Deep Learning

Aymeric DIEULEVEUT

May 2024

1. Goals
2. The first Neural Network: the Perceptron
3. Convolutional Neural Networks
4. Applications
   - Image classification
   - Pose, action detection
   - Object detection and segmentation
   - Scene labeling - Semantic segmentation
   - Object tracking - videos

Outline

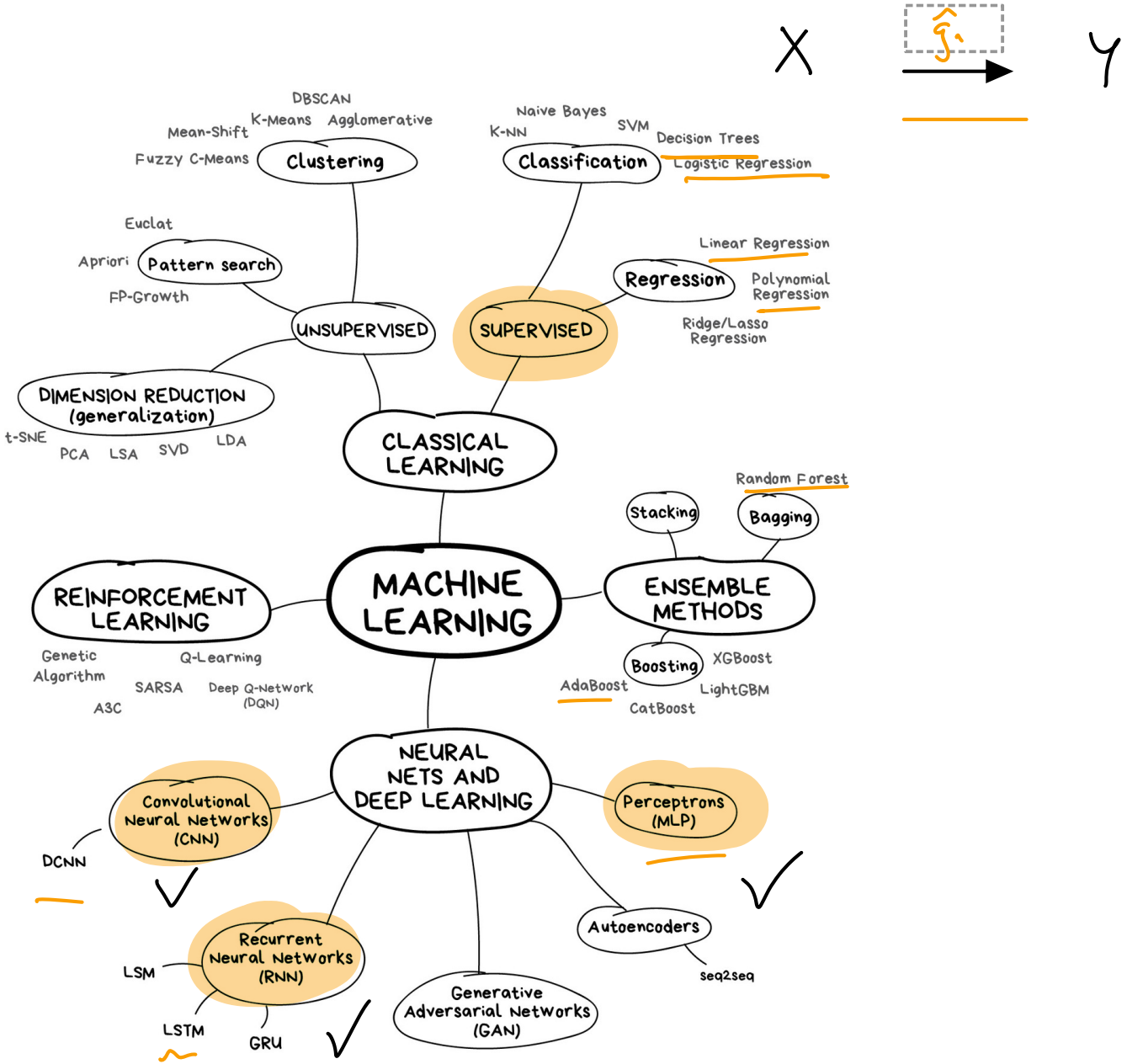
Deep Learning in one slide

- **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

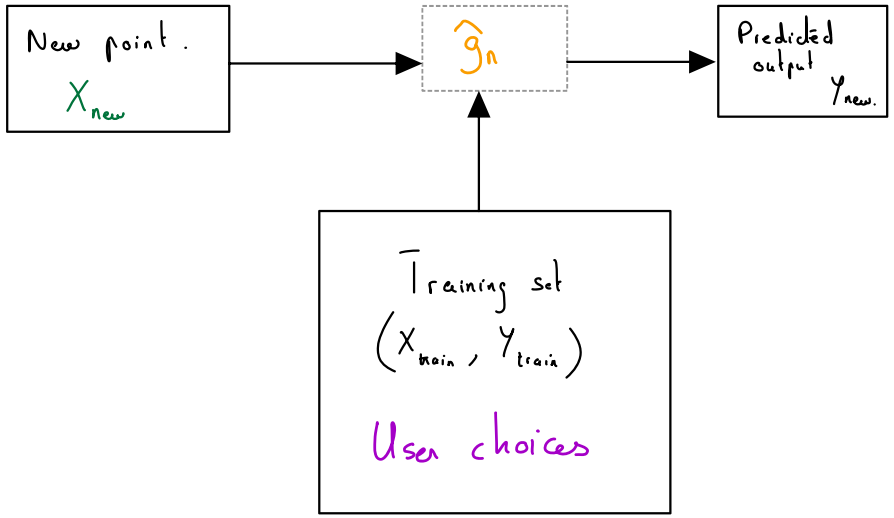


Summary of previous lectures on Machine Learning

Take home message (in 2 words) : generalize + overfitting

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

$$R(\hat{g}_n) = \mathbb{E}\left[\ell\left(\hat{g}_n(X_{new}), Y_{new}\right)\right]$$



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

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

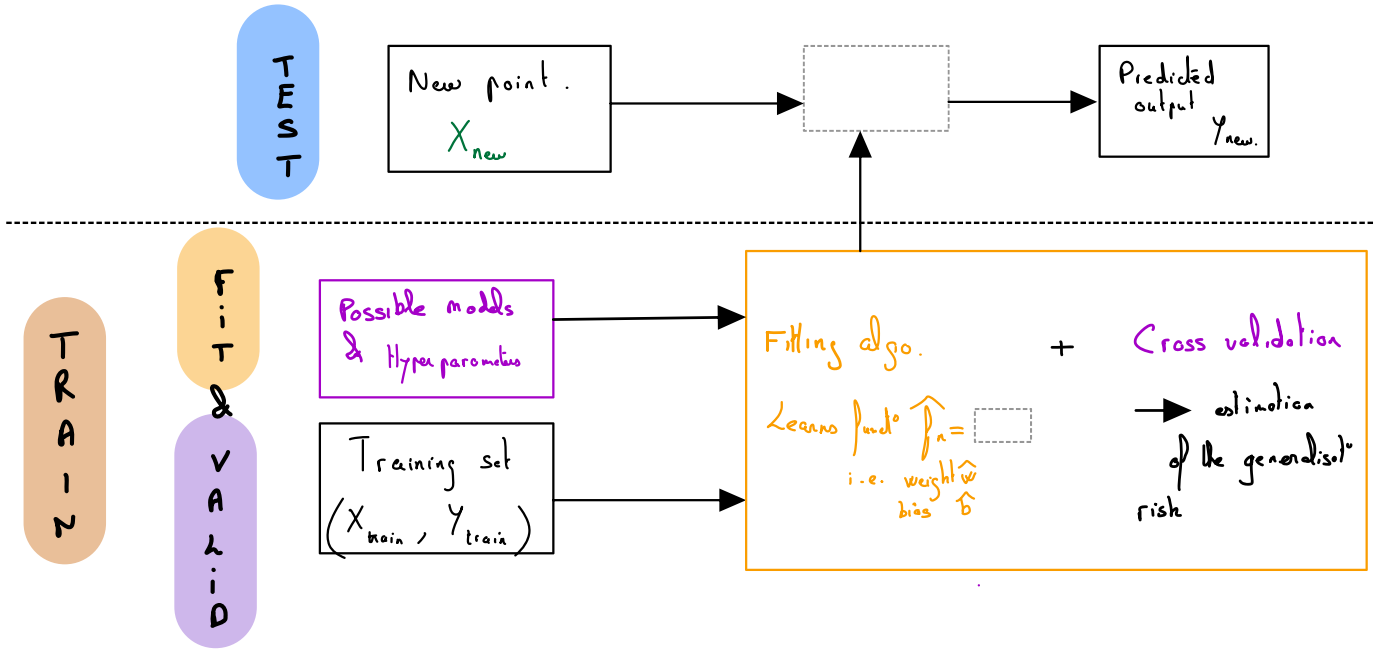
$\leftrightarrow$ Our goal is to predict well on inputs/outputs pairs that **we have not seen in the dataset**, i.e. *generalize* well.

THM 1.

Summary of previous lectures on Machine Learning

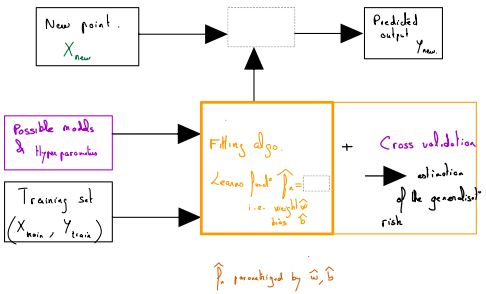
THM : Cross-Validation

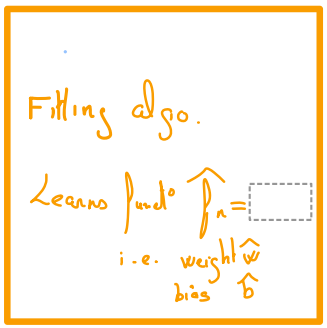
1. General framework: loss, risk, generalization risk vs training risk
2. SML Pipeline
   - Train (Fit + validation) / Test / Deployment
   - Evaluation of validation scores through cross validation to pick the best method/hyperparams

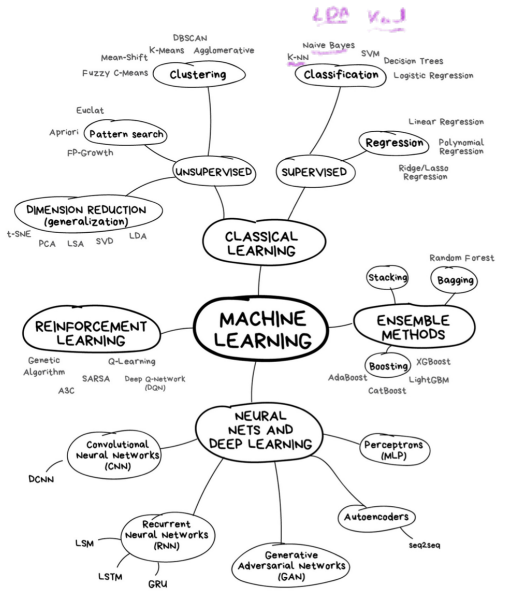


Summary of previous lectures on Machine Learning

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	explicit	→	algo. (opt)	✓	✓
Key hyperparameters	none (regularized)		not many	✓	✓
⊕	simple stable interpretable	//	//	✓	✓
⊖	underfitting risk	//	//	✓	✓
Overfitting / Underfitting	really under depends on d vs n if $\frac{d}{n}$ too close overfitting risk	→ much more overfit	//	✓	✓
→ regularized technique	Lasso* Ridge*		//	✓	✓
Approach	ERM	ERM	ERM	not ERM	not ERM

✓ = seen in lecture.

* Advanced

ERM : empirical risk minimisation

* Define class of funct° $\{f_n$ possible $\} = \mathcal{C}$

find $\min_{f \in \mathcal{C}} \{ \hat{R}_n(f) := \frac{1}{n} \sum_{i=1}^{n} \ell(f(x^i), y^i) \}$

$\hat{R}_n$: on the trainset

choice 1 (1) define loss → general. risk
→ empirical risk ER $\hat{R}_n$

choice 2 (2) choose a class $\mathcal{C}$

ERM = (3) → find the best $f \in \mathcal{C}$ for the ER

ie on train dat

Vérif (4) Check overfitting (risk is high if $\mathcal{C}$ large)

ERM encapsules : (1) & (2) → LR, LogR, NN, ..

	Seen		Seen
* Linear Reg	✓	* Poly Log Reg	~
* Logistic Reg	✓	* Poly Lin Reg	~
* NN	:)	* Kernels	×

Outline

From logistic regression to Multi Layer Perceptron 1

Supervised ML problem:

1. Data $D_n = (X_i, Y_i)_{i=1,\dots n}, X_i \in \mathbb{R}^d, Y_i \in \{0, \dots, K\}$
2. Goal: Predict a the label for a new point.

2 approaches:

Generative approach

- Define a model on your data (e.g., logit model).
- Estimate parameter (likelihood maximization).

→ LDA, QDA.

→ stat. based.

Discriminative approach

- No statistical model !
- Define a loss function ℓ
- Goal:

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

- Approach:

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

From logistic regression to Multi Layer Perceptron 1

Supervised ML problem:

1. Data $D_n = (X_i, Y_i)_{i=1,\dots n}, X_i \in \mathbb{R}^d, Y_i \in \{0, \dots, K\}$
2. Goal: Predict a the label for a new point.

2 approaches:

Generative approach

- Define a model on your data (e.g., logit model).
- Estimate parameter (likelihood maximization).

not seen afterwards

Discriminative approach – ERM

- No statistical model ! ✓
- Define a loss function ℓ ✓
- Goal:

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)]$$ → gen risk

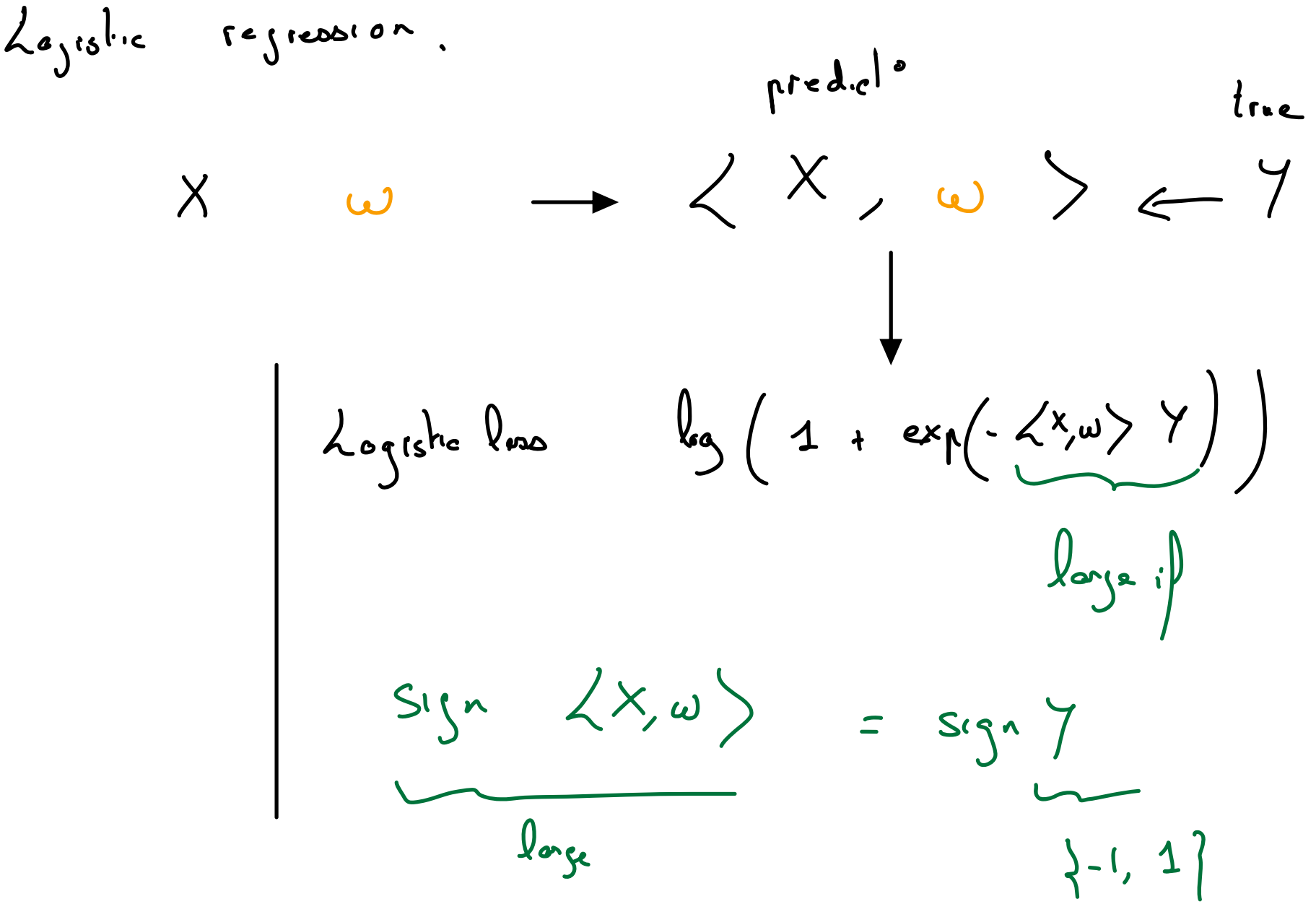
- Approach: ↕ overfitting ! ⚠

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

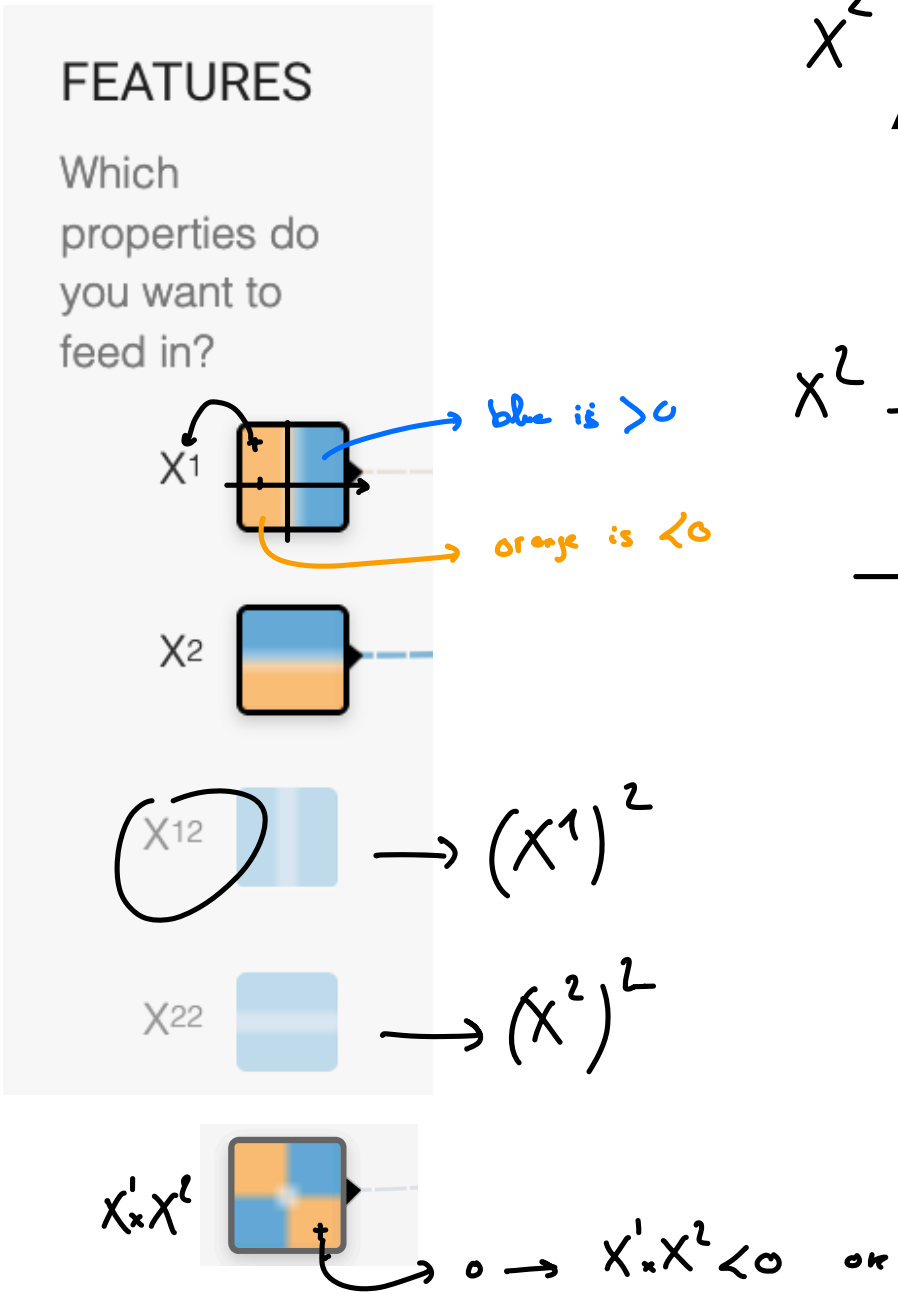
Logistic regression can be seen both ways !

