Deep Learning - Part 1

Aymeric DIEULEVEUT

DS4M - April 2025

1. Goals

2. The first Neural Network: the Perceptron
   - Logistic regression
   - Discovering Fully connected NN through the NNPG.
   - Formalization, advanced topics
   - Further discussion

Outline

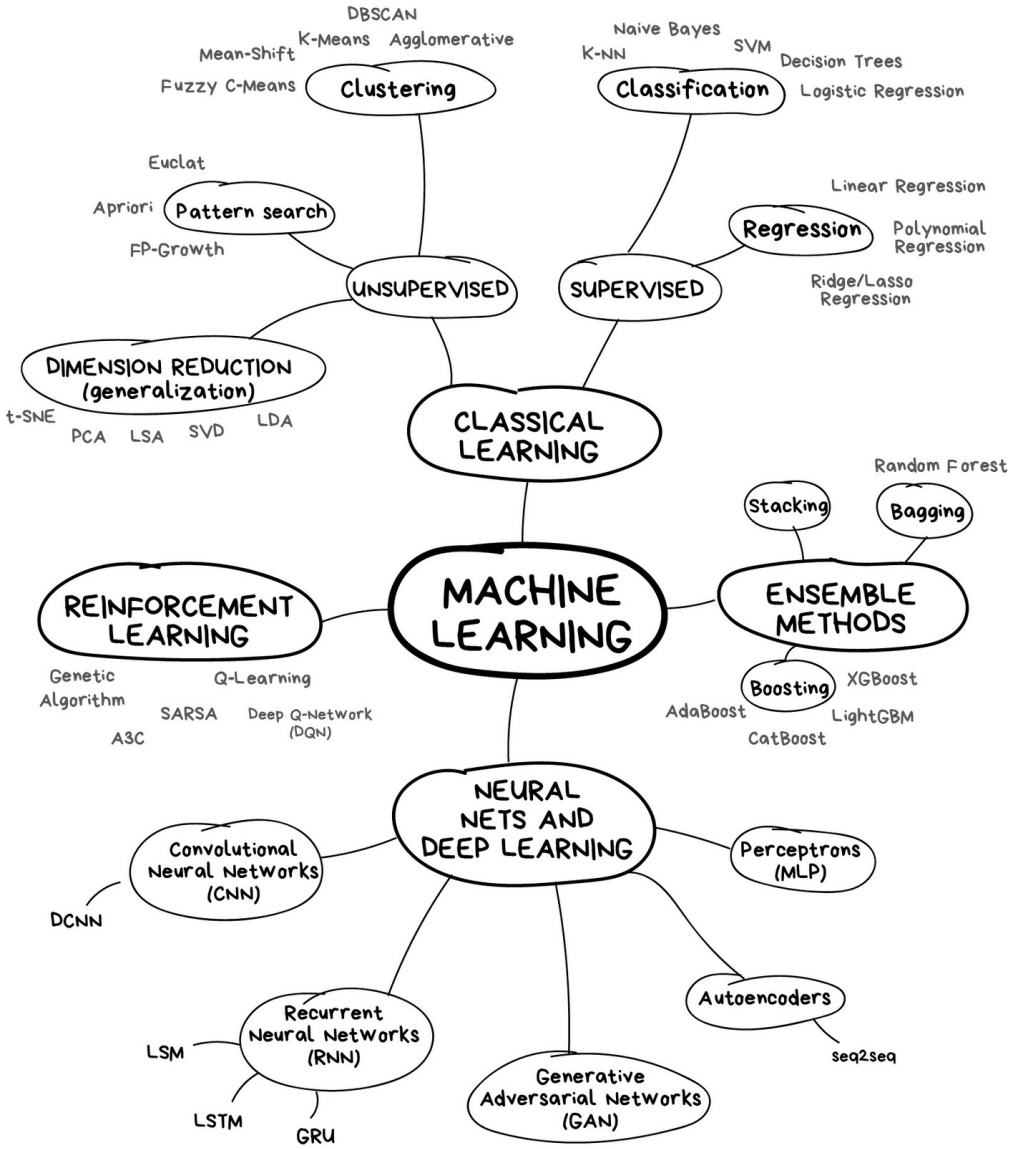
Deep Learning in one slide + Goals of the afternoon

- **How does it work:**
  - Automatically learn representations of observations
  - Learn highly non-linear models.
- **What does it require:**
  - Large datasets with structure
  - Computational power
- **Why now:**
  - Combination of the 2 points above
  - investment !
- **Some Applications**
  - Image classification; object / face recognition
  - Self driving cars
  - Automatic Translation, Information extraction
  - Caption Generation
  - Ads, recommendation systems, etc.

Goals: Understanding core concepts of Deep Learning.

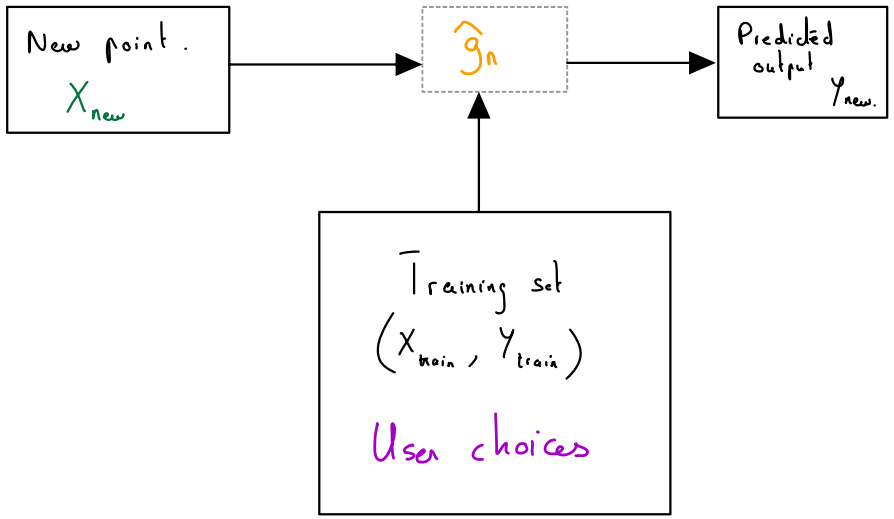
- When and how it works on paper (data, models, architecture)
- How to implement a simple neural network with Python.
- Overview of some of the main applications and challenges.

Machine Learning



Summary of previous lectures on Machine Learning

1 General framework: loss, risk, generalization risk vs training risk



Loss function, Generalization Risk and data-dependent predictors

Loss Function

- Loss function: $\ell(y, g(x))$ quantifies the quality of the prediction $g(x)$ of y, e.g.:
  - 0-1 Loss (classification): $\ell(y, g(x)) = 1_{y \neq g(x)}, \quad y \in \mathcal{Y} = \{-1, 1\}$
  - Quadratic Loss (regression): $\ell(y, g(x)) = \|y - g(x)\|^2$, $y \in \mathbb{R}^d$

Risk of a Decision Rule = average loss

- Risk: $R(g) = \mathbb{E}[\ell(Y, g(x))] = \int \ell(y, g(x)) d\mathbb{P}(x, y)$, e.g.:
  - 0-1 Risk (classification): $R(g) = \mathbb{P}(Y \neq g(x))$
  - Quadratic Risk (regression): $R(g) = \mathbb{E}[|Y - g(x)|^2]$

$\mapsto$ R is also referred to as *Generalization* risk, or *True* risk.

⚠ The distribution $\mathbb{P}$ is unknown.

Data-dependent predictors

- Training set: $\mathcal{D}_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ $(i.i.d. \sim \mathbb{P})$
- Goal: build a data dependent predictor / classifier $\widehat{g}_n$ based on the training data
- That has a minimal generalization risk $R(\widehat{g}_n)$.

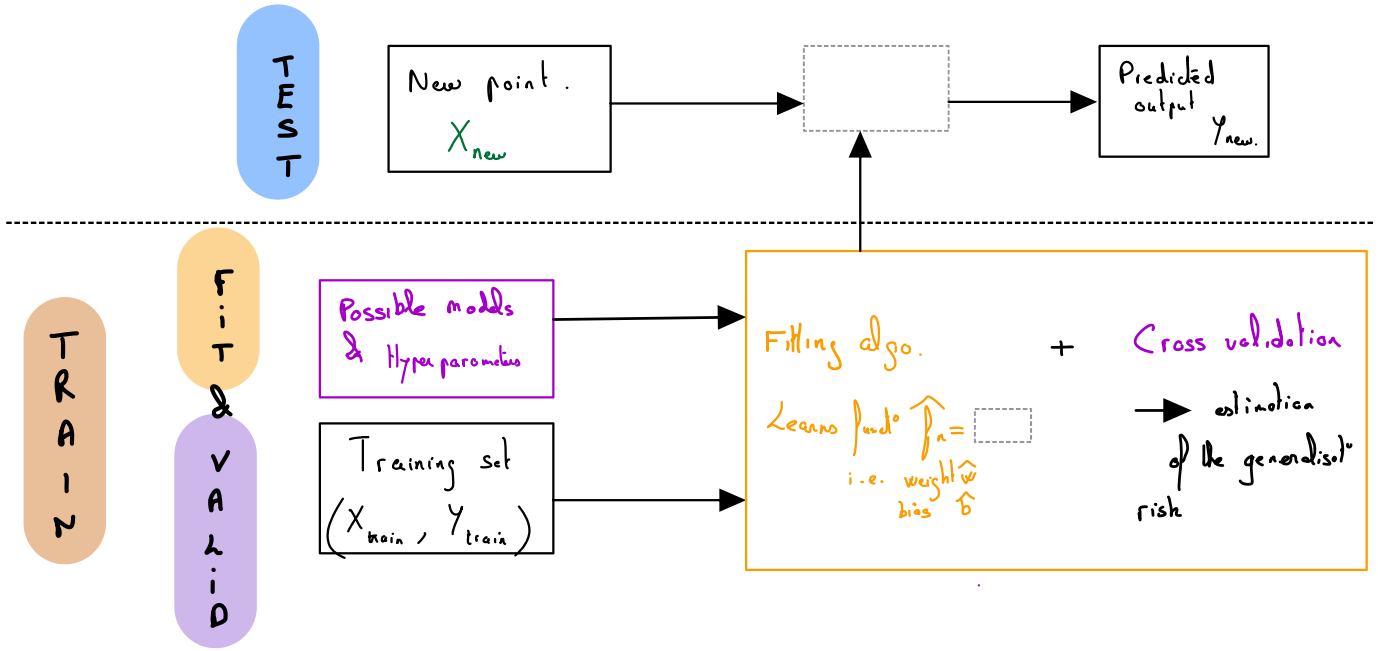
$R(\widehat{g}_n)$ = average loss on a "new point" $(X, Y) \sim \mathbb{P}$, independent from D_n

$\leftrightarrow$ Our goal is to predict well on inputs/outputs pairs that **we have not seen in the dataset**, i.e. *generalize* well.

THM 1.

