Neural Network & Deep Learning

Keras & TensorFlow 2

Hichem Felouat

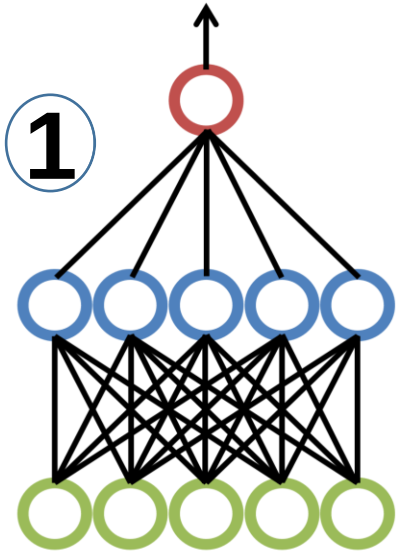
hichemfel@gmail.com

https://www.linkedin.com/in/hichemfelouat/

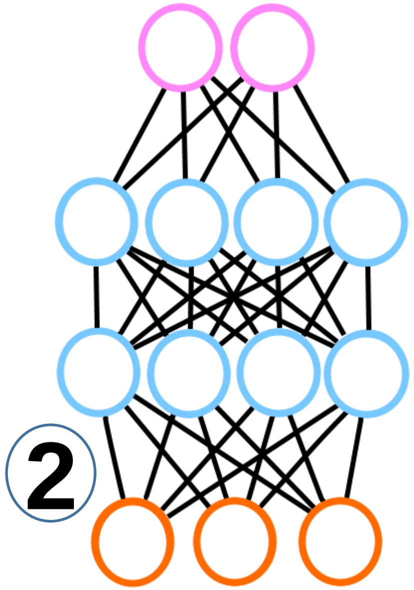
Regression MLPs

Regression MLPs

1) If you want to **predict a single value** (e.g., the price of a house, given many of its features), then you just need a **single output neuron**, its output is the predicted value.



2) For **multivariate regression** (i.e., **to predict multiple values at once**), you need **one output neuron per output dimension**. For example, to locate the center of an object in an image, you need to predict 2D coordinates, so you need two output neurons.



Regression MLPs

Typical regression MLP architecture:

Hyperparameter	Typical value
input neurons	One per input feature
hidden layers	Depends on the problem, but typically 1 to 5
neurons per hidden layer	Depends on the problem, but typically 10 to 100
output neurons	1 per prediction dimension
Hidden activation	ReLU (relu)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE (mean_squared_error) or MAE/Huber (if outliers)

Regression MLPs - Available losses

Regression losses

- MeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- mean_squared_error function
- mean_absolute_error function
- mean_absolute_percentage_error function
- mean_squared_logarithmic_error function
- cosine_similarity function
- Huber class
- huber function
- LogCosh class
- log_cosh function

https://keras.io/api/losses/

Classification MLPs

Classification MLPs

Binary classification:

- We just need a **single output neuron** using the **logistic** activation function (**sigmoid**) **0 or 1**.

Multilabel Binary Classification:

- We need **one output neuron for each positive class**.

For example: you could have an email classification system that predicts whether each incoming email is **spam or not-spam** and simultaneously predicts whether it is an **urgent or nonurgent** email. In this case, you would need **two output neurons**, both using the **logistic** activation function.

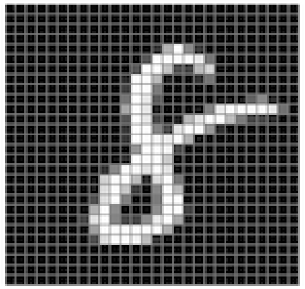
Classification MLPs

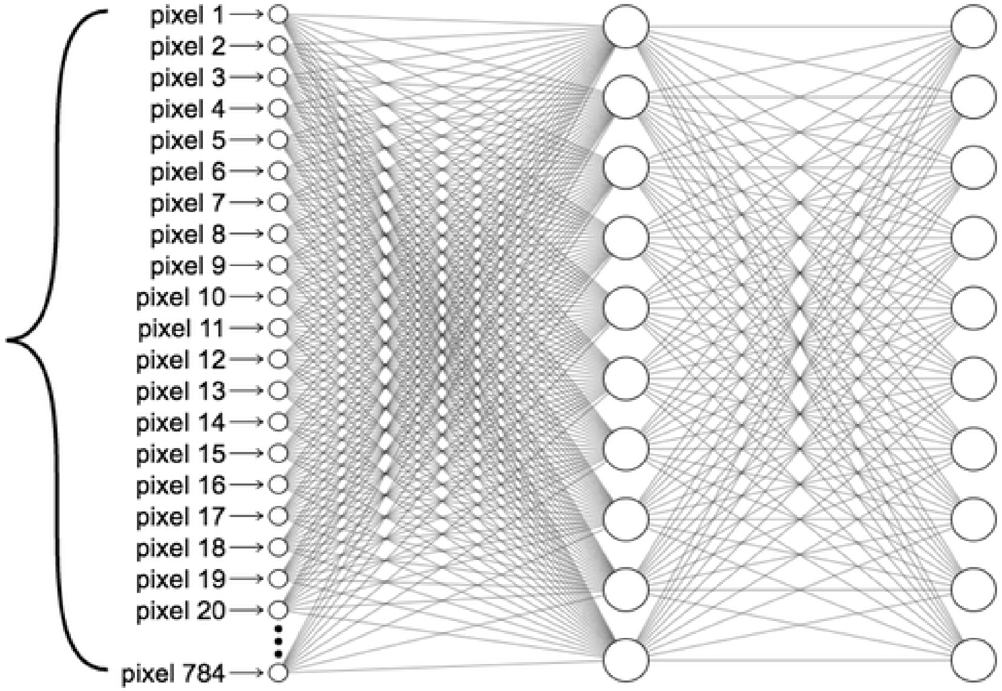
Multiclass Classification:

We need to have **one output neuron per class**, and we should use the **softmax** activation function for the whole output layer.