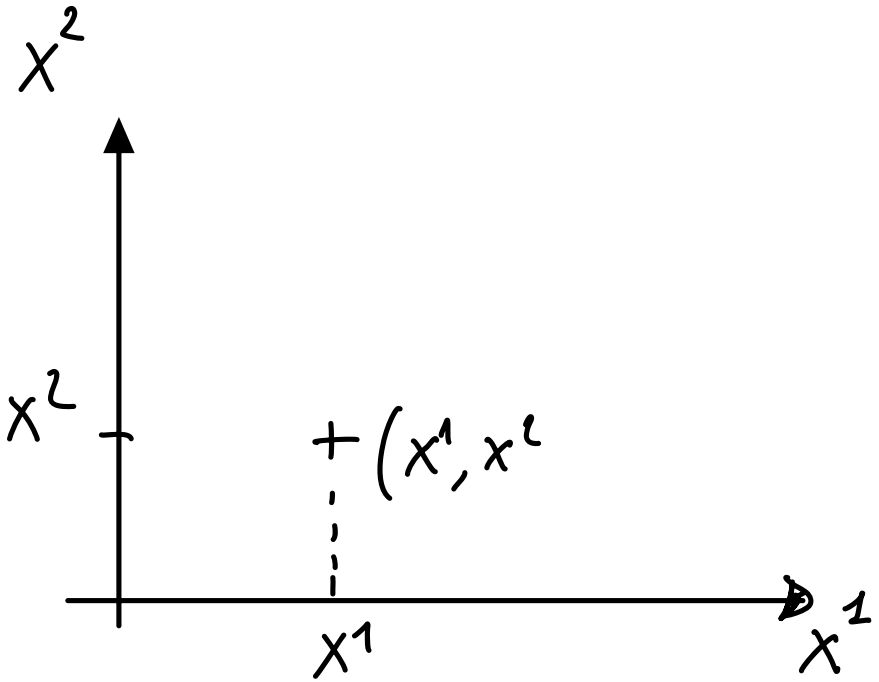
$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-Y_i \; w^T X_i))$$

Let's NN playground: Link



Let's NN playground: Link

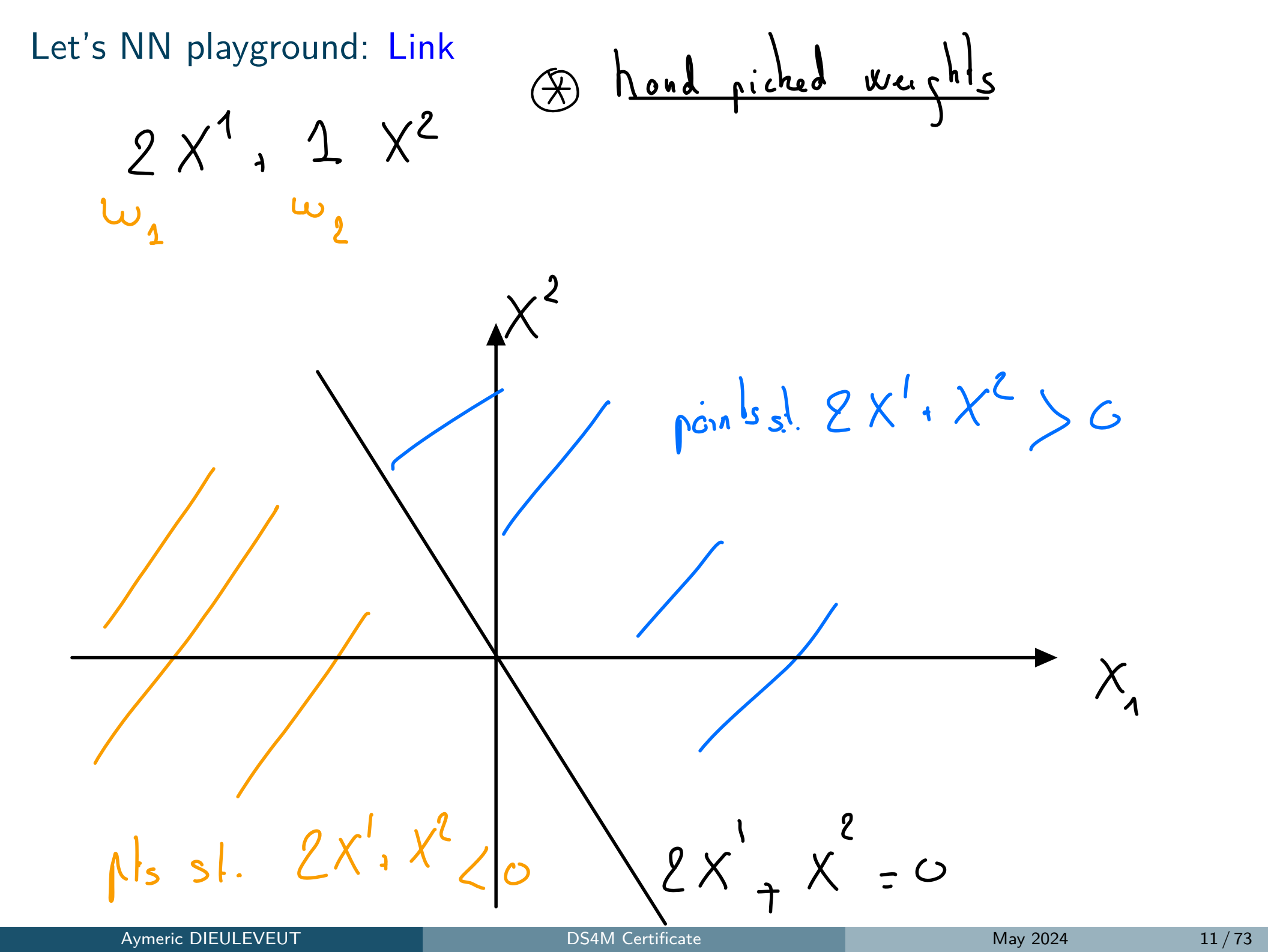




understanding the NNPG:

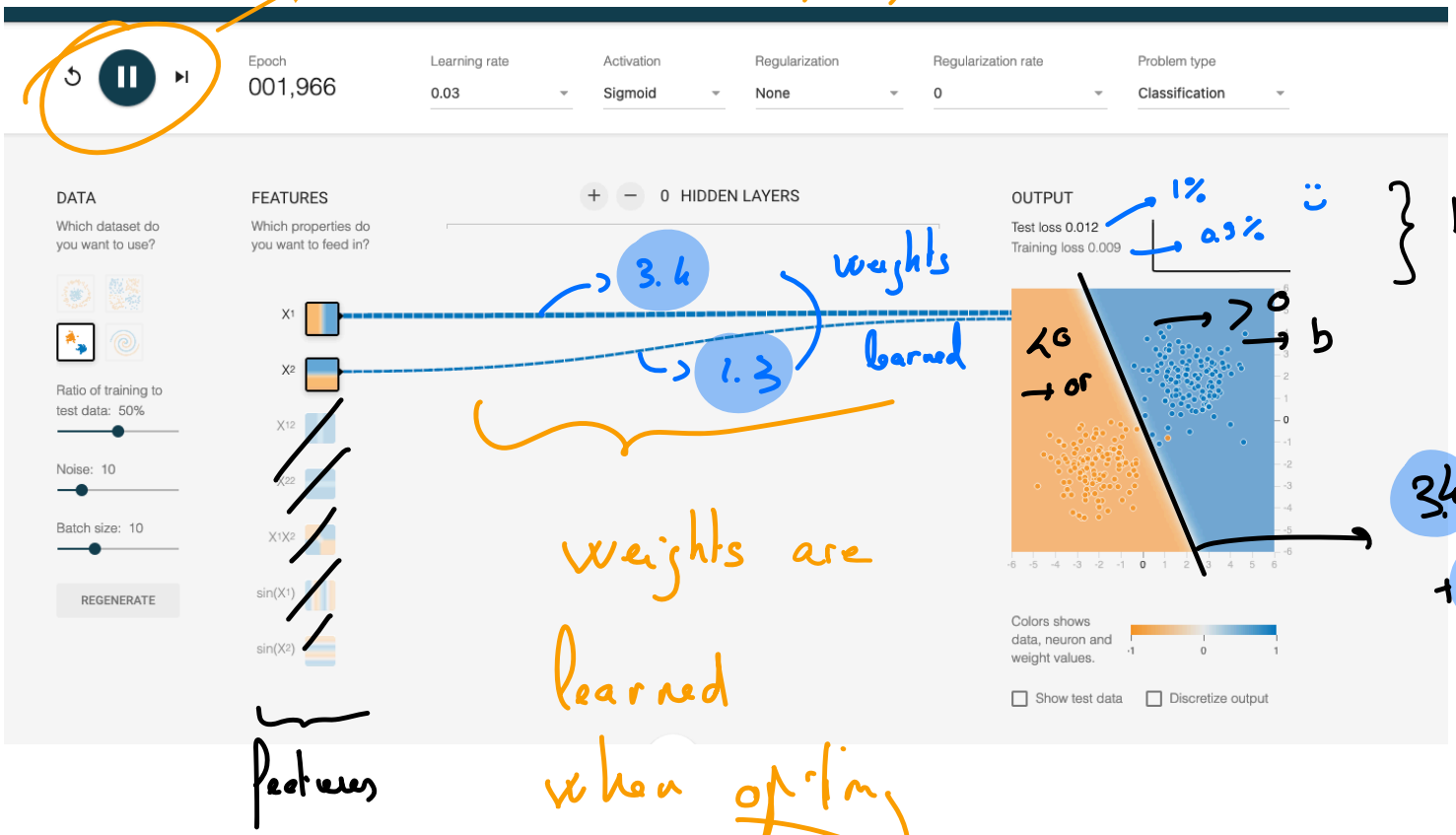
* features
* colors.

Let's NN playground: Link



Let's NN playground: Link

* Let us run the optim process, with 0 hid layer
* this is equivalent to logistic regression

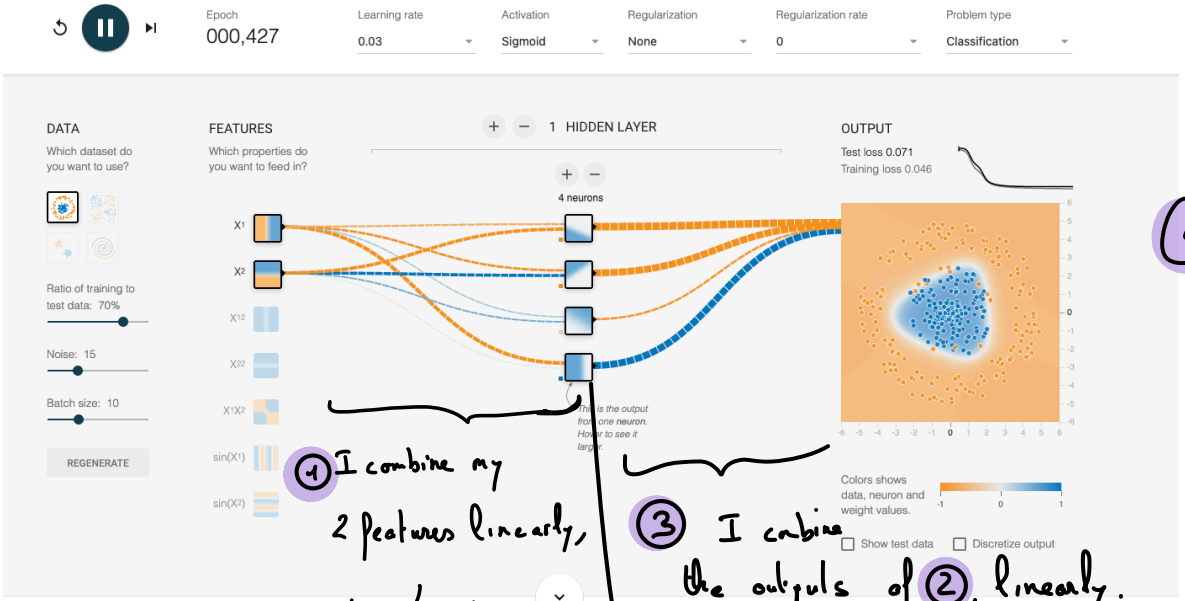


rk:

Logistic regression → stable : 2 ≠ init give the same sol° !!

Let's NN playground: Link

* Let us move to non linearly sep data → Log reg fails
  → So does single neuron hiddlayer
* Let us add more "neurons".



(1) I combine my 2 features linearly, in 4 ≠ manners

(2) I pass that through a non linear activ° (sigmoid)

(3) I combine the outputs of (2) linearly.

(4) Success!!

I learn output weight and the inner weights simultaneously

$-0.6X^1-2X^2$ → $-1X^1+2X^2$

$1.4X^1+2X^2$

→ $-3X^1+0.1X^2$

Take home mess → I have learned a non linear fct°!!

→ I have learned 4 good features!!

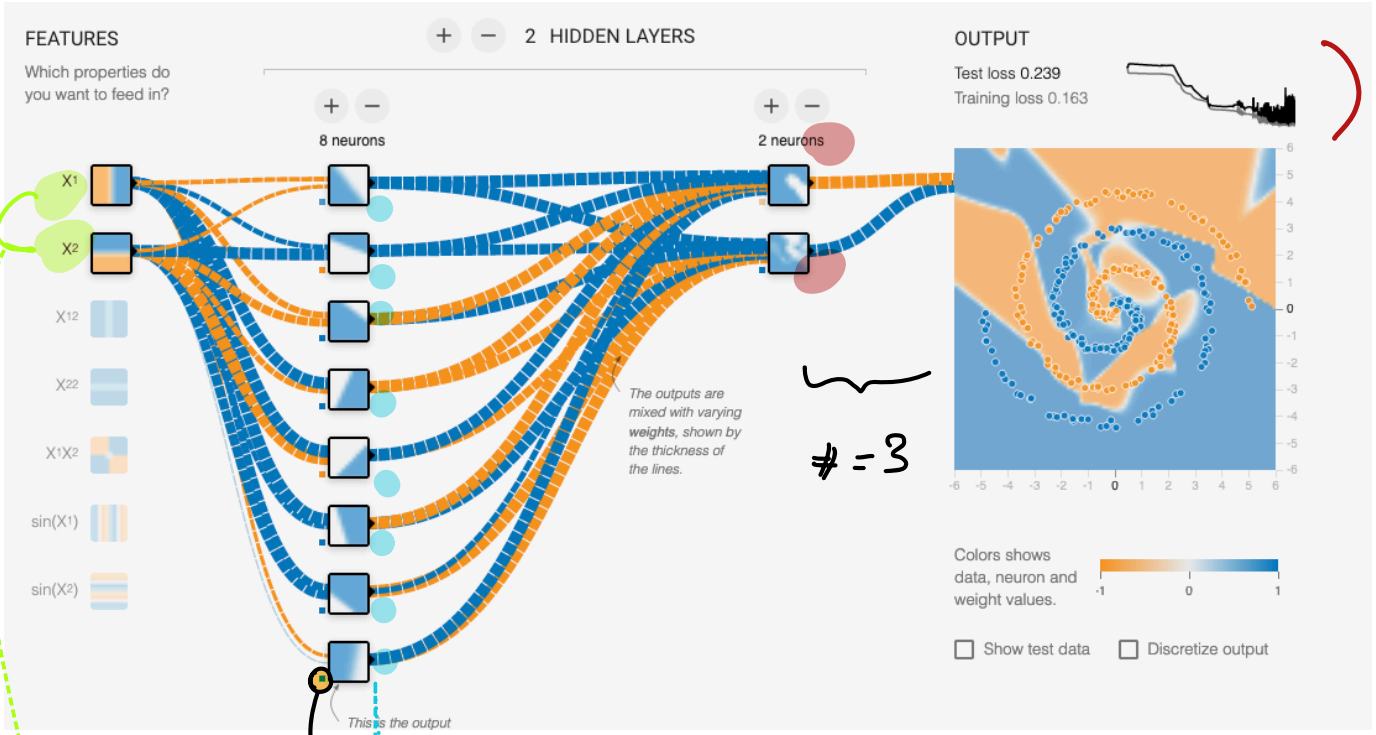
$(\sigma(\ ),\sigma(\ ),\sigma(\ ),\sigma(\ ))$

→ that can be linearly combined to obtain the output!!

Let's NN playground: Link

*Let us move to more advanced data

* Let us try with 8 neurons ? 8N ?
1 L ×2L



→ crazy!! nothing happened for 3 min the drop then occurred.

= 3

x how many learnable parameters do we have ?

bias = offset = b in $w_1 X_1^1 + w_2 X_1^2 + b$

Layer 0-1

Layer 1-2

weights & biases

$\# = (2+1) \times 8$

$\# 18 = (8+1) \times 2$ → 45 parameter overall

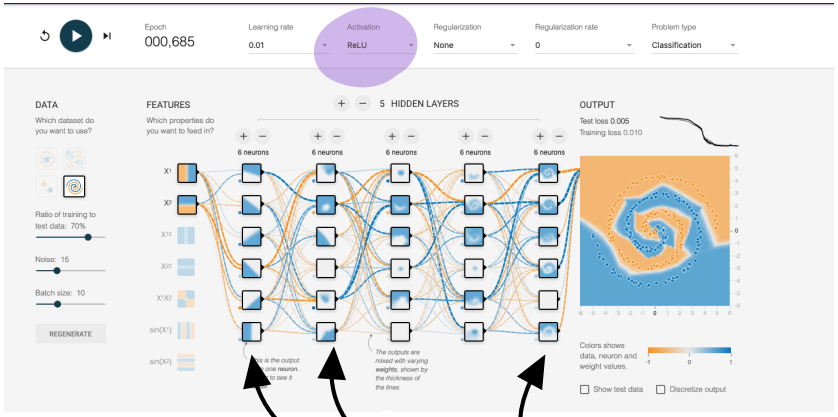
(vs GPT 4 : 10^{11} +)

Let's NN playground: Link

Impact of activation :

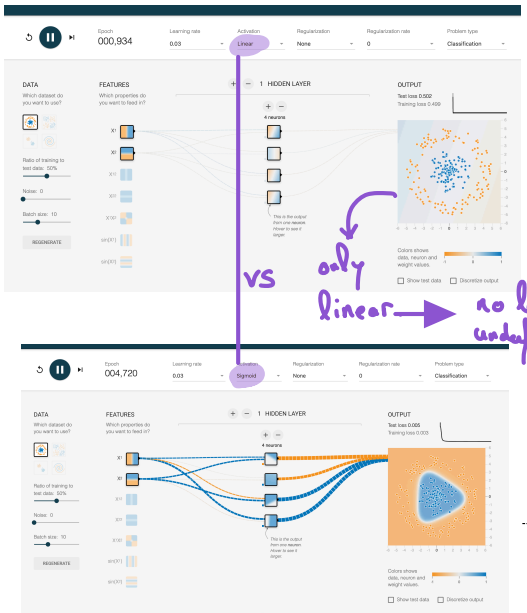
1. linear activ° → no learning of non linear behavior
2. Sigmoid → gradient killer → alg does not progress ☹
3. Tanh → improves on Tanh → overfitting
4. Relu → work amazingly well here

4



♡ note the evolution of features learned on each layer !

