Deep Learning Fundamentals

Mihai DOGARIU

www.mdogariu.aimultimedialab.ro

Summary

M4. Practical considerations

M5. Supervised learning applications

M6. Transfer learning

M4. Practical considerations

Steps in designing a neural network model

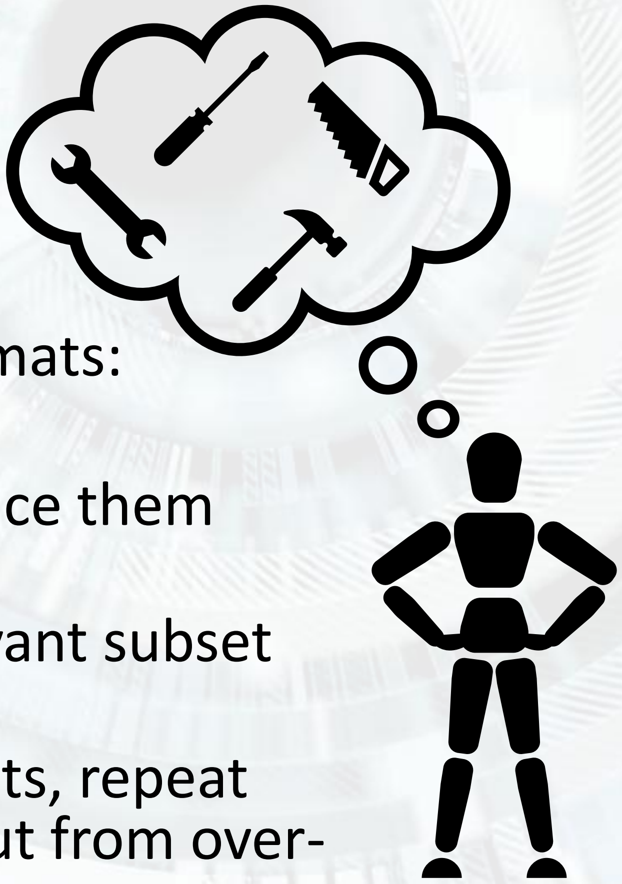
1. Choosing data
   - Number of examples in the dataset:
     - General: 10x no. of model parameters
     - Regression: 10 examples / prediction variable
     - Classification: 1000 examples / class
   - Sample quality – labeling done by experts, reduced noise
   - Dataset resources:
     - https://archive.ics.uci.edu/ml/index.php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

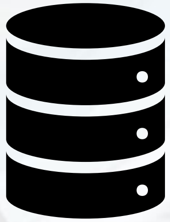
Steps in designing a neural network model

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

M5. Supervised learning applications

Supervised Deep Learning

Machine learning = we say that a system „learns” from experience E regarding a set of tasks T and a performance measure P, if the performance in solving the tasks T, measured by P, grows with the experience E.

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

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

input output task dimension

Supervised Deep Learning

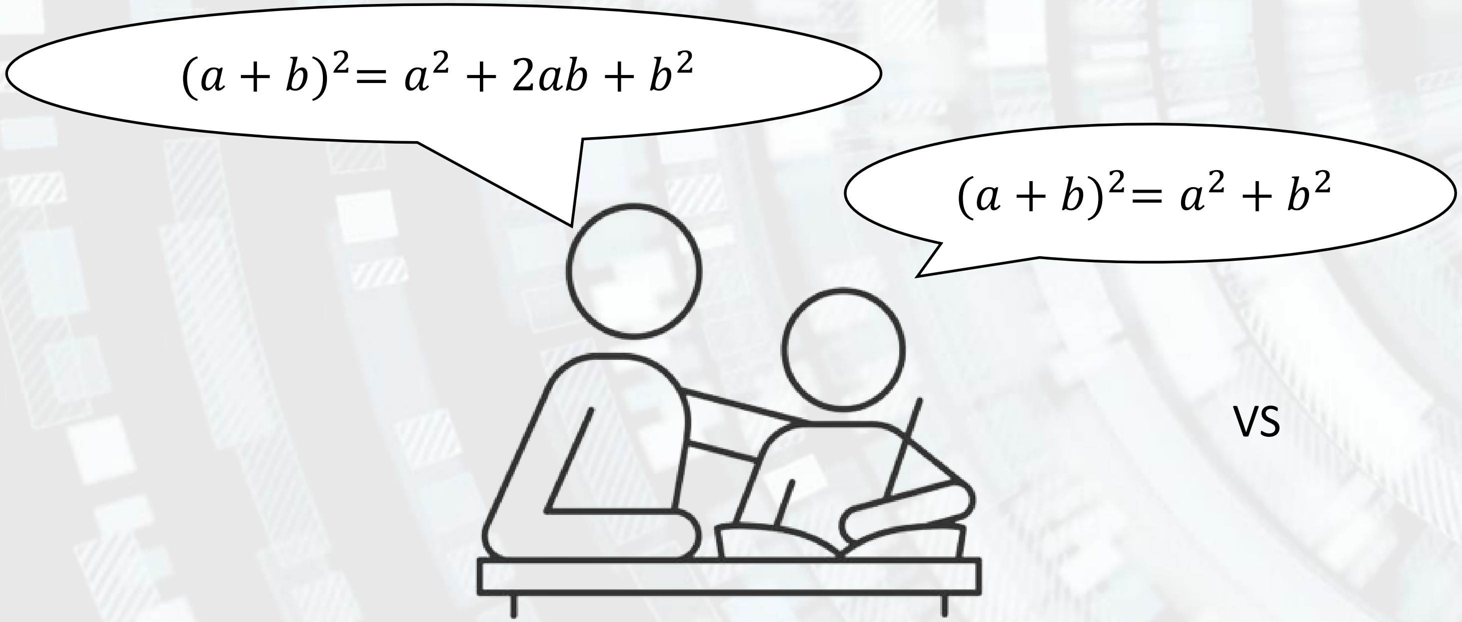
$$D = \{((x_i, y_i)|T), 1 \leq i \leq M\} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_M, y_M)\}$$

$$\left.\begin{array}{c} f(x_1) = y_1 \\ f(x_2) = y_2 \\ f(x_3) = y_3 \\ \dots \\ f(x_M) = y_M \end{array}\right\} f = ?$$

