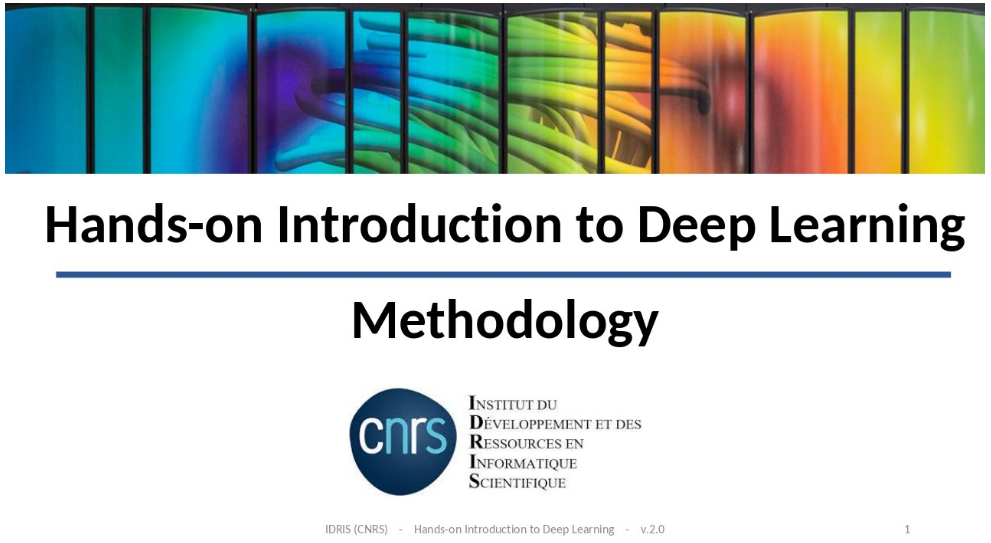


Hands-on Introduction to Deep Learning

Methodology

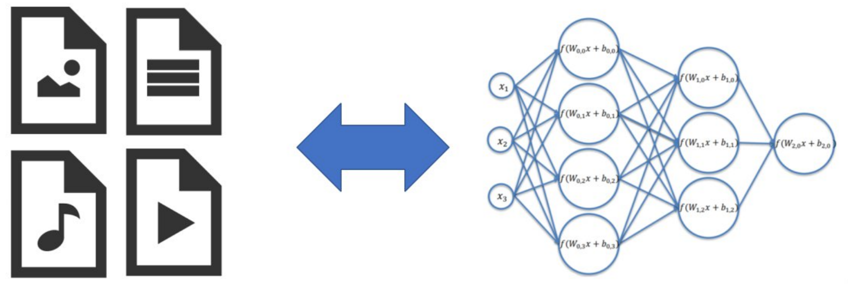
INSTITUT DU DÉVELOPPEMENT ET DES RESSOURCES EN INFORMATIQUE SCIENTIFIQUE

Objectives of this sequence :

- Become aware of the generic difficulties linked to training a model.
- Establish a methodology to follow during dataset and model development.

Duration: ½ day
Document annex : TP2 dossier
Aspects addressed :

- Model evaluation
- Data preprocessing
- Data management
- Optimizer
- Regularization
- Hyperparameters
- Transfer learning

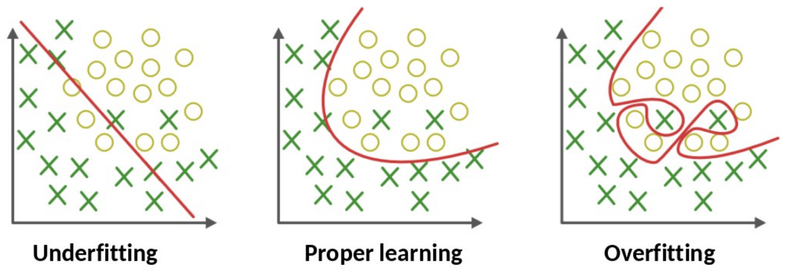


Working basis

To obtain an efficient Deep Learning model, it is necessary to have :

- Data
- An architecture to train

Each of these elements can be the cause of a poor performance for intrinsic or extrinsic reasons.



Bias variance trade-off

There are multiple challenges in training a model, the most well-known being the bias-variance dilemma.

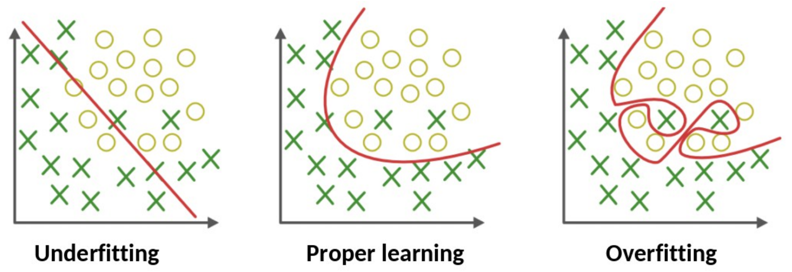
1. Bias-variance dilemma : Compromise appearing during the optimization of a model. The more the model represents complex relations, the more it risks representing the learning data noise. On the other hand, if it is not complex enough, it cannot correctly represent the relations of the observed phenomena.

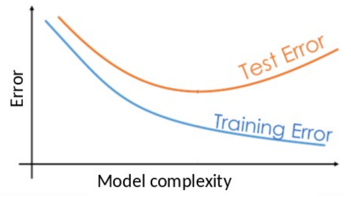
Simple model :

- Difficulty in learning the data noise
- Incapable of representing the data with precision
- Underfitting: sub-optimal performance, the determined boundaries could be better

Complex model :

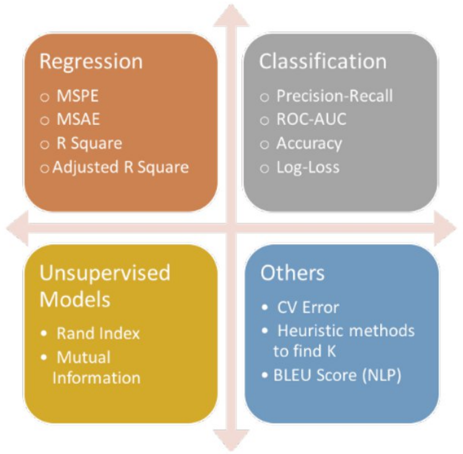
- Permits representing the relations between more complex data
- Risk of representing the data noise => overfitting





Bias variance trade-off - Detection

Several tools exist for characterizing the qualities and flaws of a model.
It is necessary to split the database into training and testing datasets. Observing the cost function variance or the evaluation function on the database training and testing sets is a good measure of the bias-variance dilemma.
Objective : Have a good test performance
Underfitting : Mediocre performance on both datasets
Overfitting : Performance is good in training, poor in test



Metrics

The choice of evaluation metrics is especially important when used on new data to avoid a disappointing performance.
The metrics should permit evaluating :

- The model for a given task.
- A solution without bias.
- Having distance properties allows this.