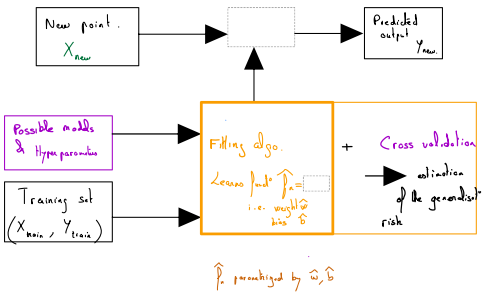
Summary of previous lectures on Machine Learning

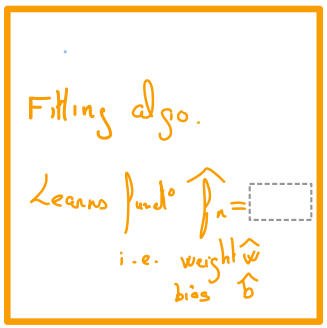
1. General framework: loss, risk, generalization risk vs training risk
2. SML Pipeline
   - Train (Fit + validation) / Test / Deployment
   - Evaluation of validation scores through cross validation to pick the best method/hyperparams

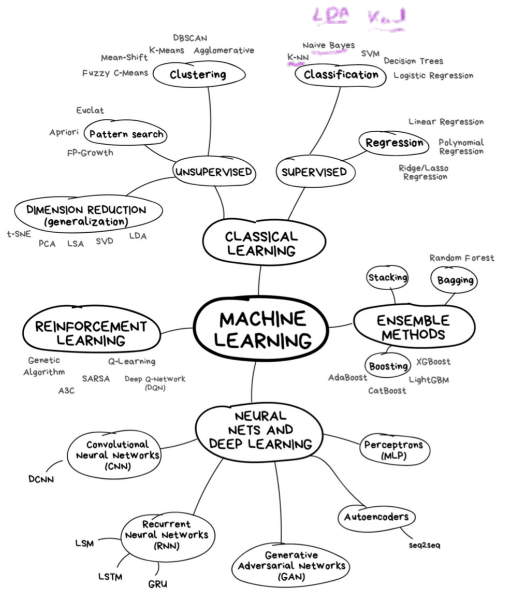


Summary of previous lectures on Machine Learning

1. General framework: loss, risk, generalization risk vs training risk
2. SML Pipeline
   - Train (Fit + validation) / Test / Deployment
   - Evaluation of validation scores through cross validation to pick the best method/hyperparams
3. Methods: Linear, Logistic regression, Decision Trees, Random Forests, Boosting







Summary of previous lectures on Machine Learning

1. General framework: loss, risk, generalization risk vs training risk
2. SML Pipeline
   - Train (Fit + validation) / Test / Deployment
   - Evaluation of validation scores through cross validation to pick the best method/hyperparams
3. Methods: Linear, Logistic regression, Decision Trees, Random Forests, Boosting
4. Opening the black box: for each method
   - How to predict ($\leftrightarrow$ interpretability)
   - How fit is made ($\leftrightarrow$ speed, complexity) [Computer learning parameters]
   - Important Hyper-parameters [User chosen]
   - Drawbacks and Strengths
   - Underfitting, overfitting risks ($\leftrightarrow$ Performance)
     $\rightarrow$ regularization techniques (avoiding overfitting by modifying the method)

$f_n =$ ☐ :	Lin. Reg	Polynomial Linear reg	Logistic Reg	Dec Trees	RF / Boosting
How to predict					
How to fit					
Key hyperparameters ⊕					
⊖					
Overfitting / Underfitting ↳ regularization technique					
Approach					

Outline

Outline

From logistic regression to Multi Layer Perceptron 1

Supervised ML problem:

1. Data $D_n = (X_i, Y_i)_{i=1,\dots n}, X_i \in \mathbb{R}^d, \quad Y_i \in \{-1, 1\}$
2. Goal: Predict a the label for a new point.

Empirical risk minimization (Discriminative approach)

- No statistical model!
- Define a loss function ℓ
- Choose a function class $\mathscr{F}$

- Goal:

$$\min_{f \in \mathscr{F}} \mathbb{E}[\ell(f(X), Y)]$$

- ERM

$$\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$$

Linear models

Class of *soft* linear separators:

$$\mathscr{F}_{lin} = \{g_w : X \mapsto w^\top X \quad , w \in \mathbb{R}^d\}$$

Binary prediction from a soft predictor: $X \mapsto \text{sign}(w^\top X)$

Linear models

Class of *soft* linear separators:

$$\mathscr{F}_{lin} = \{g_w : X \mapsto w^\top X \quad , w \in \mathbb{R}^d\}$$

Binary prediction from a soft predictor: $X \mapsto \text{sign}(w^\top X)$

01 loss:

$$\frac{1}{n}\sum_{i=1}^{n} 1_{Y_i \neq \text{sign}(w^T X_i)} = \frac{1}{n}\sum_{i=1}^{n} 1_{-Y_i w^T X_i > 0}$$

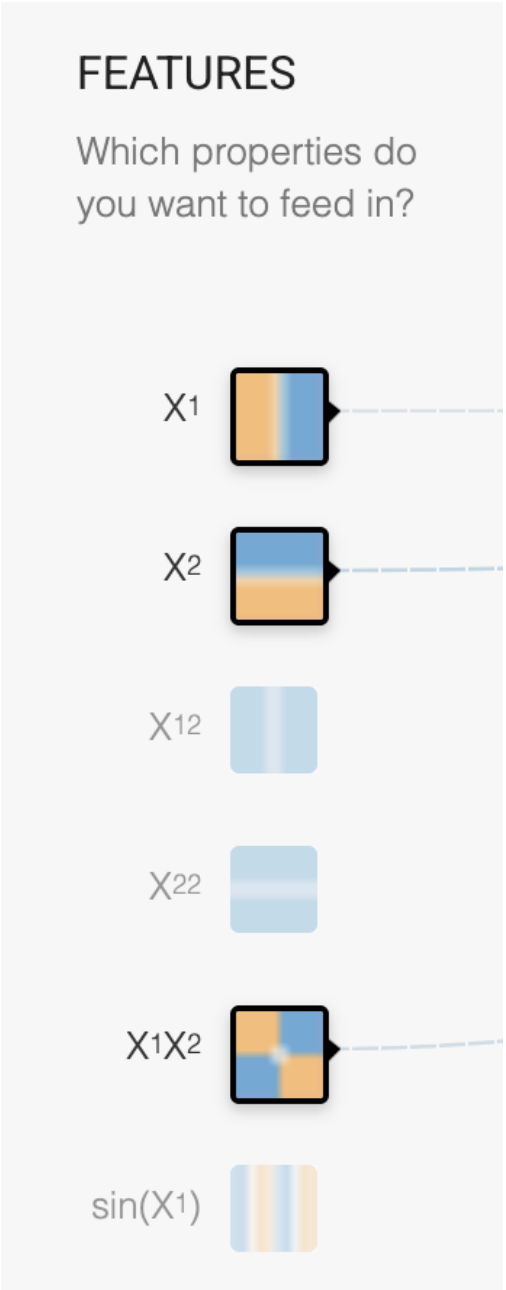
Logistic regression loss

$$\frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(-Y_i \quad w^T X_i))$$

Outline

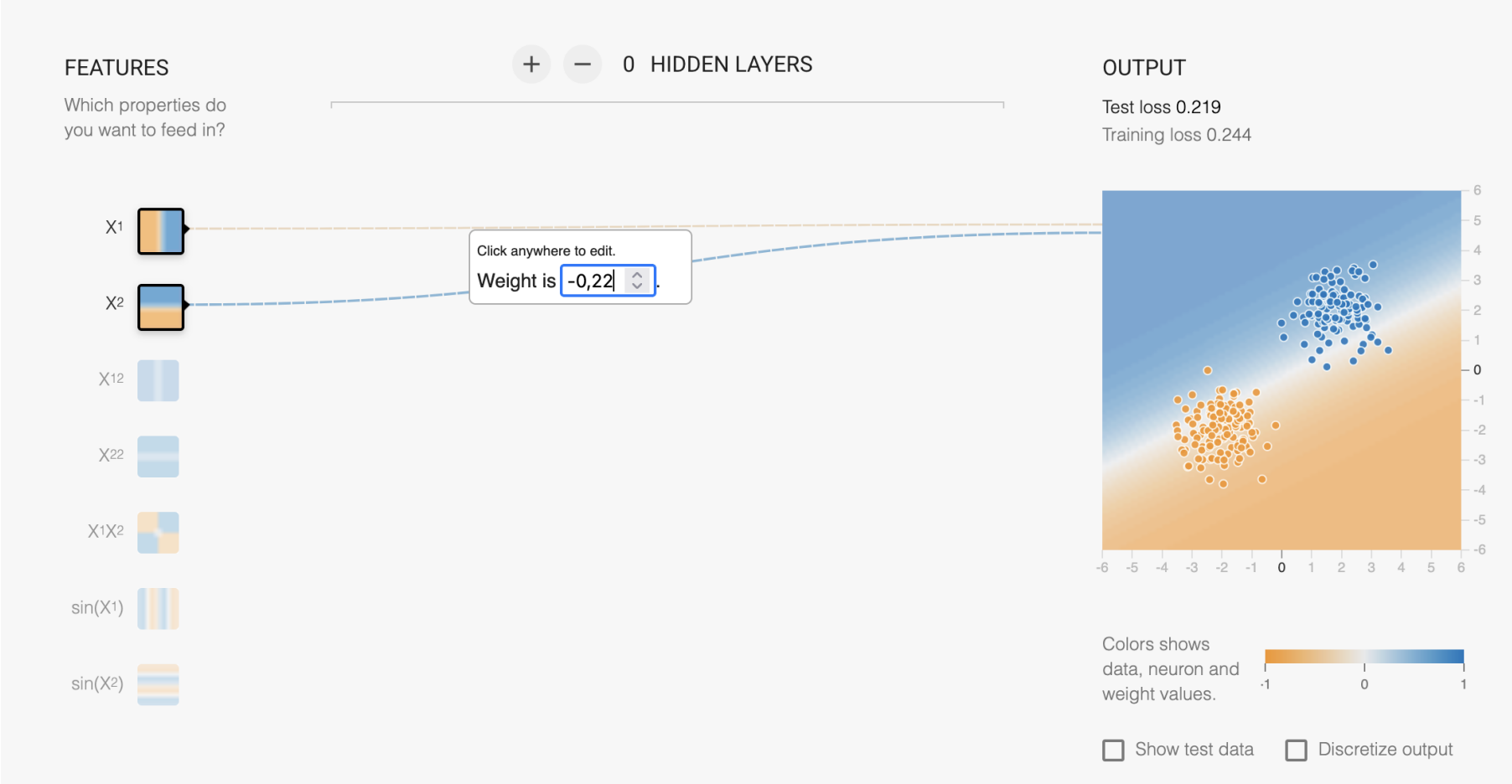
Let's NN playground: Link

1. Understand what the features are. Understand what colors represent.



Let's NN playground: Link

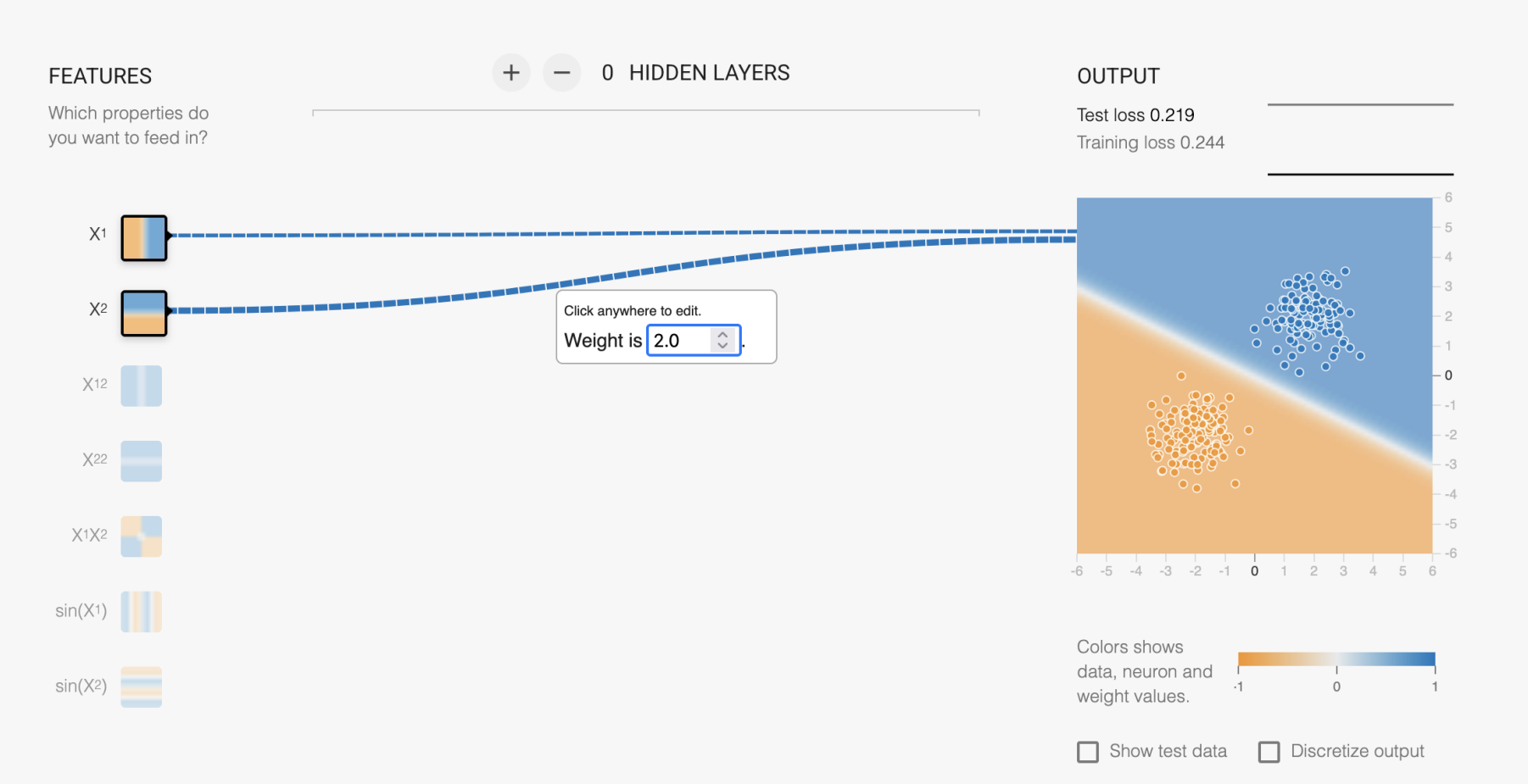
2. Consider the simplest dataset, an 0 hidden layer. Manually pick weights to obtain a good classification.