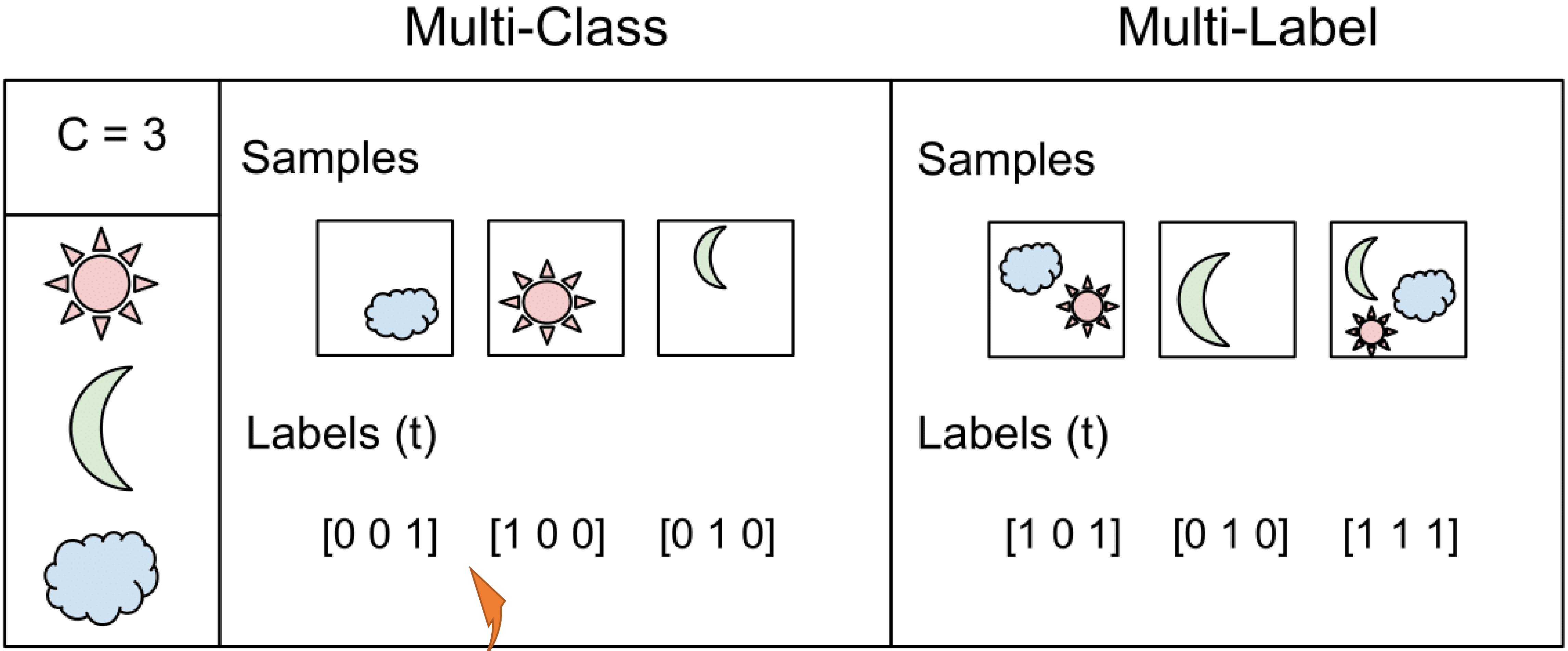
For example:

classes **0 through 9** for digit image classification [28, 28].





Classification MLPs



Integers : Labels **[2, 0, 1]**
one-hot : Labels **[[0 0 1], [1 0 0], [0 1 0]]**

Classification MLPs

To convert categorical features to such integer codes, we can use the **OrdinalEncoder**. This estimator transforms each categorical feature to one new feature of integers (0 to n_categories - 1).

```
from sklearn import preprocessing
#genders = ['female', 'male']
#locations = ['from Africa', 'from Asia', 'from Europe', 'from US']
#browsers = ['uses Chrome', 'uses Firefox', 'uses Safari']

X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Safari'],
    ['female', 'from Asia', 'uses Firefox'], ['male', 'from Africa', 'uses Chrome']]
enc = preprocessing.OrdinalEncoder()
X_enc = enc.fit_transform(X)
print(X_enc)
```

```
[[1. 3. 2.]
 [0. 2. 2.]
 [0. 1. 1.]
 [1. 0. 0.]]
```

Classification MLPs

Datetime Feature Engineering: we can extract the component of the date-time part **(year, quarter, month, day, day_of_week, day_of_year, week_of_year, time, hour, minute, second, day_part)** from the given date-time variable.

	date_issued	date_issued:year	date_issued:month	date_issued:day
0	2013-06-11	2013	6	11
1	2014-05-08	2014	5	8
2	2013-10-26	2013	10	26
3	2015-08-20	2015	8	20
4	2014-07-22	2014	7	22

```
df["Date"] = pd.to_datetime(df["Date"])

df["year"]    = df["Date"].dt.year
df["month"] = df["Date"].dt.month
df["day"]     = df["Date"].dt.day
df["week_of_year"] = df["Date"].dt.weekofyear
df["day_of_year"]    = df["Date"].dt.dayofyear

df= df.drop(["Date"], axis=1)
print(df.info())
```

Classification MLPs

Label Encoding (LabelEncoder) is a popular encoding technique for handling categorical variables. In this technique, each label is assigned a unique integer based on alphabetical orderingwith value between 0 and n_classes-1.

```
# Encoding Categorical Labels
from sklearn import preprocessing

labels = ["India", "US", "Japan", "US", "Japan"]
le = preprocessing.LabelEncoder()
new_labels = le.fit_transform(labels)

print(new_labels)

print("inverse_transform : \n",
                                  le.inverse_transform([2, 0, 1]))
```

The country names do not have an **order** or **rank**. But, when label encoding is performed, the country names are ranked based on the alphabets. Due to this, there is a very high probability that the model captures the relationship between countries such as India < Japan < US.

Classification MLPs

One-Hot Encoding is another popular technique for treating categorical variables. It simply creates additional features based on the number of unique values in the categorical feature. Every unique value in the category will be added as a feature.

Country	Age	Salary
India	44	72000
US	34	65000
Japan	46	98000
US	35	45000
Japan	23	34000

Country	Age	Salary
0	44	72000
2	34	65000
1	46	98000
2	35	45000
1	23	34000

0	1	2	Age	Salary
1	0	0	44	72000
0	0	1	34	65000
0	1	0	46	98000
0	0	1	35	45000
0	1	0	23	34000

Classification MLPs

```
# Importing one hot encoder
from sklearn.preprocessing import OneHotEncoder

# Creating one hot encoder object
onehotencoder = OneHotEncoder()

# Reshape the 1-D country array to 2-D as fit_transform expects 2-D and finally fit the object
X = onehotencoder.fit_transform(my_data.Country.values.reshape(-1,1)).toarray()
print(X)
```

Classification MLPs

Typical classification MLP architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
input neurons	One per input feature	One per input feature	One per input feature
hidden layers neurons per hidden layer	Depends on the problem	Depends on the problem	Depends on the problem
output neurons	1	1 per label	1 per class
Hidden activation	ReLU (relu)	ReLU (relu)	ReLU (relu)
Output layer activation	Logistic (sigmoid)	Logistic (sigmoid)	Softmax (softmax)
Loss function	**binary_crossentropy**	**binary_crossentropy**	Cross entropy

Multiclass classification : sparse_categorical_crossentropy [Labels are Integers]
categorical_crossentropy [Labels are one-hot]

Classification MLPs - Cross Entropy

Loss function for binary (yes/no) classification

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = -\sum_{i=1}^{n} (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

Loss function for multi-class classification.

