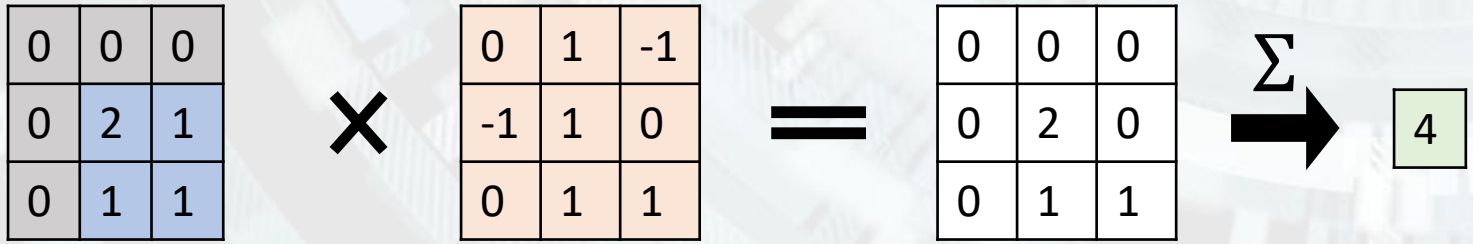
8. Popular layers – convolutional layer



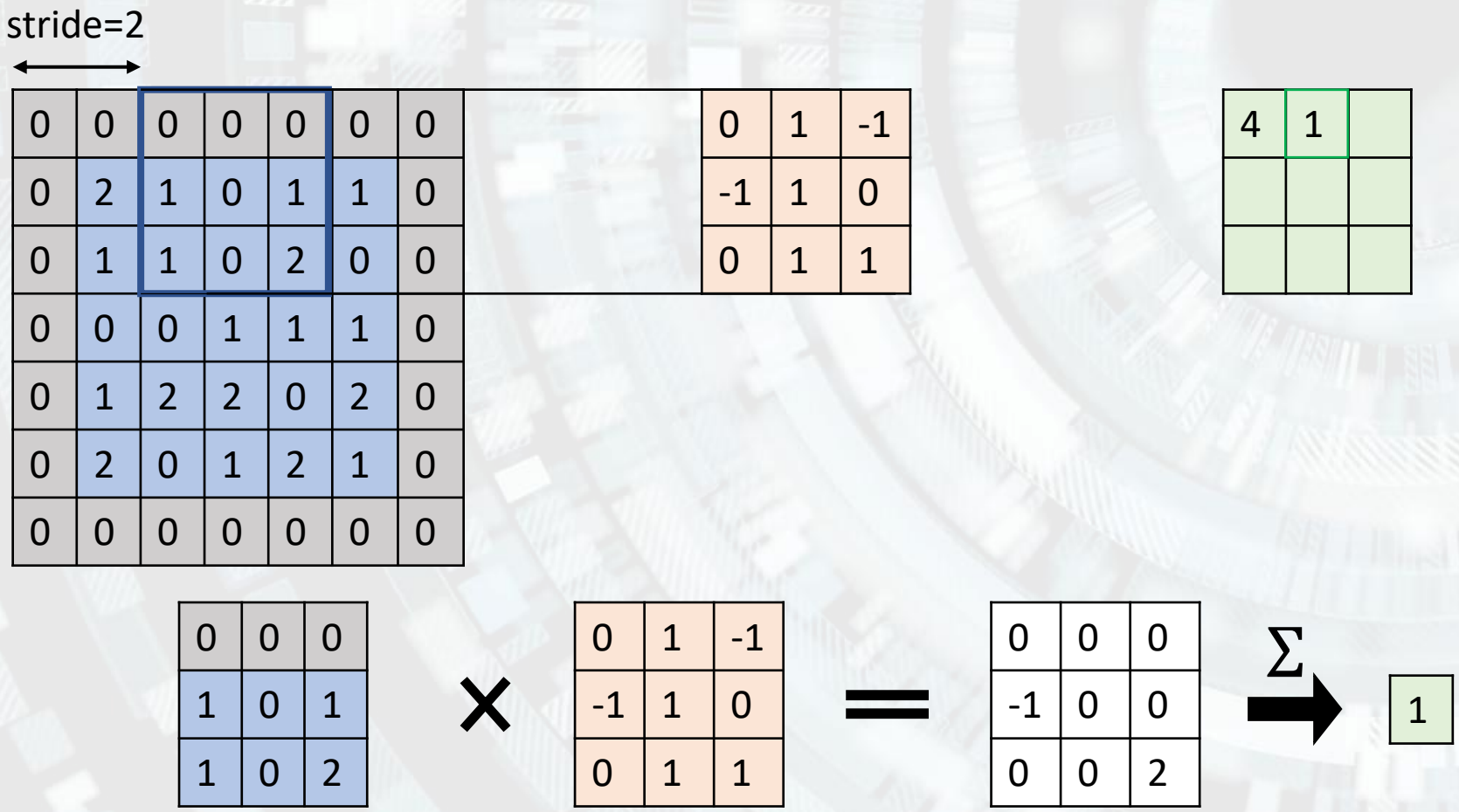


8. Popular layers – convolutional layer

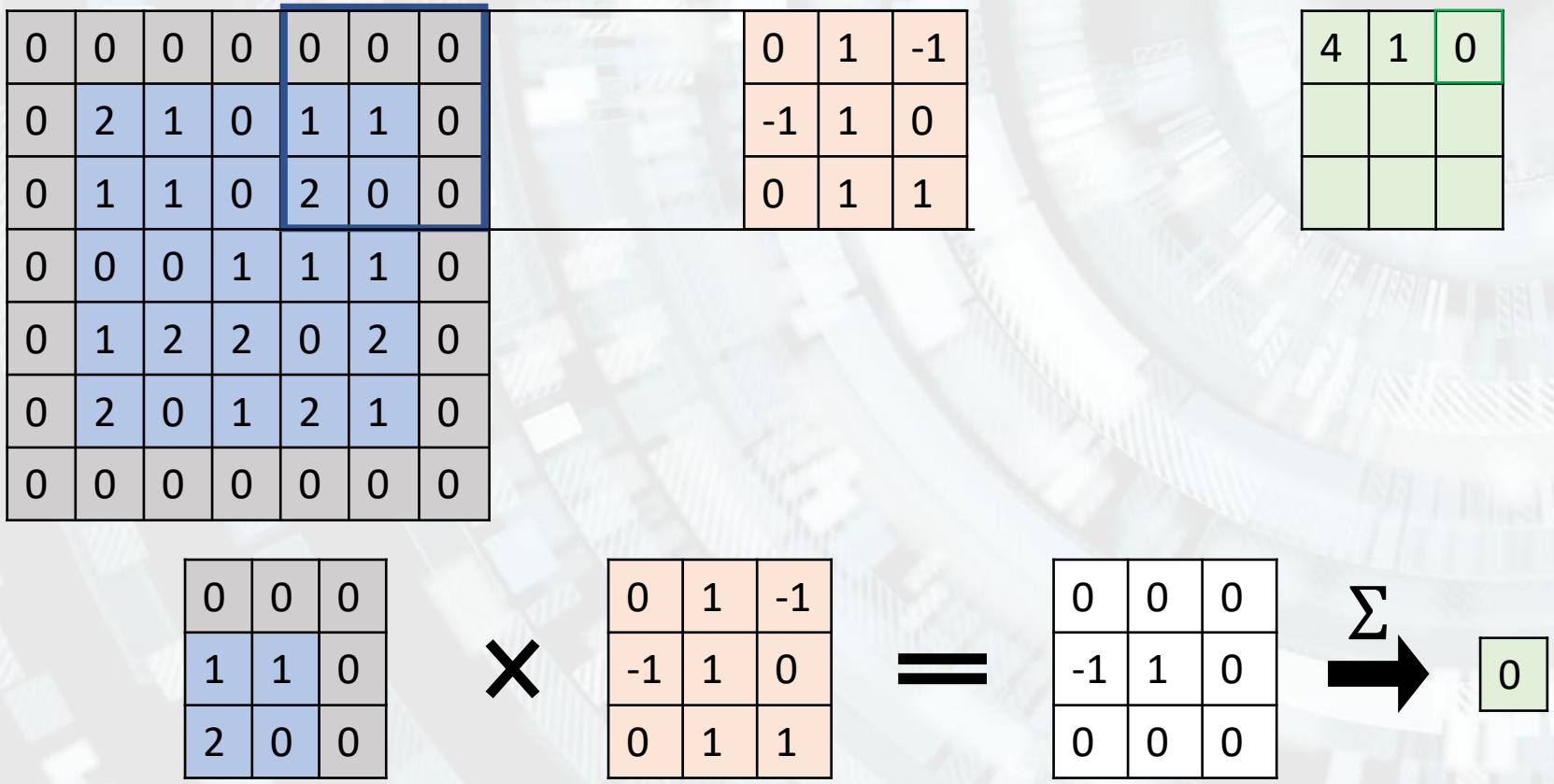




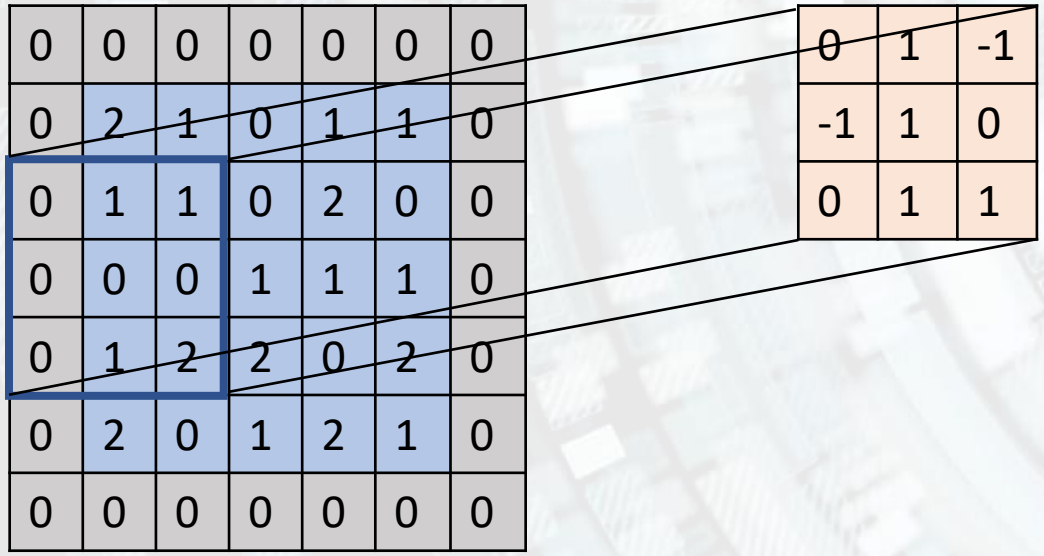
8. Popular layers – convolutional layer



8. Popular layers – convolutional layer