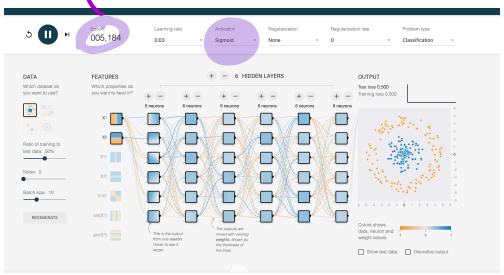
1



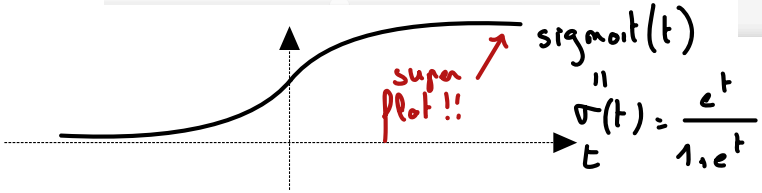
vs

only linear → no learn. underfit. ☹

2

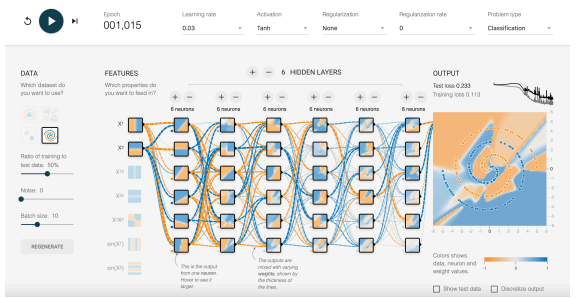


5000 epochs and no learning !! even on simple data !!



3

now, learning happens → but overfit



Let's NN playground: Link

Optimization : * iterative

↳ "million $ question"

* batch : nb of points "looked at" at each iteration

* step size : size of the update at each step

→ very complicated

* ok: stable for logistic (convex function, unique minimum)

unstable otherwise → non convex, many minima!

* very long

↳ for architecture choice → CV not possible → pick standard architecture

Let's NN playground: Link

* Architect : "fully-connected" here

* We have visualized & discussed :

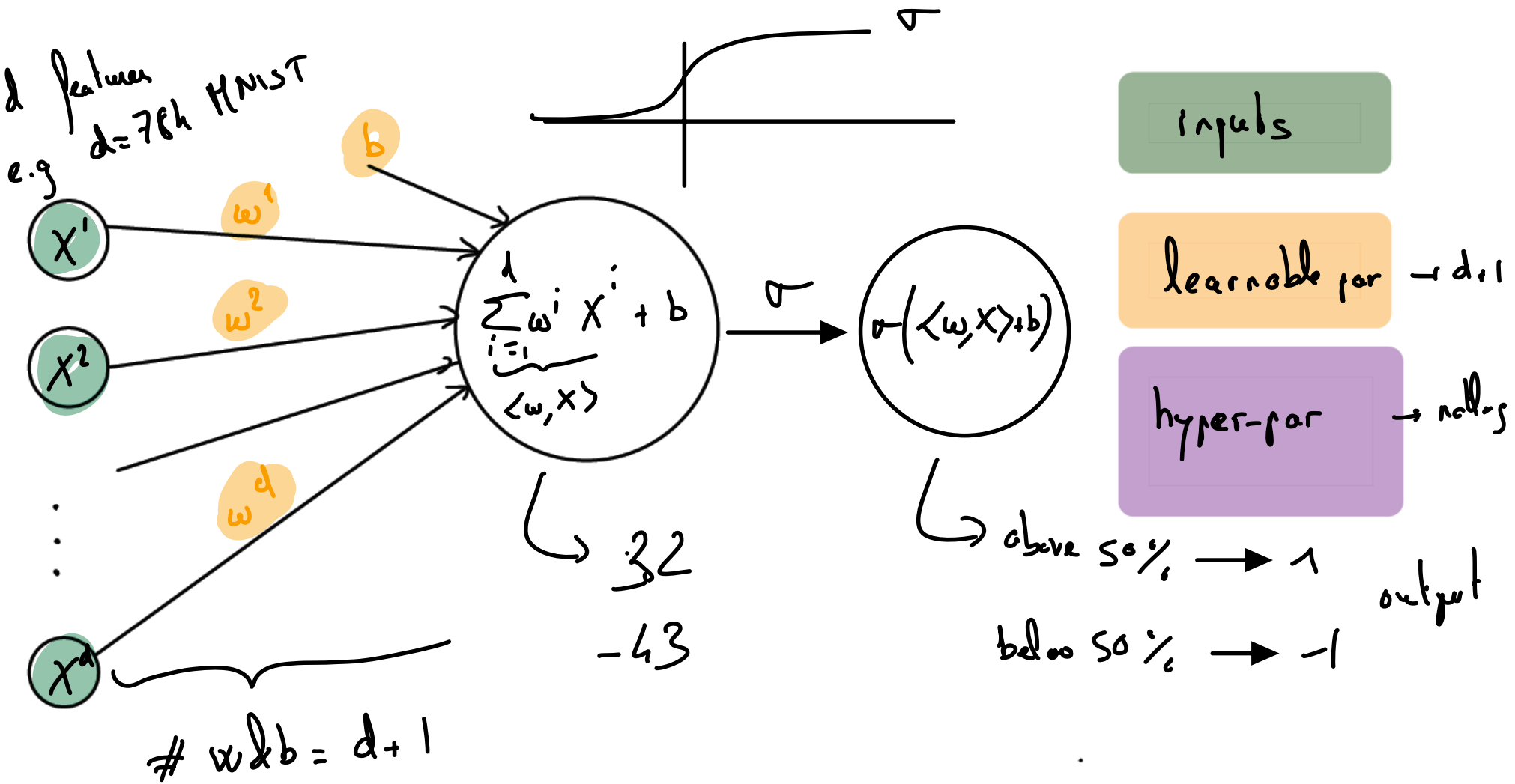
→ Neuron, layer, bias, weights, activd°, optimizat°, learned features, etc

Pro tip: try again the playground in one month - one year

add a task in your calendar not to forget

From logistic regression to Multi Layer Perceptron 3

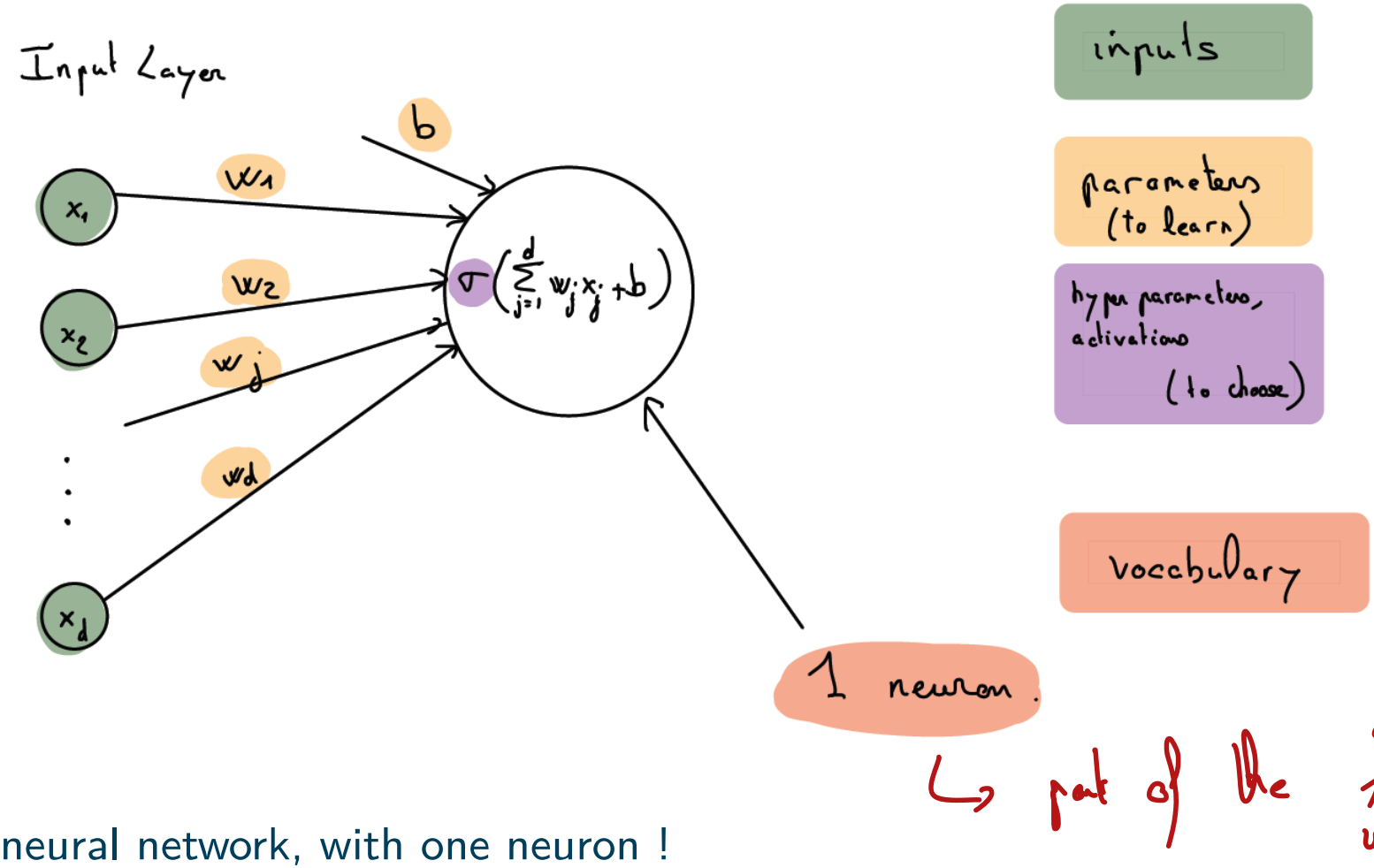
- Class of functions for logistic regression: $\mathscr{F} = \{X \mapsto w^T X,\ w \in \mathbb{R}^d\}$.
- Prediction for a new point is $1_{\sigma(X_i)>1/2}$
- Can be seen as predicting a probability $\sigma(X_i)$ that the output is 1



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

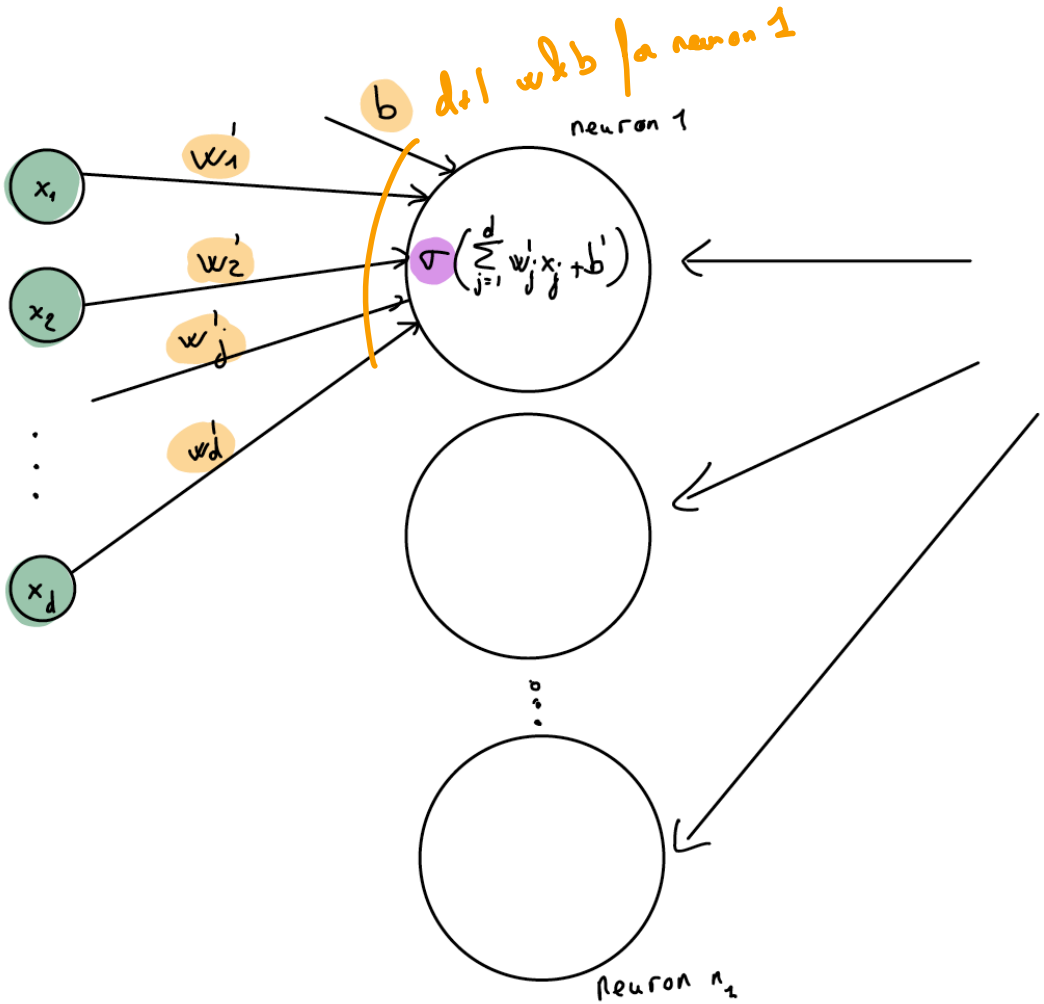


This is a neural network, with one neuron !

From logistic regression to Multi Layer Perceptron 4

Extend to

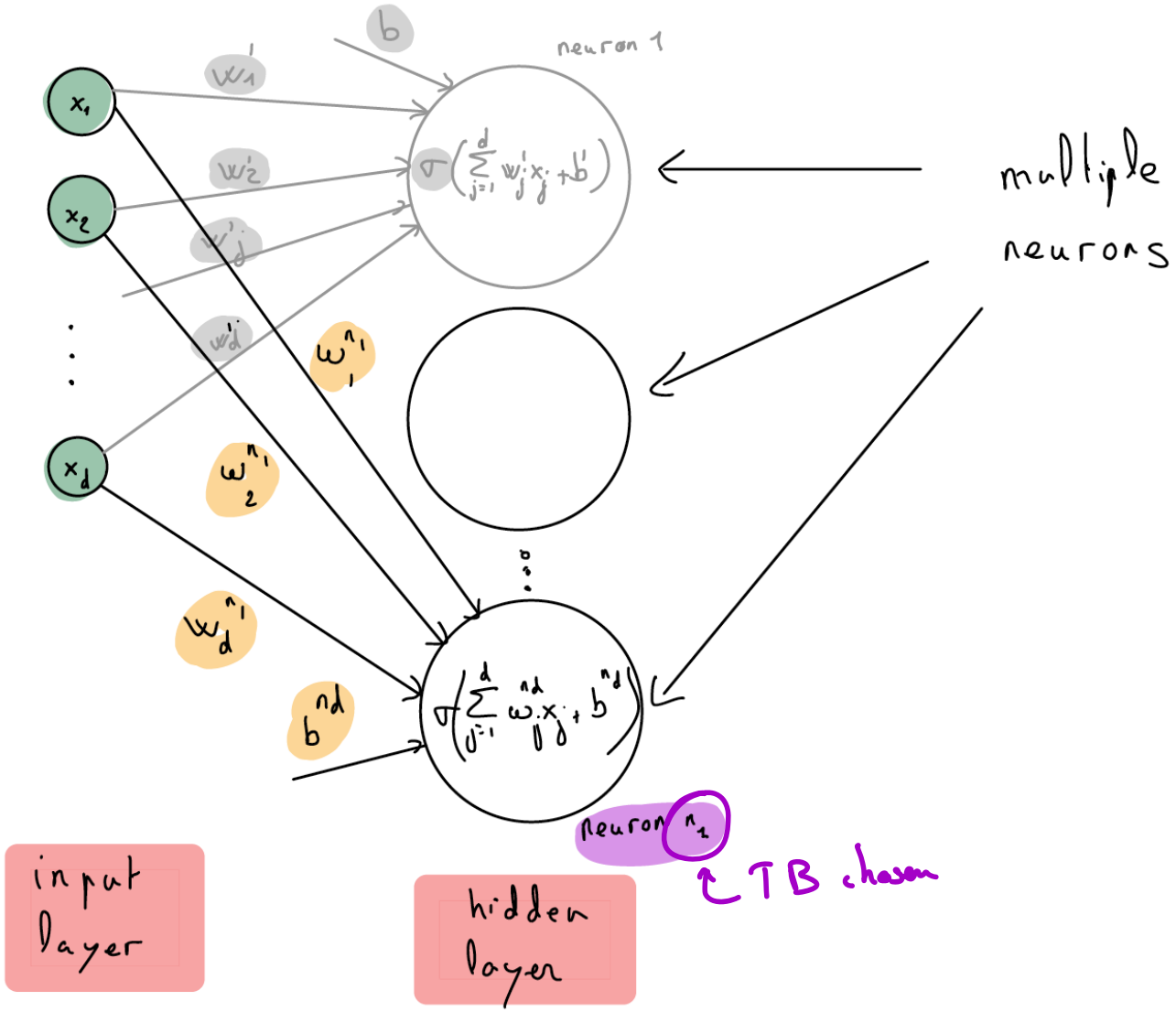
- Multiple Neurons ✓
- Multiple Layers



From logistic regression to Multi Layer Perceptron 4

Extend to

- Multiple Neurons
- Multiple Layers

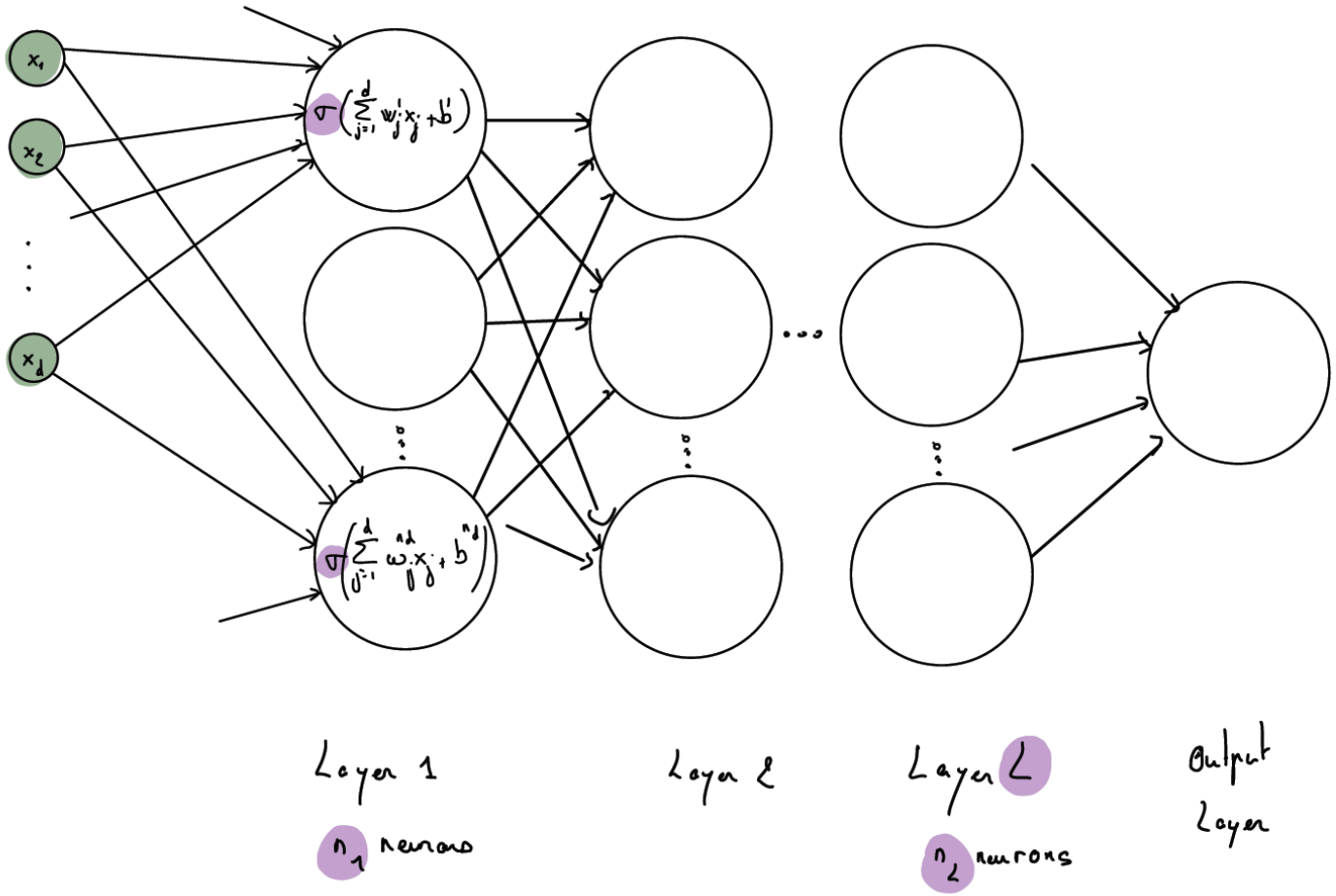


From logistic regression to Multi Layer Perceptron 5

Extend to

- Multiple Neurons
- Multiple Layers

values after non linearity act as inputs for the next layer. and again . .