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = -\sum_{i=1}^{n} \sum_{k=1}^{k} (y_{ik} \cdot \log \hat{y}_{ik})$$

- Ground truth: $\boldsymbol{y}$
- Prediction: $\hat{\boldsymbol{y}}$
- Loss function: $L(\boldsymbol{y}, \hat{\boldsymbol{y}})$

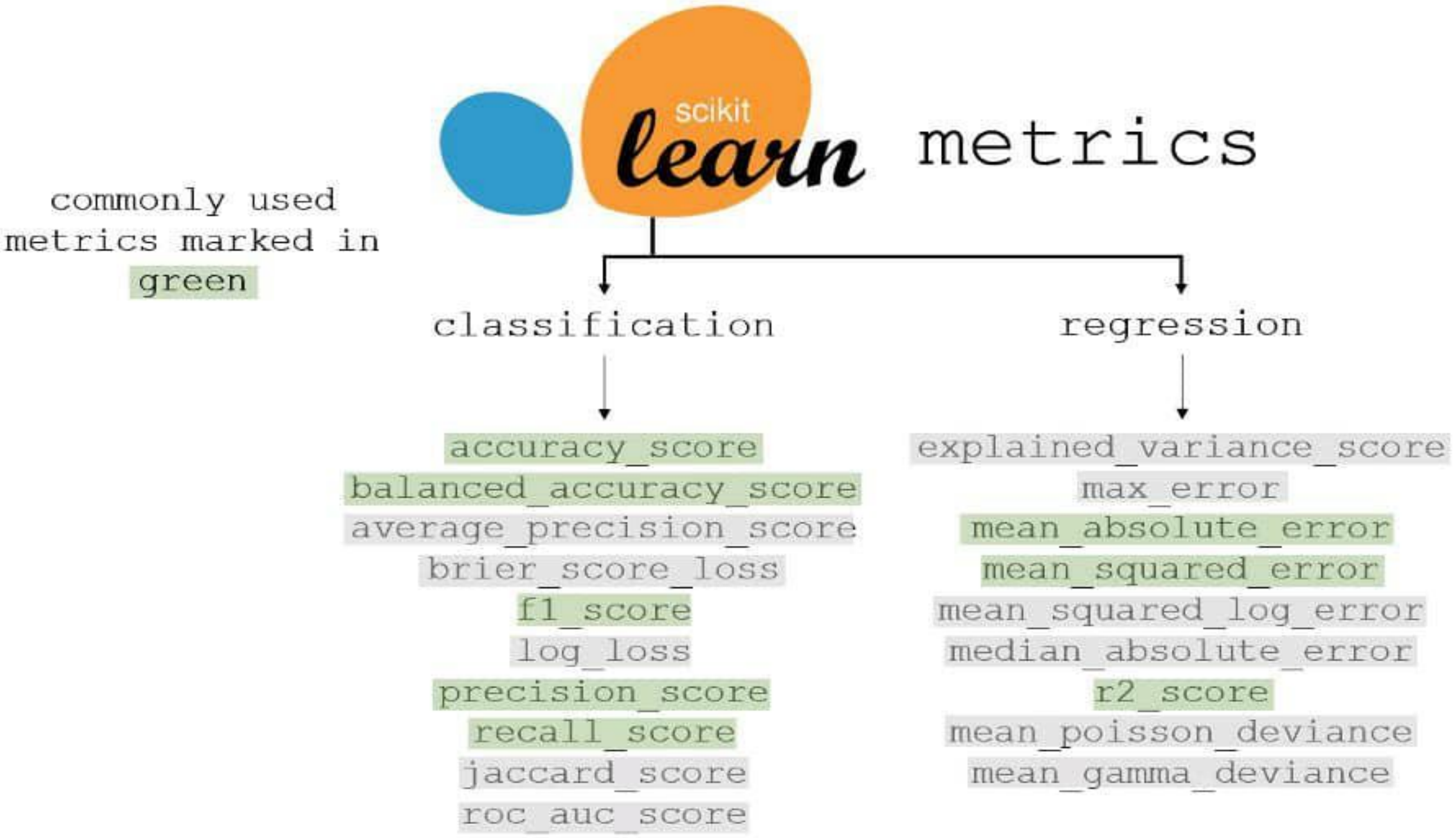
Classification MLPs - Available losses

Probabilistic losses

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- Poisson class
- binary_crossentropy function
- categorical_crossentropy function
- sparse_categorical_crossentropy function
- poisson function
- KLDivergence class
- kl_divergence function

https://keras.io/api/losses/

Evaluation Metrics



Evaluation Metrics - Available metrics

Accuracy metrics

- Accuracy class
- BinaryAccuracy class
- CategoricalAccuracy class
- TopKCategoricalAccuracy class
- SparseTopKCategoricalAccuracy class

Probabilistic metrics

- BinaryCrossentropy class
- CategoricalCrossentropy class
- SparseCategoricalCrossentropy class
- KLDivergence class
- Poisson class

Regression metrics

- MeanSquaredError class
- RootMeanSquaredError class
- MeanAbsoluteError class
- MeanAbsolutePercentageError class
- MeanSquaredLogarithmicError class
- CosineSimilarity class
- LogCoshError class

Classification metrics based on True/False positives & negatives

- AUC class
- Precision class
- Recall class
- TruePositives class
- TrueNegatives class
- FalsePositives class
- FalseNegatives class
- PrecisionAtRecall class
- SensitivityAtSpecificity class
- SpecificityAtSensitivity class

Image segmentation metrics

- MeanIoU class

Hinge metrics for "maximum-margin" classification

- Hinge class
- SquaredHinge class
- CategoricalHinge class

https://keras.io/api/metrics/

Evaluation Metrics - Regression

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y})^2}{\sum (y_i - \bar{y})^2}$$

Where,

$\hat{y}$ $-$ *predicted value of* y

$\bar{y}$ $-$ *mean value of* y

Evaluation Metrics - Classification

Confusion matrix

		Predicted class	
		+	−
Actual class	+	**TP** True Positives	**FN** False Negatives Type II error
	−	**FP** False Positives Type I error	**TN** True Negatives

Evaluation Metrics - Classification

Metric	Formula	Interpretation
Accuracy	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}$	Overall performance of model
Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	Coverage of actual positive sample
Specificity	$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$	Coverage of actual negative sample
F1 score	$\frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$	Hybrid metric useful for unbalanced classes

Evaluation Metrics - Classification

Multilabel binary classification:

- Accuracy
- Hamming loss

Hamming loss

True labels / Predicted labels

TEXT	SERVICE	FOOD	ANECDOTES	PRICE	AMBIENCE	SERVICE	FOOD	ANECDOTES	PRICE	AMBIENCE
but the staff was so horrible to us	1	0	0	0	0	0	1	0	0	0
to be completely fair the only redeeming facto...	0	1	1	0	0	1	1	0	0	0
the food is uniformly exceptional with a very ...	0	1	0	0	0	0	0	0	1	0
where gabriela personaly greets you and recomm...	1	0	0	0	0	1	0	0	0	0
for those that go once and dont enjoy it all i...	0	0	1	0	0	1	0	0	0	0

Total number of predictions (TNP) = 25
Total number of incorrect predictions (TNIP) = 8