8. Popular layers – convolutional layer



4	1	0
3		

0	1	1
0	0	0
0	1	2

×

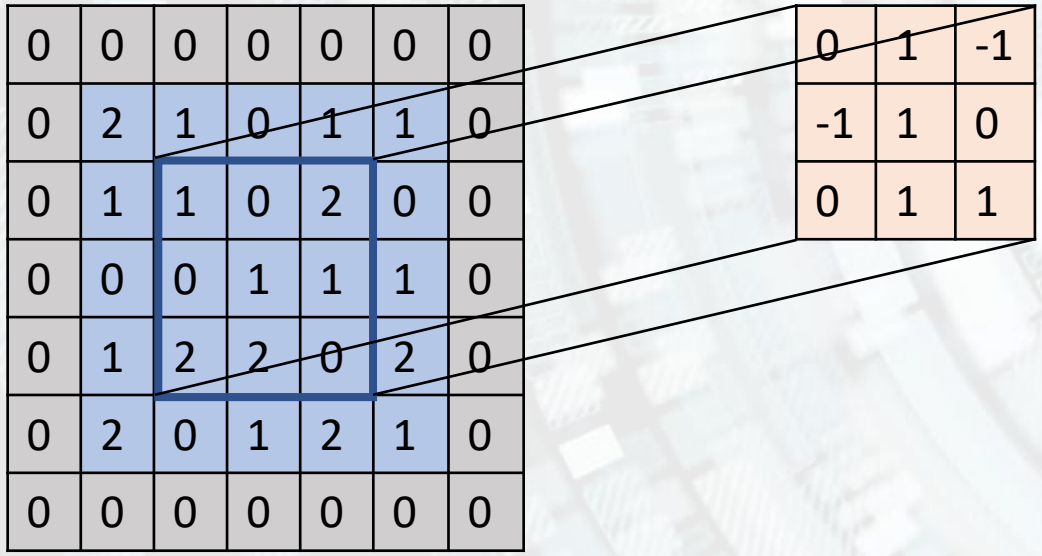
0	1	-1
-1	1	0
0	1	1

=

0	1	-1
0	0	0
0	1	2

Σ → 3

8. Popular layers – convolutional layer



4	1	0
3	1	

1	0	2
0	1	1
2	2	0

×

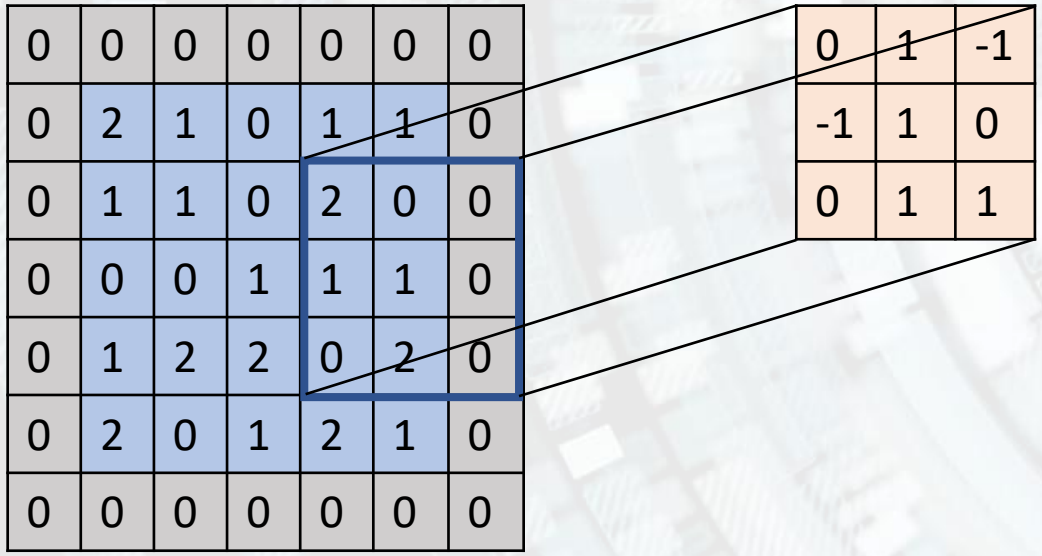
0	1	-1
-1	1	0
0	1	1

=

0	0	-2