What model does this correspond to? How many weights does it have?

Let's NN playground: Link

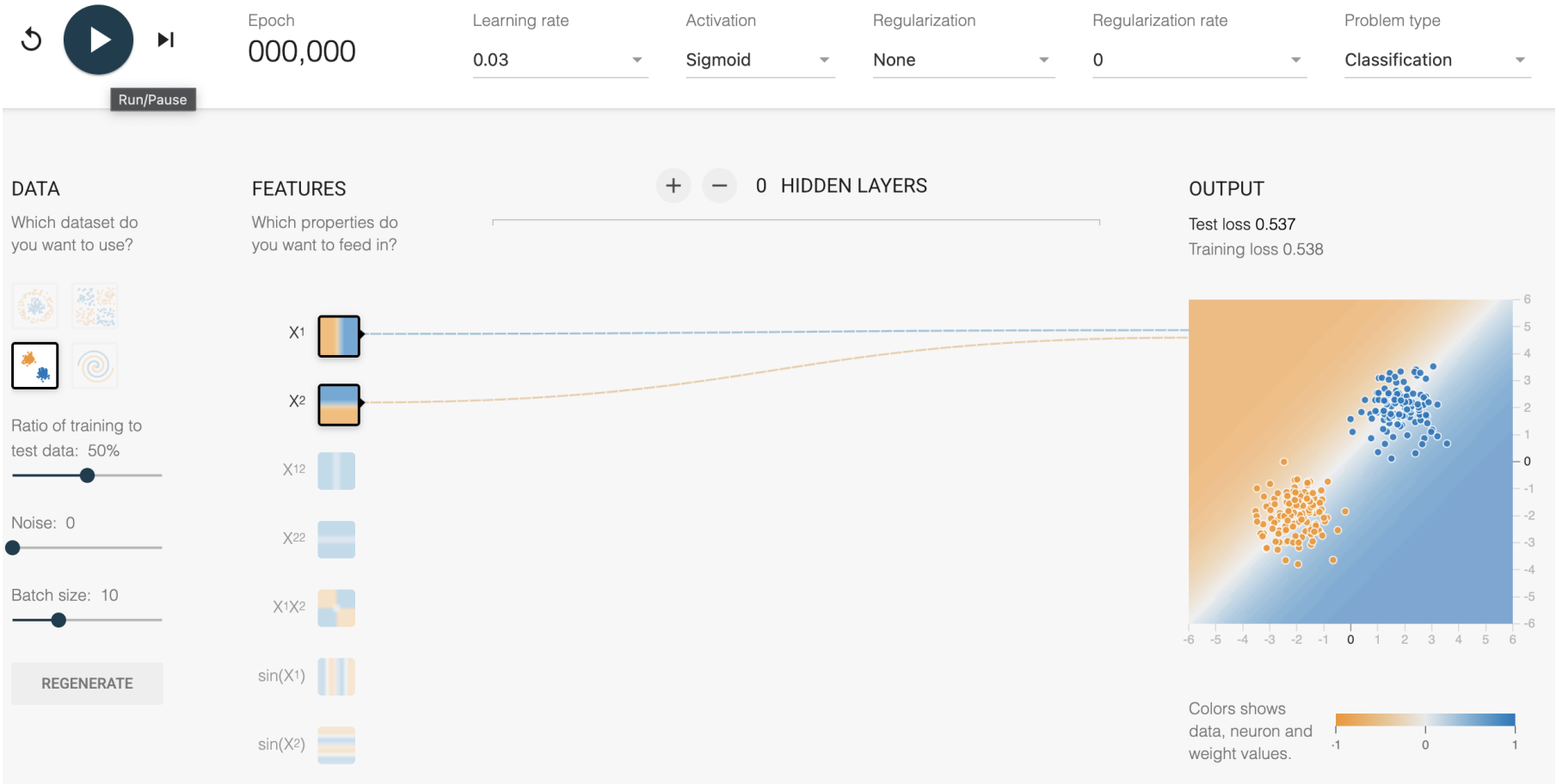
2. Consider the simplest dataset, an 0 hidden layer. Manually pick weights to obtain a good classification.



What model does this correspond to? How many weights does it have?

Let's NN playground: Link

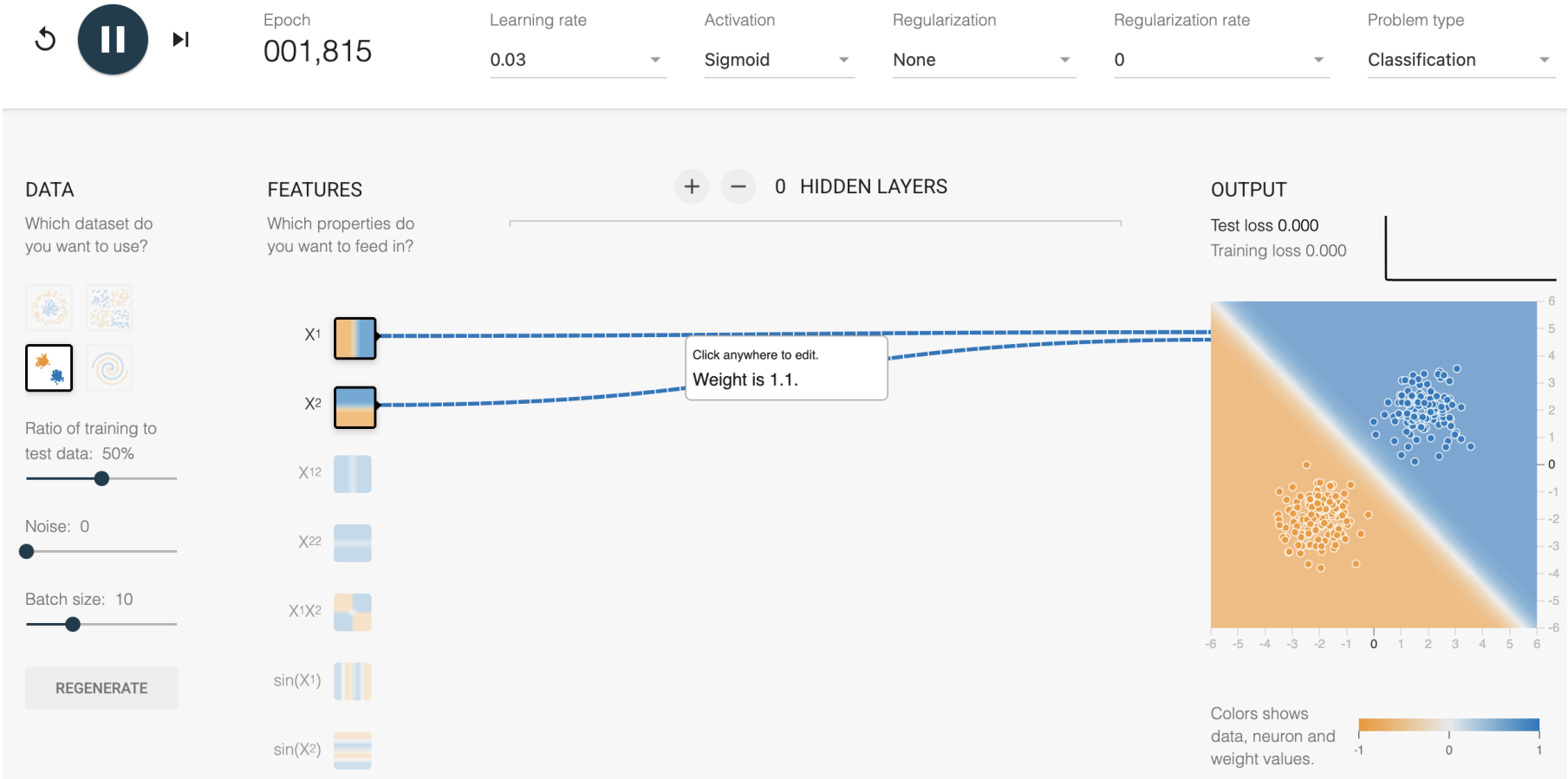
3. Consider the simplest dataset, an 0 hidden layer. Instead of manually tuning the weights, obtain them by fitting!



What algorithm are we running?
What can you say about the loss?
Rerun the algorithm? What happens?

Let's NN playground: Link

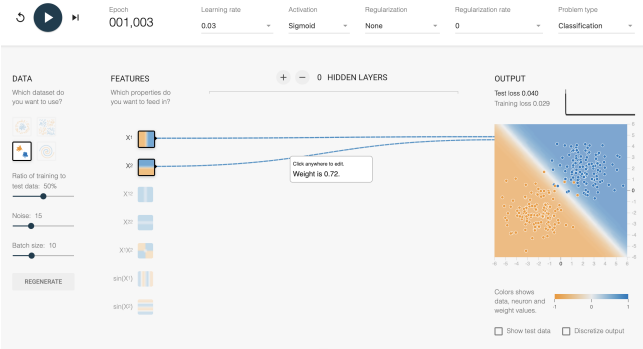
3. Consider the simplest dataset, an 0 hidden layer. Instead of manually tuning the weights, obtain them by fitting!

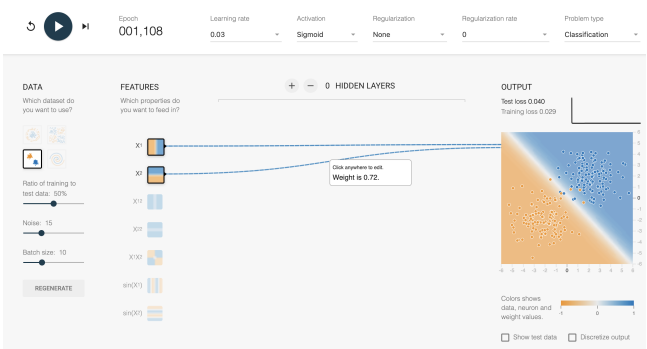


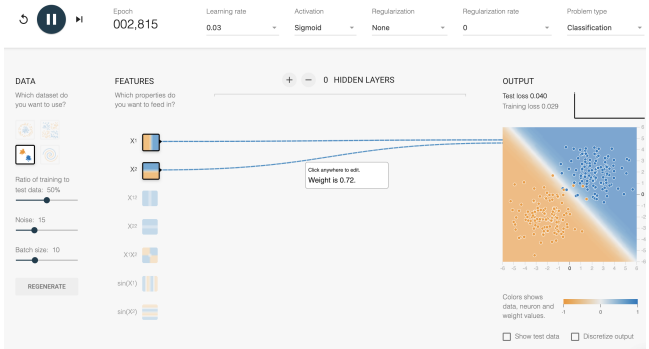
What algorithm are we running?
What can you say about the loss?
Rerun the algorithm? What happens?