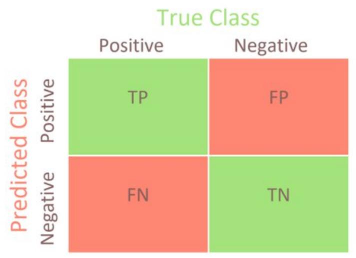
There is a great variety of generic metrics depending on the type of problem (regression, classification, ...) and data.
The metrics do not all have the same goal :

- Evaluate the model’s global performance.
- Show the biases.
- Evaluate the model's confidence in its predictions.
- ...

New system for illness detection – the accuracy is not a good metric for this case

Metrics – Bad Choice

Example of a poor metric choice: The case of an automatic diagnostic system.
The database which is at our disposal contains more data from healthy patients than from ill patients : 1% from ill patients.
Choosing Accuracy (a very common metric in ML) for this case would be an error.
Indeed, this is a poor system because it would diagnosis everyone as healthy with a 1% accuracy which is a very good score.
Therefore, we must turn to other metrics.



		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

$$precision = \frac{TP}{TP + FP}$$

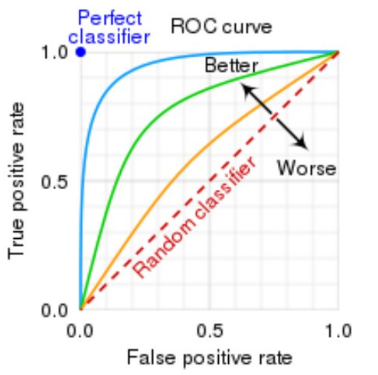
Above all positive prediction, how many are positive data

$$recall = \frac{TP}{TP + FN}$$

Above all positive data, how many have been predicted positive

Metrics – Precision/Recall

Precision and Recall give evaluations which correspond a little better with the objective of the system (for a classification task).
Precision is a good metric if we prefer avoiding False Positives even if it means missing some True Positives. Ex. : A doctor who, above all, does not want to diagnosis a patient as ill when the patient is not ill.
A coefficient based on precision and recall (the F score) provides a metric which unites both of them.



Variation of the acceptance threshold of a class to obtain the curve

$$TPR = \frac{TP}{TP + FN}$$

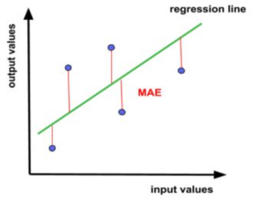
Above all positive data, how many have been predicted positive

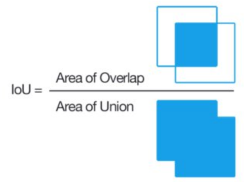
$$FPR = \frac{FP}{FP + TN}$$

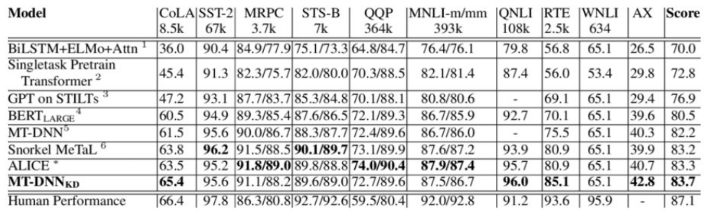
Above all negative data, how many have been predicted positive

Metrics – ROC curve

The ‘’ROC curve’’ is another alternative to Accuracy and Precision/Recall.
With this, we can evaluate the model visually.
In addition, we can visually determine which probability threshold is the most appropriate for our data.



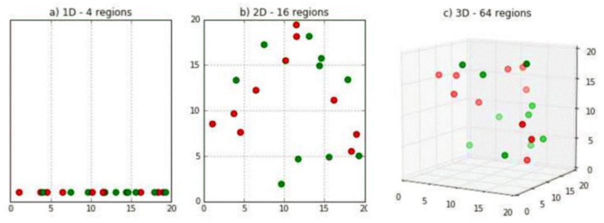


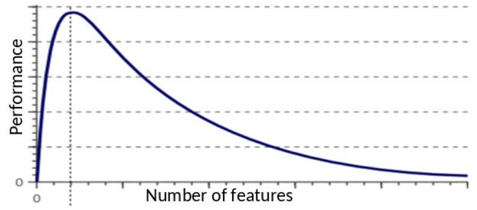


Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	**Score**
BiLSTM+ELMo+Attn [1]	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	79.8	56.8	65.1	26.5	70.0
Singletask Pretrain Transformer [2]	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	87.4	56.0	53.4	29.8	72.8
GPT on STILTs [3]	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	-	69.1	65.1	29.4	76.9
$\text{BERT}_{\text{LARGE}}$ [4]	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	92.7	70.1	65.1	39.6	80.5
MT-DNN[5]	61.5	95.6	90.0/86.7	88.3/87.7	72.4/89.6	86.7/86.0	-	75.5	65.1	40.3	82.2
Snorkel MeTaL [6]	63.8	**96.2**	91.5/88.5	**90.1/89.7**	73.1/89.9	87.6/87.2	93.9	80.9	65.1	39.9	83.2
ALICE [*]	63.5	95.2	**91.8/89.0**	89.8/88.8	**74.0/90.4**	**87.9/87.4**	95.7	80.9	65.1	40.7	83.3
$\text{MT-DNN}_{\text{KD}}$	**65.4**	95.6	91.1/88.2	89.6/89.0	72.7/89.6	87.5/86.7	**96.0**	**85.1**	65.1	**42.8**	**83.7**
Human Performance	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0/92.8	91.2	93.6	95.9	-	87.1

Metrics – Other examples

There are a multitude of other metrics which are adapted to different cases.
RMSE or MAE for regression tasks.
IoU (Intersection over Union) is very suitable for object detection in computer vision. It compares the real and predicted zones of the
objects to be detected.
The GLUE benchmark is a test suite which we use on language models to evaluate their linguistic competences (grammar tests, sentiment detection, ...). Despite the number of tests realized, this evaluation metric is still considered to be poor.