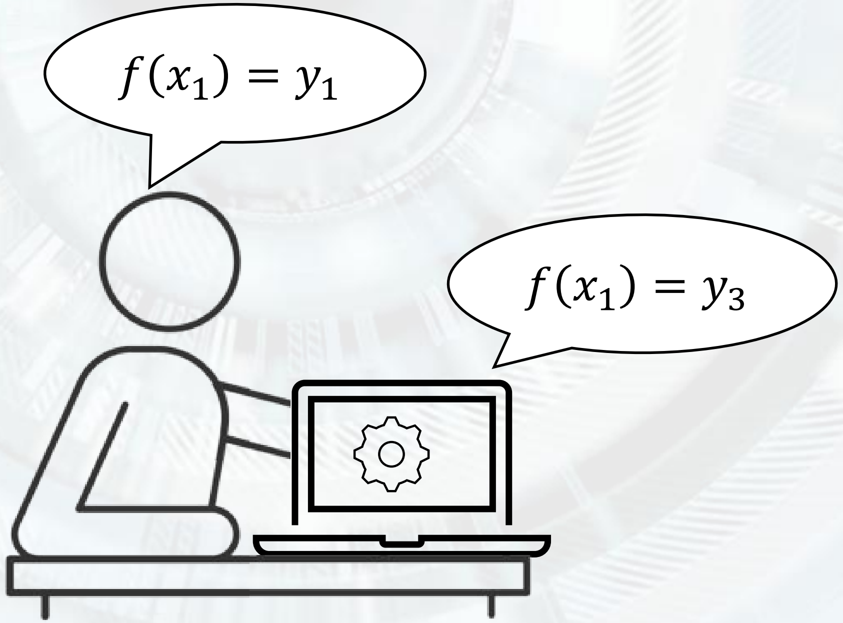
- Each (x_M, y_M) pair is called a training sample;
- x_M = input vector;
- y_M = real output/label.

Sursă imagine: iStock

Supervised Deep Learning



VS



Supervised Deep Learning

Supervised Learning = machine learning paradigm in which training data is labeled. Each training sample is composed of a descriptor and a label. The goal of supervised learning is to learn the association between the input features and their corresponding labels.

Supervised Deep Learning = supervised machine learning paradigm applied to deep neural networks (deep = several (>3, >7, >30, >50) layers).

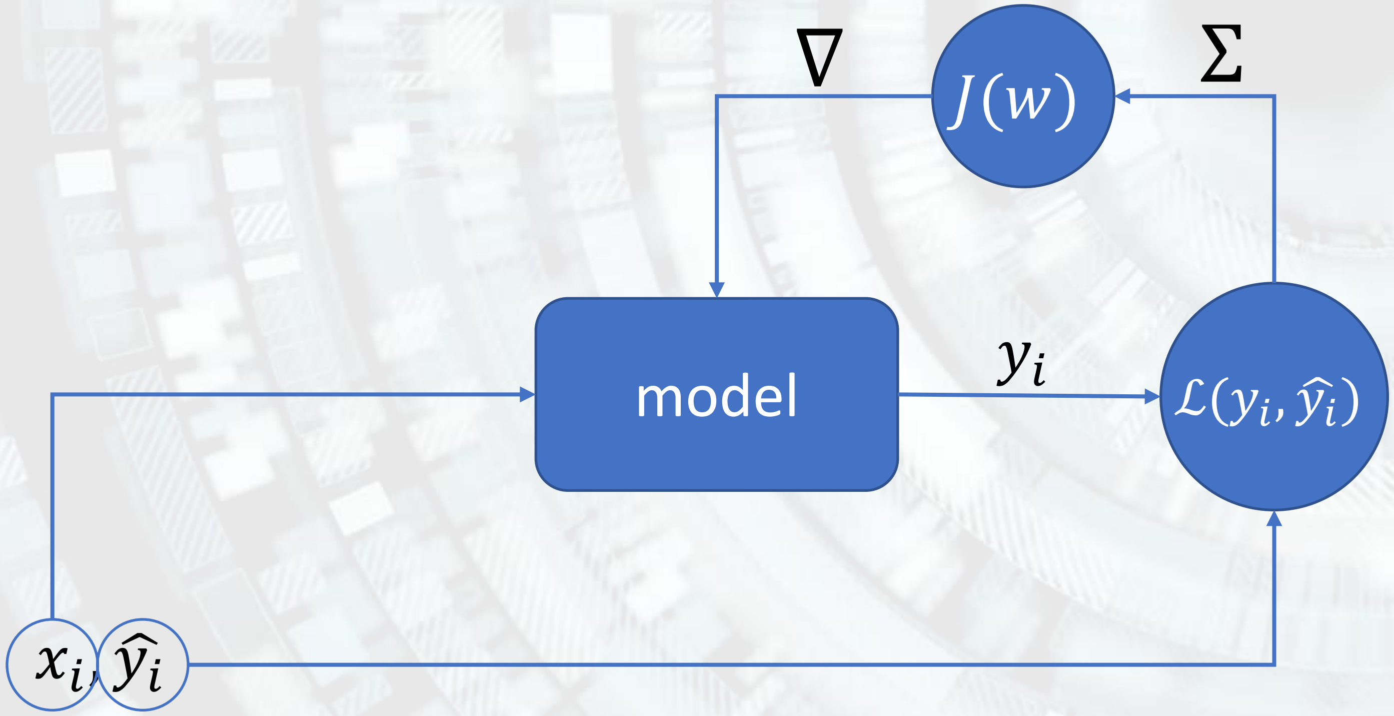
Supervised Deep Learning

- All training samples contain a label of their own;