Accuracy = TNIP/TNP = 8/25 = 0.32

Custom Loss and Custom Metrics

Sometimes there is no good loss/metrics available or you need to implement some modifications.

```
# calculate cross entropy
def custom_loss_function(y_true, y_pred):
  .
  return loss
# Calculate accuracy percentage between two lists
def accuracy_metric(y_true, y_pred):
  .
  return accuracy

# Compiling the model
model.compile(optimizer="sgd", loss=custom_loss_function, metrics=[accuracy_metric])
# Training and evaluating the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

Custom Loss and Custom Metrics

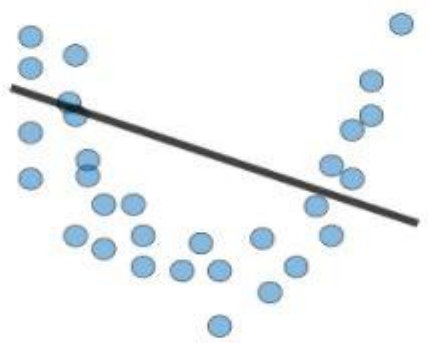
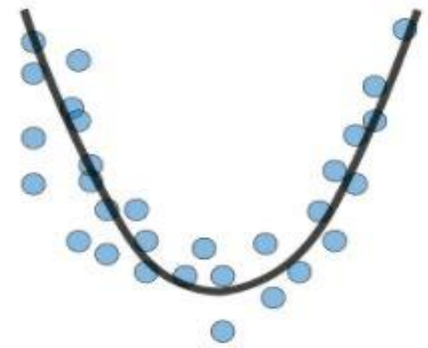
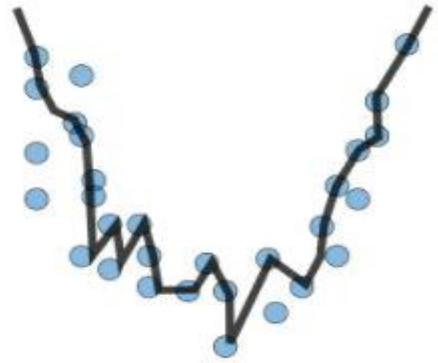
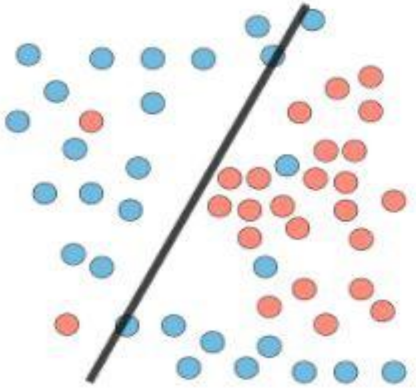
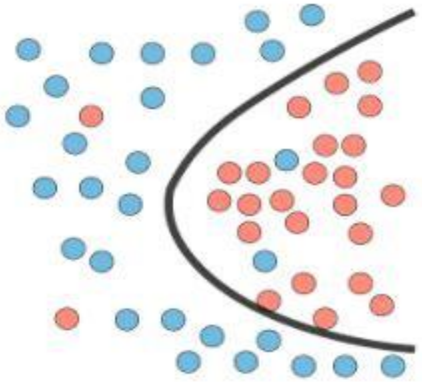
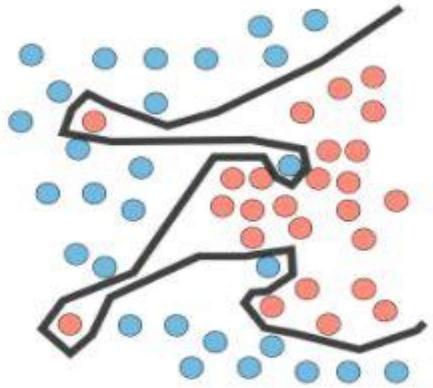
If you want a different threshold :

```
def create_my_loss(threshold=1.0):
    def custom_loss_function(y_true, y_pred):
        .
        .
        return loss