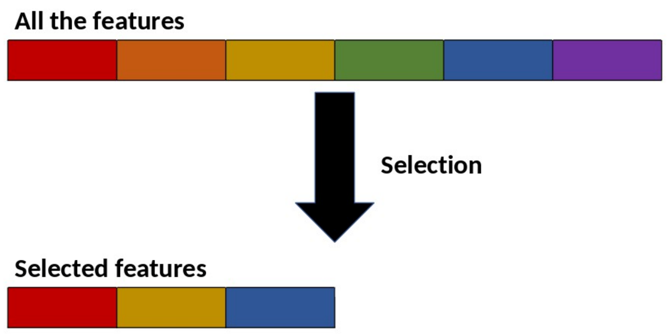
The curse of dimensionality

Curse of Dimensionality
Refers to the growth of the space when the number of features increases causing the data to become sparser in this space.
Problems :

- Loss of the meaning of distance because of the equidistance between the data which can occur for different reasons.
- Requires more data for a better representativeness of the space.
- Requires more neurons in the network (explosion of the memory occupation).

Objectives :

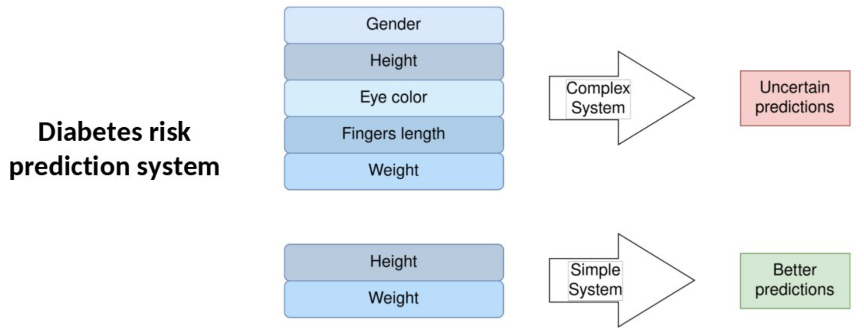
- Avoid the curse of dimensionality.
- Reduce the data and their memory occupation.
- Decrease the training time.



Features selection

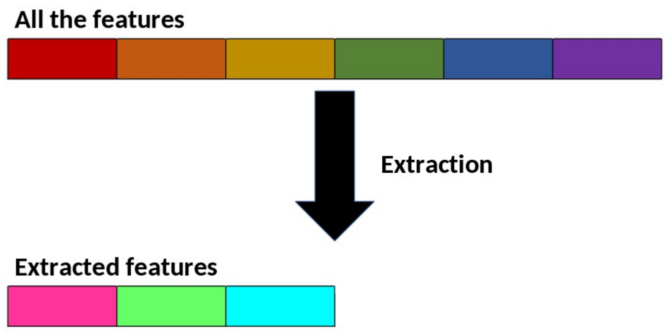
Diverse strategies for feature selection.
Strategy of supervised selection:

- Wrapping
  - Explores the space of possible features
  - Evaluates each subset by executing a model
  - Costly in terms of calculations
  - Risk of over-fitting to the model
- Filtering
  - A search approach similar to the wrapping method
  - Evaluation in relation to a simple and predefined criterion
  - Comparatively inexpensive in calculations
- Embedding
  - Selection by the model and determined during the training
  - Intermediary calculation cost



Features selection - Example

In this example of a system predicting the risk of diabetes, we can quickly see that we don't need all of the personal features to make the prediction.



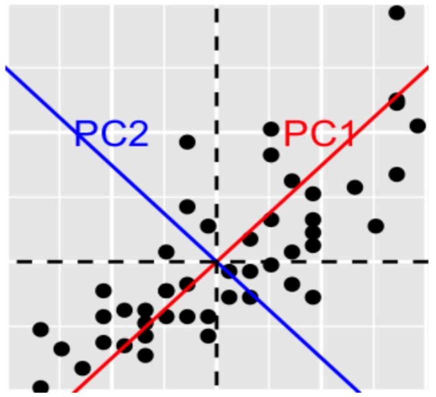
Features extraction

Data preprocessing for feature identification :

- Differentiating the data while reducing their complexity
- Improving the model performance for a given task
- Non-redundant
- Robust in face of various changes in data capturing methods
- Facilitating human interpretation

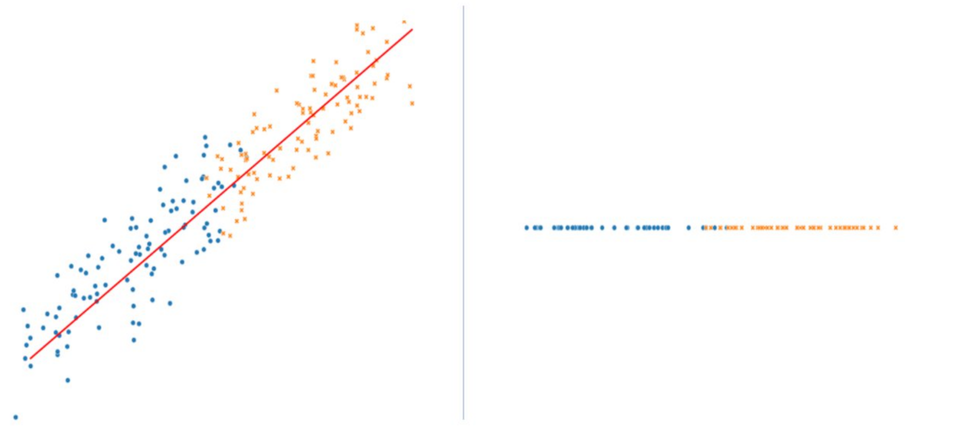
Methods :

- General : Preserving the maximum information according to a statistical criterion
    - Principal component analysis
    - Independent component analysis
    - ...
- Specialized : Using human understanding of the data to define specific transformations
- Histogram o   f gradients for images
    - Frequency decomposition for sound
    - Less and less popular, we let models independently determine the features