- 2 main subcategories:
  1. M5.1. Regression – we would like to predict continuous values, adapted to the model that describes the dataset.
  2. M5.2. Classification – we would like to predict a discrete value, representing the class to which an input sample belongs.

- Definition collision:
  - A classifier can predict a continuous value under the form of a probability distribution.
  - A regressor can predict a discrete value under the form of an integer quantity.

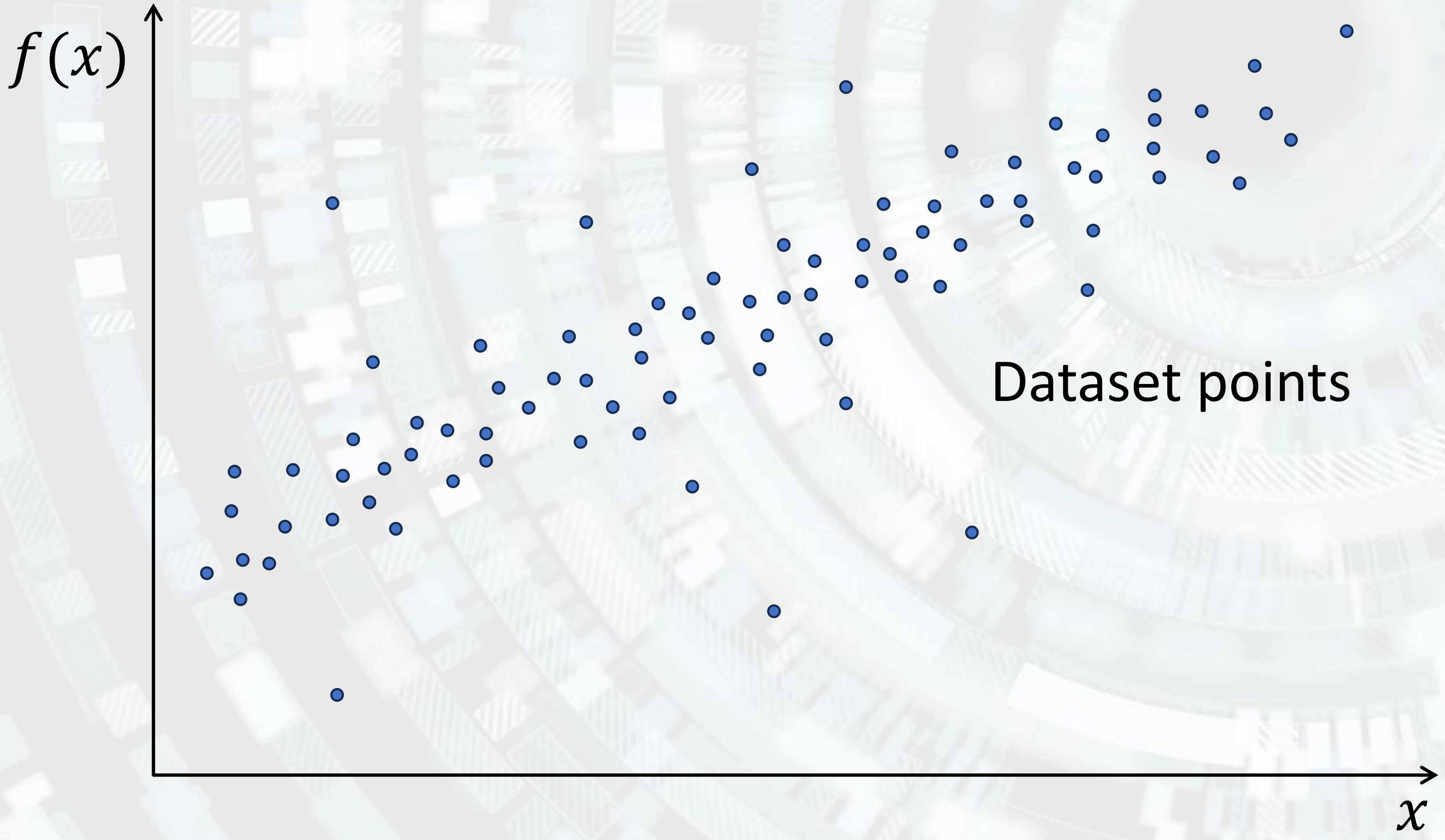
Supervised Deep Learning



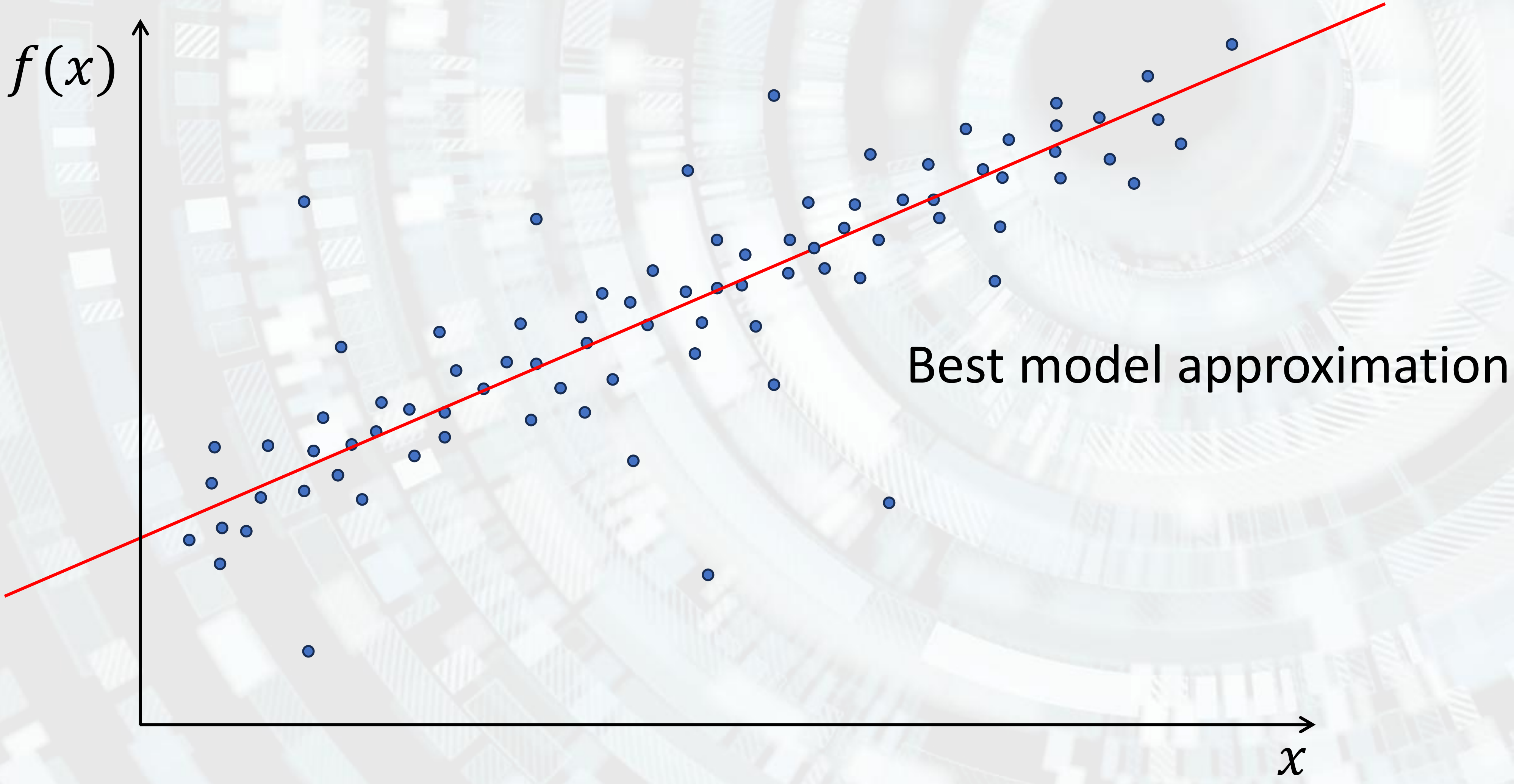
M5.1. Regression

- Maps the data points to the most optimized linear functions that can be used for prediction on new datasets.
- The model learns the equation of a path that interpolates all seen data.
- Predicted values are usually unbounded (any real value).
- Can depend on any number of variables and become quite abstract, e.g. combination of multivariate polynomials.