0	1	0
0	2	0

Σ → 1

8. Popular layers – convolutional layer



4	1	0
3	1	2

2	0	0
1	1	0
0	2	0

×

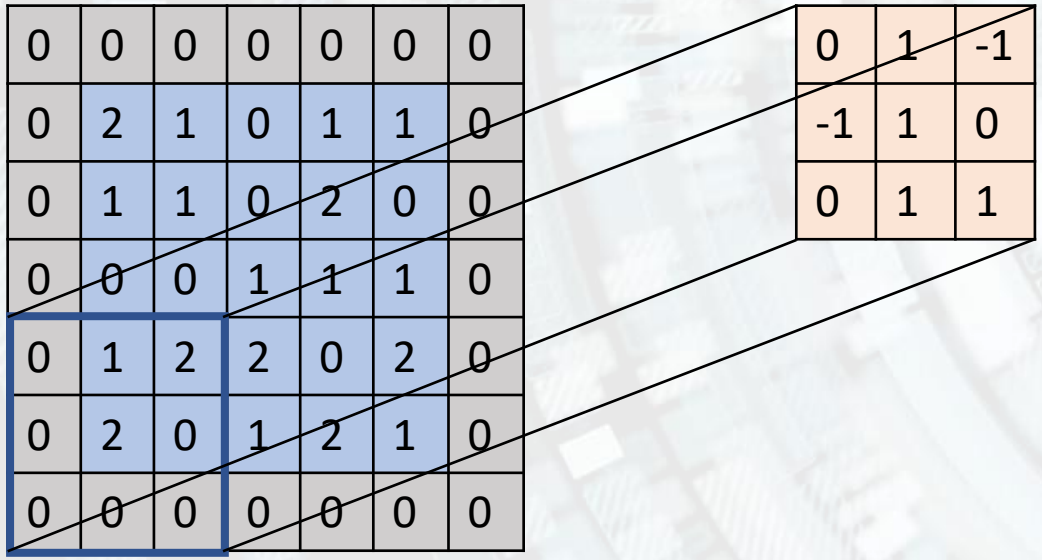
0	1	-1
-1	1	0
0	1	1

=

0	0	0
-1	1	0
0	2	0

Σ → 2

8. Popular layers – convolutional layer



4	1	0
3	1	2
1		

0	1	2
0	2	0
0	0	0

×

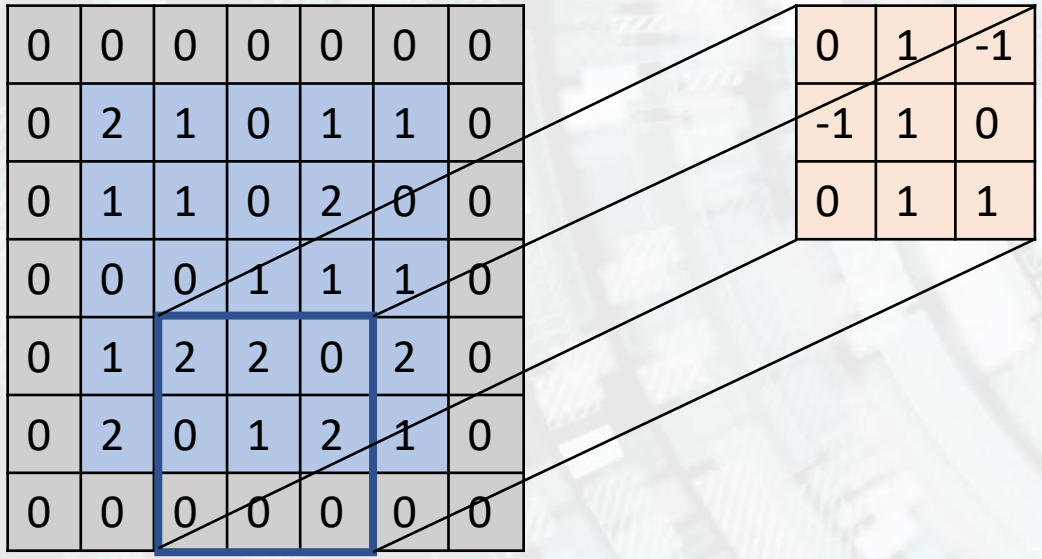
0	1	-1