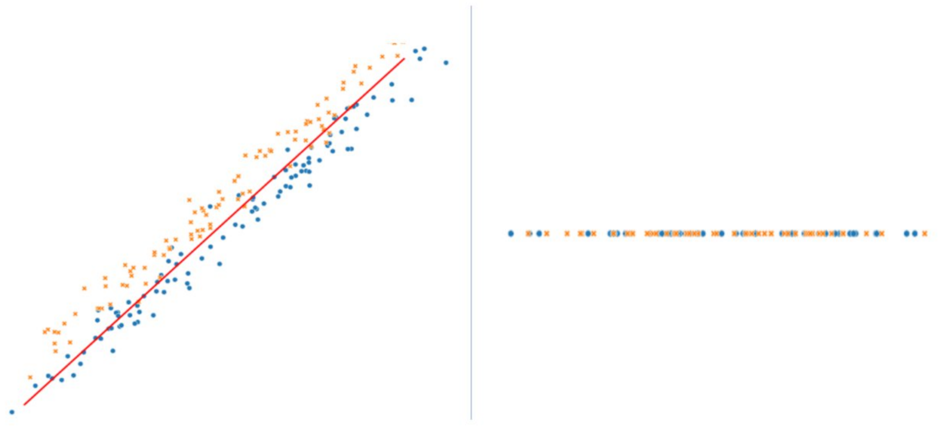
Feature extraction – PCA example

Principal component analysis (PCA) is a good example of data feature identification.
In fact, it enables reducing the dataset dimensions through projecting the data on the space which retains the most variance.



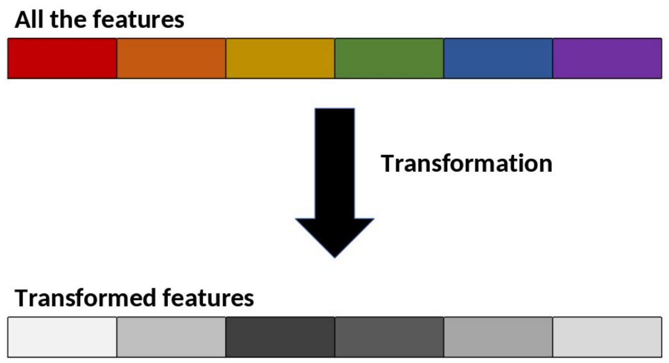
Feature extraction – PCA example

Here is an example where PCA works well. We see that the projection of data on the axis which « keeps » the most variance enables dimension reduction while leaving enough data relationships so that the model can learn to classify them.



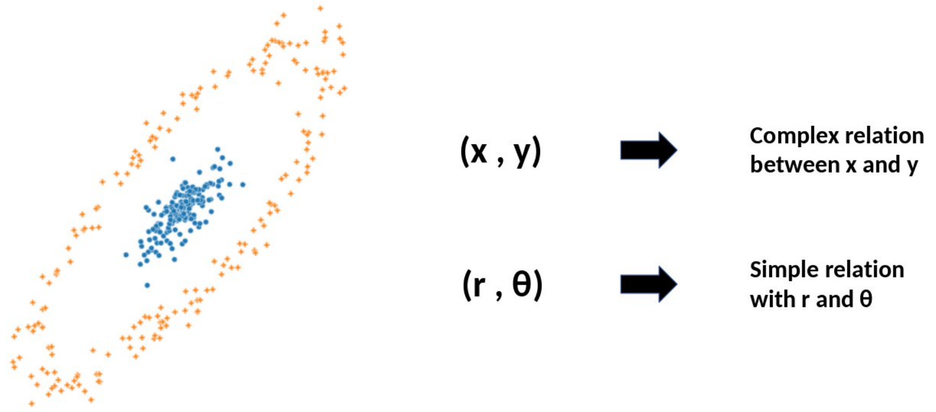
Feature extraction – PCA example

Here is an example where PCA does not work well. The dimensions were reduced but it is no longer possible to determine the relationships between data in order to classify them.



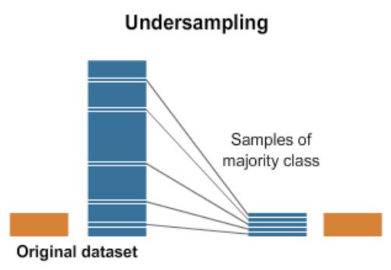
Feature transformation

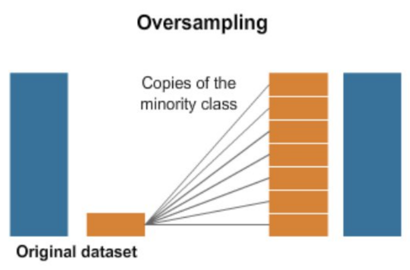
Feature transformation makes the data more ‘’digestible’’ for the model.
In fact, it can improve training stability and also reduce task complexity if the transformation is appropriate.


(x , y)
Complex relation between x and y
(r , θ)
Simple relation with r and θ

Feature transformation - Example

Transforming the dataset coordinates into polar coordinates enables training a less complex model.





Balance the classes

Avoid biases linked to :

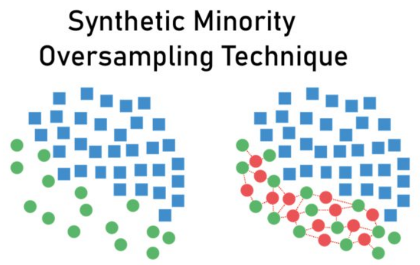
- Imbalances in each data instance
- Imbalances in the database

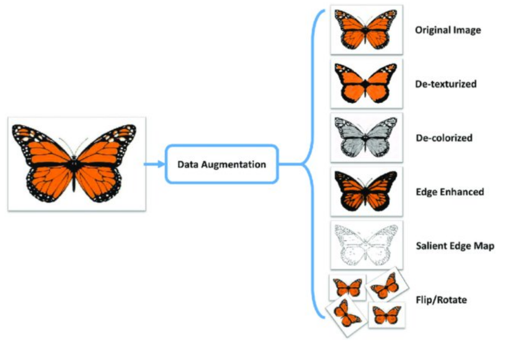
Consequences :

- Creates bias in the model
- Slows down training
- High memory occupation

Example of problem :
Binary classification where 95% of the data are positive. The model could simply learn to predict everything in a positive way.
To counter these effects, several types of actions are possible :

- Integrating weighting coefficients in the evaluation metrics
- Directly correcting the data imbalance by creating synthetic data
- Filtering out part of the data