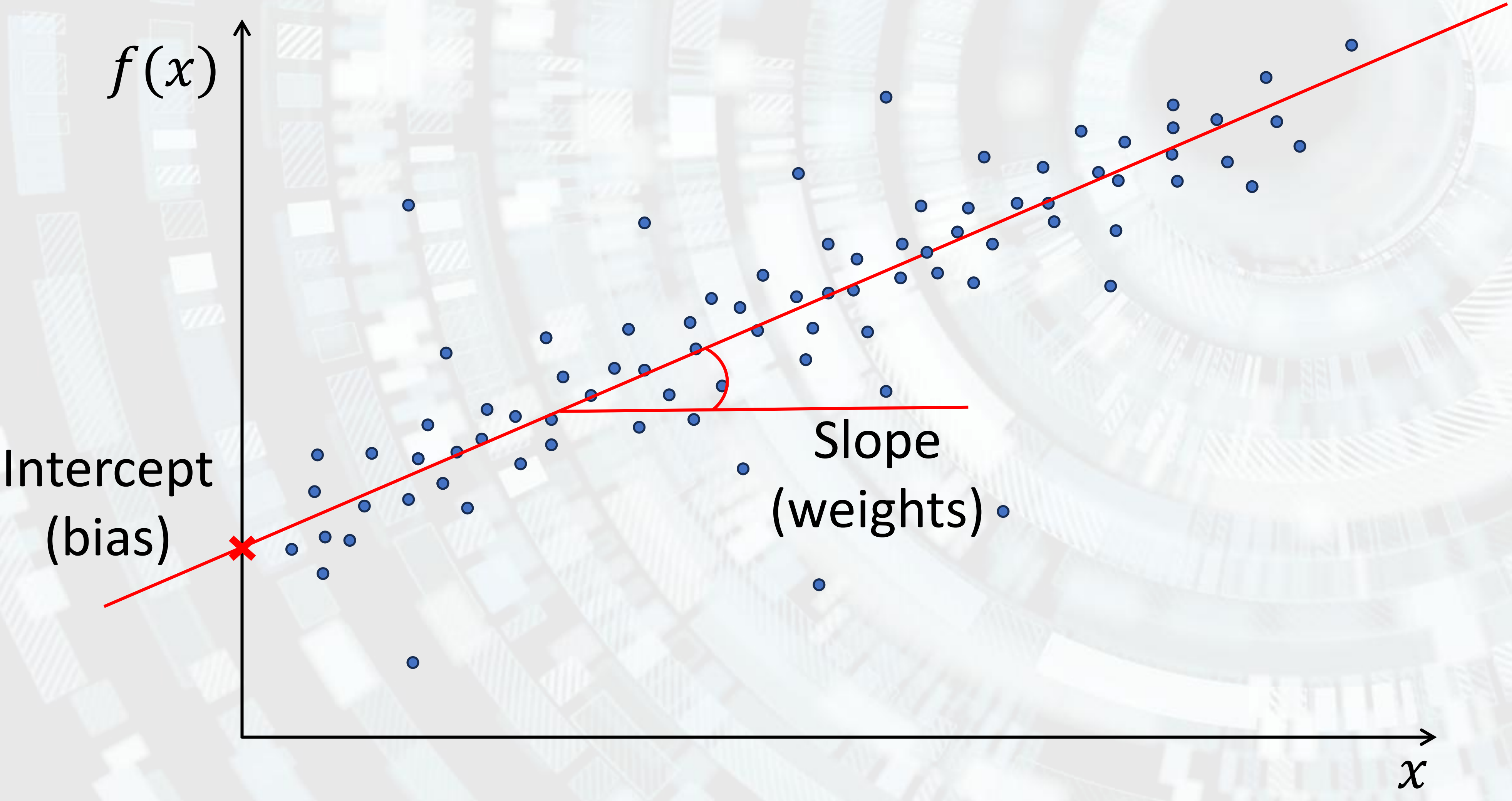
M5.1. Regression



M5.1. Regression



M5.1. Regression



M5.1. Regression

- Best loss function: Mean-Squared Error.
- Activation function depends on the application – we can even end the neural network with a simple fully connected layer, without any activation.
- Multiple inputs lead to multi-variable functions.
- Multi-layer neural networks lead to more complex function compositions (which can be derived from the network structure).

Let’s test it out – unit #8

M5.2. Classification

Classification = the task of assigning a label to a sample.

Characteristics:

- The content of the sample is treated as a whole ⇔ the label describes the entire input, not just a part of it.
- Any sample belongs to a single class.
- The output of a classifier is usually a probability distribution, not a categorical decision.
- Strongly depends on the qualitative and quantitative aspects of the training dataset.
- Multi-class classifiers usually have the last layer fully connected and a softmax activation.