-1	1	0
0	1	1

=

0	1	-2
0	2	0
0	0	0

Σ → 1

8. Popular layers – convolutional layer



4	1	0
3	1	2
1	3	

2	2	0
0	1	2
0	0	0

×

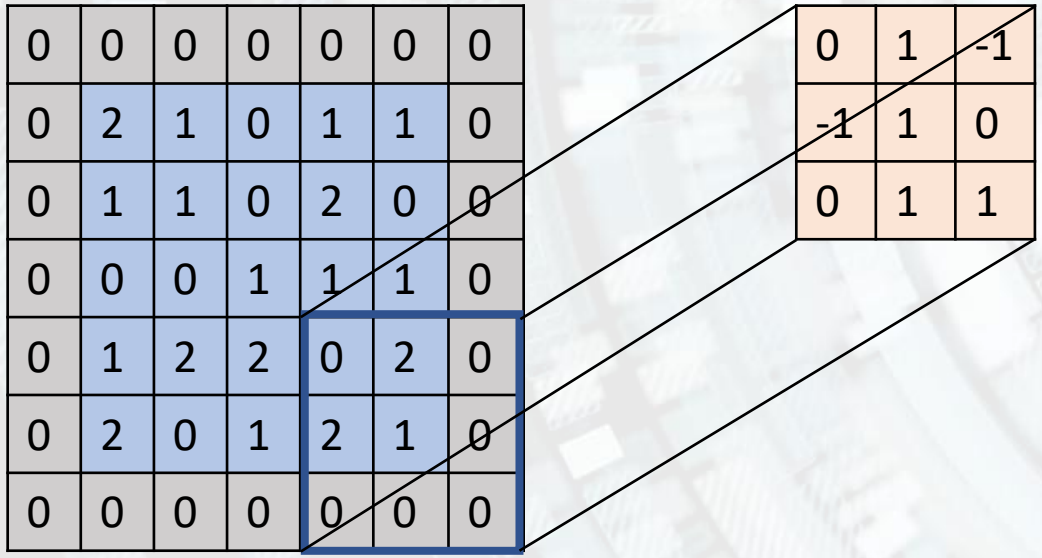
0	1	-1
-1	1	0
0	1	1

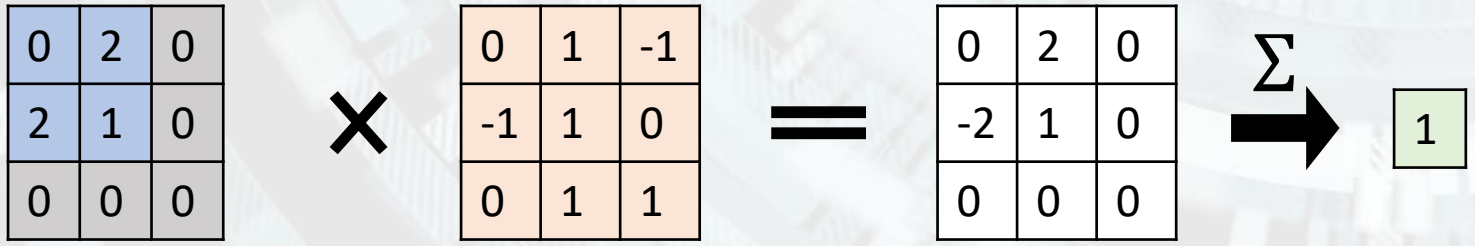
=

0	2	0
0	1	0
0	0	0

Σ → 3

8. Popular layers – convolutional layer





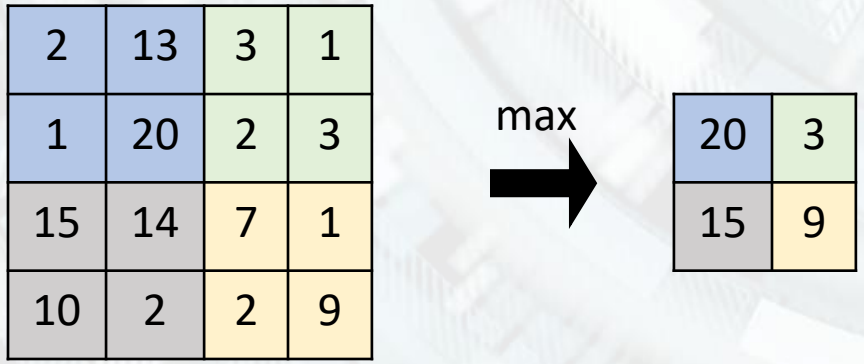
8. Popular layers – convolutional layer

- Convolutional layer hyperparameters:
  - Number of filters/kernels: K
  - Filter/kernel size: F
  - Stride: S
  - Padding: P
  - For an input of dimensions $W_1 \times H_1 \times D_1$ we will obtain an output of dimensions $W_2 \times H_2 \times D_2$, where:
    - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
    - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
    - $D_2 = K$

Let's test it out – unit #6

8. Popular layers

- Pooling layer
  - Can be max/average pooling – similar concept.
  - Hyper-parameters similar to conv layer: filter size - F, stride - S, padding - P