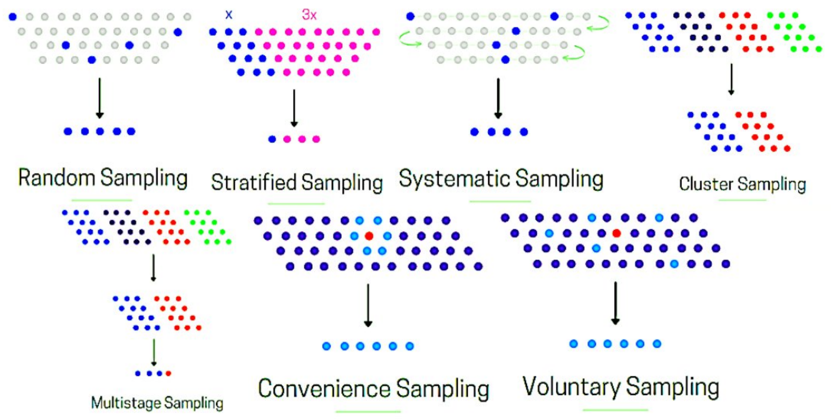
Data augmentation

Strategies :

- Duplicate the data or use them frequently during the training
- Generate interpolated data between the existing data
- Generate data by applying transformations to them

Finding the right data augmentation strategies is rather complex.
It requires knowledge of the data to know which transformation is realistic ; also of the model to know which transformations can be beneficial to it and which transformations would be useless. In fact, some feature identifiers or models are naturally robust when facing certain transformations.
Data augmentation is often an effective solution for decreasing model bias and variance.
Disadvantages :

- Adds a preprocessing time which can be substantial
- Increases the training duration

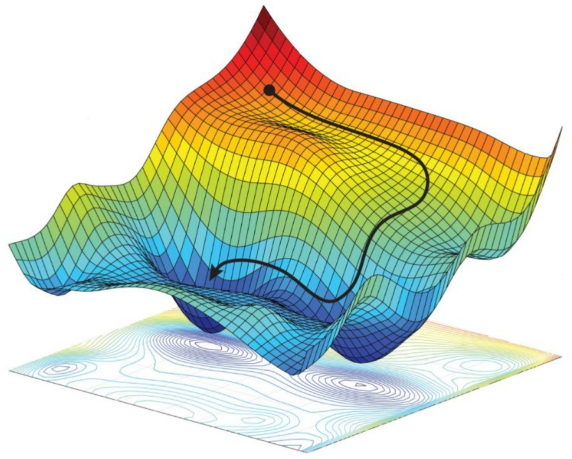


Under sampling

It is possible to have too many data or too many data in one class.
This can impair model performance or could unnecessarily increase training time.
Sampling strategies can :

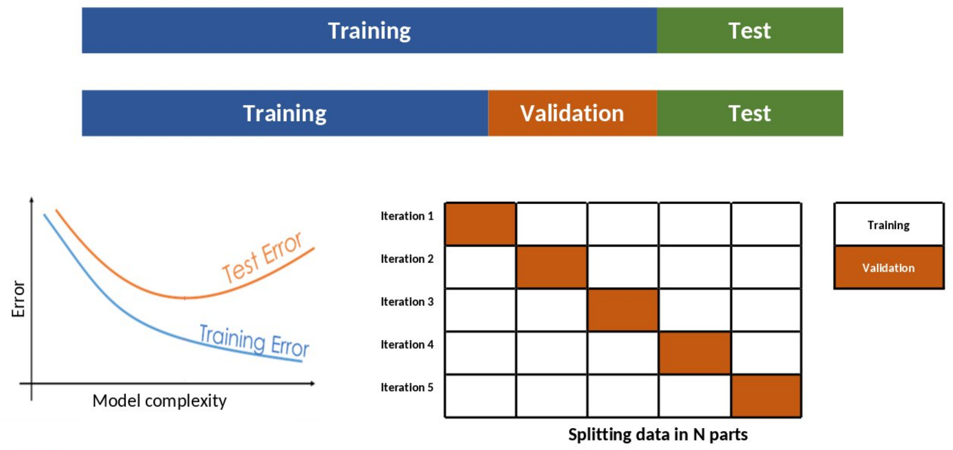
- Be based on random samples.
- Seek to preserve the data which is most difficult to identify.
- Delete the data which is too ambiguous, or the outliers.
- Filter out the data which is too similar.

All of the cited augmentation and reduction techniques can be combined in order to diversify the data and improve the dataset while controlling its size.



Reminder : training a neural network

What is achieved during the training of a neural network is the minimization of a cost function in the space of the model parameters. This illustration is an example of such a function for a model with only two parameters.
Since the cost function reflects the difference between the predicted data and the labels, the smaller the result of this function, the better the prediction. We therefore look for the global minimum of this function.
In practice, we cannot calculate the cost function for all points in space but only locally in the state where the network is. We can therefore never be sure if the minimum we are in is a global or a local minimum. We hence consider that the network is trained if it’s in a good minimum of the cost function.
We characterize the quality of the network with other metrics (accuracy, MAE …).



Data splitting

The first thing to respect is separating the database into at least two parts:

- The train set : for training the model
- The test set : for evaluating the quality of the model

This separation allows determining if there is overfitting and thus if the model generalizes correctly. It is this separation of the data set that is the origin of the two curves in the figure. Be careful, depending on how the model is built, there could be a data leak from the test set to the training set.

To find the correct hyper-parameters, it is useful to separate the train set into a smaller train set and a validation set. This way hyper-parameters can be adapted according to the score of the model on the validation test. The final score is the one on the test set.

When the quantity of data is reduced and to avoid again overfitting the model for certain hyper-parameters, more complex strategies can be implemented for the selection of the validation base, such as the K-fold cross validation illustrated on the right.

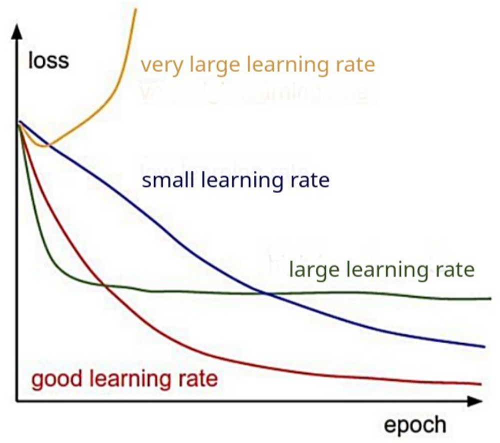
Hyper-parameters	Methods
• Learning rate	• Manual research
• Regularization	• Grid search
• Optimizer	• Random research
• Model architecture	• Gradient
• Batch size	• Evolutionary algorithms
• …	• …

Find the hyper-parameters

We train a model so that its weights (parameters) are learned automatically. However, hyper-parameters must be defined before the training and they have a great impact on it. Two models that are identical and and trained on the same data can give very different results depending on the parameters defined for the training.