M5.2. Classification

Classifier #1 → 78% Tabby cat, 15% Egyptian cat, 2% Siamese cat

Classifier #2 → 85% cat, 10% dog, 3% table

Classifier #3 → 90% cat, 10% not cat

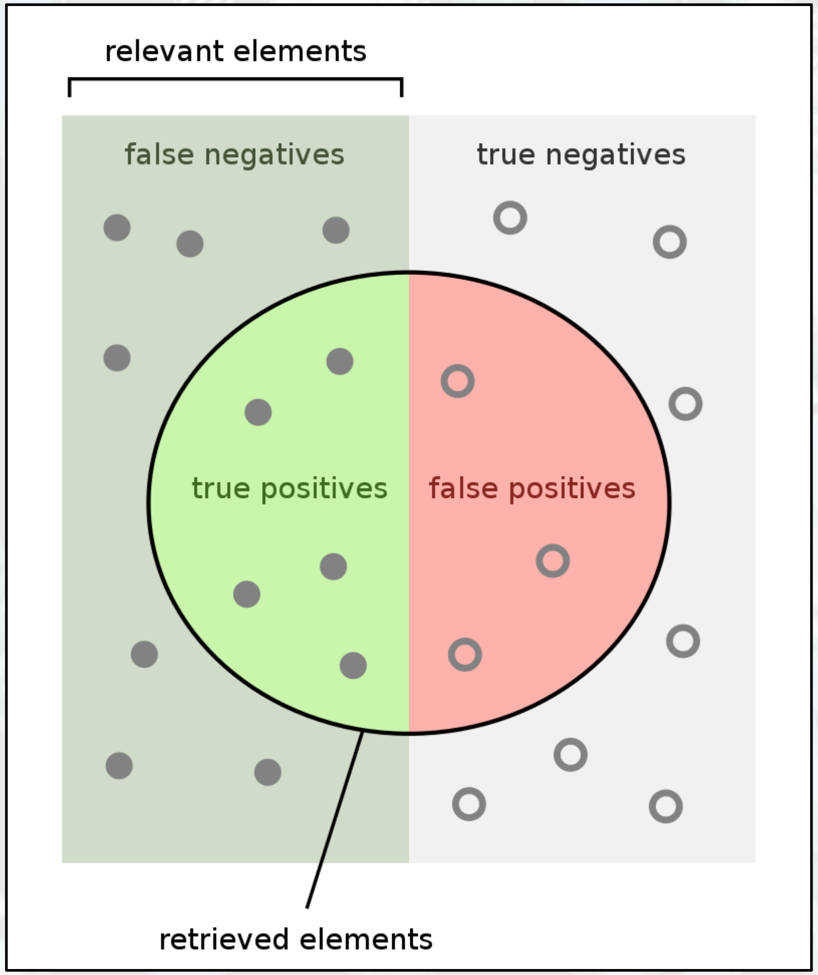
Dataset | Models | Metrics

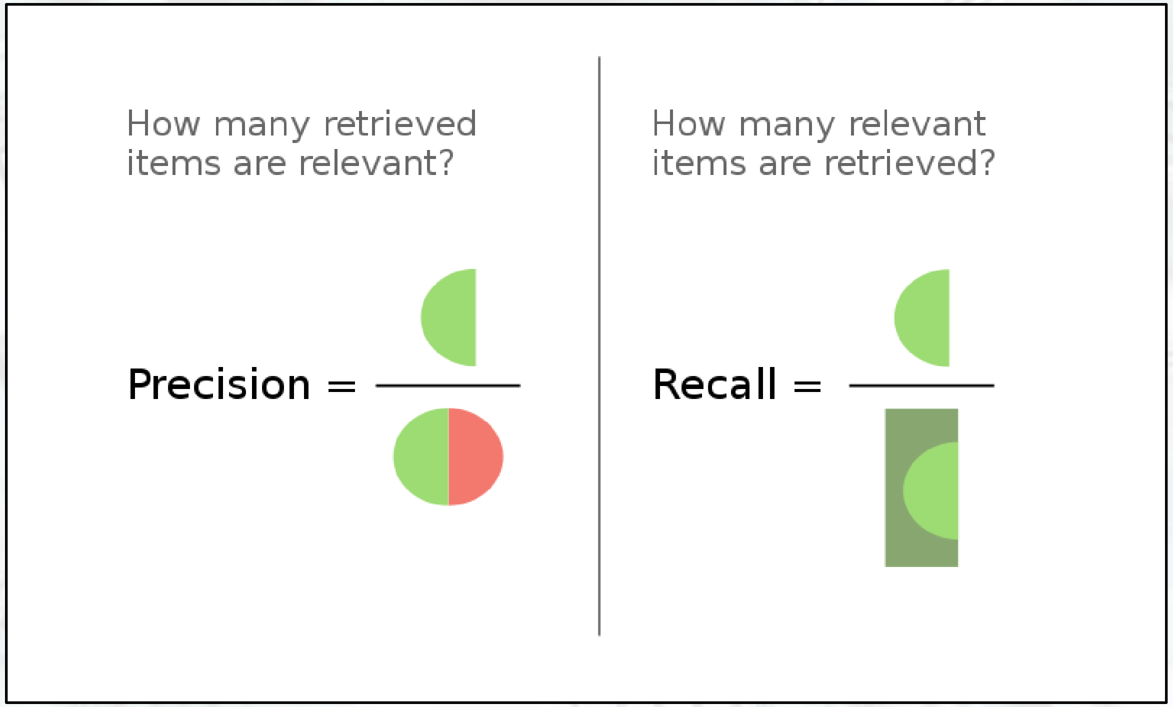
M5.2. Classification – metrics

Metric = quantitative method that is used to measure the performance of a system. It offers sortable numerical values such that we can quantify the progress made by a trainable model.

- It is calculated at the end of an epoch, on the entire dataset (train, val, test).
- It is usually not differentiable, so it cannot be used to drive the learning process.
- In some cases, though, it can be synonymous with the cost function (e.g. MAE, MSE).
- It must be interpreted in context. Usually, the values/ranges of values that represent the desired situation are mentioned.

M5.2. Classification – metrics





*Only for binary classification!

M5.2. Classification – metrics

Confusion matrix:

		Predicted	
		Positive	Negative
Real	Positive	True positive (tp)	False negative (fn)
	Negative	False positive (fp)	True negative (tn)

$$precision(P) = \frac{tp}{tp + fp}$$

$$recall(R) = \frac{tp}{tp + fn} = rata\ TP\ (TPR)$$

$$TN\ rate\ (TNR) = \frac{tn}{tn + fp}$$

$$FP\ rate(FPR) = \frac{fp}{tn + fp}$$

$$accuracy\ (ACC) = \frac{tp + tn}{tp + tn + fp + fn}$$

$$F - score\ (F) = 2\frac{PR}{P + R}$$

*Only for binary classification!

M5.2. Classification – metrics

Accuracy for imbalanced datasets

- For a binary classifier, the accuracy is defined as:

$$ACC = \frac{tp + tn}{tp + tn + fp + fn}$$

- Scenario:
  - 95 examples from class A and 5 examples from class B.
  - Classifier predicts class A in 100% of the cases => 95% accuracy (wrong).
  - Solution: use balanced accuracy.

M5.2. Classification – metrics

Accuracy for imbalanced datasets

- Balanced accuracy is defined as:

$$ACC = \frac{\mathrm{TPR} + \mathrm{TNR}}{2} = \frac{\frac{tp}{tp + fn} + \frac{tn}{tn + fp}}{2}$$

- Scenario:
  - 95 examples from class A and 5 examples from class B.
  - Classifier predicts class A in 100% of the cases.

Predicted

		Positive	Negative
Real	Positive	tp = 95	fn = 5
	Negative	fp = 0	tn = 95

$$ACC = \frac{\mathrm{TPR} + \mathrm{TNR}}{2} = \frac{\frac{0}{0 + 5} + \frac{95}{95 + 0}}{2} = 0.5$$

M5.2. Classification – metrics

Accuracy for multi-class classification:

$$ACC = \frac{\#correct\ classifications}{\#total\ classifications}$$

- E.g.: from 100 analyzed examples, 83 were correctly classified => 83% accuracy.

Top-k accuracy

- For an example from the database, the relative probability of belonging to each class is calculated.
- The output probabilities are ordered.
- If n among the first k best rated classes is also the real class => correct classification.