    return custom_loss_function

model.compile(loss=create_my_loss(2.0), optimizer="adam")
```

Overfitting and Underfitting

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			

Overfitting and Underfitting

- **The bias** is known as the **difference** between the **prediction** of the values by the ML model and the **correct** value. Being high in biasing gives a large error in training as well as testing data.
- **Its recommended** that an algorithm should always be **low biased** to avoid the problem of underfitting.

- The variability of model prediction for a given data point which tells us **spread of our data** is called the **variance** of the model. The model with **high variance** has a very **complex fit** to the training data and thus is not able to fit accurately on the data which it hasn't seen before.

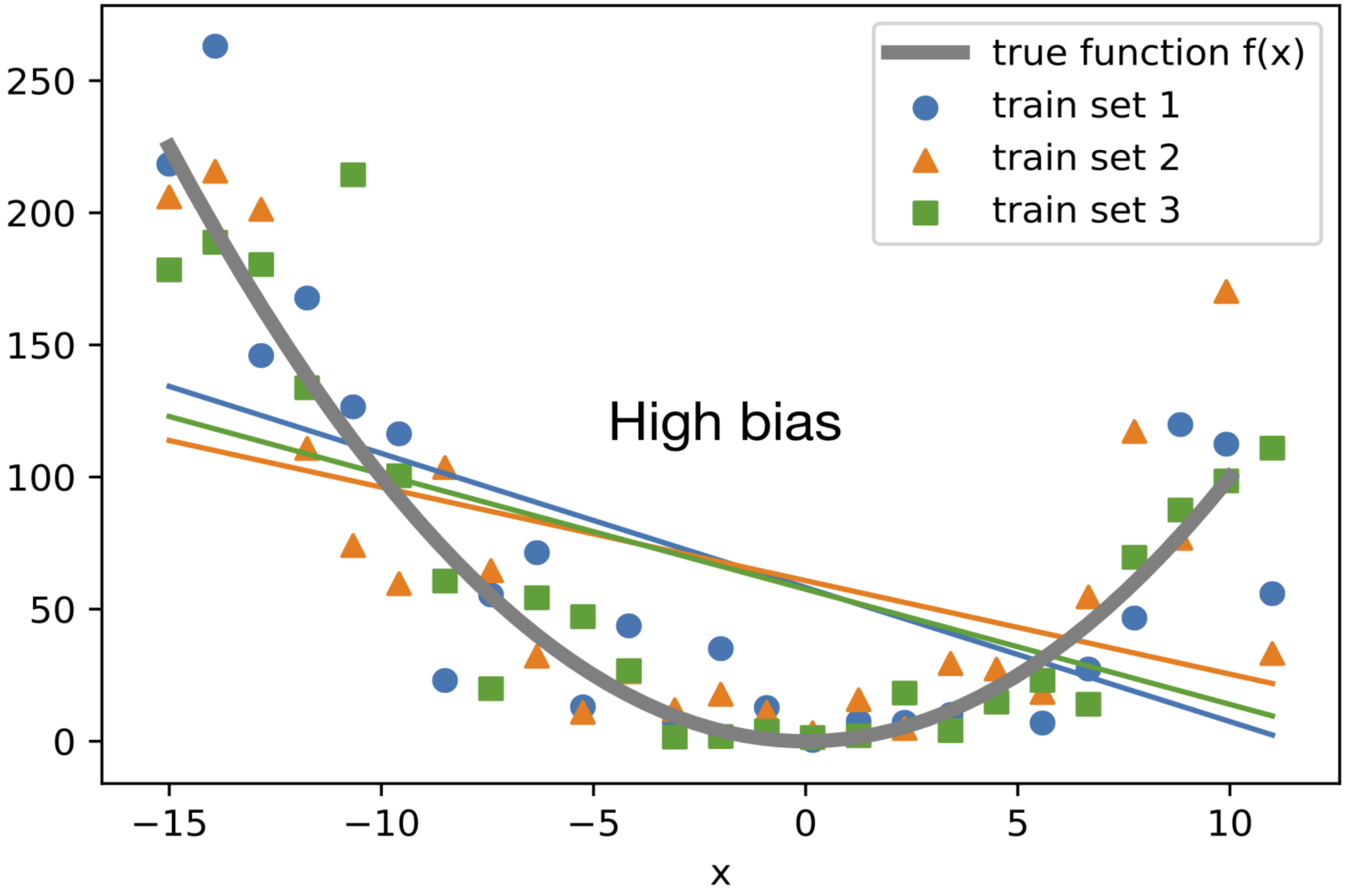
Overfitting and Underfitting

- Supervised Learning - Under the Hood
  - supervised learning: $y = f(x)$, f is unknown.
  - Goals
    - Find a model $\hat{f}$ that best approximates: f : $\hat{f} \approx f$
    - $\hat{f}$ can be Logistic Regression, Decision Tree, Neural Network,...
    - Discard noise as much as possible
    - **End goal**: $\hat{f}$ should achieve a low predictive error on unseen datasets.
- Difficulties in Approximating f
  - Overfitting: $\hat{f}(x)$ fits the training set noise.
  - Underfitting: $\hat{f}$ is not flexible enough to approximate f.
- Generalization Error
  - Generalization Error of $\hat{f}$: Does $\hat{f}$ generalize well on unseen data?
  - It can be decomposed as follows:

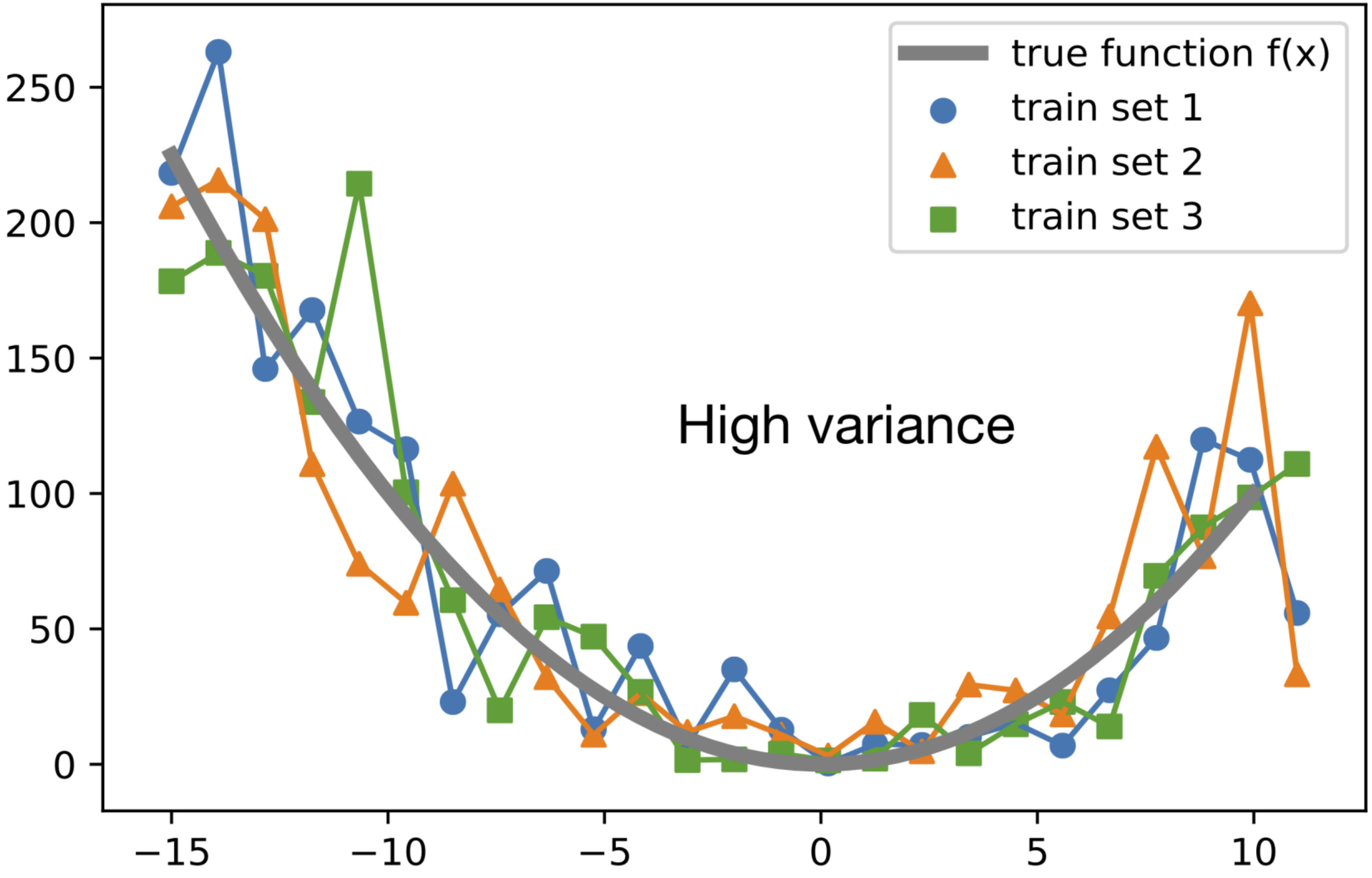
$$\hat{f} = \text{bias}^2 + \text{variance} + \text{irreducible error}$$

  - Bias: error term that tells you, on average, how much $\hat{f} \neq f$.

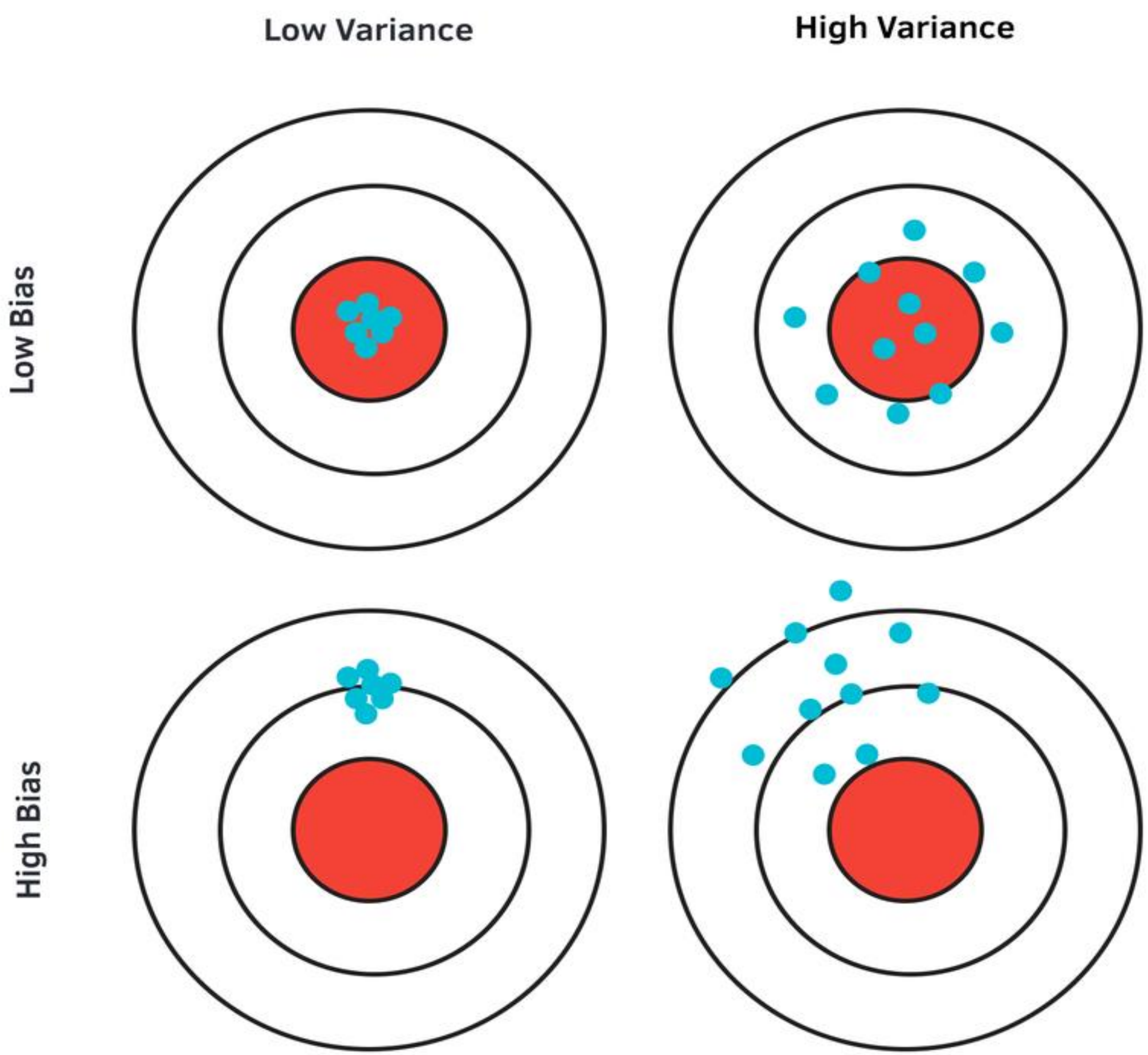
Overfitting and Underfitting



Overfitting and Underfitting



Overfitting and Underfitting



Classification MLPs

If the training set was very skewed, with some classes being **overrepresented and others underrepresented**, it would be useful to set the **class_weight** argument when calling the fit().

sample_weight: Per-instance weights could be useful if **some instances were labeled by experts** while others were labeled using a crowdsourcing platform: you might want to give more weight to the former.

Classification MLPs

- **If you are not satisfied with the performance of your model, you should go back and tune the hyperparameters.**

- Try another **optimizer** — **Gradient Descent, Momentum Optimization, Nesterov Accelerated Gradient, AdaGrad, RMSProp, Adam, Nadam**
- Try tuning model hyperparameters such as **the number of layers**, **the number of neurons per layer**, and the **types of activation functions** to use for each hidden layer.
- Try tuning other hyperparameters, such as **the number of epochs** and **the batch size**.

☐ *Once you are satisfied with your model's validation accuracy, you should evaluate it on the test set to estimate the generalization error before you deploy the model to production.*

How to Save and Load Your Model

Keras use the **HDF5 format** to save both the **model's architecture (including every layer's hyperparameters)** and the values of all **the model parameters** for every layer **(e.g., connection weights and biases)**. It also saves **the optimizer** (including its hyperparameters and any state it may have).