Again:

learn weights

that create features

that can be combined

in features that can

be combined in features

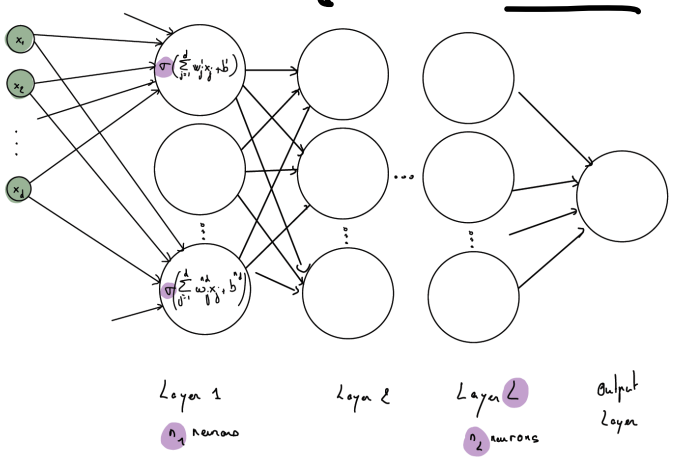
that ... can be combined

in output

From logistic regression to Multi Layer Perceptron 5

Neural network:

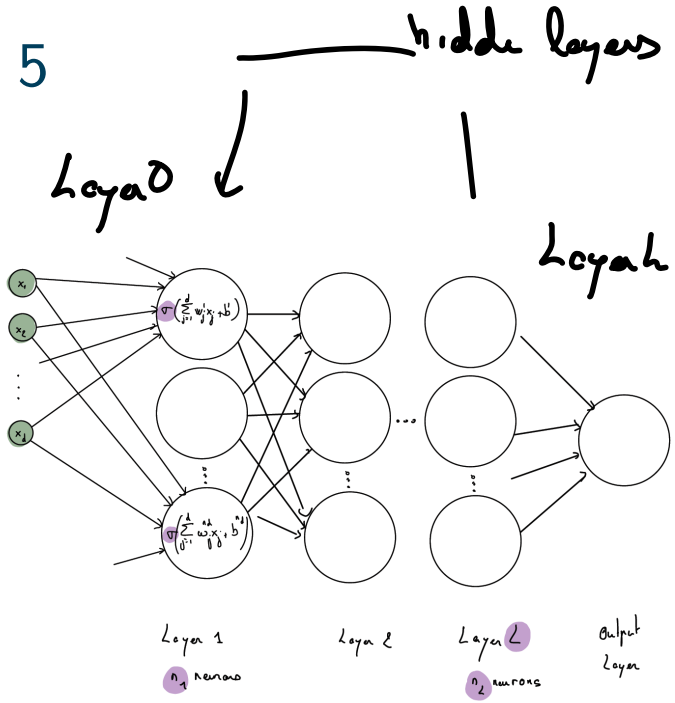
- Goal $\min_{f \in \mathcal{F}_{\text{MLP}}} \frac{1}{n} \sum_{i=1}^{n} [\ell(f(X_i), Y_i)]$
- Class of non linear functions.
  $\mathcal{F}_{\text{MLP}} = \{f(X; W, b) = \sigma(b_n + W_n \sigma(...(b_1 + W_1 x))),$
  $W \in \mathbb{R}^{\cdots}, b \in \mathbb{R}^{\cdots}\}$



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

learnable weights on layer l

$n_0 \times (n_{i+1} + 1)$ on fortable

$(d+1)\, n_1$

What needs to be chosen

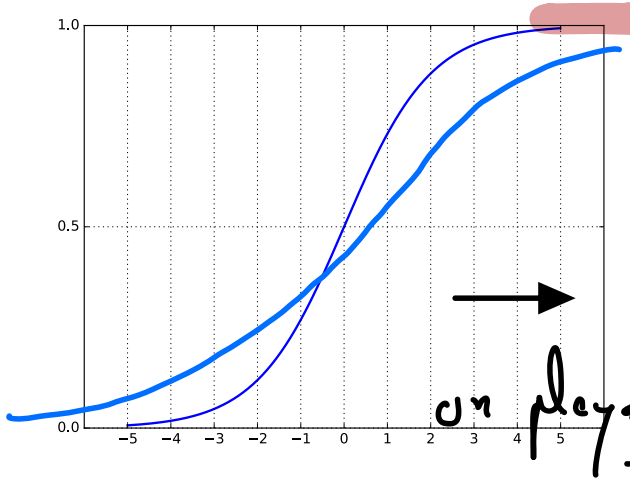
- Activation σ
- Number of layers L
- Number of neurons per hidden layer $n_i, i = 1, \ldots, L$.

Vocabulary

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

2 possible activations: Sigmoid and Rectified Linear Unit

Linear activate



Tanh

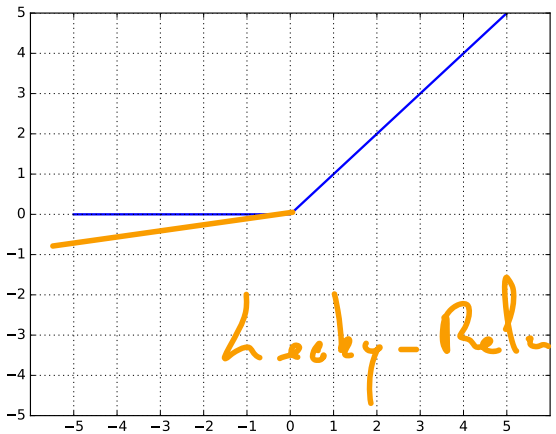
Sigmoid function

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

almost exactly one

no learning

on playground with 6 layers → algo does not progress



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

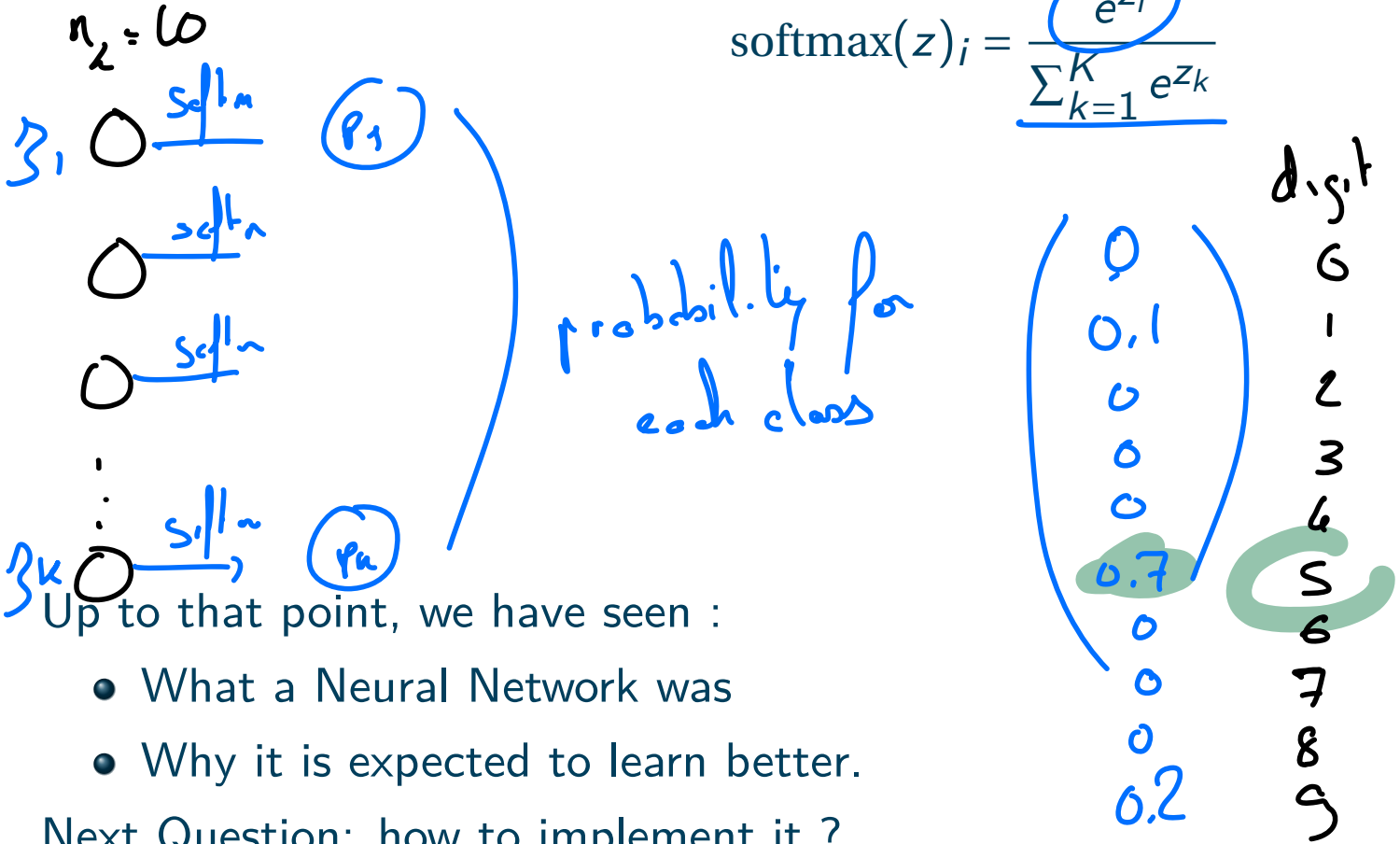
How to deal with multi-class output: softmax activation

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 $\{1, \ldots, K\}$:

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



Last layer:

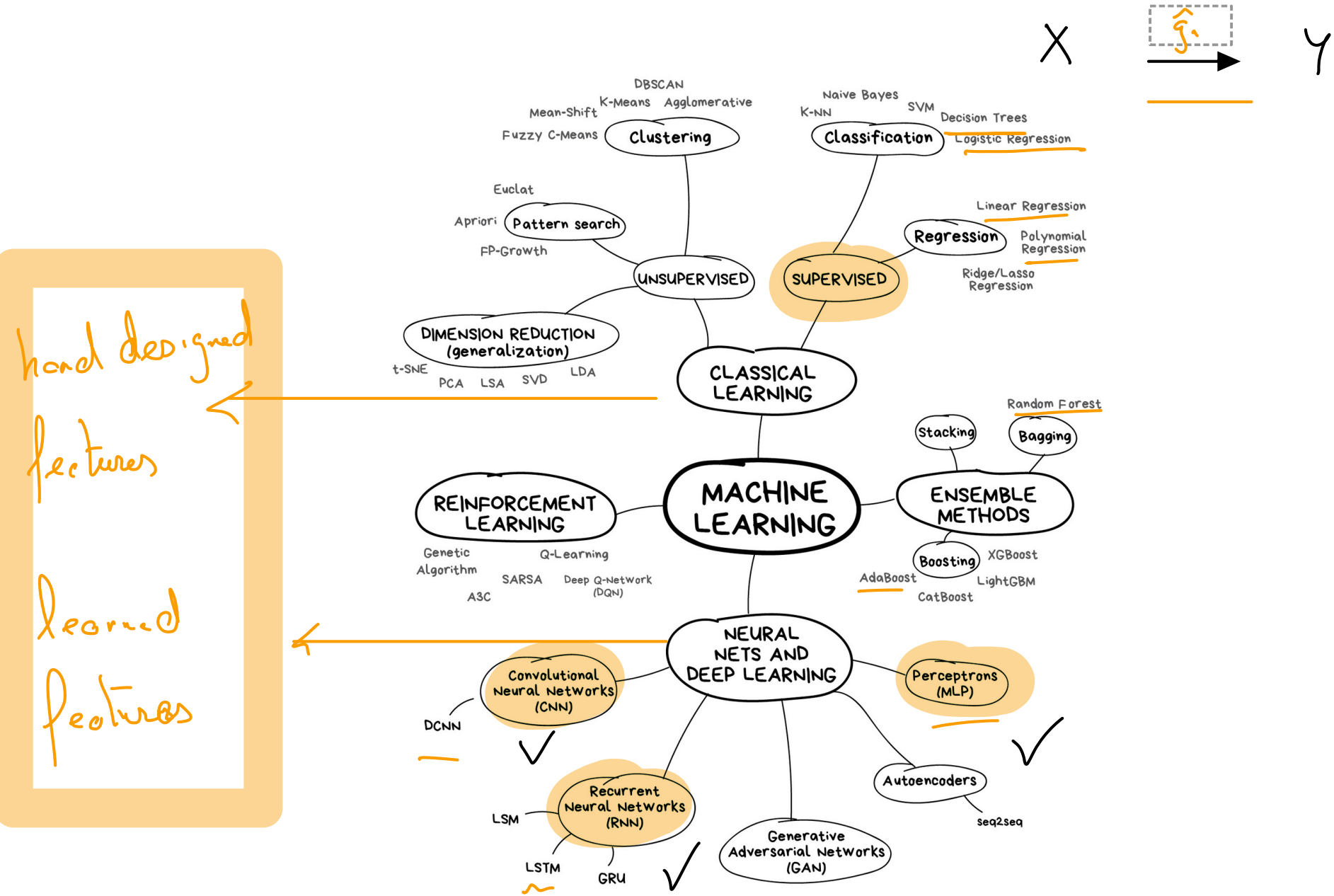
- in regression, activation = Id
- no activation
- in binary classif: sigmoid. → gives you a probability

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 ?

Machine Learning



Python - Keras

1. Hardware:
   - CPU
   - GPU x 10 times faster
   - 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?

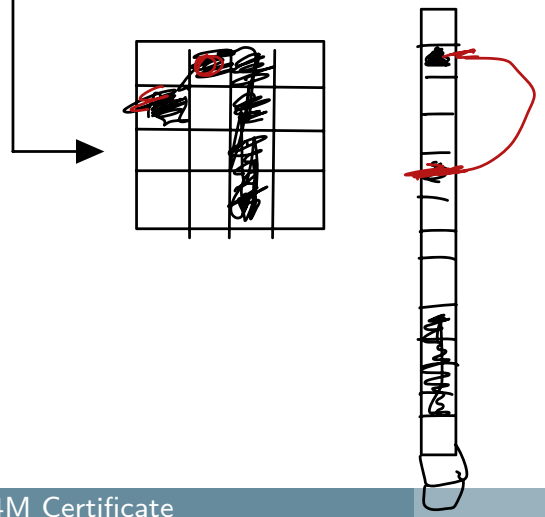
- architect (# layer, # neurons per h)
- activate
  - algo for optimizer, loss

What do we need to do next?

fit !

predict!

```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(10, activation=tf.nn.softmax)
])

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

# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)

# make predictions
predictions = model.predict(test_images)
```

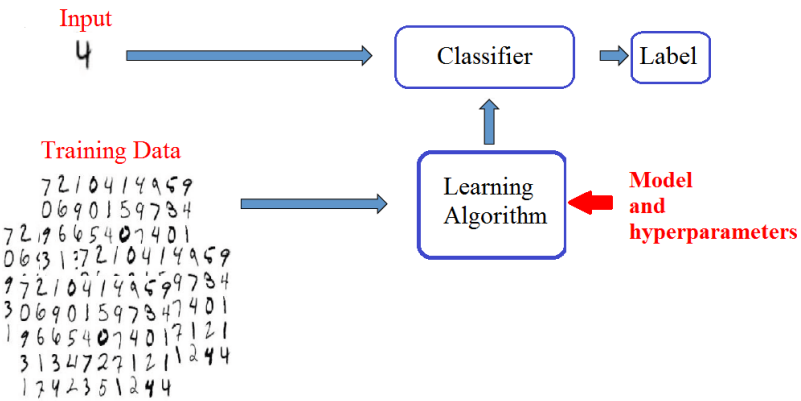


Keras - A few lines ![1]

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(10, activation=tf.nn.softmax)
])

model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# train model
model.fit(train_images, train_labels, epochs=5)

# evaluate
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test accuracy:', test_acc)

# make predictions
predictions = model.predict(test_images)

```



```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)
# 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 - loss: 1.5450 - accuracy: 0.8669
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2851 - accuracy: 0.9290
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1913 - accuracy: 0.9474
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1578 - accuracy: 0.9548
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1373 - accuracy: 0.9607
313/313 [==============================] - 1s 2ms/step - loss: 0.1607 - accuracy: 0.9586
test accuracy: 0.9585999846458435
313/313 [==============================] - 1s 1ms/step
```

randomness in training due to adj.

→ Final accuracy

784

```python
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
_________________________________________________________________
```

128 neurons HL # 1

$128 \times (784 + 1)$

$64 \times (128 + 1)$

$10 \times (64 + 1)$