M5.2. Classification – metrics

Classifier

35% cat
3% table
1% velociraptor
12% airplane
6% truck
40% dog
2% baseball
1% mouse

40% dog
35% cat
12% airplane
6% truck
3% table
2% baseball
1% velociraptor
1% mouse

top-1: miss
top-2: hit
top-3: hit
top-5: hit

Let’s test it out – unit #9

M6. Transfer learning

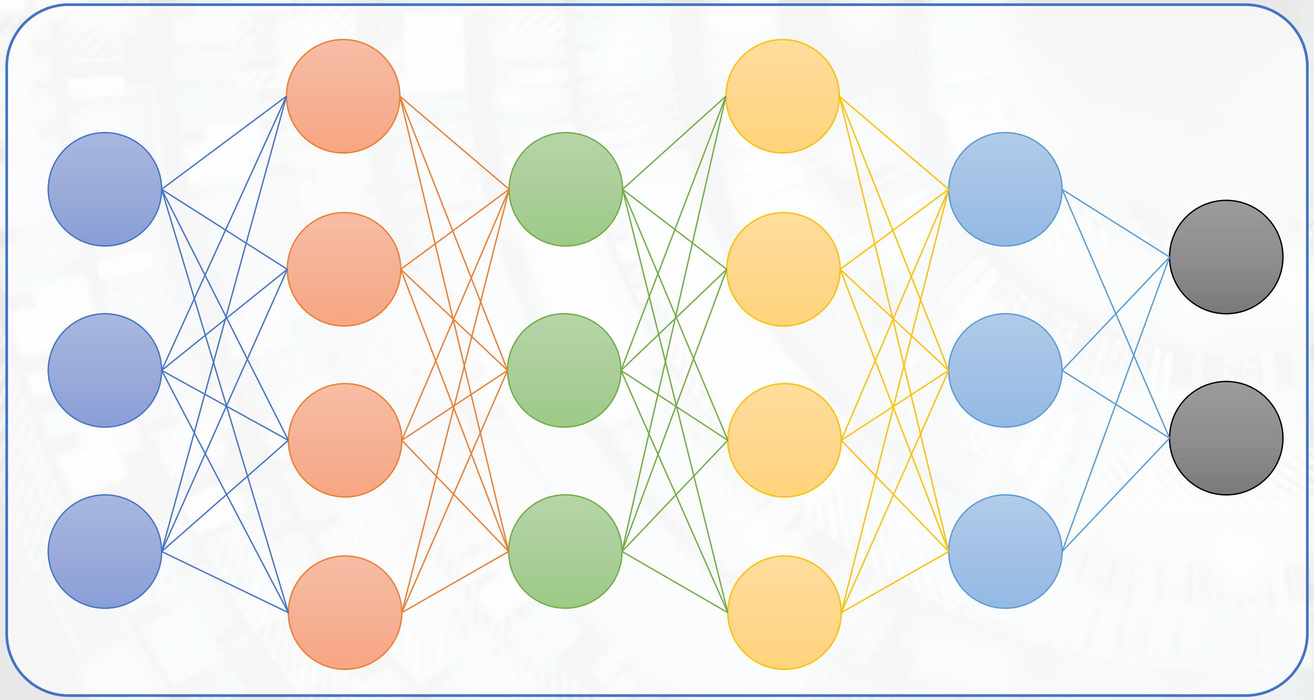
M6. Transfer learning

Transfer Learning = a machine learning technique where a model trained on one task is re-purposed and fine-tuned for a related, but different task.

- In general, model training takes a lot of time => so does retraining from scratch.
- Models learn more abstract features in the first layers (closer to input) and more task-specific layers in the last layers (closer to output).
- Part of the network model can be reused and repurposed for different, related, tasks.
- We can retrain the entire model or only the newly added components.

M6. Transfer learning - training on Task_A

Input Features
(Dataset Task_A)