Let's NN playground: Link

4. Increase the data noise to 15. Is the data linearly separable then? Run 3 times for 1000 epochs, compare the weights obtained



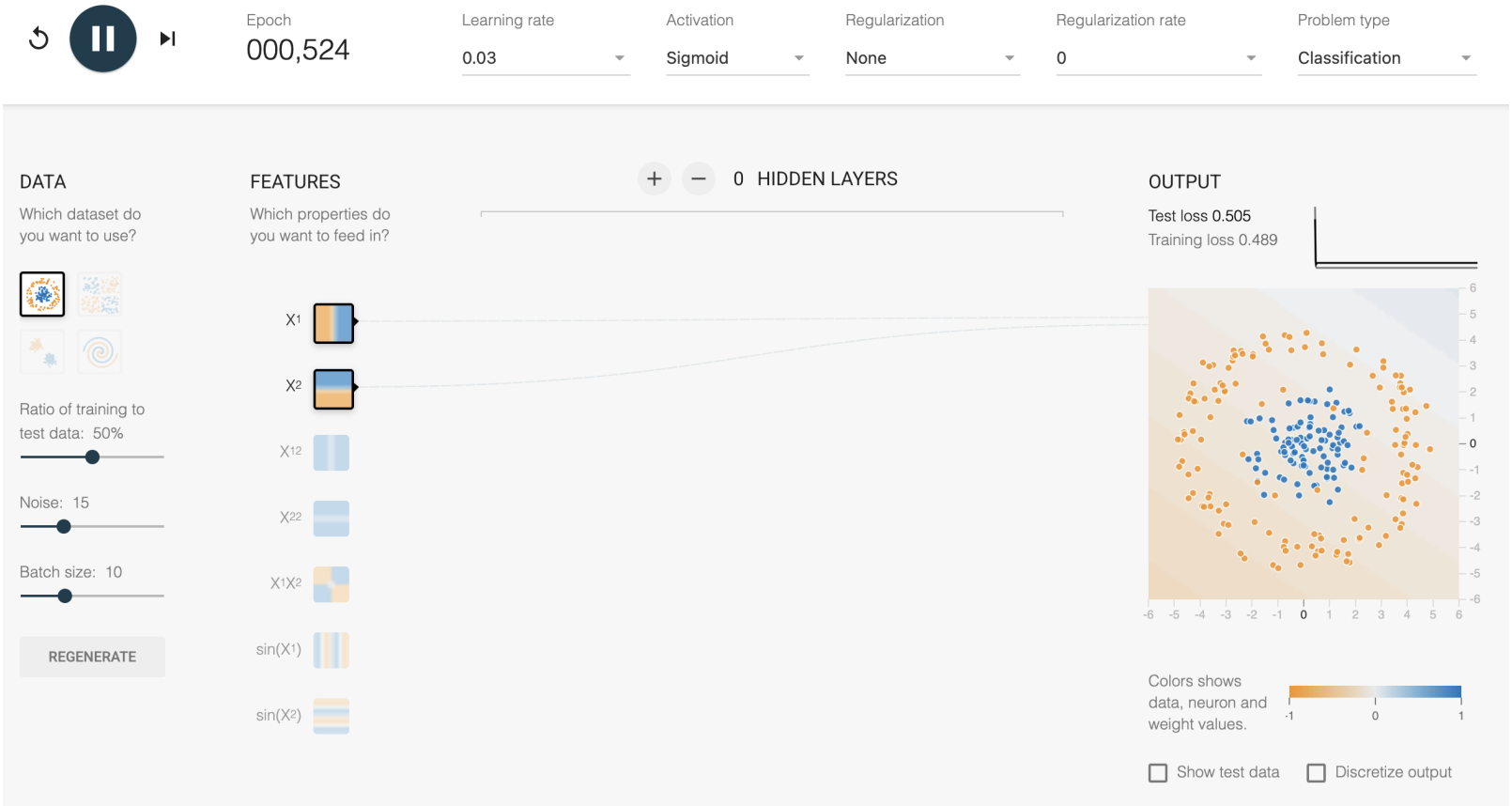




What do you conclude / confirm on logistic regression?

Let's NN playground: Link

5. Change the dataset to  . Run logistic regression. What do you observe?



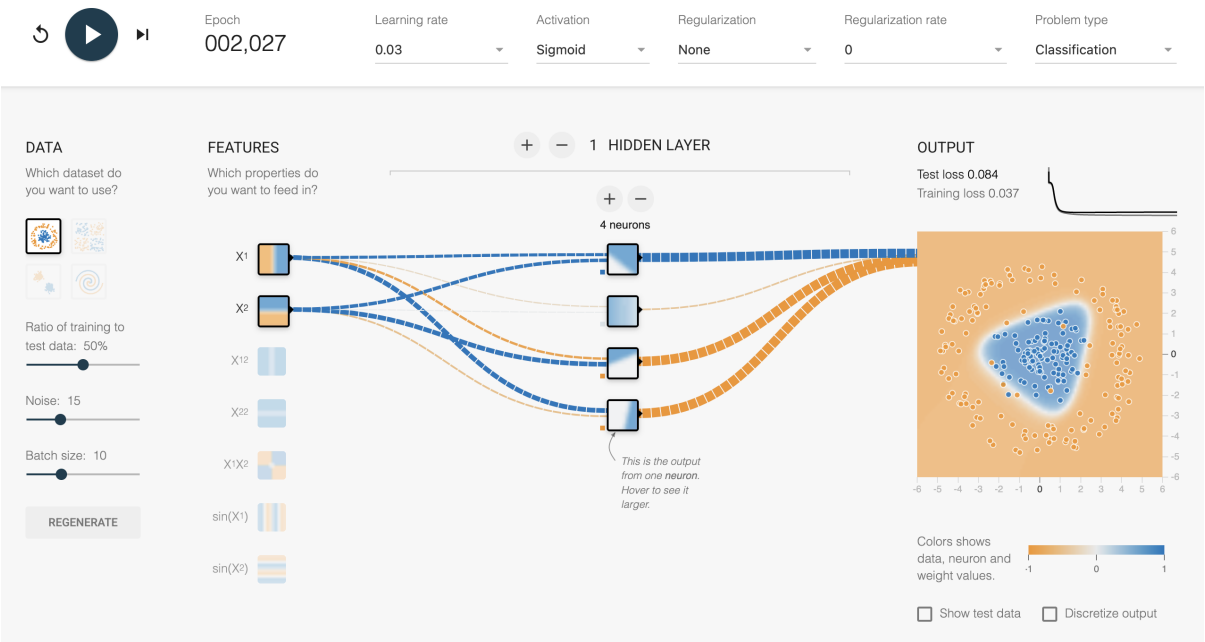
What do you suggest?

Let's NN playground: Link

6. Add a single hidden layer, first with a single neuron, then with 4 neurons.
Keep only features X^1 and X^2.

Let's NN playground: Link

6. Add a single hidden layer, first with a single neuron, then with 4 neurons.
Keep only features X^1 and X^2.



What do you observe?
What have we learned?
How does that illustrate *feature learning*?
What happens if you restart learning?

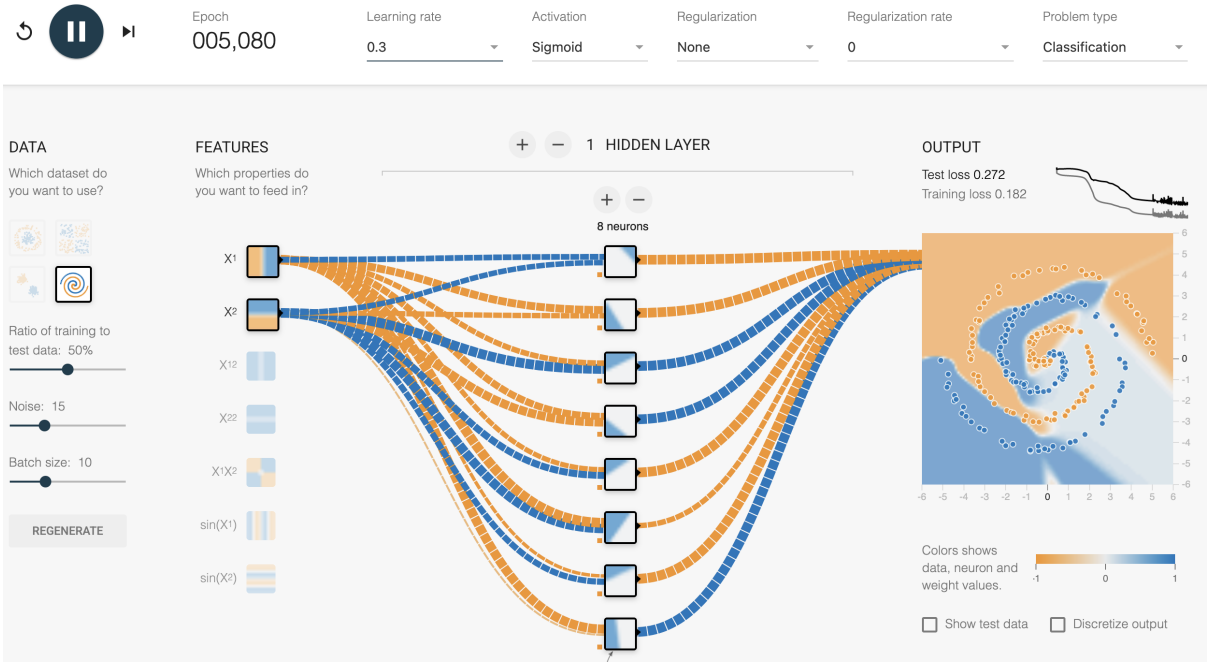
Let's NN playground: Link

6. Switch to the more complex spiral dataset  **. Increase the number of neurons. Fit the model.**
Keep only features X^1 and X^2.

Let's NN playground: Link

6. Switch to the more complex spiral dataset  . Increase the number of neurons. Fit the model.

Keep only features X^1 and X^2.



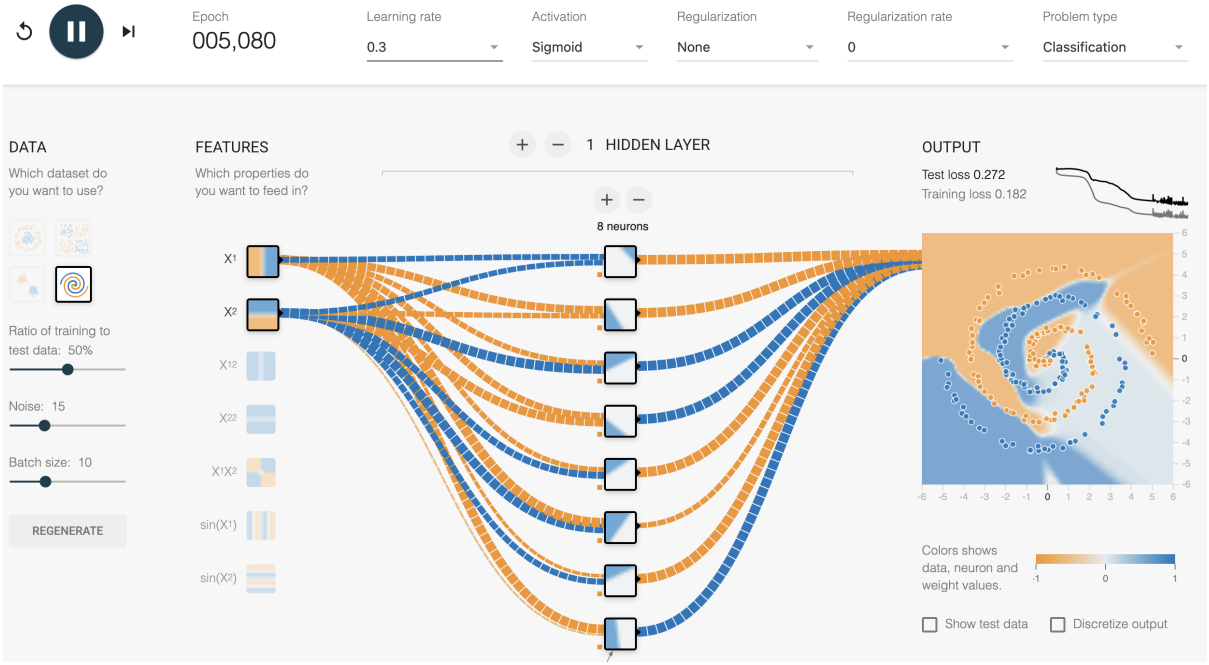
What happens? How stable is the training?