Choosing the hyper-parameters is very complex. Today, the most popular method is to tweak them manually because of the compute cost of automated methods. We tend to use them at the end of a project in order to obtain the best possible model performance.

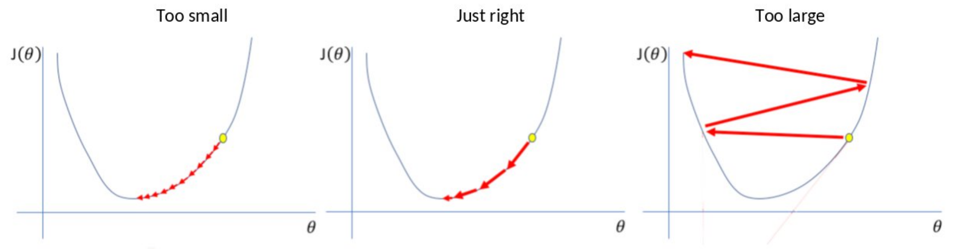
In the next slides, the learning rate, regularization and optimizer are presented.

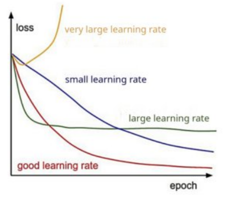


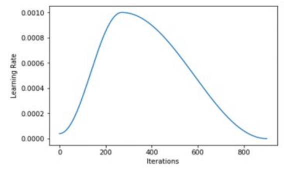
Learning rate

The learning rate, in particular, strongly influences the convergence of the model. The learning rate defines the extent to which the calculated gradient will change the weights. With a large learning rate, each weights update will greatly change the model.

We can see here the effects of different learning rates on network training.







Learning rate

Learning rates :

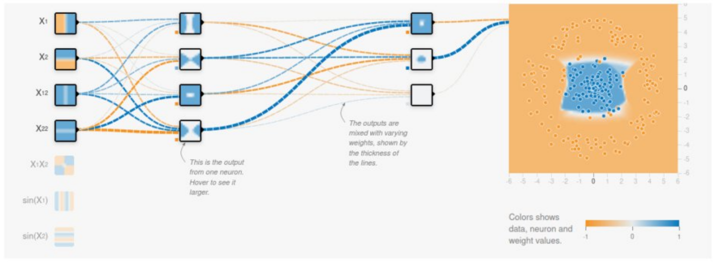
- Small
  - o Makes the model converge slowly towards a solution.
  - o Can cause the model to converge towards a mediocre solution.
- Large
  - o Can prevent the model from converging towards a solution.

It is possible to use a variable learning rate with a « warm-up » phase during which it starts rather low and increases quickly to a maximum learning rate. This step assures starting in the right direction before drastically changing the network weights. The learning rate then decreases at each epoch. This procedure enables converging rapidly towards a correct solution and then, to follow it with optimizing this solution.

$$\Theta_{t+1} = \Theta_t - \eta \nabla_\Theta [\mathscr{L}(\hat{y}_i, y_i)]$$

Updated weights = Weights before update − Learning rate * Gradient [Cost function (Prediction , Label)]

Weight update equation


X₁
X₂
X₁²
X₂²
X₁X₂
sin(X₁)
sin(X₂)
This is the output from one neuron. Hover to see it larger.
The outputs are mixed with varying weights, shown by the thickness of the lines.
Colors shows data, neuron and weight values.

Regularization : motivation

Here is the weights updating equation without regularization. Weights are updated at each step by removing a part of their gradients on the cost function. For example , if the model gives a bad prediction , the weights that were the most activated are reduced.
Sometimes, during the training , the model identifies certain data features that are really useful for giving good predictions on the dataset. It will then give great importance to these features , even to relying only on these to make its prediction.
This behavior could be something desired (if the objective is to find the most important features) ,or not (if we want to avoid overfitting, for example).
When weights become very large, it is problematic for another reason : exploding gradients. In fact, the gradient regarding the cost function is proportional to the weight amplitude. We encounter the same problems as if the learning rate was set too large(the network changes a lot with each update and it diverges).

The purpose of regulation is, therefore, to limit the divergence of certain weights of the network.

$$\Theta_{t+1} = \Theta_t - \eta \nabla_\Theta [\mathscr{L}(\hat{y}_i, y_i) + \lambda R(\Theta_t)]$$

Updated weights = Weights before update − Learning rate * Gradient [Cost function (Prediction , Label) + Regularization rate * Regularization function (Weights before update)]

Weight update equation

- L1 Regularization
- L2 Regularization
- Max norm Regularization
- Regularization with the cost function
- Dropout

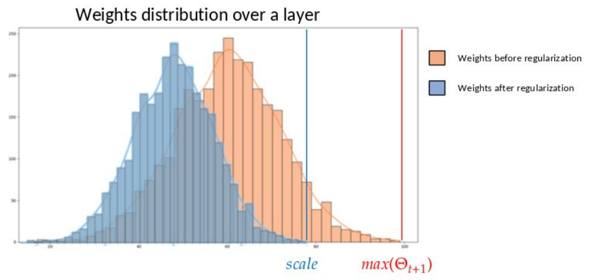
L1 : LASSO	L2 : Ridge
$\lvert\Theta\rvert$	Θ^2

Regularization: L1 and L2

This is the function which updates weights with regularization.
We see here that regularization is applied at the same time as weights updating through a regularization function added to the cost function. The strength of this regularization is modulated by the regularization rate.
The regularization methods mentioned are presented below.
For the L1 and L2 regularization (seen in Hands-on Practice 1) :