```python
785*128
```

```
100480
```

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

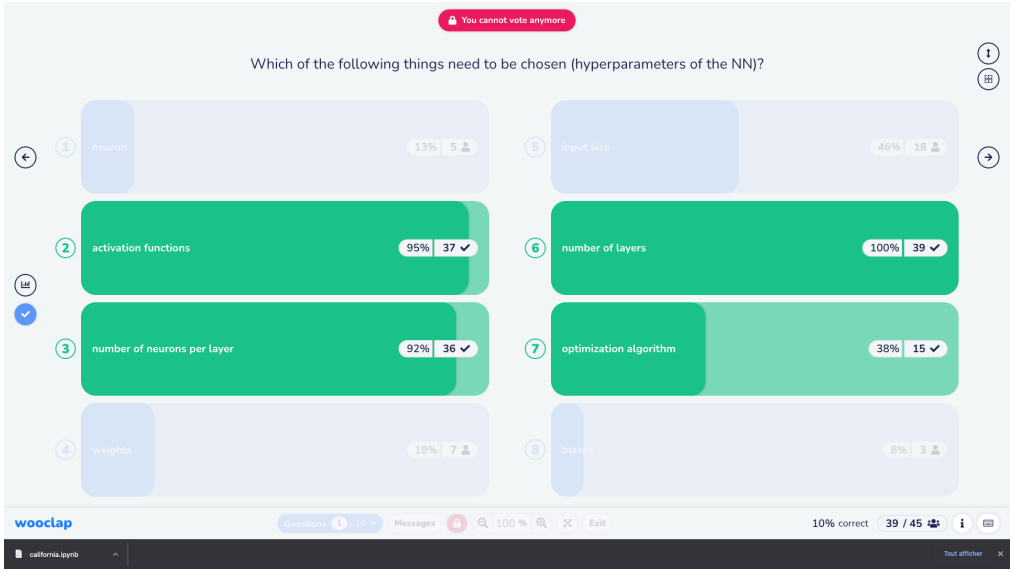
Up to that point, we have seen :

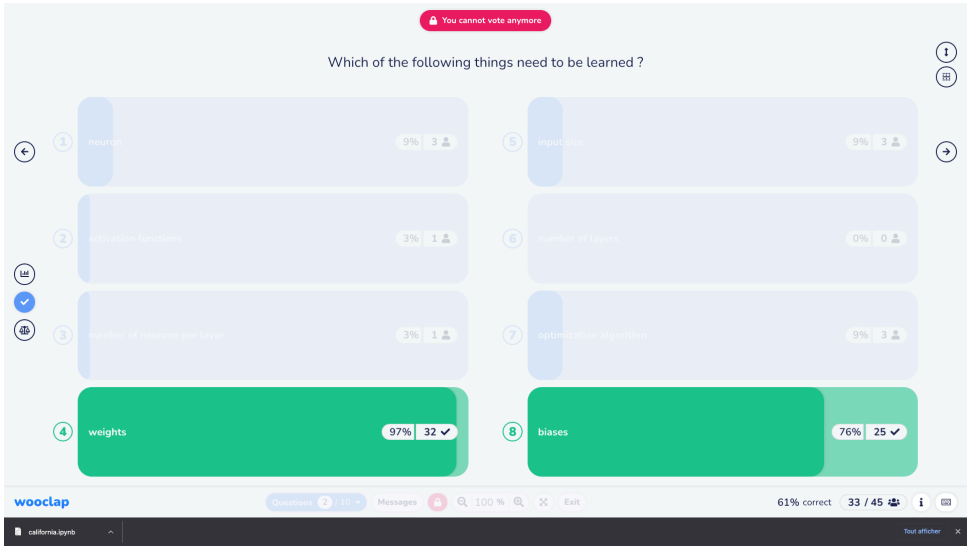
- What a Neural Network was.
- How to implement it in Python.

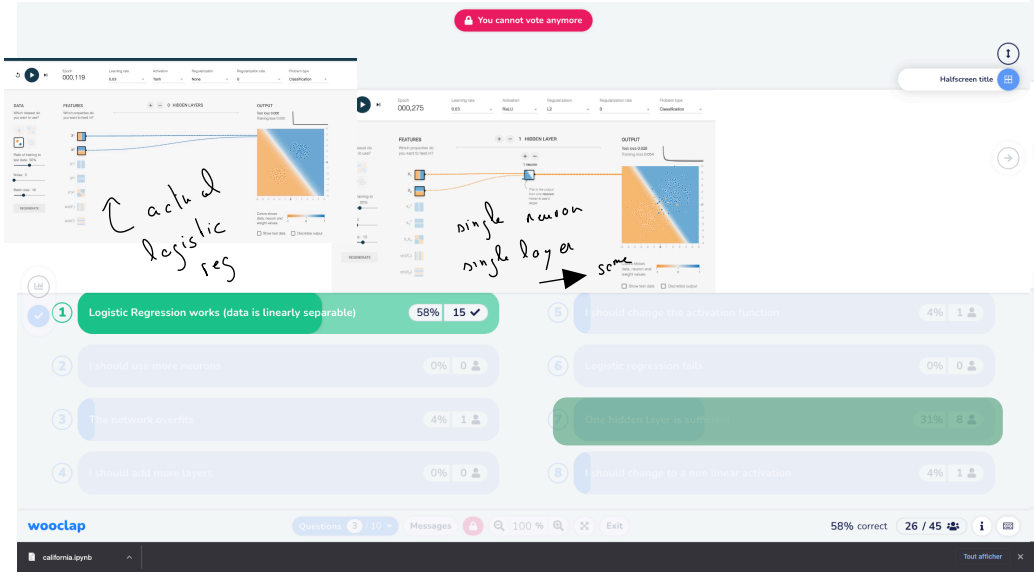
Next Question: why and when does it work?

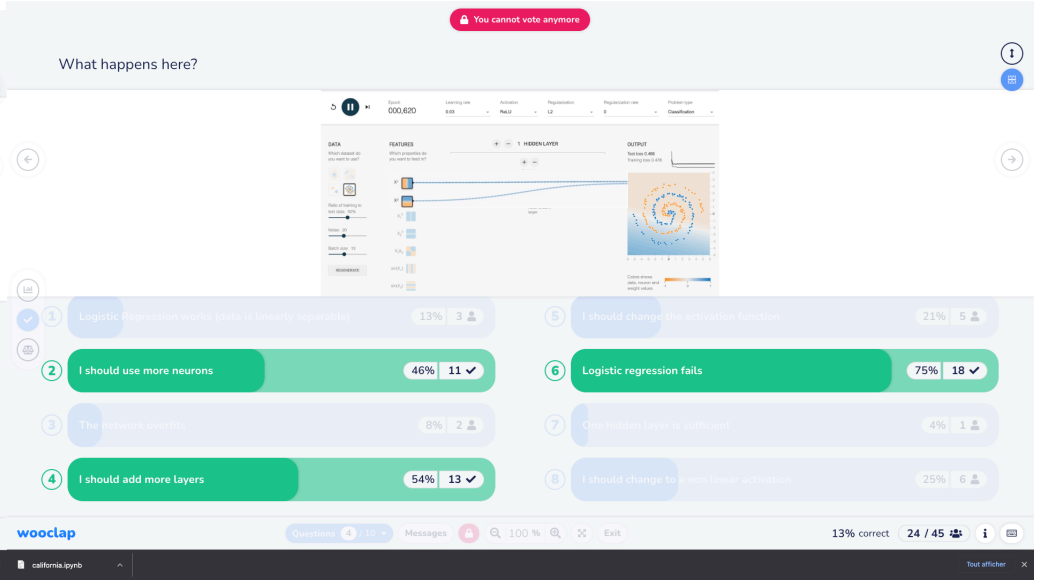
1. Approximation theorem
2. Optimization

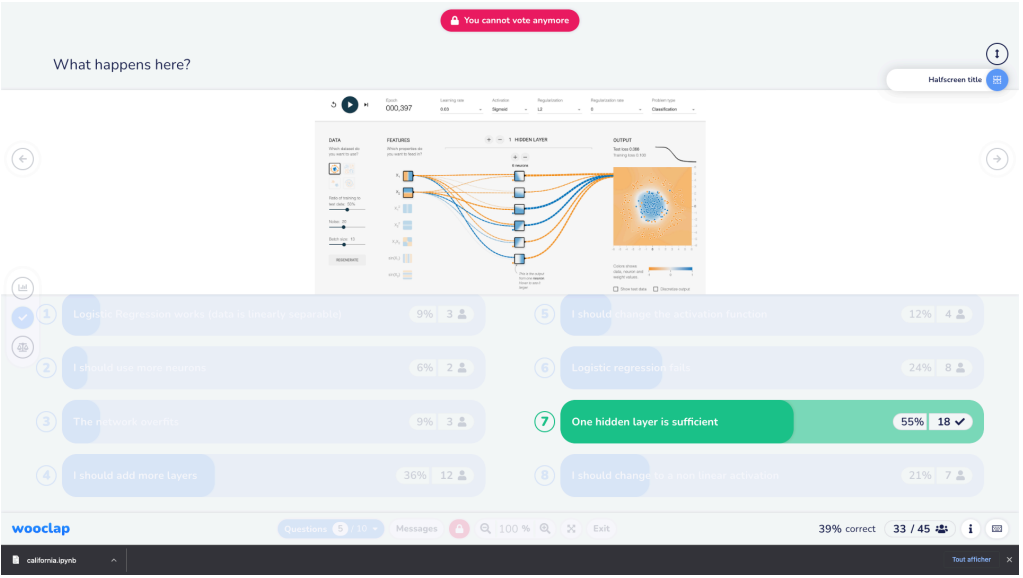
Wooclap

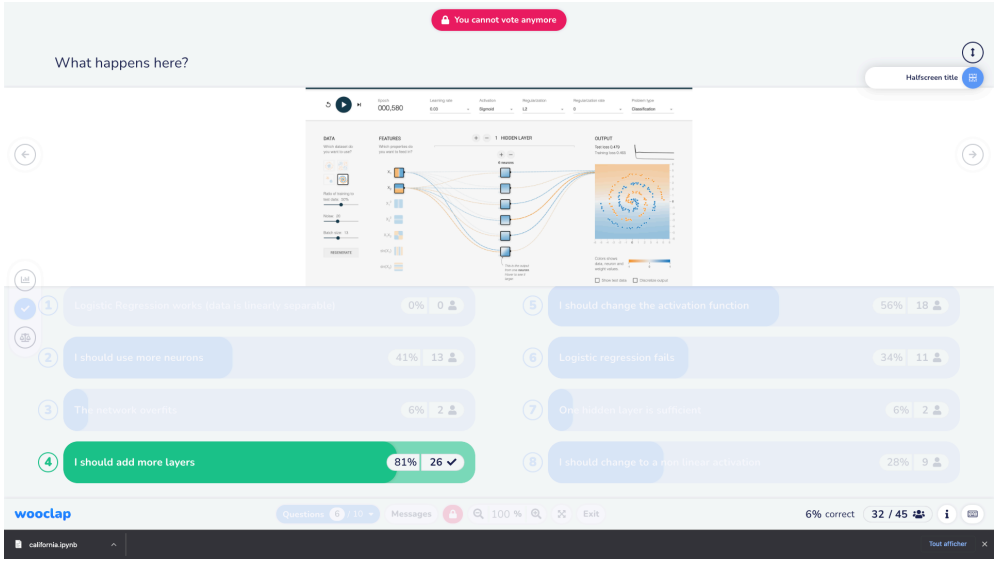

You cannot vote anymore
Which of the following things need to be chosen (hyperparameters of the NN)?
1 neuron 13% 5
2 activation functions 95% 37
3 number of neurons per layer 92% 36
4 weights 18% 7
5 input size 46% 18
6 number of layers 100% 39
7 optimization algorithm 38% 15
8 biases 8% 3
wooclap
10% correct 39 / 45

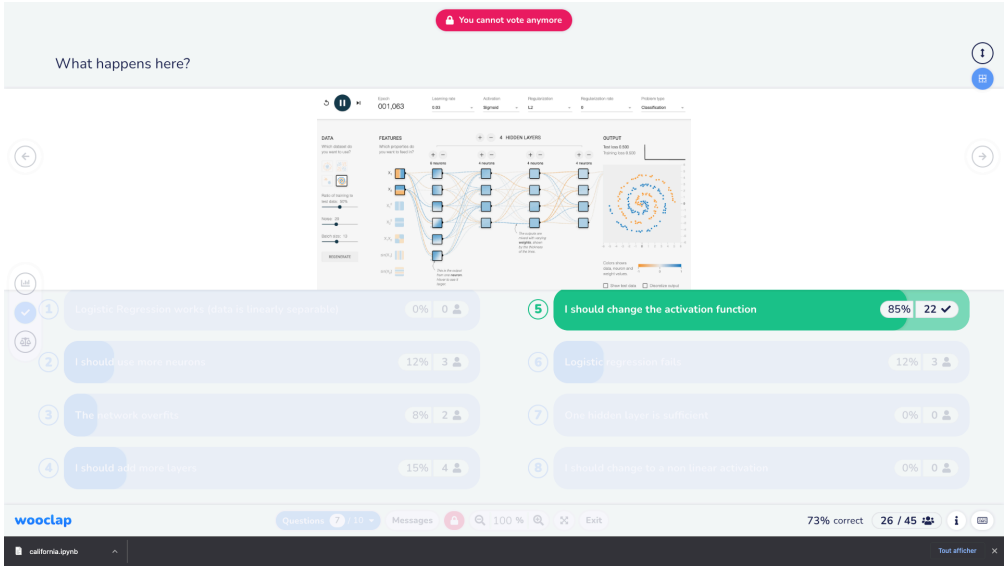

You cannot vote anymore
Which of the following things need to be learned ?
1 neuron 9% 3
2 activation functions 3% 1
3 number of neurons per layer 3% 1
4 weights 97% 32
5 input size 9% 3
6 number of layers 0% 0
7 optimization algorithm 9% 3
8 biases 76% 25
wooclap
61% correct 33 / 45

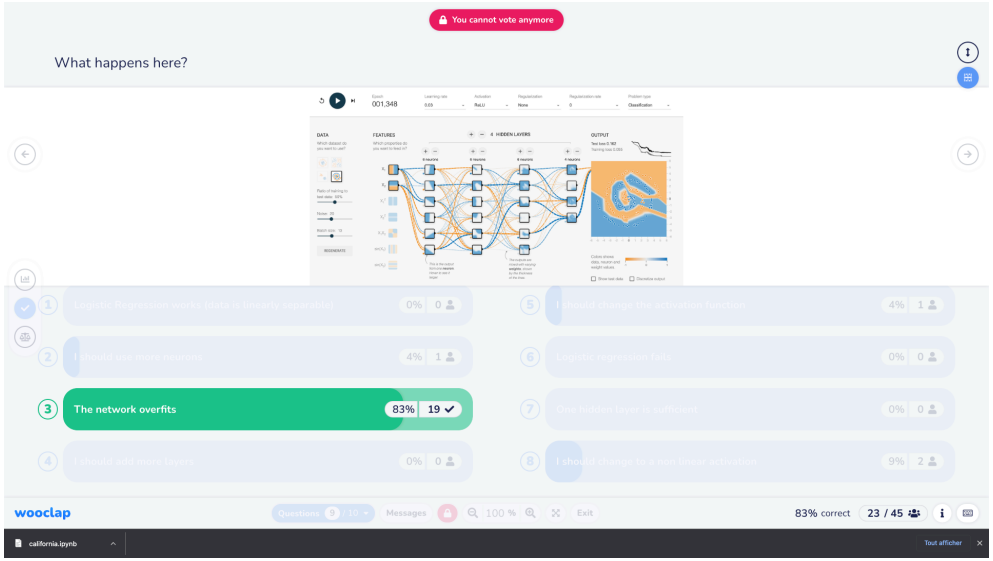

You cannot vote anymore
Halfscreen title
actual logistic reg
single neuron single layer
1 Logistic Regression works (data is linearly separable) 58% 15
wooclap
58% correct 26 / 45

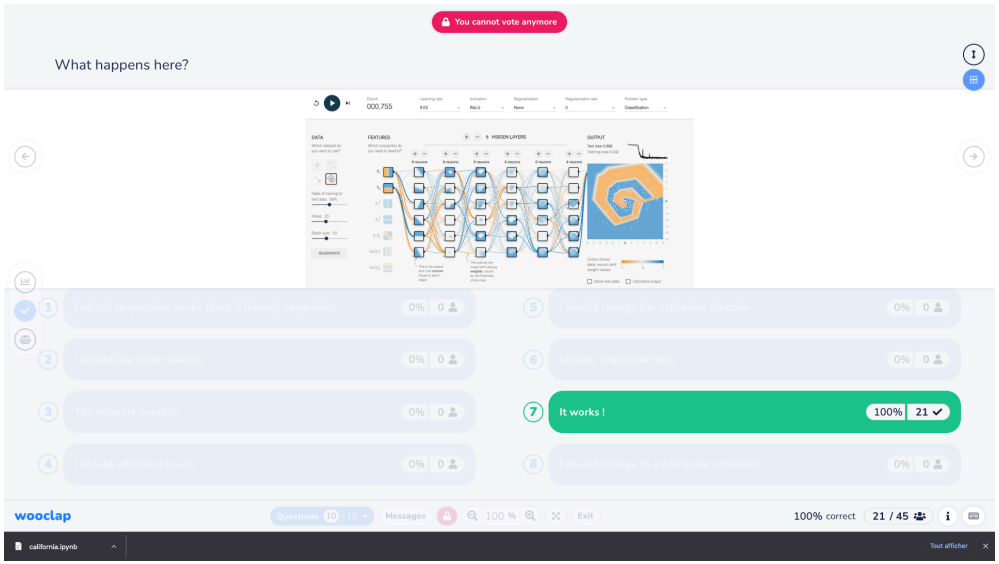

You cannot vote anymore
What happens here?
2 I should use more neurons 46% 11
4 I should add more layers 54% 13
6 Logistic regression fails 75% 18
wooclap
13% correct 24 / 45


You cannot vote anymore
What happens here?
Halfscreen title
1 Logistic Regression works (data is linearly separable) 9% 3
2 I should use more neurons 6% 2