How many weights are you learning now? Are there biases? What are those?

Let's NN playground: Link

6. Switch to the more complex spiral dataset  . Increase the number of neurons. Fit the model.

Keep only features X^1 and X^2.

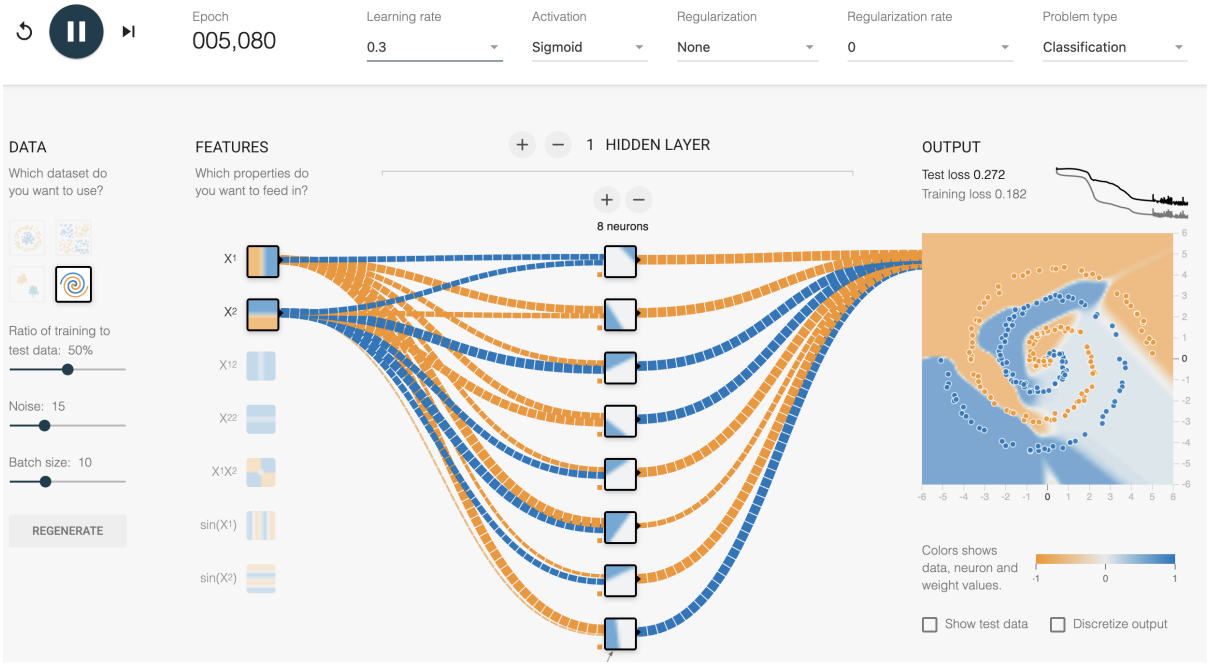


What can you say about the algorithm dynamic ?

Let's NN playground: Link

6. Switch to the more complex spiral dataset  . Increase the number of neurons. Fit the model.

Keep only features X^1 and X^2.



What do you suggest next?

Let's NN playground: Link

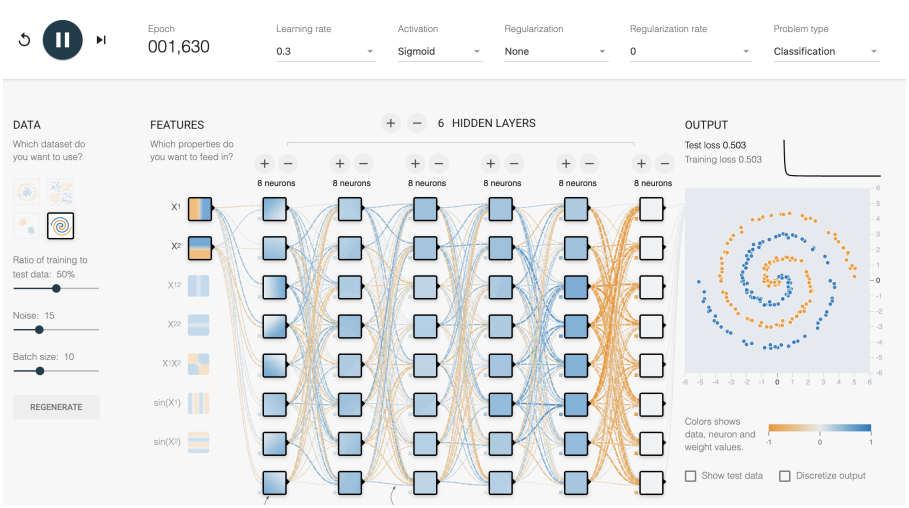
7. Explore configurations : what do you obtain?

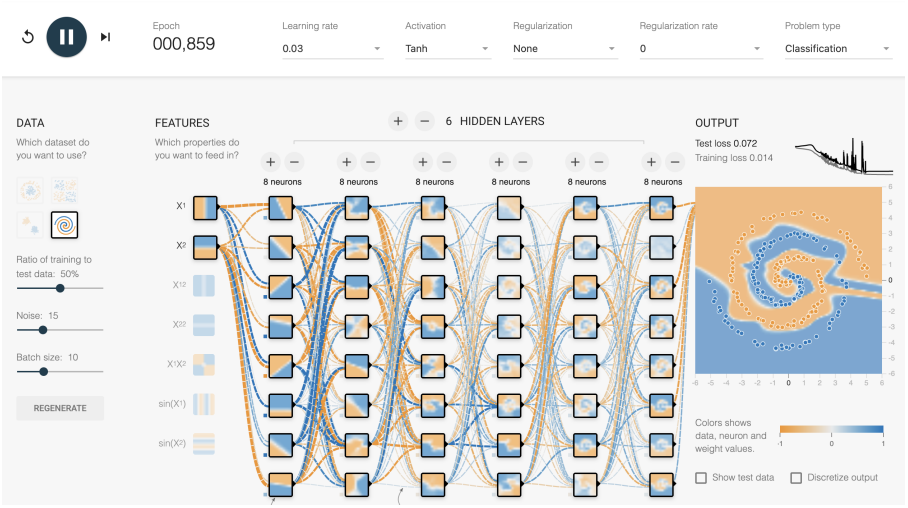
Let's NN playground: Link

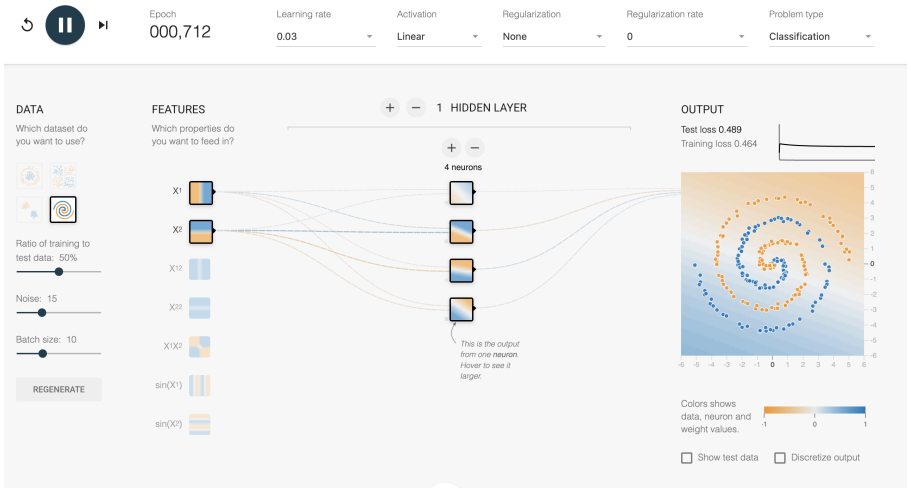
7. Explore configurations : what do you obtain? What is the role of the activation function? What happens for a deep architecture for Linear / Sigmoid / Relu / Tanh?

Let's NN playground: Link

7. Explore configurations : what do you obtain? What is the role of the activation function? What happens for a deep architecture for Linear / Sigmoid / Relu / Tanh?





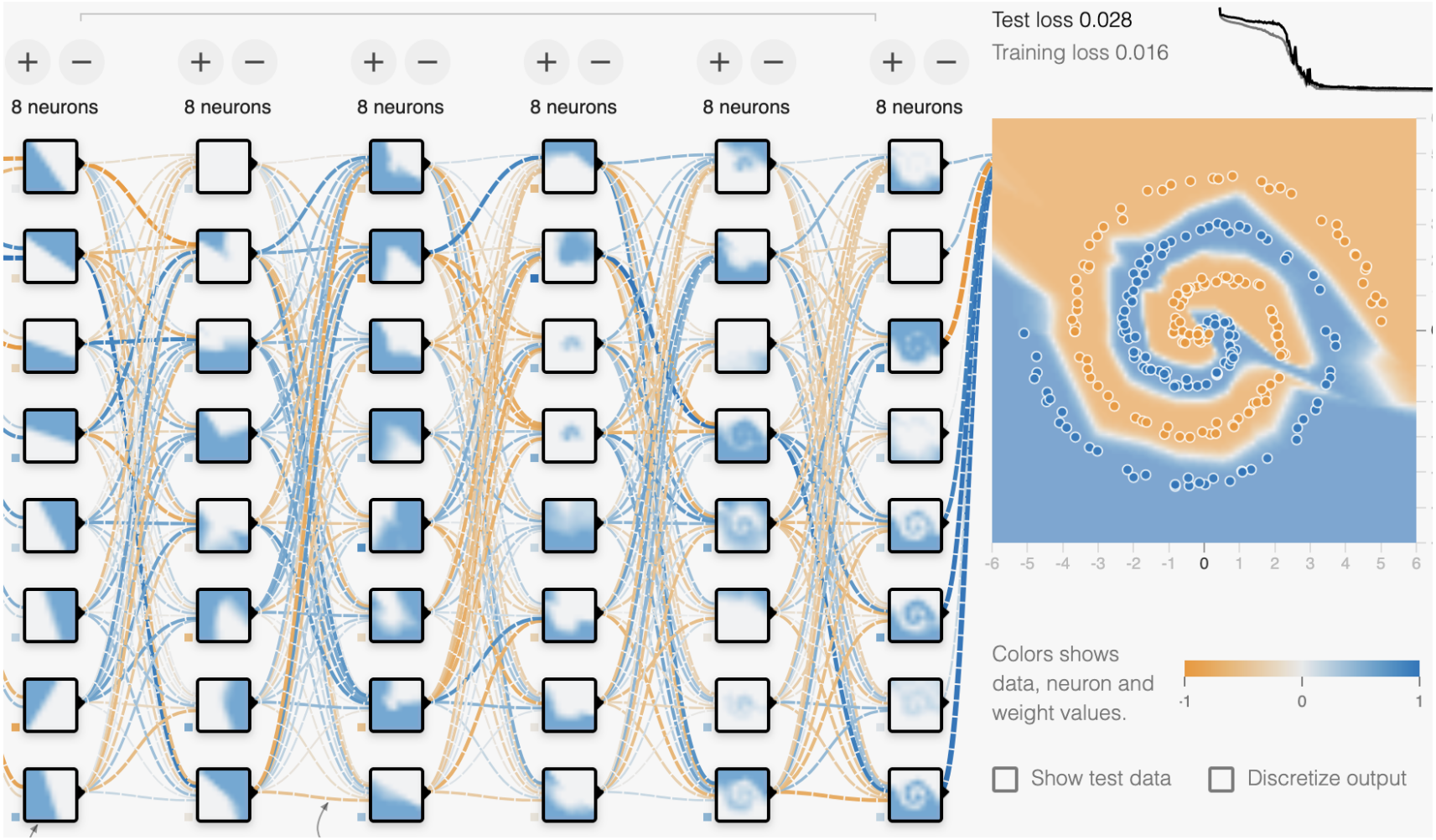


Let's NN playground: Link

8. How do the features learned evolve depending on the layer?

Let's NN playground: Link

8. How do the features learned evolve depending on the layer?



Let's NN playground: Link

9. What are the hyper-parameters of the SGD algorithm? Play with them.

Advanced

1. What happens if you initialize all weights to 0?

Conclusion on NNPG

1. "Architecture":

2. What we have seen

3. *Protip*

Wooclap time!