```
model.save("my_keras_model.h5")
```

Loading the model:

```
model = keras.models.load_model("my_keras_model.h5")
```

In case we use custom loss/metric function:

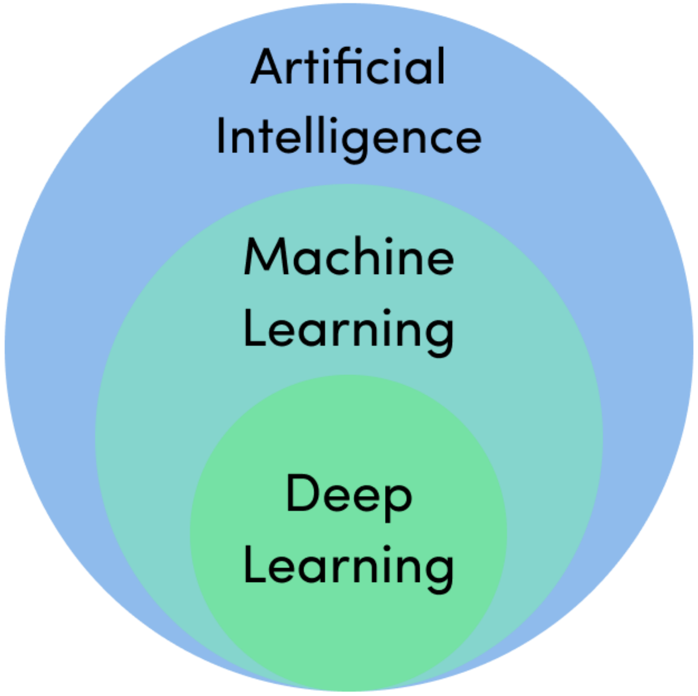
```
model = keras.models.load_model("my_keras_model.h5", compile=False)