3 The network overfits 9% 3
4 I should add more layers 36% 12
5 I should change the activation function 12% 4
6 Logistic regression fails 24% 8
7 One hidden layer is sufficient 55% 18
8 I should change to a non linear activation 21% 7
wooclap
39% correct
33 / 45
california.ipynb
Tout afficher


You cannot vote anymore
What happens here?
Halfscreen title
1 Logistic Regression works (data is linearly separable) 0% 0
2 I should use more neurons 41% 13
3 The network overfits 6% 2
4 I should add more layers 81% 26
5 I should change the activation function 56% 18
6 Logistic regression fails 34% 11
7 One hidden layer is sufficient 6% 2
8 I should change to a non linear activation 28% 9
wooclap
6% correct
32 / 45
california.ipynb
Tout afficher


You cannot vote anymore
What happens here?
5 I should change the activation function 85% 22
wooclap
73% correct 26 / 45


You cannot vote anymore
What happens here?
3 The network overfits 83% 19
wooclap
83% correct 23 / 45


You cannot vote anymore
What happens here?
7 It works ! 100% 21
wooclap
100% correct 21 / 45

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

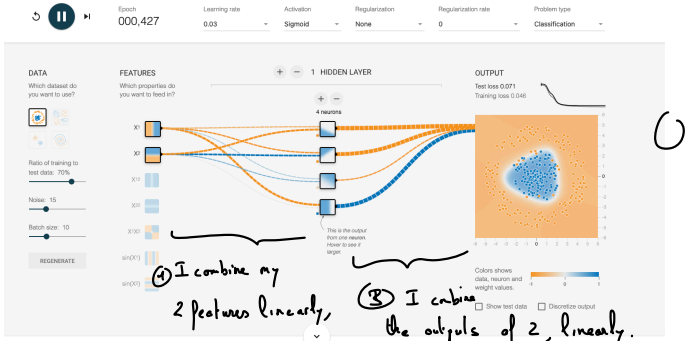
Summary is

is uni. Neural !!

Goal

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

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



0

except 1

h neurons 1 Layer "approx. ok"

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.

↝ nb of neurons if $L=1$ can be exponentially huge

much smaller if $L \gg 1$

↝ Very powerful in terms of approximations.

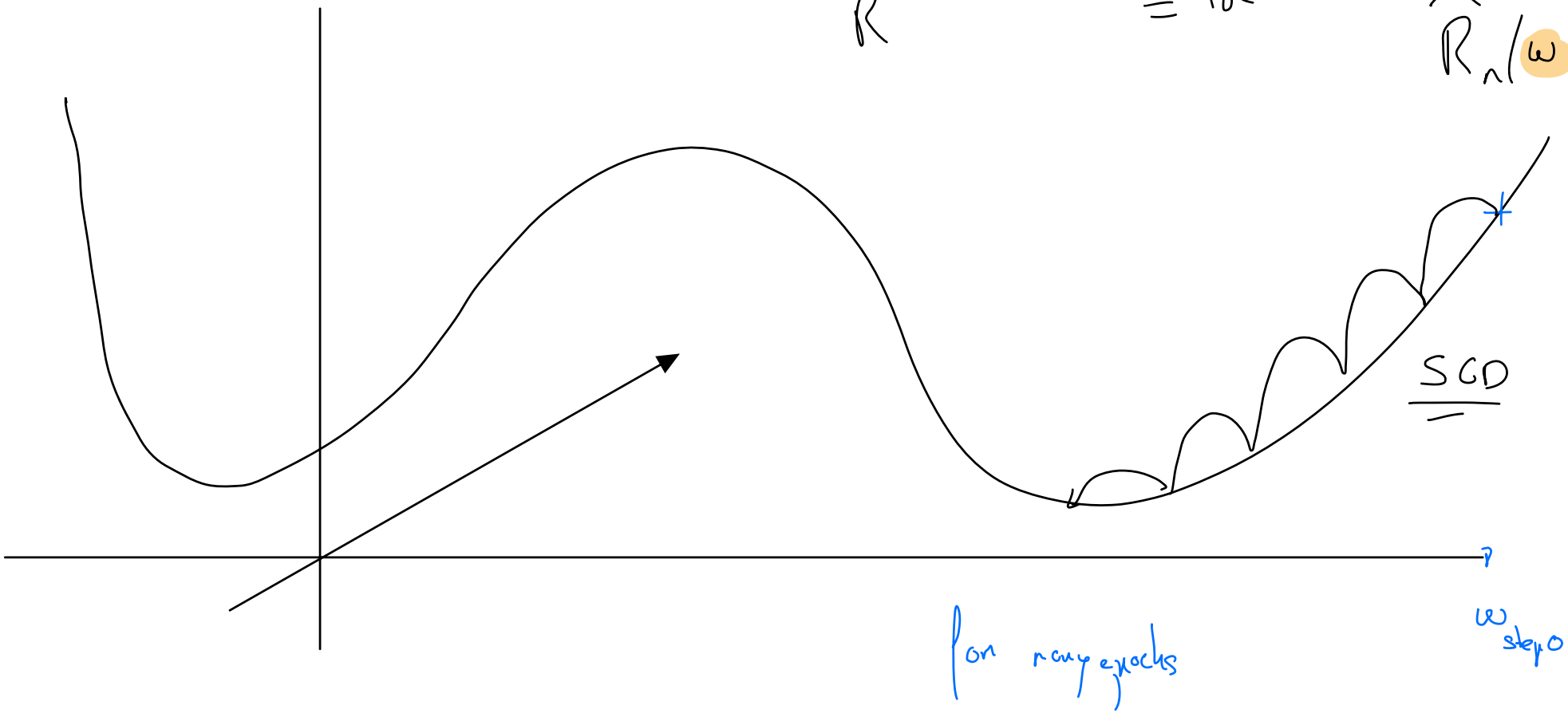
Not very surprising: non linear function with millions of parameters!

Optimization - Fitting the network

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

How to minimize a function?