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
BLCOFS

1 Envoyez @BLCOFS au 06 44 60 96 62

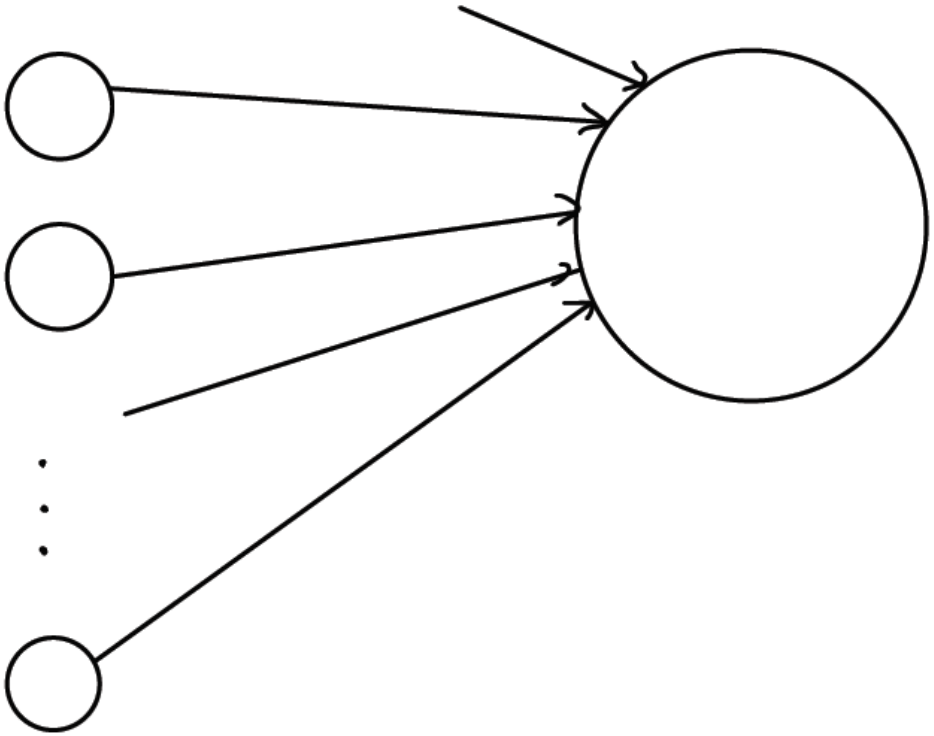
2 Vous pouvez participer

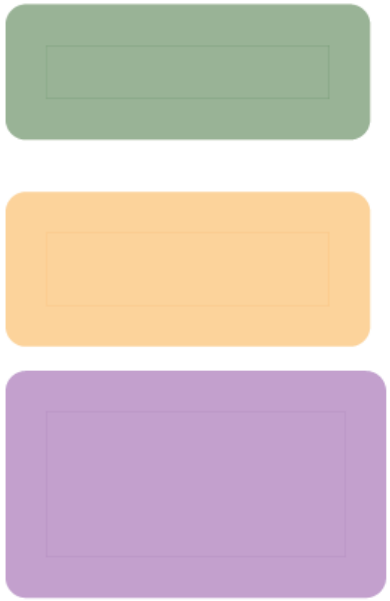
Désactiver les réponses par SMS

Outline

From logistic regression to Multi Layer Perceptron 3

- Class of functions for logistic regression: $\mathscr{F} = \{X \mapsto w^T X,\ w \in \mathbb{R}^d\}$.
- Prediction for a new point is $1_{\sigma(X_i)>1/2}$
- Can be seen as predicting a probability $\sigma(X_i)$ that the output is 1

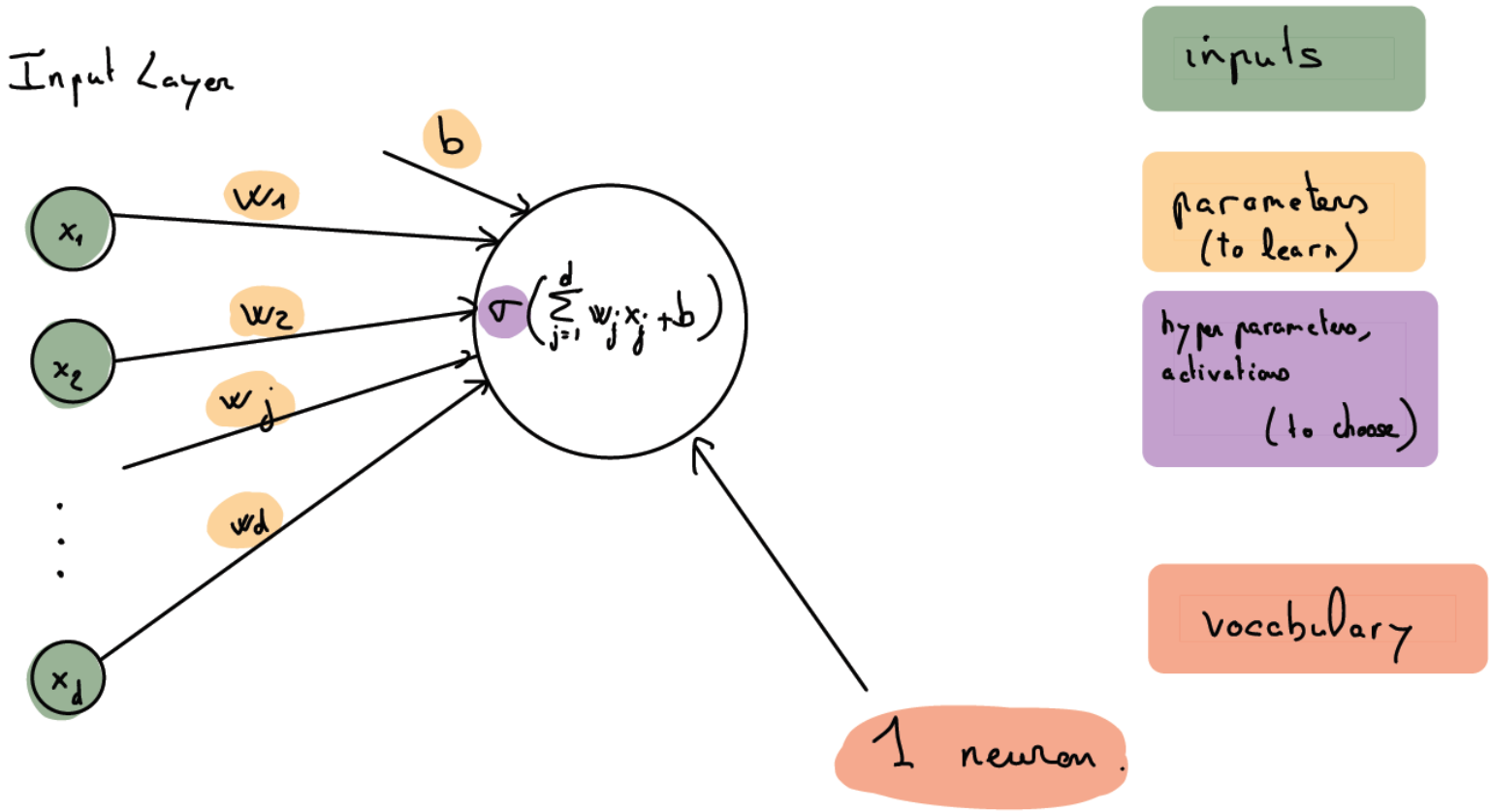




This is a neural network, with one neuron !

From logistic regression to Multi Layer Perceptron 3

- Class of functions for logistic regression: $\mathscr{F} = \{X \mapsto w^T X,\ w \in \mathbb{R}^d\}$.
- Prediction for a new point is $1_{\sigma(X_i)>1/2}$
- Can be seen as predicting a probability $\sigma(X_i)$ that the output is 1

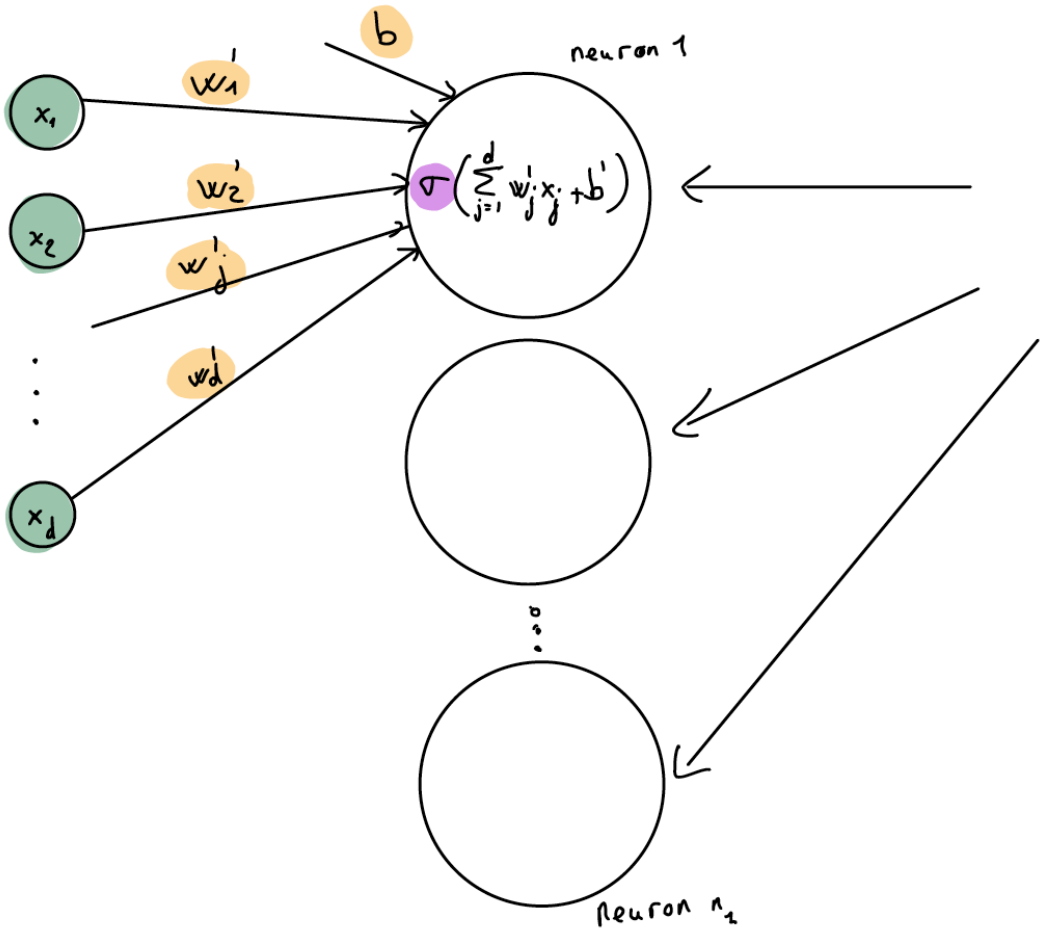


This is a neural network, with one neuron !