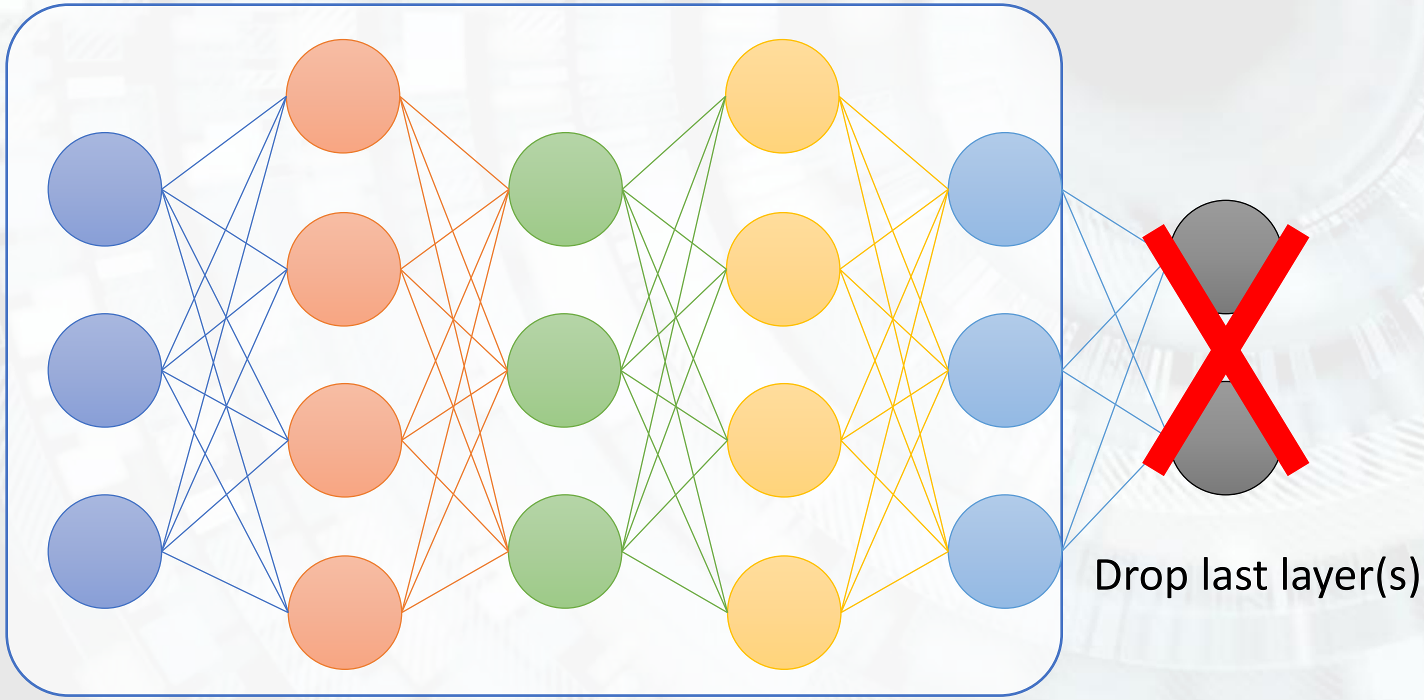


Output Values
(Dataset Task_A)

More generic features
(task independent)

More specific features
(task dependent)

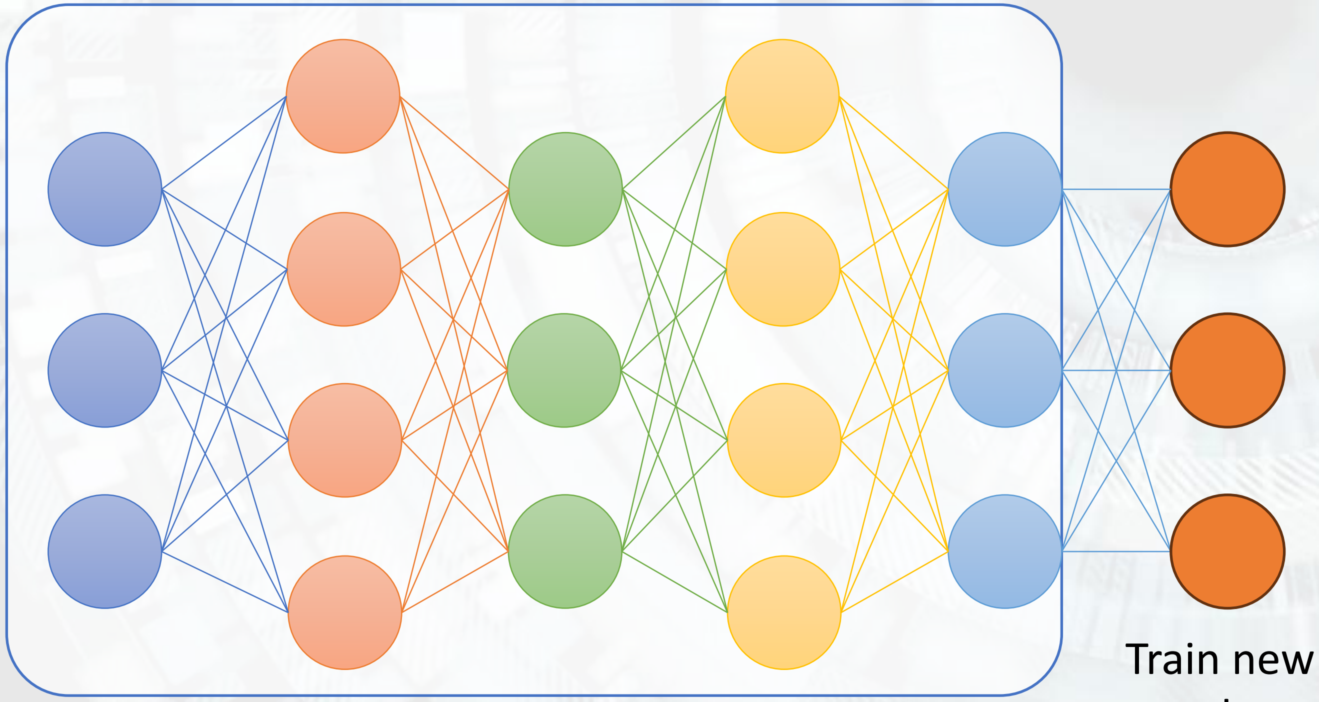
M6. Transfer learning - fine-tuning on Task$_B$



Keep the first part of the network (use pretrained weights as a very good initialization or freeze them completely)

M6. Transfer learning - fine-tuning on $Task_B$

Input Features
(Dataset $Task_B$)



Output Values
(Dataset $Task_B$)

Train new
output layer(s)

Keep the first part of the network (use pretrained weights as a very good initialization or freeze them completely)

M6. Transfer learning

Transferring learning from Task$_A$ to Task$_B$ depends on the datasets:

1. New dataset is **small** and **similar** to original dataset => train a linear classifier on the output layer only.

2. New dataset is **large** and **similar** to the original dataset => fine-tune the entire network (acts as a continuation of the original training).

3. New dataset is **small** but very **different** from the original dataset => train a linear classifier from somewhere earlier in the network.

4. New dataset is **large** and very **different** from the original dataset => fine-tune the entire network (use the pre-trained weights as a solid initialization for the new task).

Let's test it out – unit #10