M4. Practical considerations

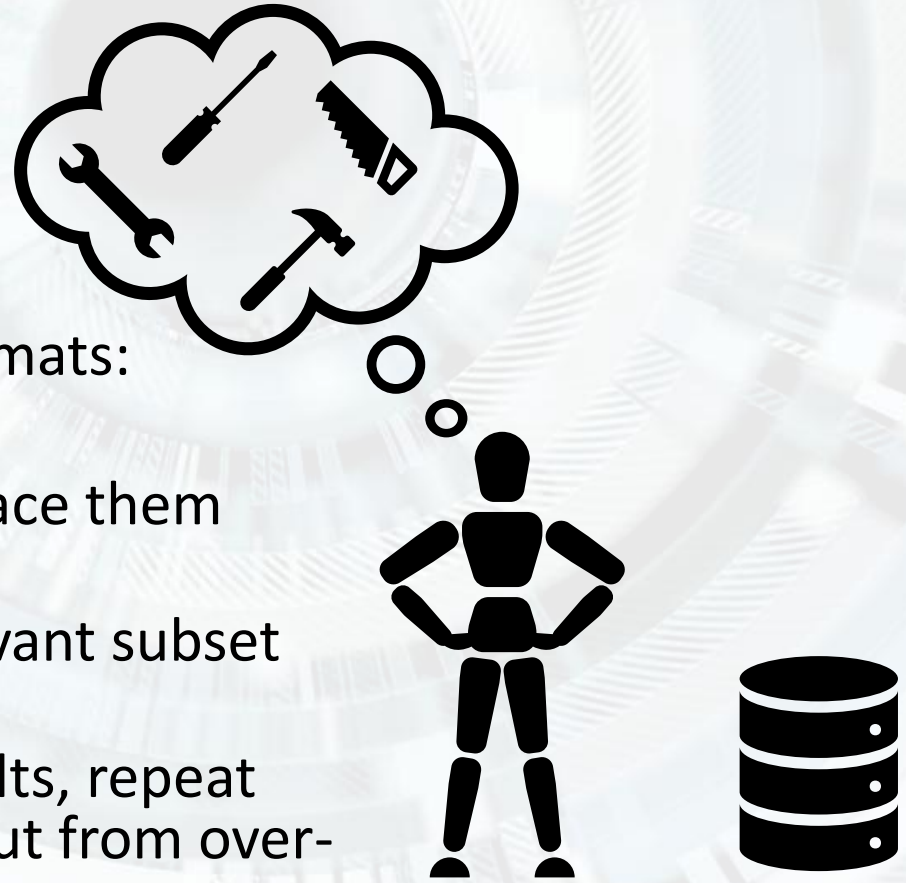
Steps in designing a neural network model

1. Choosing data
   - Number of examples in the dataset:
     - General: 10x no. of model parameters
     - Regression: 10 examples / prediction variable
     - Classification: 1000 examples / class
   - Sample quality – labeling done by experts, reduced noise
   - Dataset resources:
     - https://archive.ics.uci.edu/ml/index.php
     - https://www.kaggle.com/datasets
     - https://paperswithcode.com/datasets
     - https://datasetsearch.research.google.com/

Steps in designing a neural network model

2. Data pre-processing
   - Dataset split: train (70%), val (15%), test (15%)
   - All subsets contain similar data
   - Data formatting, switching between various formats:

   .csv <-> .pkl <-> .json <-> database;
   - Missing data (Null, NaN) – what should we replace them with?
   - Dataset is too large – use just a statistically relevant subset for training
   - Class imbalance – different weights for the results, repeat input from under-represented class, reduce input from over-represented class, etc.
   - Normalization – bring features in the same range of values



Steps in designing a neural network model

3. Model training
   - Initialize weights
   - Load data
   - Forward propagation
   - Compute cost function
   - Backpropagation
   - Update weights
   - Rinse & repeat

Steps in designing a neural network model

4. Model validation
   - Test the model on the validation subset to test its generalizing ability
   - This validation is not run at each iteration, but at a multiple of it (N=30, 100 etc.).

5. Model optimization
   - Hyperparameter tuning: adjusting model hyperparameters
   - Solve over-fitting with regularization methods: L1, L2, dropout, early stopping, dataset augmentation, etc.
   - Pre-train + fine-tune
   - Feature transfer
   - At the end of the optimization process we run the model one last time, on the test set. The metrics obtained on this subset are the officially reported ones.

Let's test it out – unit #7