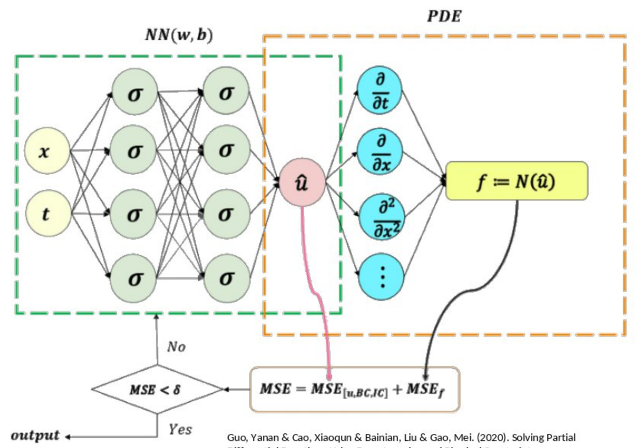
- L1-LASSO (Absolute value of the weights) : Focuses the attention of the neurons on certain features.
- L2 (Ridge) (weight decay) : Forces using all the information.
- ElasticNet : Combines LASSO and Ridge.

$$\boxed{\Theta_{t+1} = \frac{\Theta_{t+1}}{max(\Theta_{t+1})} * scale}$$



Regularization: max norm

Another type of regularization is called max-norm regularization.
The idea is to prevent certain weights from becoming too large (to avoid overfitting and possible exploding gradients).
This implementation is really basic since the weights are automatically returned below a threshold value for each neuron layer. The weights are updated with the displayed formula.
This regularization is carried out immediately after the classic weights updating with a gradient.


PDE
$NN(w, b)$
x
t
σ
$\hat{u}$
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\vdots$
$f := N(\hat{u})$
No
$MSE < \delta$
$MSE = MSE_{[u,BC,IC]} + MSE_f$
Yes
output

Guo, Yanan & Cao, Xiaoqun & Bainian, Liu & Gao, Mei. (2020). Solving Partial Differential Equations Using Deep Learning and Physical Constraints

Regularization: with the cost function (PINN example)

The training can also be constrained or monitored with the cost function.
The PINNs (Physics-Informed Neural Networks) are a good example. These networks aim to resolve partial differential equations.
In this case, the computation of the cost function is done in two parts : The first part is the classic function which computes the difference between the prediction and the label. The second part reflects the respect of the constraints of the solution and/or of their partial derivatives. In this way, the calculated gradient takes into account the constraints imposed from the PDE (initial conditions, boundary conditions, ...).
Be careful, if the cost function depends on several terms, this can make the training of the network more complex.

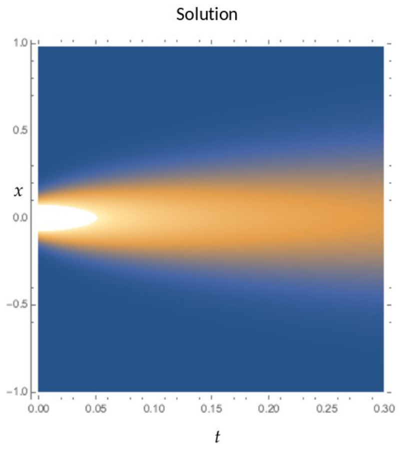
Exercise :
The heat diffusion over time along one dimension is governed by the following equation :

$$\frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial^2 x} = 0$$

with :
- u : the heat distribution along x,
- k : the thermal diffusivity of the material

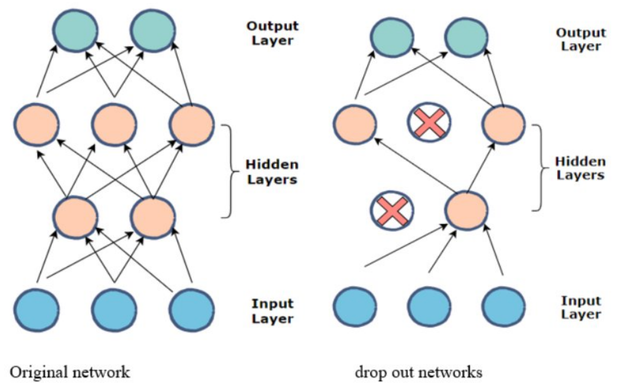
We set theses constraints :
1. $u(x,0) = exp\left(-\left(\frac{x}{0.1}\right)^2\right)$ pour $x \in [-1,1]$
2. $u(-1,t) = u(1,t) \quad \forall\, t$

Result : The final loss is below 10^{-6}



PINN : 1D heat equation

Here is an example of a classic PDE solution : heat diffusion on a 1D axis.
The time is from left to right, and the x axis from top to bottom. The colors are proportional to the temperature, from blue (cold) to white (hot).
The limitations of this PDE are the initial conditions (heat distribution for t=0) and the boundary conditions (equality in x=1 and x=-1).
We can see that the network has found an optimal solution.
This kind of network is applied to cases where the PDE is ill-posed and for those which are very resource-consuming for their numerical resolutions.

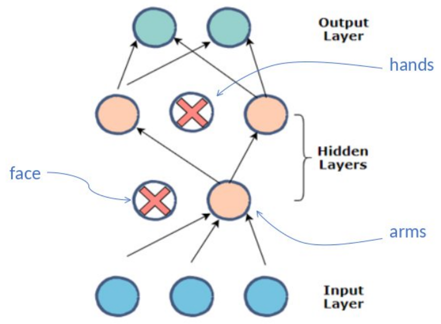


Regularization: Dropout

Dropout is a technique which consists of randomly deactivating certain neurons during the training. At each epoch, we randomly change the neurons which are deactivated.
By inference, all the neurons are reactivated.
The goal is to force the learning of different features. This can permit stimulating certain neurons more strongly by forcing them to contribute for a larger part of the solution. In this way, we can reduce overfitting by forcing diversification of the model.
To obtain a similar result without the dropout, it is possible to randomly hide part of the entry data (this is less effective than the dropout) .

Human = face ?