From logistic regression to Multi Layer Perceptron 4

Extend to

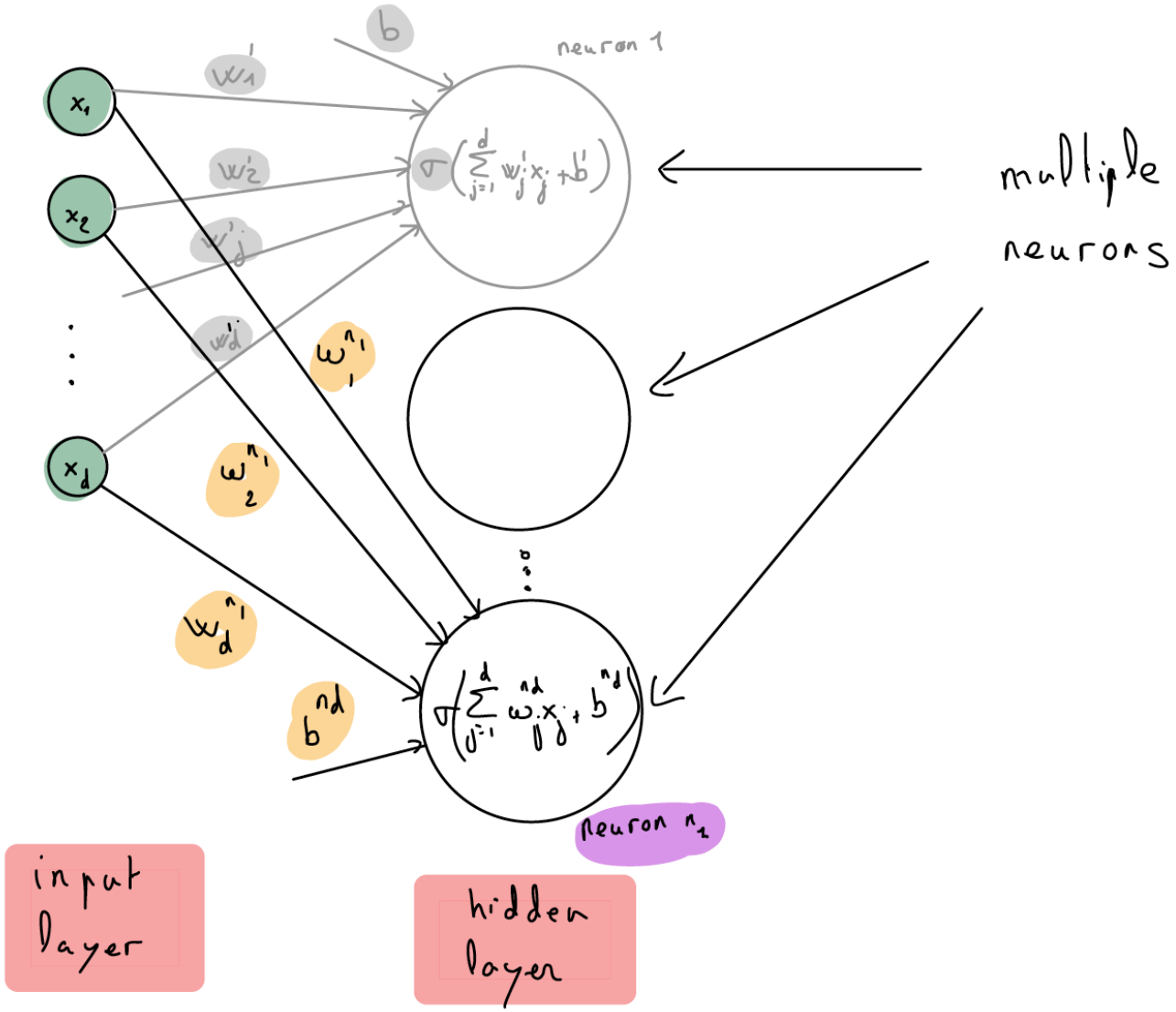
- Multiple Neurons
- Multiple Layers



From logistic regression to Multi Layer Perceptron 4

Extend to

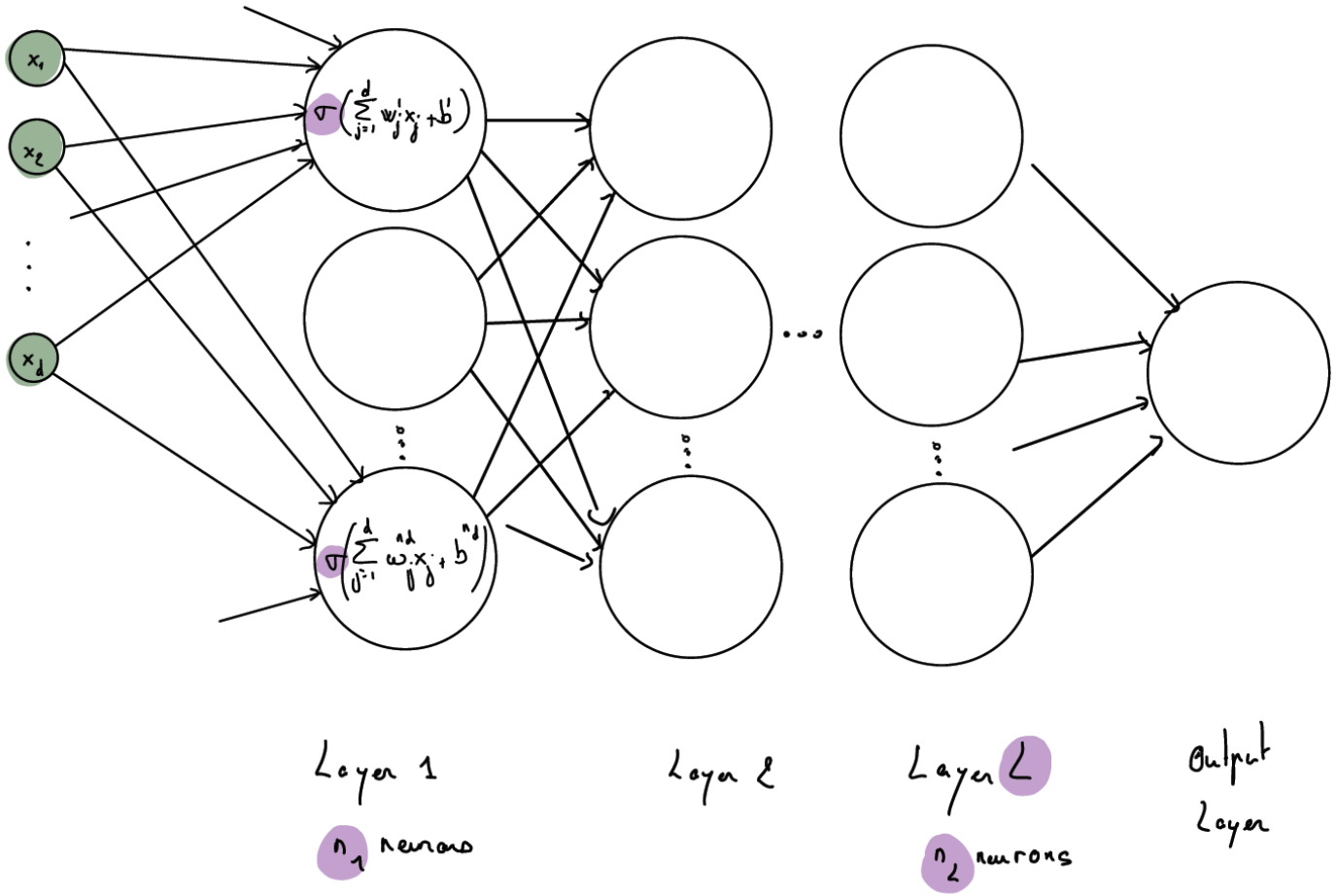
- Multiple Neurons
- Multiple Layers



From logistic regression to Multi Layer Perceptron 5

Extend to

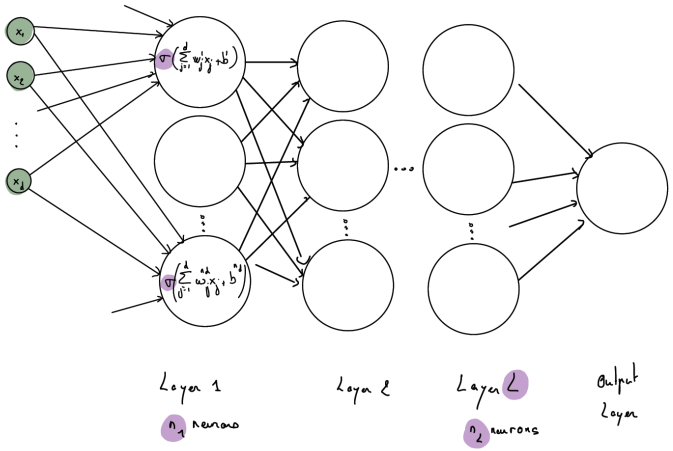
- Multiple Neurons
- Multiple Layers



From logistic regression to Multi Layer Perceptron 5

Neural network:

- Goal $\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$
- Class of non linear functions.
  $\mathscr{F}_{\text{MLP}} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(...(b_1 + W_1 x))),$
  $W \in \mathbb{R}^{\cdots}, b \in \mathbb{R}^{\cdots}\}$



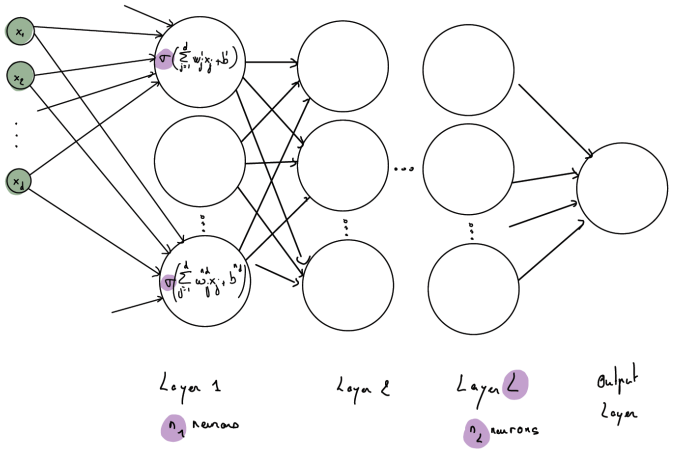
Summary: NN generalize regression by looking for candidates in a much larger class of functions than linear ones.

What needs to be learned What needs to be chosen Vocabulary

From logistic regression to Multi Layer Perceptron 5

Neural network:

- Goal $\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$
- Class of non linear functions.
  $\mathscr{F}_{\text{MLP}} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(...(b_1 + W_1 x))),$
  $W \in \mathbb{R}^{\cdots}, b \in \mathbb{R}^{\cdots}\}$



Summary: NN generalize regression by looking for candidates in a much larger class of functions than linear ones.

What needs to be learned

Parameters W, b

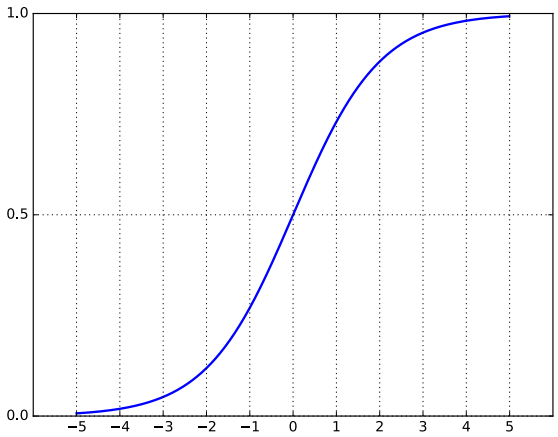
What needs to be chosen

- Activation σ
- Number of layers L
- Number of neurons per hidden layer $n_i, i = 1, \ldots, L$.