model = keras.models.load_model("my_model.h5",
                    custom_objects={"custom_loss_function": create_my_loss(2.0)})
```

Deep Learning DL

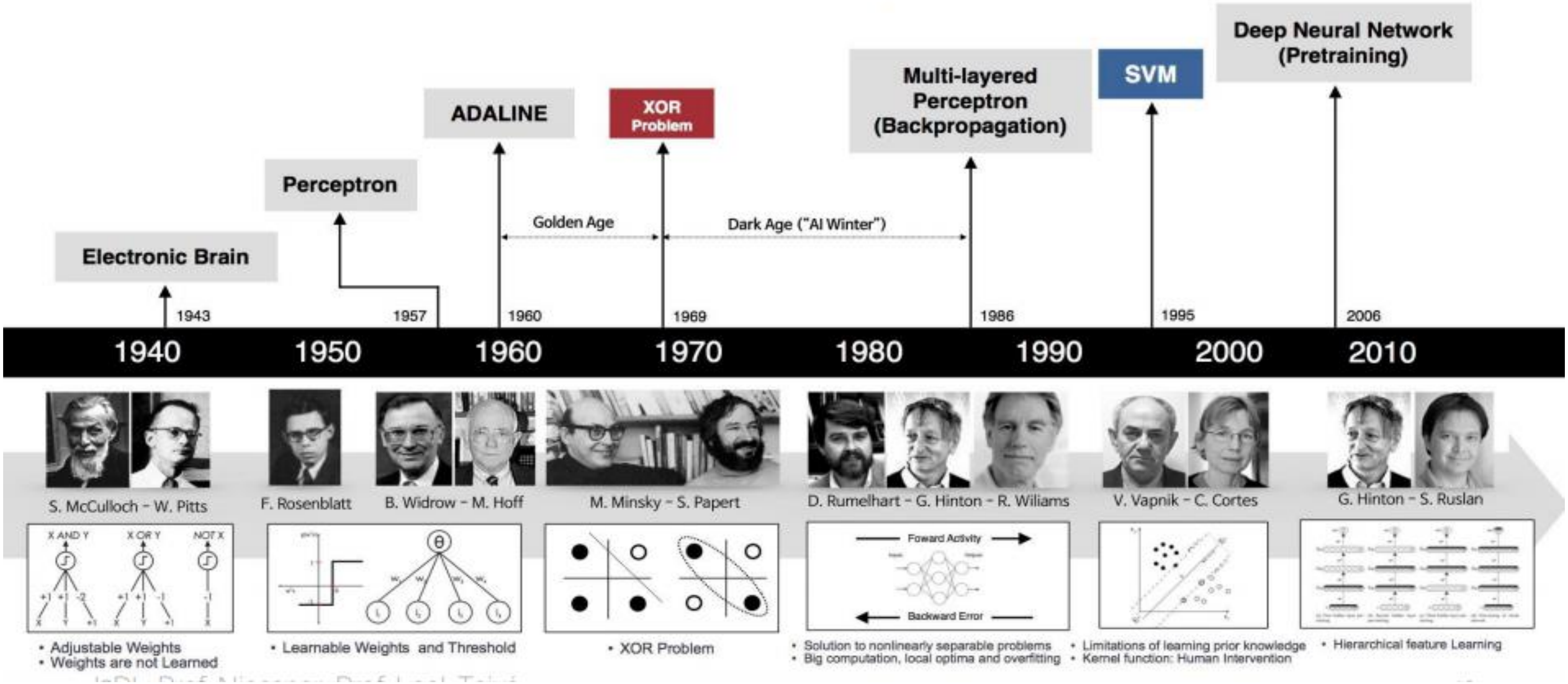
Deep Learning DL

DL is a subfield of ML, developed by several researchers.

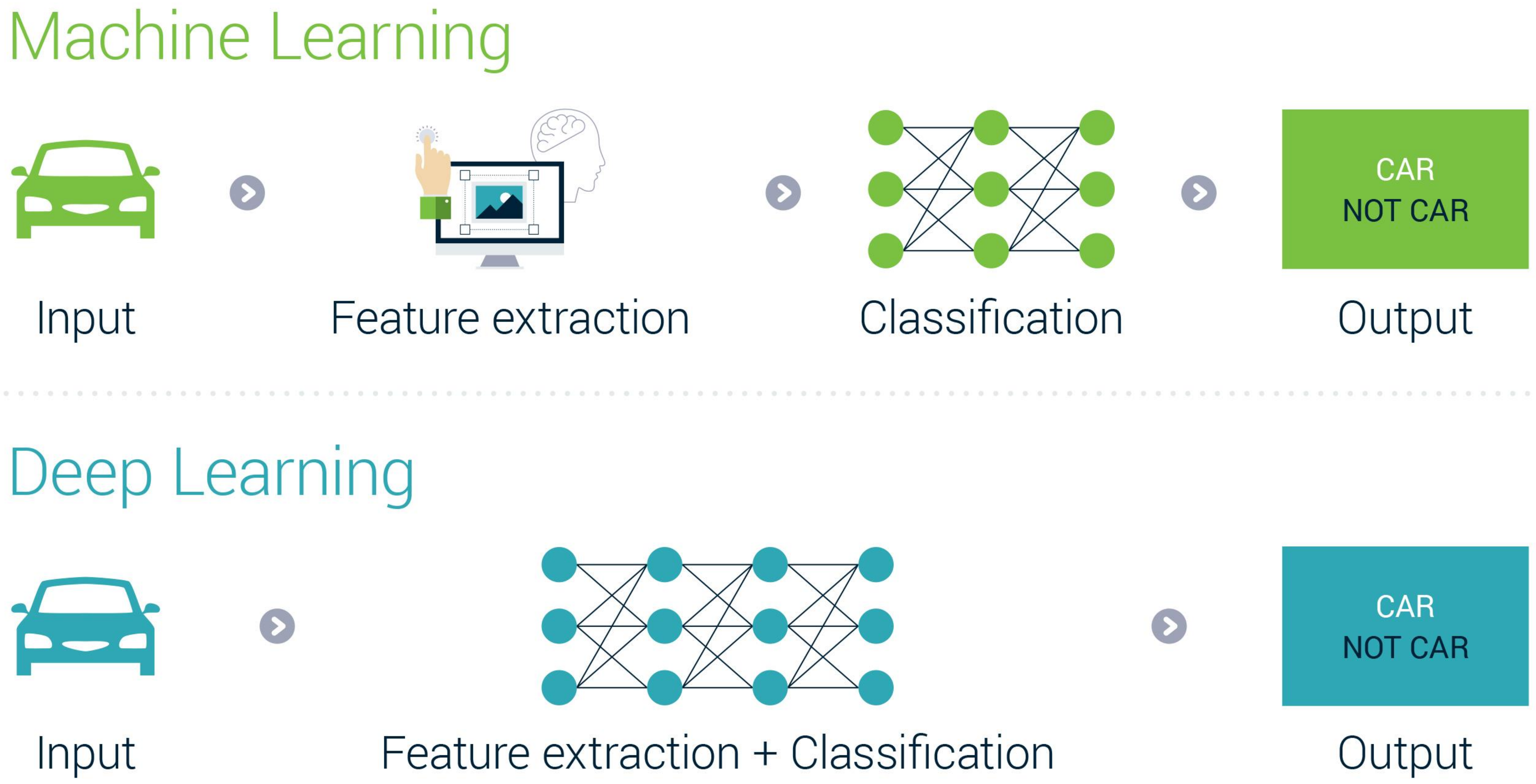


ACM Turing Award 2019 (Nobel Prize of Computing)
Yann LeCun, Geoffrey Hinton, and Yoshua Bengio

Deep Learning DL



Deep Learning DL



Deep Learning DL

- **In ML**, **features are engineered** and then **extracted** from **raw data** (e.g., text, image, sound, etc.) in a separate process to model learning for classification.

- The **features** are then passed through data to train a model that will be capable of making predictions.

- **Engineering of features** is, however, a **tedious process** for several reasons: Takes a lot of time, Requires expert knowledge.

- Extracted features often lack a **structural representation** reflecting **abstraction** levels in the problem at hand.

Deep Learning DL

- DL aims at **learning automatically representations** from large sets of labeled data:

  The machine is powered with raw data.

  Automatic discovery of representations.

- Representations at different levels of abstraction are learned by **simple non-linear transformations** (**non-linear activation functions**).
- **The composition of several transformations** can approximate very **complex functions**.

Deep Learning DL

Example:

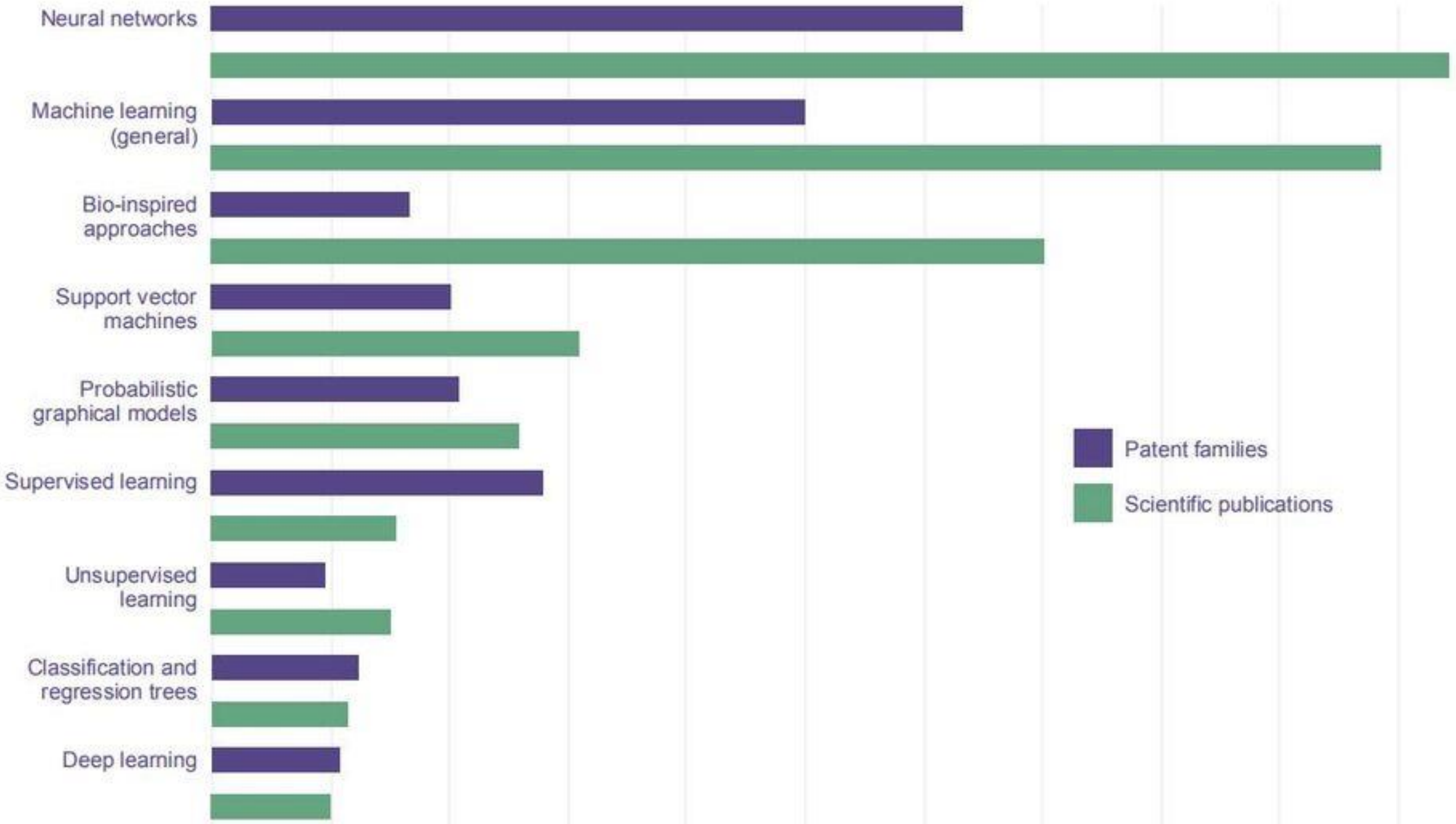
In text processing:

- Sentences are made of words
- Words are made of letters
- Letters are made of edges.

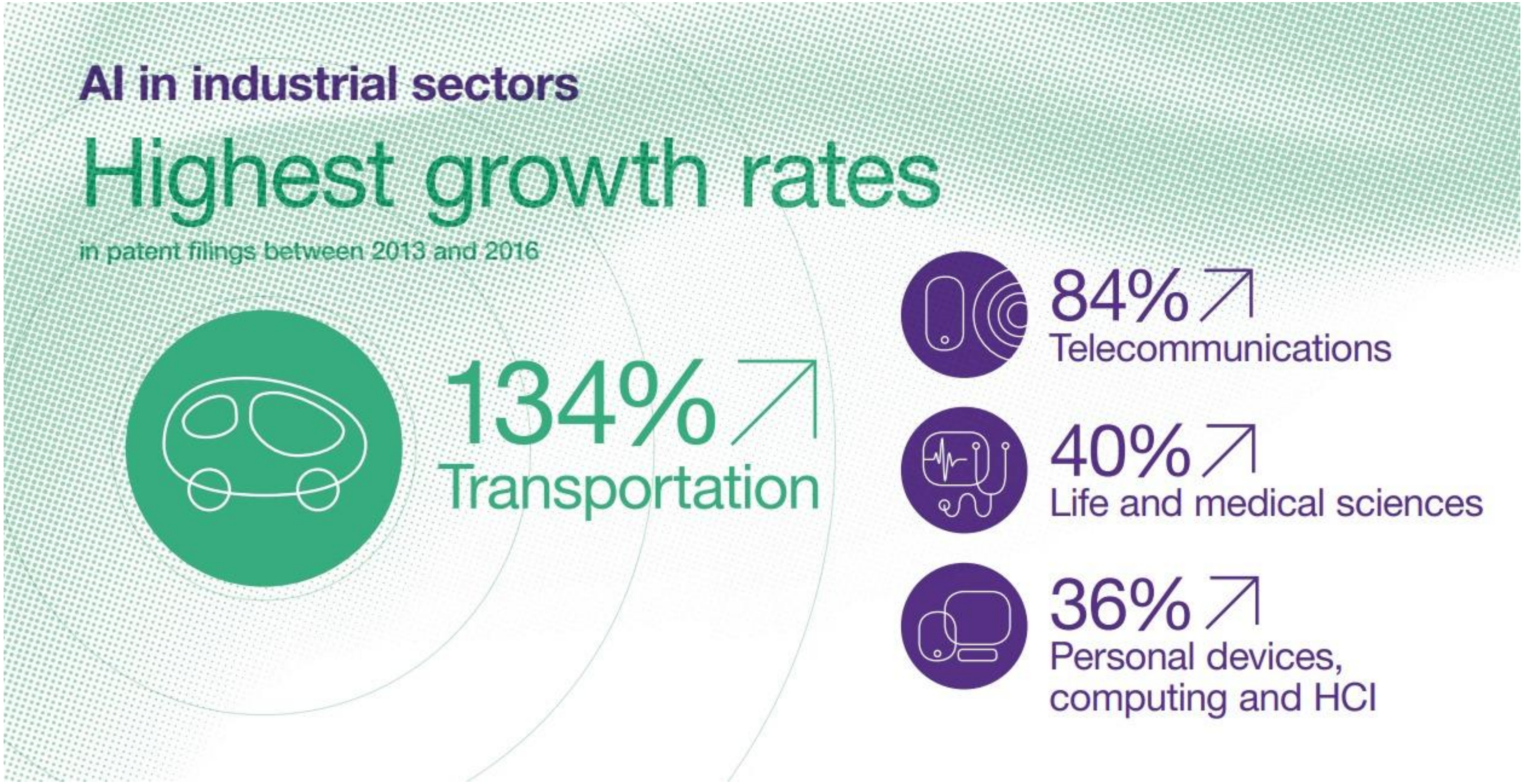
In image processing:

- Objects are made of local parts (e.g., head, torso, etc.)
- Local parts are made of lines and corners.
- Corners and lines are made of edges.

Deep Learning DL - wipo



Deep Learning DL - wipo



Deep Learning DL

Several DL models have been proposed :

- Convolutional neural networks (CNNs).
- Autoencoders (Aes).
- Recurrent neural networks (RNNs).
- Generative adversarial networks (GANs).
- Faster RCNN and Mask RCNN.

Thank you for your attention

Hichem Felouat …