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



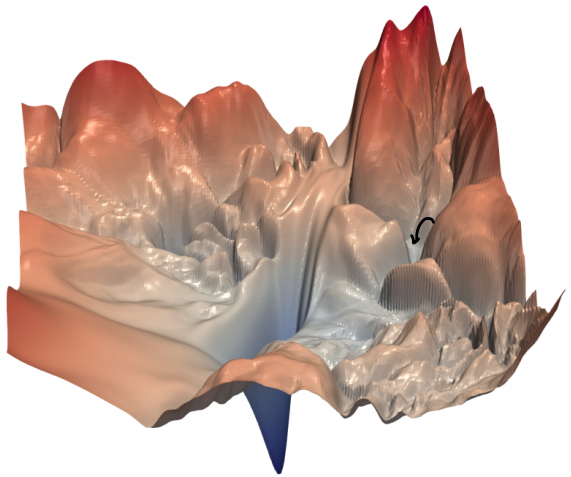
Optimization - Fitting the network

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

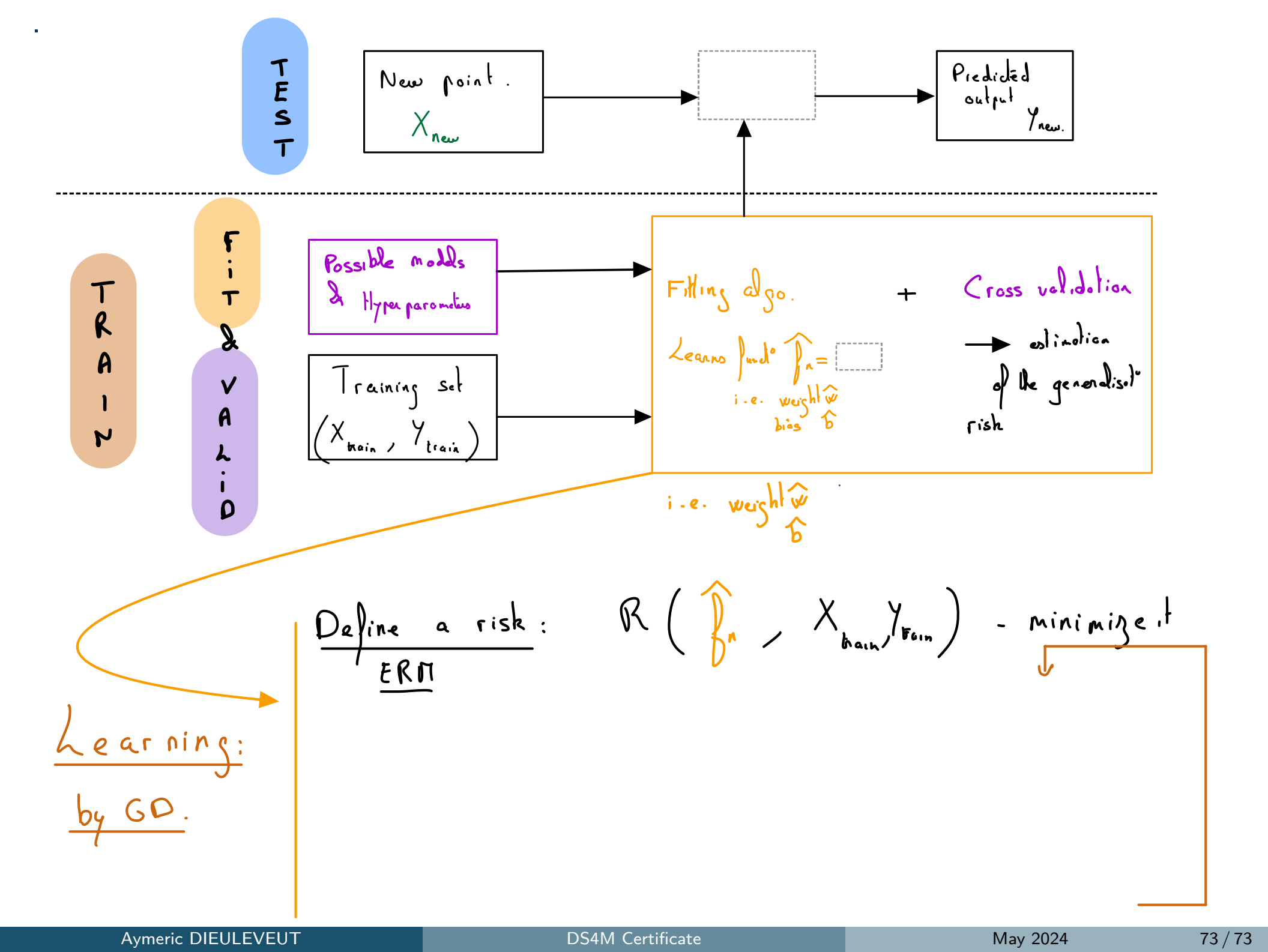
Gradient Methods. = cheapest way to update (learn) the weights iteratively to minimize the loss.

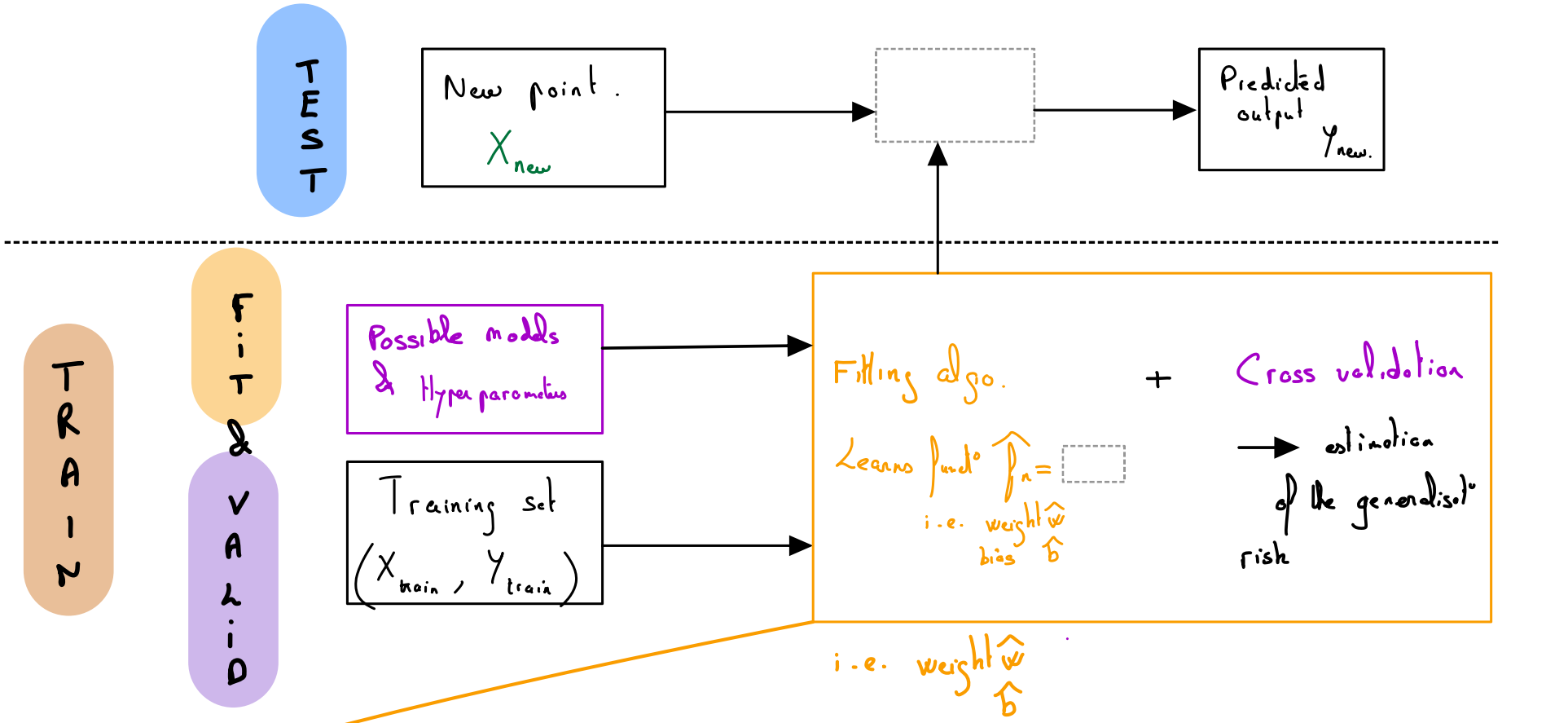
Two "miracles":

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 in 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


TEST
New point.
X_{new}
Predicted output
Y_{new}.
TRAIN
FIT & VALID
Possible models & Hyperparameters
Training set
(X_{train}, Y_{train})
Fitting algo.
+
Cross validation
Learns fonct° $\hat{f}_n =$
i.e. weight $\hat{w}$ bias $\hat{b}$
→ estimation of the generalisat° risk
i.e. weight $\hat{w}$ $\hat{b}$
Define a risk:
ERM
$R(\hat{f}_n, X_{train}, Y_{train})$ - minimize it
Learning:
by GD.



Learning: by GD.

Define a risk: ERM

$R\left(\hat{f}_n, X_{train}, y_{train}\right)$ - minimize it

Iteratively improve $\hat{f}_n$ within the class of allowed fct°.

take a training point x^p

compute $\hat{f}_n(x^p) = \hat{y}^p$

$x^p \rightarrow \square \rightarrow \hat{y}^p$

compare to y^p

improve the weights

$\times 10^6$.

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).

Outline

CNN

We consider an image classification task: we want to recognize which object is on an image.

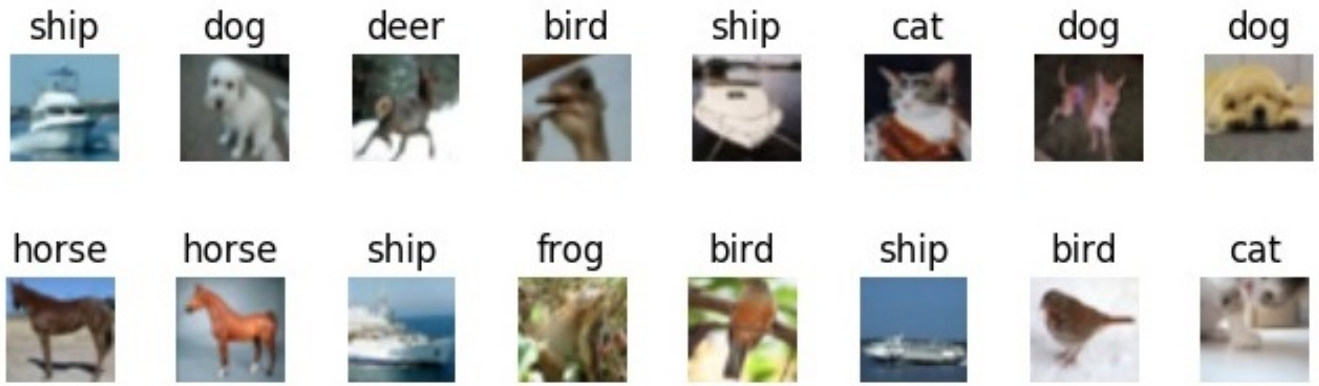


Figure: CIFAR dataset

The input is an image: about 10^3 to 10^7 pixels: these are our inputs.
Problems: The multi-layer perceptron:

- Does not take into account local information. It would work similarly on any permutation of the pixels!
- Has, just on the first layer, as many parameters as there are inputs!

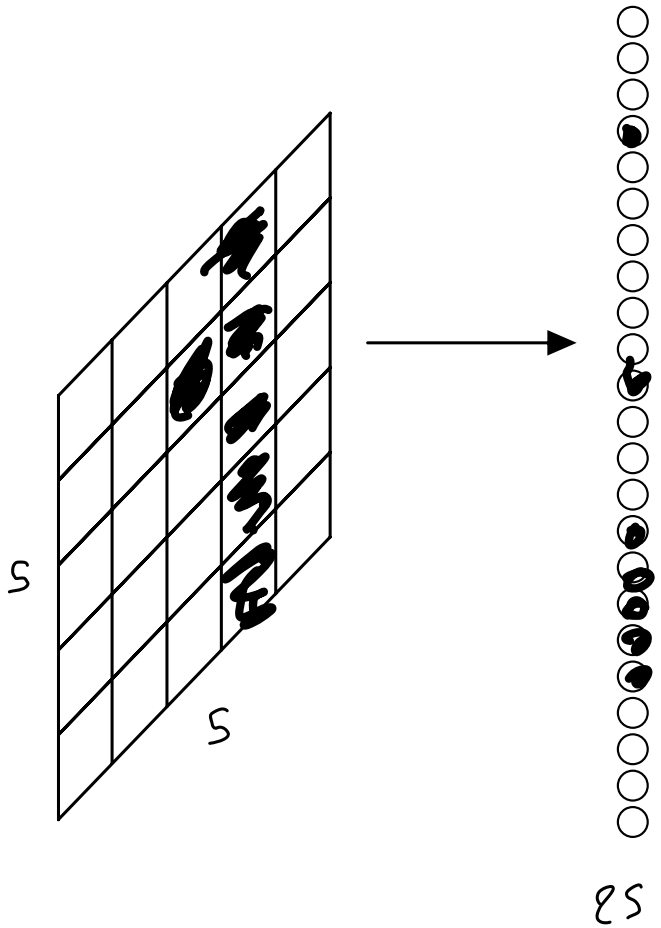