(a) Husky classified as wolf (b) Explanation

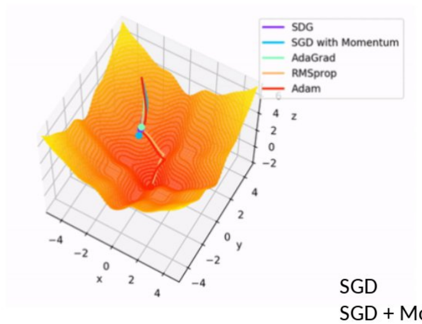


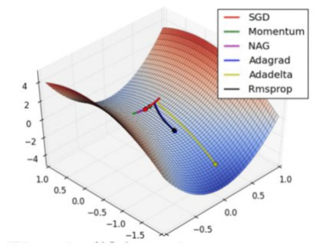
drop out networks

Source :
pytorch.org
data-flair.training
Right for the Right Reason: Making Image Classification Robust

Regularization: Dropout

Let’s get how dropout works through an example of image classification.
For the task of automatic person recognition, a face is a particular feature : It is easy to detect and assures us of the presence of a person. However, relying only on this feature would be problematic for detecting people who are not facing the camera or who are only partially included in the image.
However, the network itself builds the features it needs for identifying a person. As long as the person is identified, the network is free to rely on the features it wants (we do not have explicit control over this process). For instance, in a classification task of dogs and wolfs, researchers discovered that to predict a wolf, the network only detects if there is snow in the picture. A dog photographed in the snow is also classified as a wolf.
Dropout consists of randomly deactivating some neurons during the training. In doing this, the parts of the network in charge of detecting faces or hands are sometimes deactivated. The network then needs to rely on other features such as arms, tee-shirts or other relevant human identifiers .

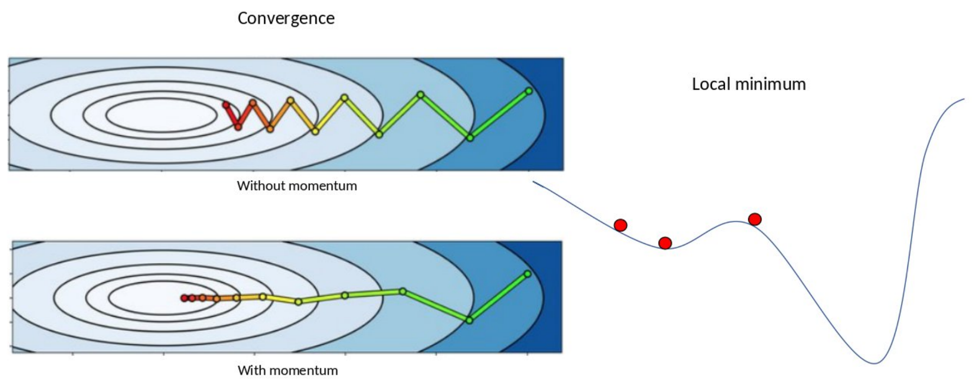




SGD
SGD + Momentum
NAG
AdaGrad
AdaDelta
RMSprop
Adam

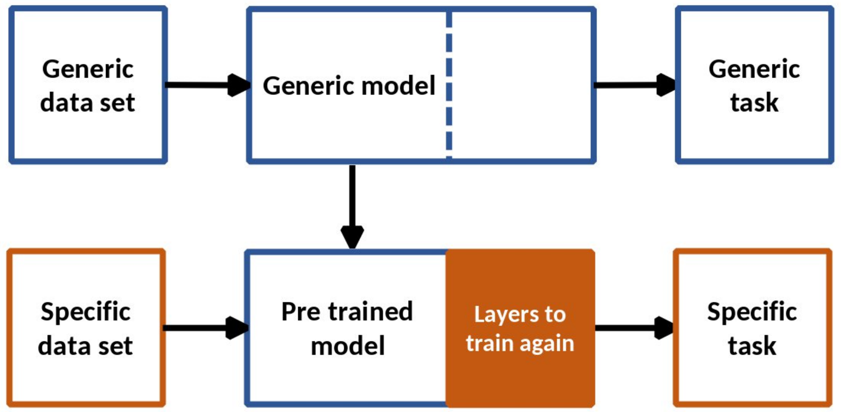
Optimizers

It is the optimizer which updates the model parameters. It integrates the regularizations during the weights updating.
Some optimizers can also update the hyper-parameters and compensate for negative effects of the learning rate.
Problematic cases for cost function minimization are when saddle points are encountered (where the cost function is minimum along only one dimension). At the saddle point, the gradient becomes zero although we are not in a local minimum. Certain optimizers treat this problem more or less effectively (see illustration).



Optimizers: Momentum

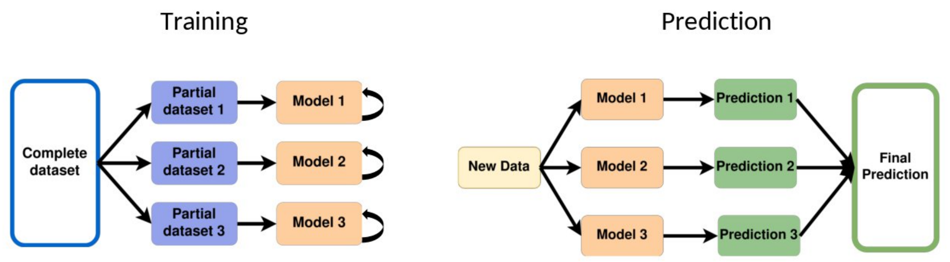
A popular solution is adding an inertia to accelerate the model convergence when it’s stable but slow. This is momentum.
Using momentum permits a good convergence of the model when the gradient is strong in one direction and weak in another (see illustration).
In the bottom figure, the zig-zags have disappeared and the result obtained for the same number of iterations is better.
Another advantage of this technique is that the network won’t be stuck in a state where there is a weak local minimum. In the right-hand picture, the gradient of the second point is null. Without momentum the network cannot change between iterations and will stay in the same state without being able to learn more.



Transfer Learning

Transfer Learning is a technique which consists of using a model trained for a given task and training it to do a similar task.
The goal is to transfer a part of the learned features , hoping that these will be meaningful for the new task.
This mean s that the training of such a model is shorter and requires fewer data.
This is a popular technique in machine learning because acquiring new data can be costly.
This method is also very efficient with neural networks due to the way they work.
The first layers learn generic features while the last layers learn specialized features. In the field of image processing, the first layers tend to detect simple geometric shapes and the last layers tend to detect more complex shapes (generally depends on the task).
Two approaches are available for transfer learning :

- Freeze the weights of the first layers and train the last ones.
- Train the whole network but share the weights which were already trained.



Ensemble learning (ex : bagging)

Finally, as networks are estimators, it is possible to use ensemble learning methods to optimize the prediction more powerfully or to make it more robust.

Bagging is a technique which consists of using multiple models for the same task by combining all their predictions.

The goal is to compensate for the variance that only one estimator can have.

For this, we divide the database into different independent subsets in order to train the different models. We can then apply a strategy for combining the predictions (majority prediction, prediction averaging).

This is a popular method for simple estimators such as decision trees to create a random forest.