Vocabulary

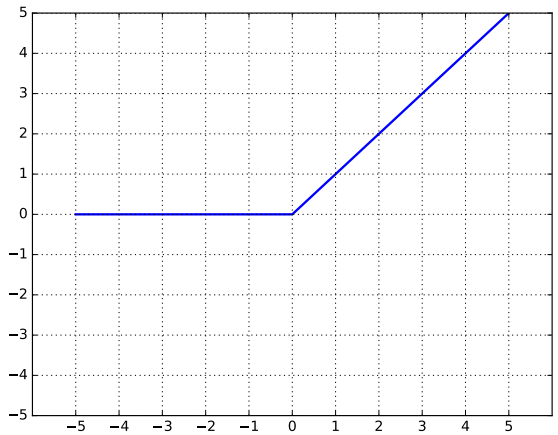
- Neuron
- Activation
- Hidden Layer
- Width, Depth
- Fully connected layer

2 possible activations: Sigmoid and Rectified Linear Unit



Sigmoid function

- $x \mapsto \frac{\exp(x)}{1+\exp(x)}$
- Problems:
  1. Saturated function: gradient killer -> need for rescaling data



Rectified Linear Unit (ReLU)

- $x \mapsto \max(0, x)$
- Pros & cons:
  1. Not a saturated function. Kills negative values.
  2. Empirically, convergence is faster than sigmoid/tanh.
  3. Plus: biologically plausible

How to deal with multi-class output: softmax activation

When we do multi-class classification, i.e., $Y_i \in 1, \ldots, K$, we output K different values:

- Each of them corresponds to the probability of belonging to the class j, $1 \leq j \leq K$
- To obtain probabilities (adding up to 1):
  1. We have K neurons on the last layer
  2. And use a Softmax activation to renormalize.

Softmax output unit, used to predict $\{1, \ldots, K\}$:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

Up to that point, we have seen :

- What a Neural Network was
- Why it is expected to learn better.

Next Question: how to implement it ?

Python - Keras

1. Hardware:
   - CPU
   - GPU
   - TPU

2. Software (Python packages):
   - Pytorch/Tensorflow
   - → Keras : TF high level API. Ideal for applications.

Keras - A few lines ![1]

What do we need to specify?

What do we need to do next?

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
# get data
(train_images, train_labels), (test_images,
                               test_labels) = keras.datasets.mnist.load_data()
# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(64, activation= tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# train models
model.fit(train_images, train_labels, epochs=5)

print('\n Evaluation' )
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
# make predictions
predictions = model.predict(test_images)
```

Wooclap!

Wooclap

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
# get data
(train_images, train_labels), (test_images,
                               test_labels) = keras.datasets.mnist.load_data()
# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(64, activation= tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# train models
model.fit(train_images, train_labels, epochs=5)

print('\n Evaluation' )
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
# make predictions
predictions = model.predict(test_images)
```

What can you say about this code bit

1. The flatten layer is a dense layer
2. I have 2 hidden layers
3. I have one hidden layer
4. I should use sigmoid activation on the last layer
5. I use SGD for the algorithm
6. AdamOptimizer is EveOptimizer's boyfriend
7. I run 5 epochs of the stochastic algorithm
8. I use relu activation on all but the last layer

Wooclap

```python
# import tensorflow and keras (tf.keras not "vanilla" Keras)
import tensorflow as tf
from tensorflow import keras
# get data
(train_images, train_labels), (test_images,
                               test_labels) = keras.datasets.mnist.load_data()
# setup model
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(64, activation= tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# train models
model.fit(train_images, train_labels, epochs=5)

print('\n Evaluation' )
# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)
# make predictions
predictions = model.predict(test_images)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8124 - loss: 4.6972
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9239 - loss: 0.3186
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9464 - loss: 0.2017
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9548 - loss: 0.1581
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9610 - loss: 0.1388

 Evaluation
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9592 - loss: 0.1602
test accuracy: 0.9621999859809875
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

We now see the output. What can we say

1. We overfit
2. There are 313 points in the test set
3. There are 10 000 points in the test set
4. 1875 is about 6 times 313
5. the loss is the accuracy
6. the loss decreases along the trajectory

Wooclap

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

Explain the number of parameters on each layer


```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 64)                8256

 dense_2 (Dense)             (None, 10)                650

=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

Outline

Intuition

Up to that point, we have seen :

- What a Neural Network was.
- How to implement it in Python.

Next Question: why and when does it work?

Intuition

Up to that point, we have seen :

- What a Neural Network was.
- How to implement it in Python.

Next Question: why and when does it work?

1. Approximation theorem
2. Optimization

Approximation Theorem

Goal

$$\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)].$$

As said before, a MLP can output non-linear functions... But how powerful is it?

Approximation Theorem

Goal

$$\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)].$$

As said before, a MLP can output non-linear functions... But how powerful is it?

Continuous Neural Networks with one single hidden layer and any bounded and non constant activation function can approximate any function in L^p, provided a sufficient number of hidden units.

$\looparrowright$ Very powerful in terms of approximations.
Not very surprising: non linear function with millions of parameters!

Optimization - Fitting the network

Goal $$\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)].$$

How to minimize a function?
Gradient Methods. = cheapest way to update (learn) the weights iteratively to minimize the loss.

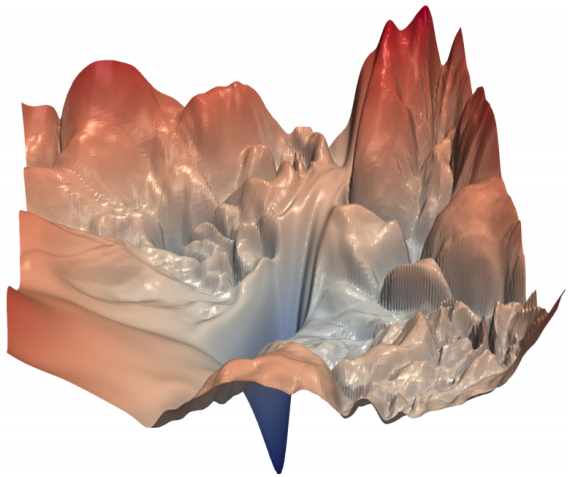
Optimization - Fitting the network

Goal $$\min_{f \in \mathscr{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)].$$

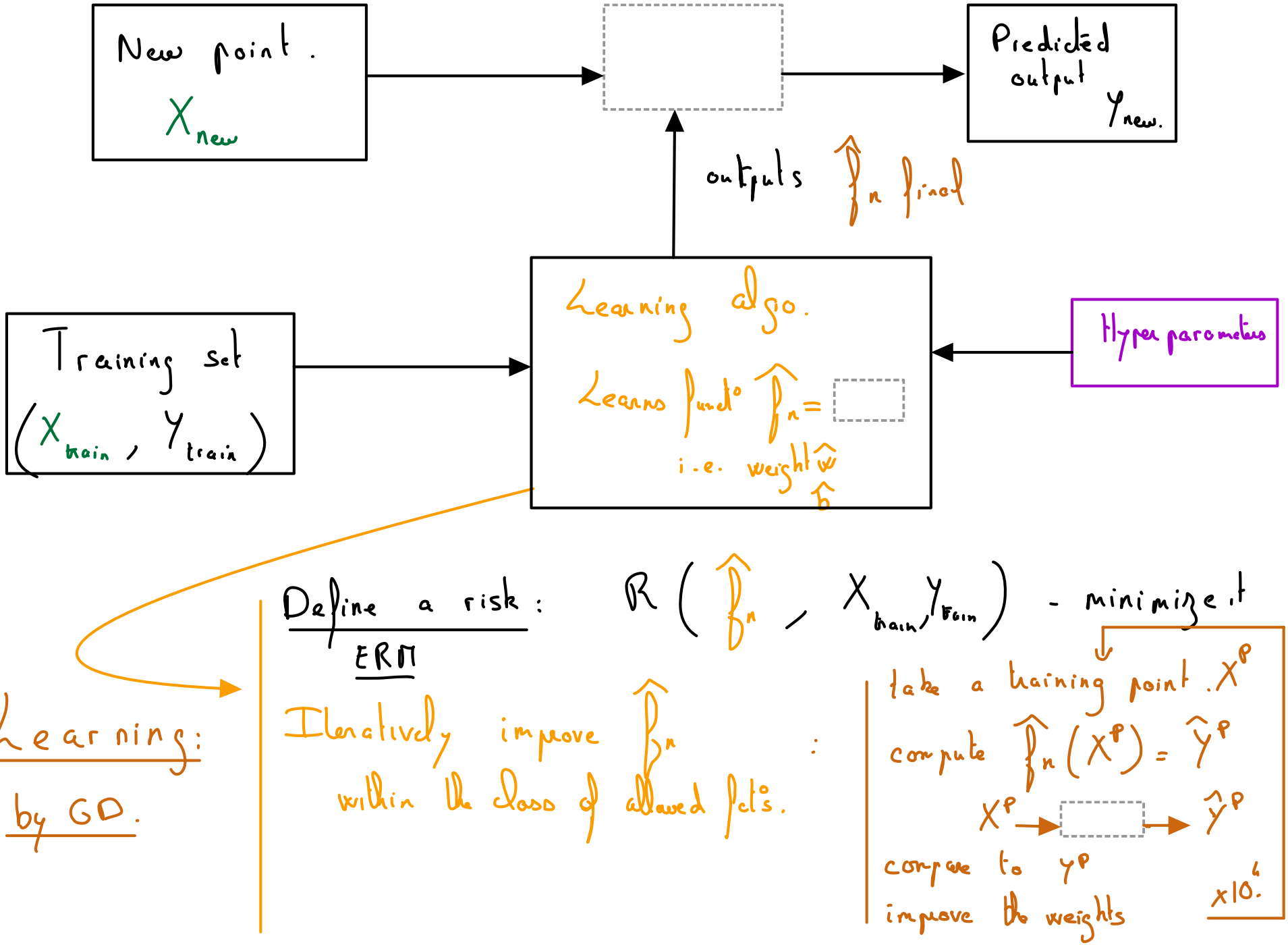
Gradient Methods. = cheapest way to update (learn) the weights iteratively to minimize the loss.

Two "miracles":

1. Autodifferentiation: even though the function is very complex and high dimensional, it is possible to compute gradients!
2. Optimization seems to work ok, though the function is non convex - we do not end up is too bad local minima. (specialized optim algos: Adam, AdaDelta, etc.)



These 2 miracles make it possible to learn a good regressor/classifier with NN and to benefit from the powerful approximation properties


New point.
X_{new}
Predicted output
Y_{new}
outputs $\hat{f}_n$ final
Learning algo.
Learns funct° $\hat{f}_n$ =
i.e. weight $\hat{w}$
$\hat{b}$
Training set
(X_{train}, Y_{train})
Hyper parameters
Learning:
by GD.
Define a risk:
ERM
$R(\hat{f}_n, X_{train}, Y_{train})$ - minimize it
Iteratively improve $\hat{f}_n$:
within the class of allowed fct°.
take a training point X^p
compute $\hat{f}_n(X^p) = \hat{Y}^p$
X^p → → $\hat{Y}^p$
compare to Y^p
improve the weights
$\times 10^4$.

Conclusion

Neural networks :

1. learn very complex non linear functions in high dimension
2. can approximate nearly any function
3. can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

Conclusion

Neural networks :

1. learn very complex non linear functions in high dimension
2. can approximate nearly any function
3. can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

What my layer wants me to say: we haven't talked about many points !!

- Initialization
- Regularization

Conclusion

Neural networks :

1. learn very complex non linear functions in high dimension
2. can approximate nearly any function
3. can be optimized even though highly non convex.

Are thus expected to work:

- When depth or width increases.
- Thus with very large datasets
- Requiring a lot of computational power.

⇒ **Explains why they were so successful since 2010**

What my layer wants me to say: we haven't talked about many points !!

- Initialization
- Regularization

Next:

- Brief history
- How to use the structure?

A brief history of NN

- 1943: Neural networks
- 1957: Perceptron
- 1974-86: Backpropagation, RBM, RNN
- 1989-98: CNN, MNIST
- 2009: ImageNet
- 2012: AlexNet,
- 2014: GANss
- 2016: AlphaGo
- 2018: BERT

The Computational power made the major change (+ investment and creativity).

Directions

- When does it work?

Large or huge datasets. Structured tasks.

↬ **Images and text.**

⚠do not try to apply DL everywhere

Each application requires a special architecture:

- Images : Convolutional Neural Network (CNN).
- Text : Recurrent Neural Networks (RNN) (2012-2018), Transformers (2018-2024).