x nb of units hidden

Convolutional Layer

Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5...

Convolutional Layer

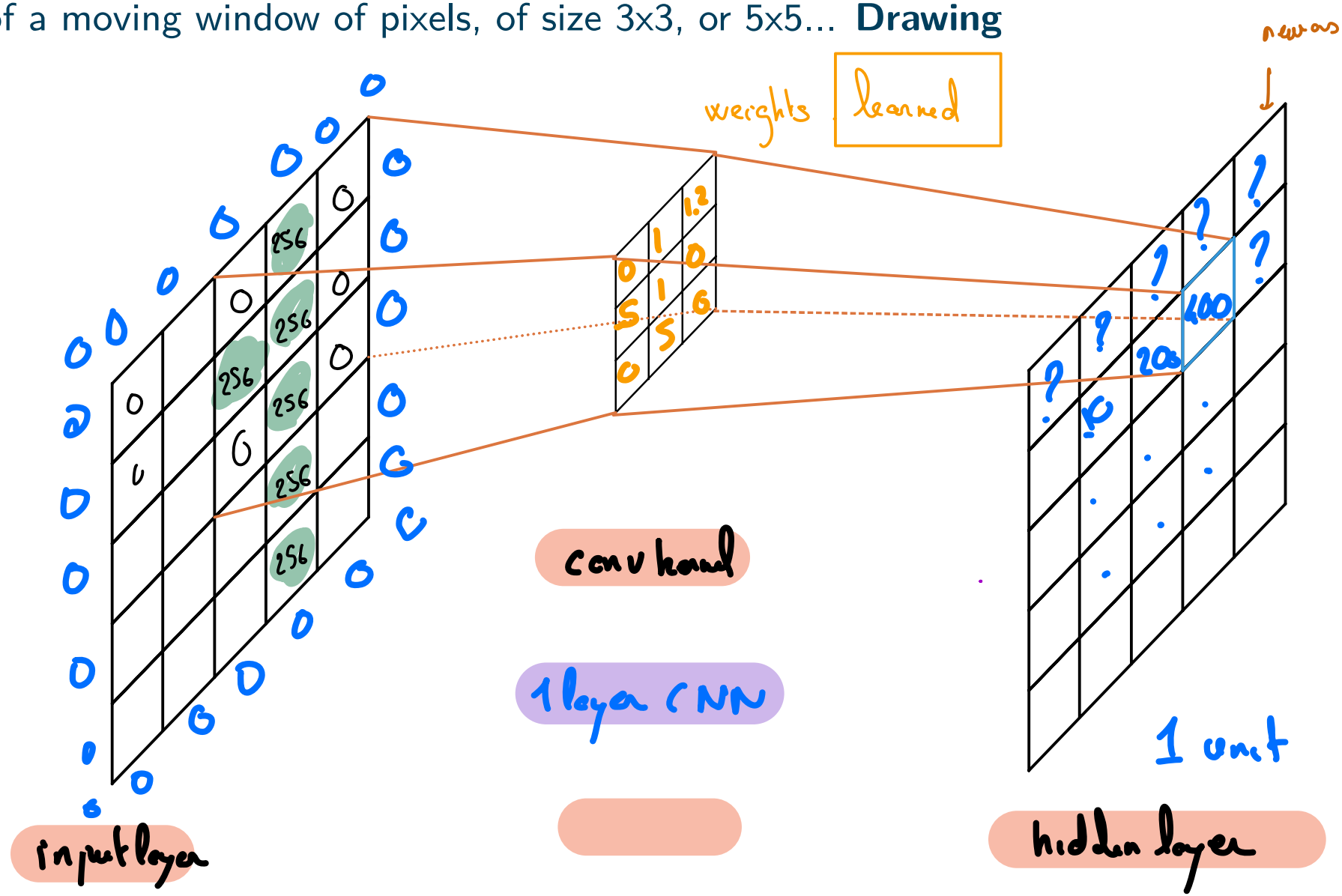
Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5...



do not flatten an image too soon

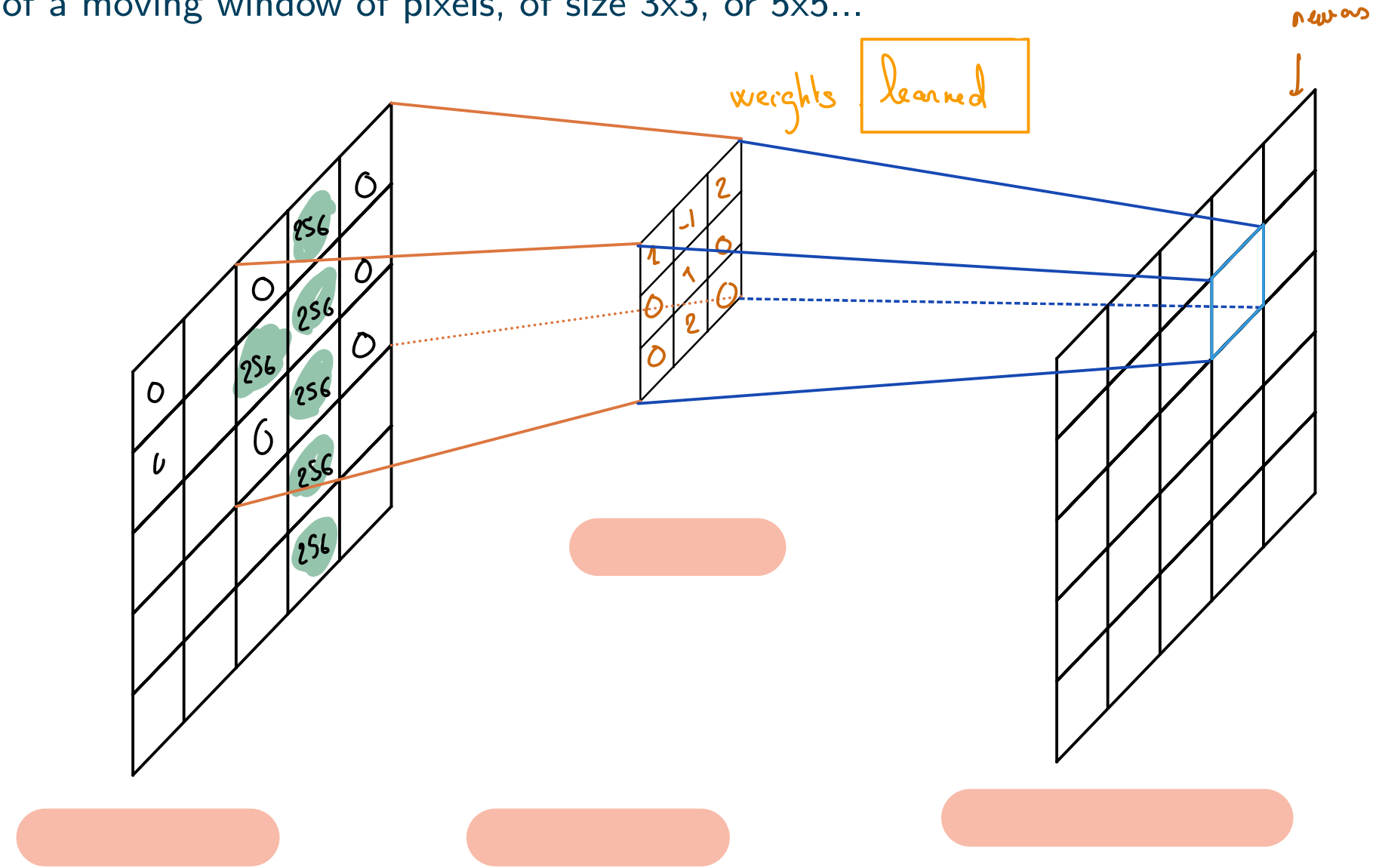
Convolutional Layer

Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5... **Drawing**

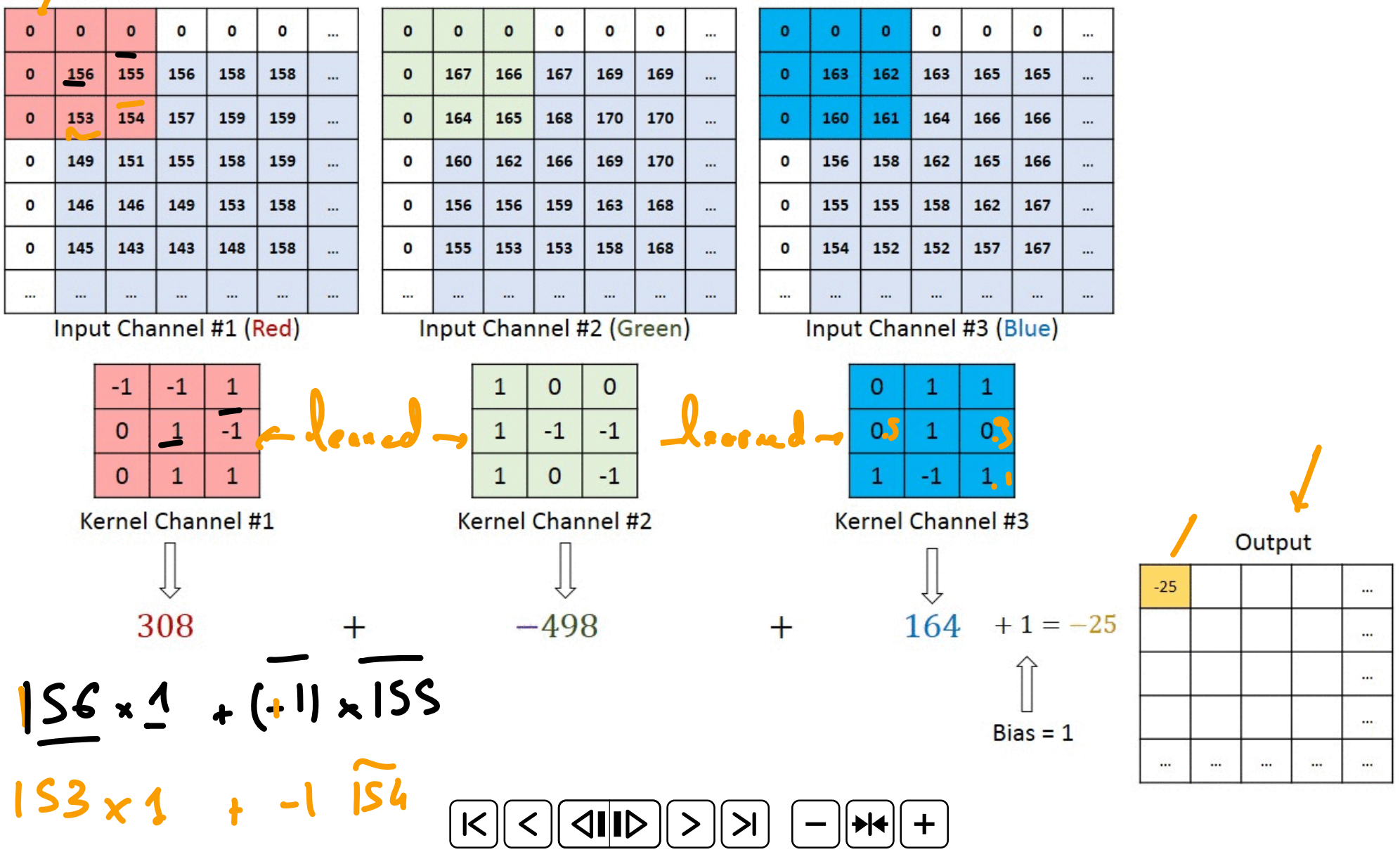


Convolutional Layer

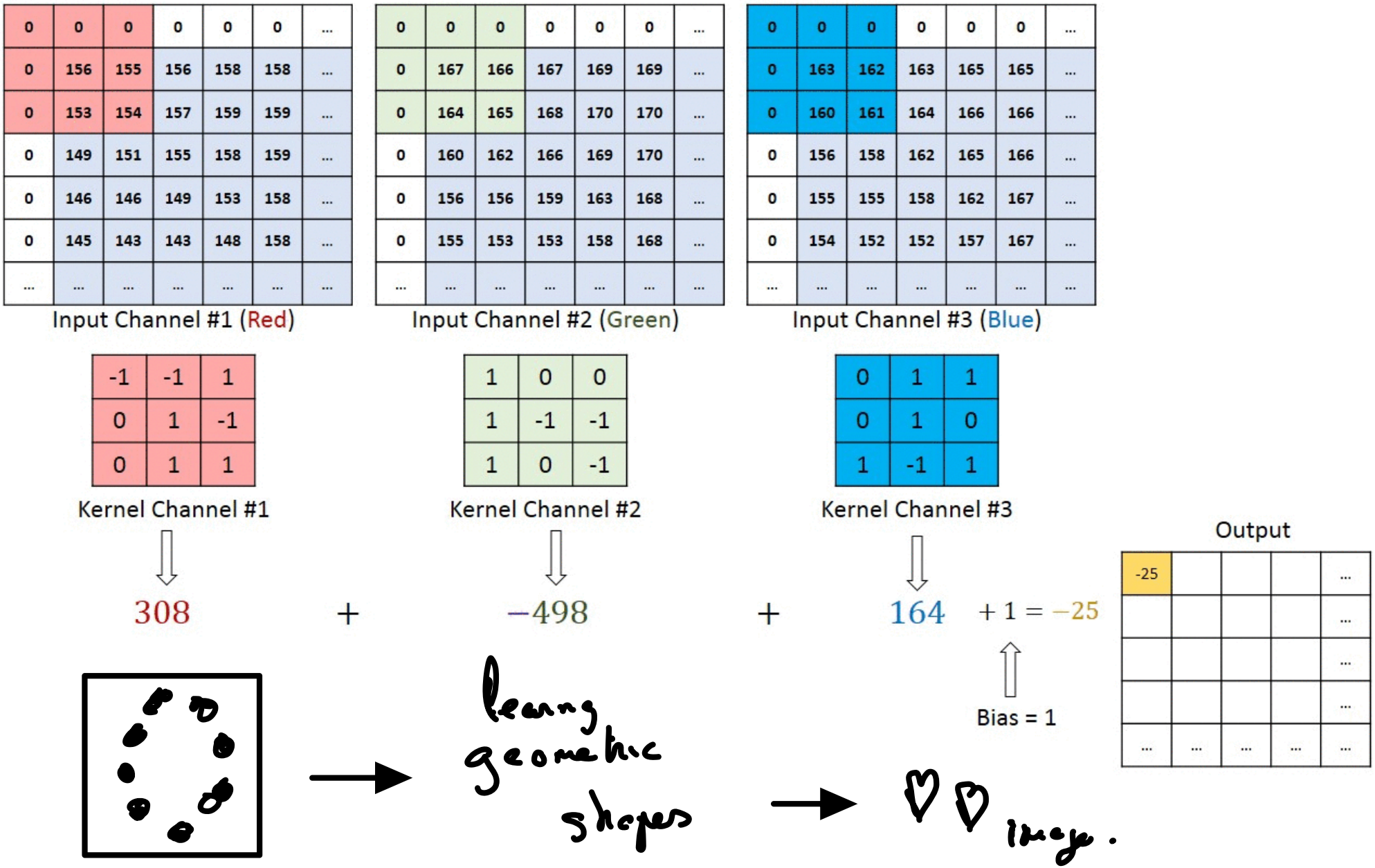
Instead of making a linear combination of all the pixels, we consider a weighted average of a moving window of pixels, of size 3x3, or 5x5...



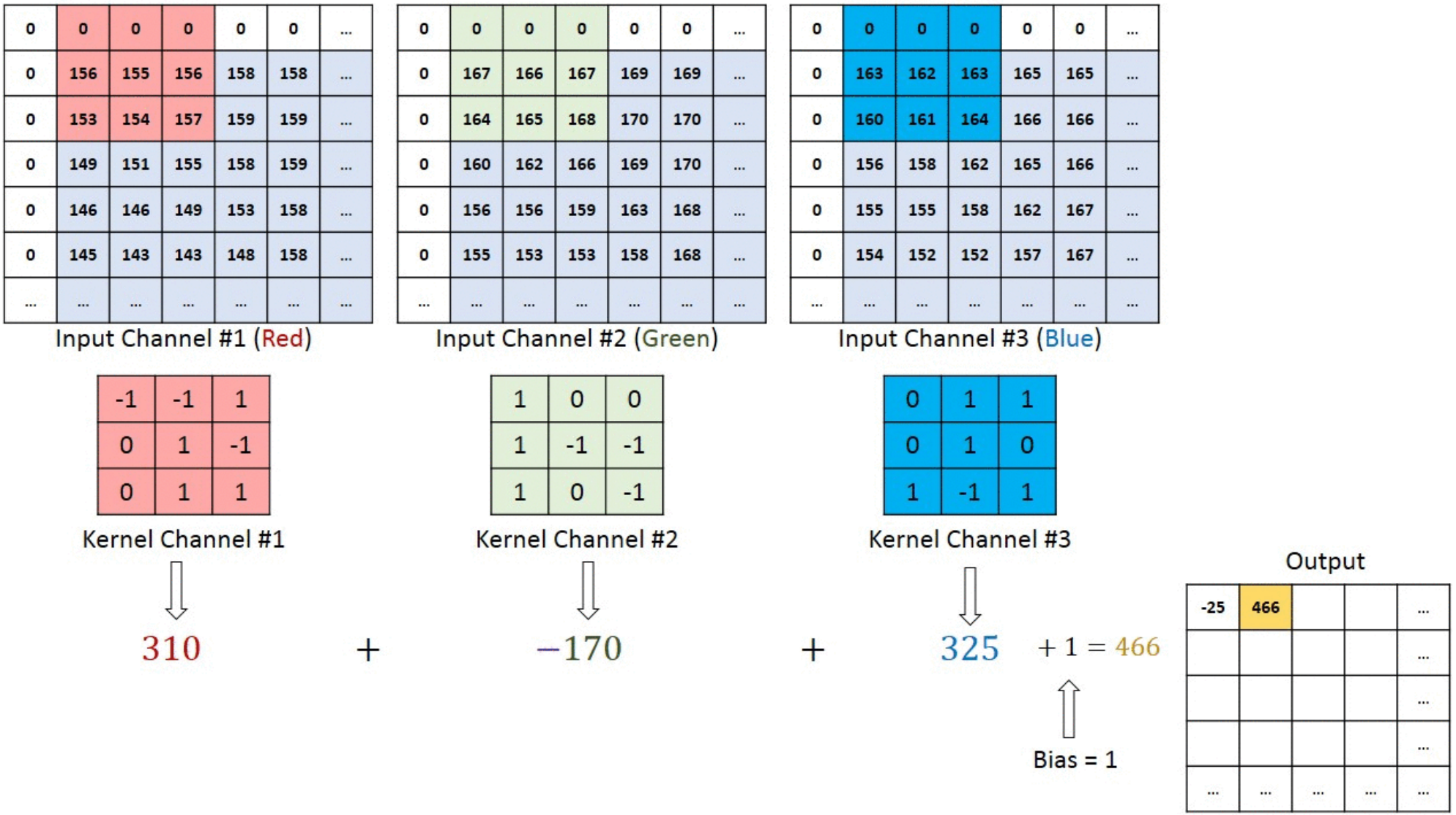
Convolutional Layer



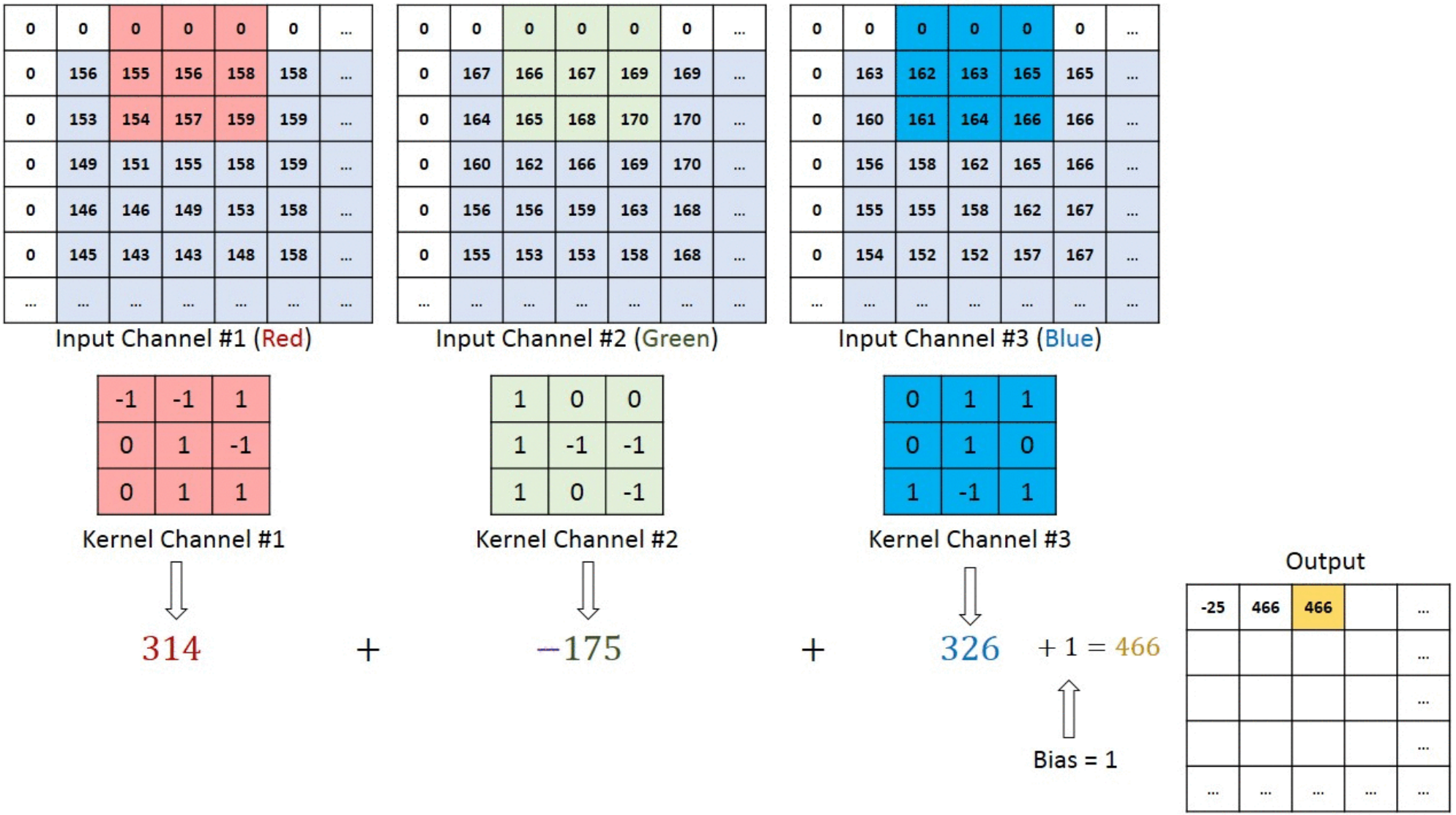
Convolutional Layer



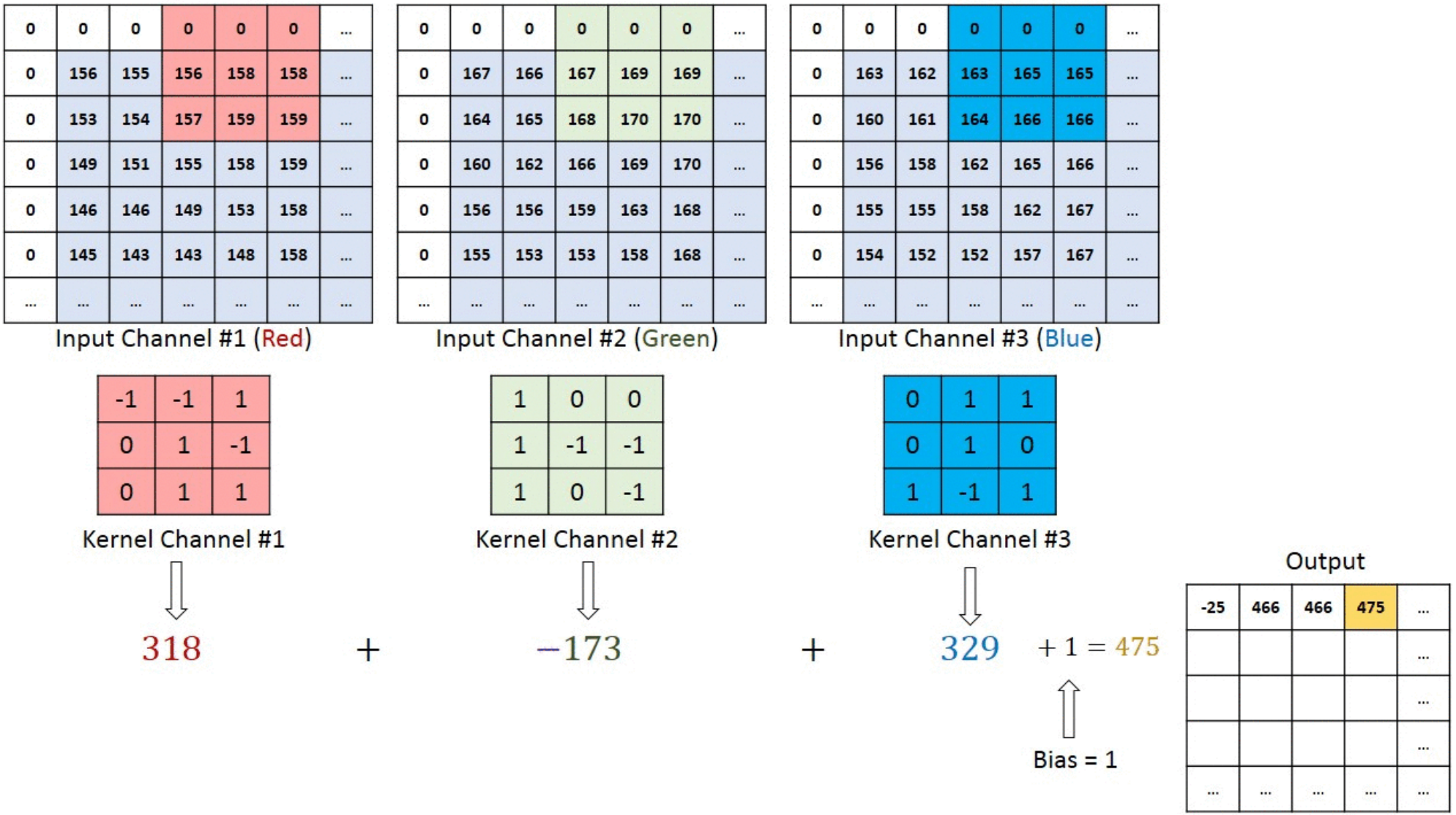
Convolutional Layer



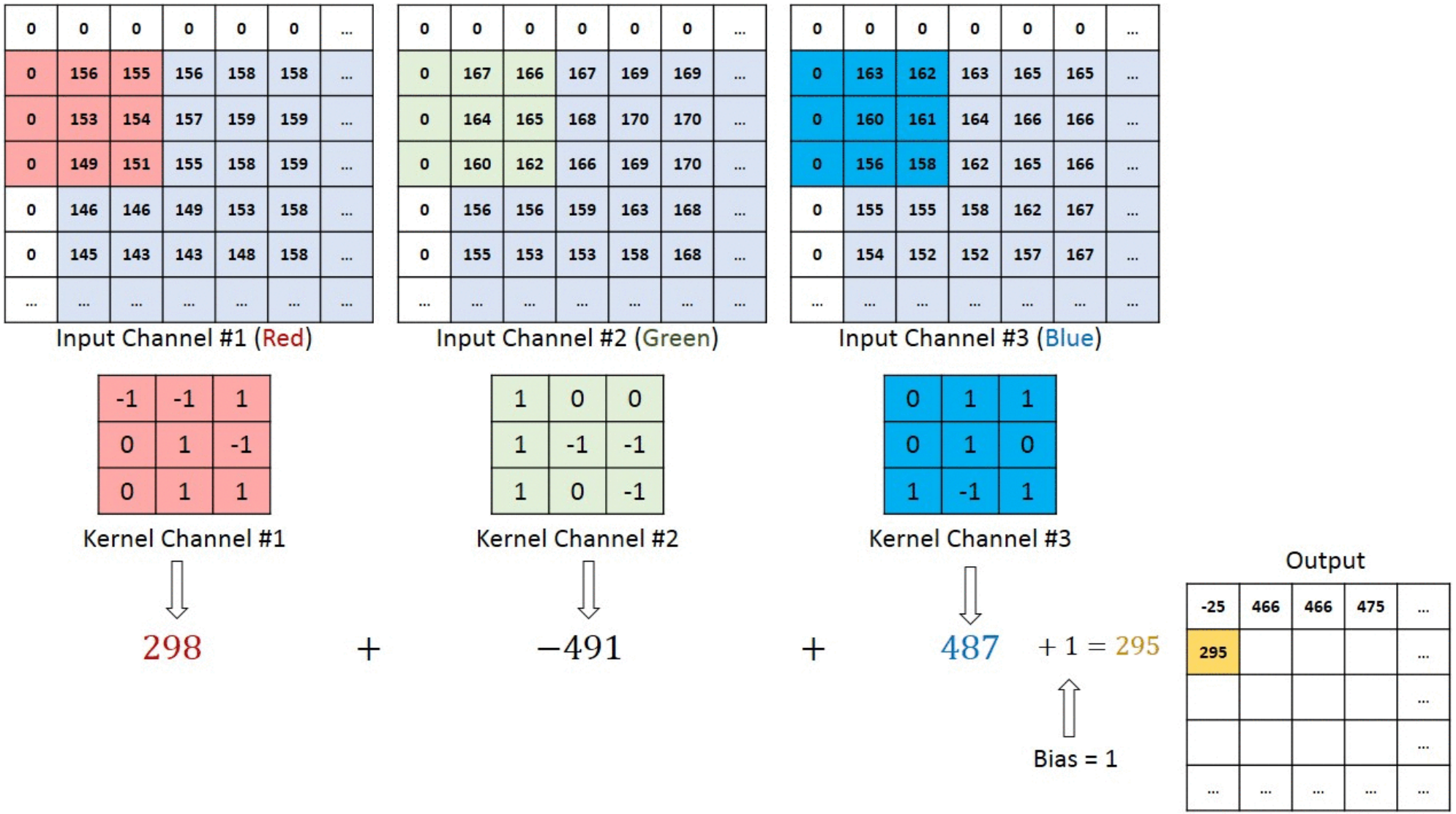
Convolutional Layer



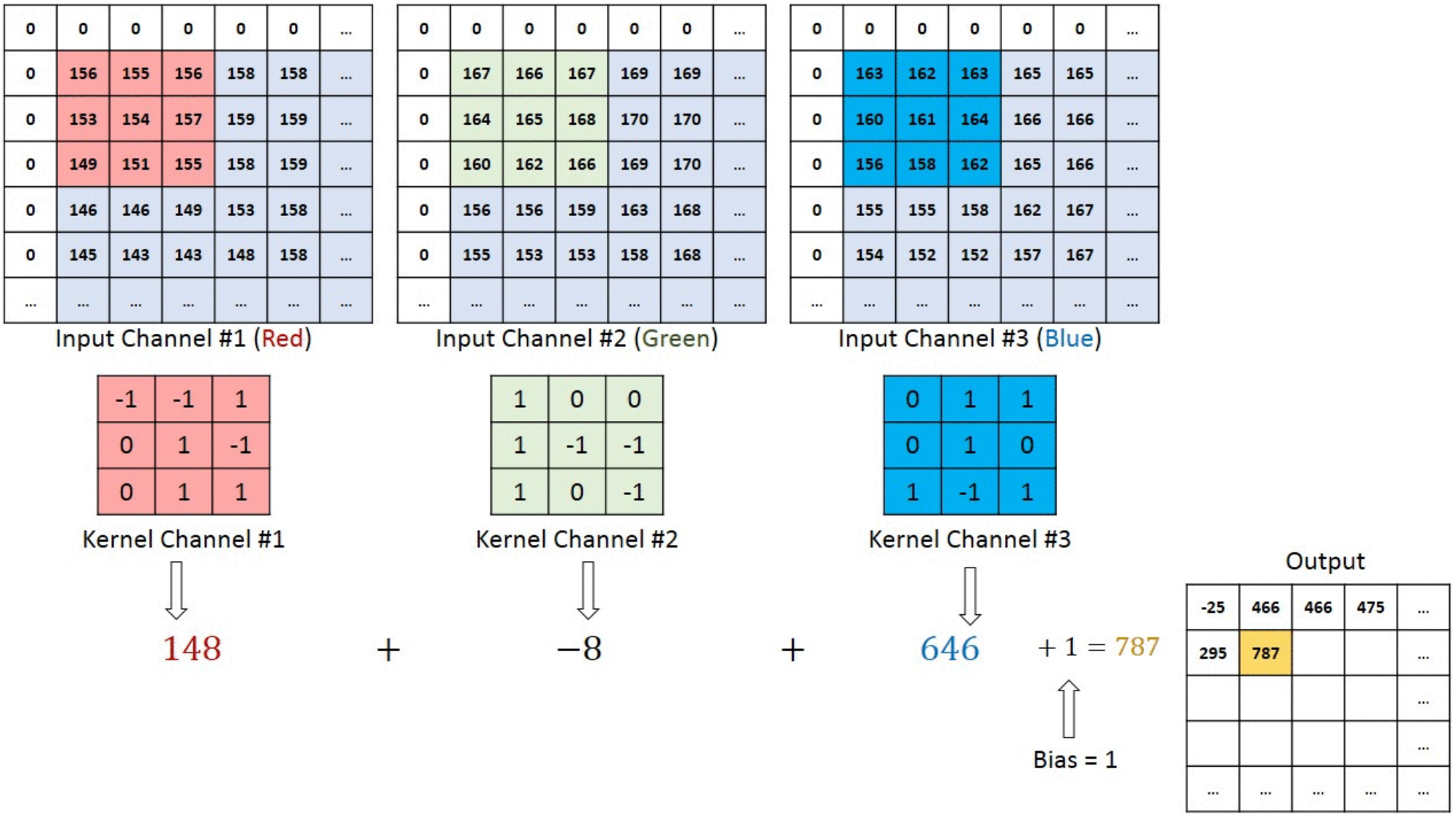
Convolutional Layer



Convolutional Layer



Convolutional Layer



Summary

CNN:

- allow to take spacial information into account
- reduce the number of parameters per layer. We can have deeper networks!

Summary

CNN:

- allow to take spacial information into account
- reduce the number of parameters per layer. We can have deeper networks!

LeNet 1998:

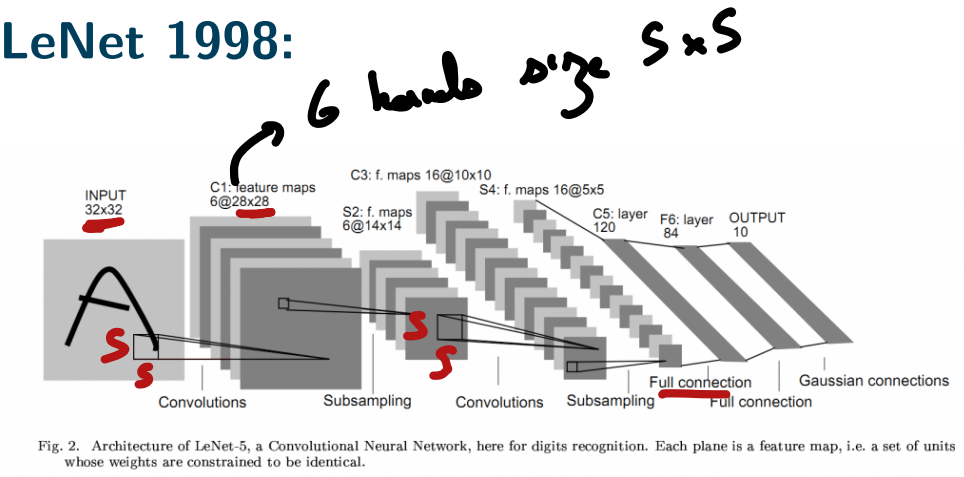
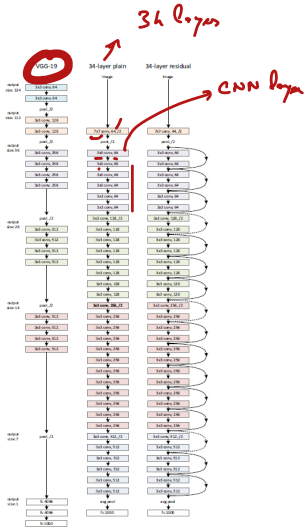


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

8 layers, 60k parameters.

Resnet 2016



~40 Layers, 140M parameters

Outline

Outline

Image classification - CNN Tree

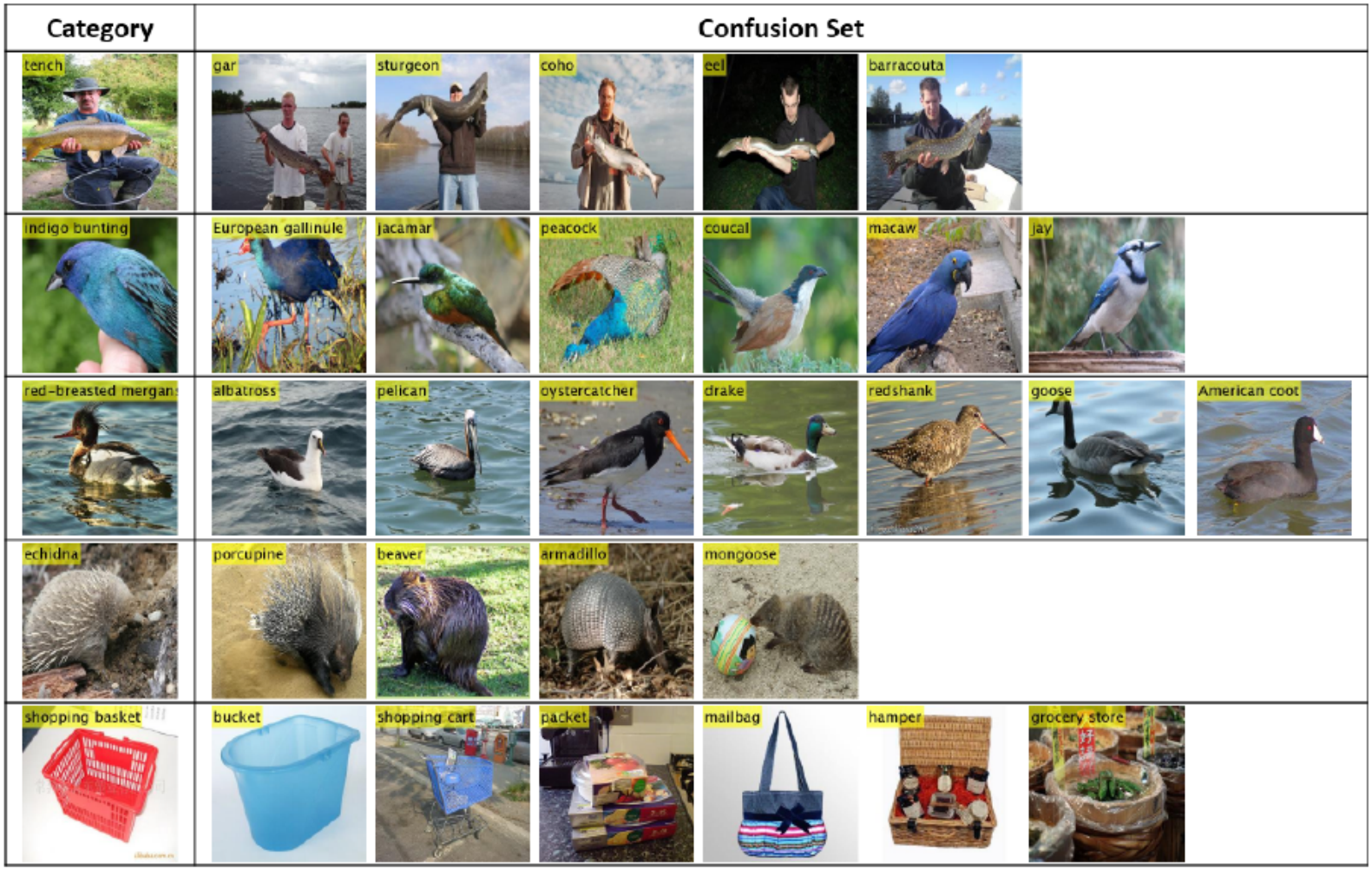


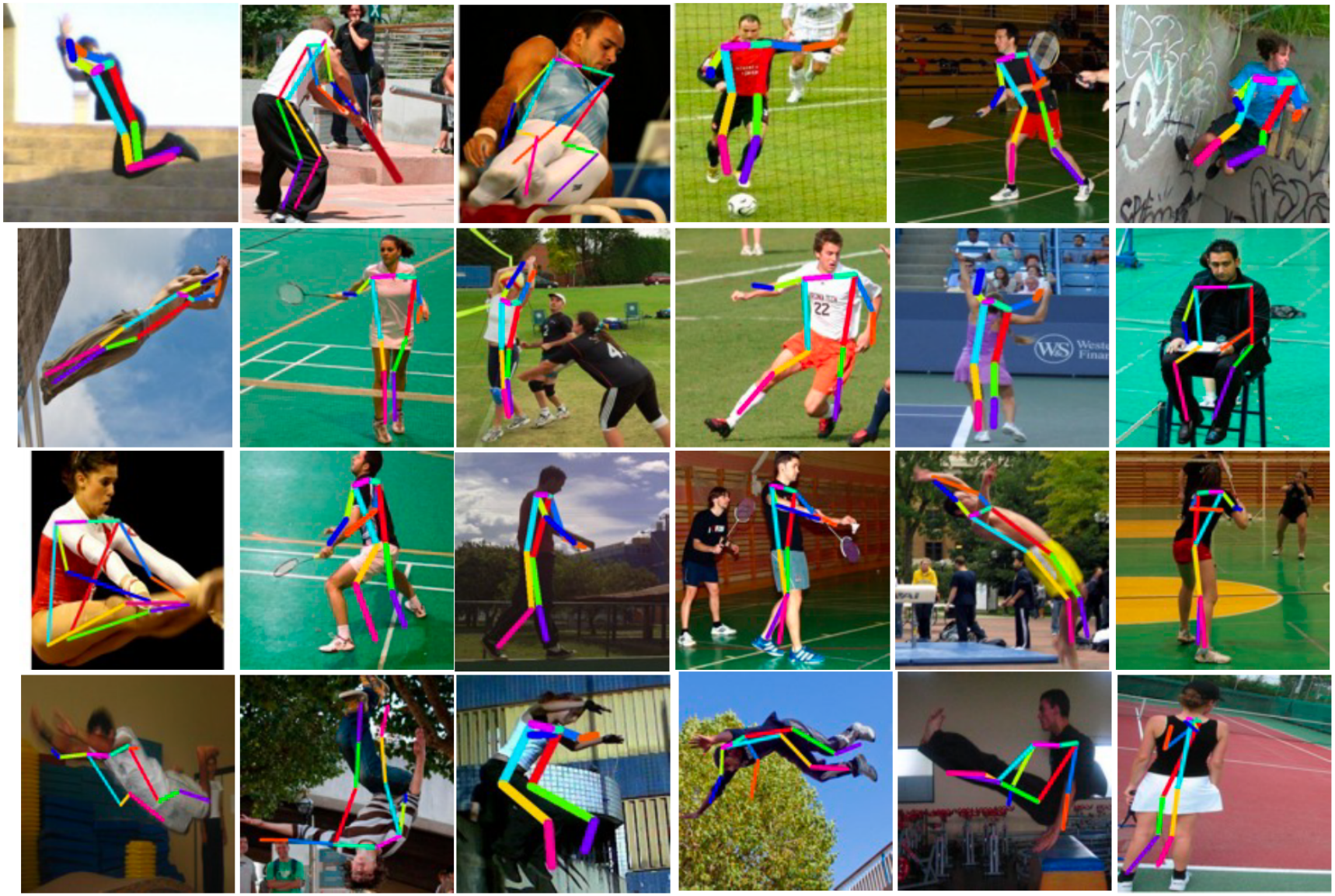
Figure: Confusion set outputs by AlexNet softmax prediction on validation set of ILSVRC 2015.

Image classification - CNN Tree

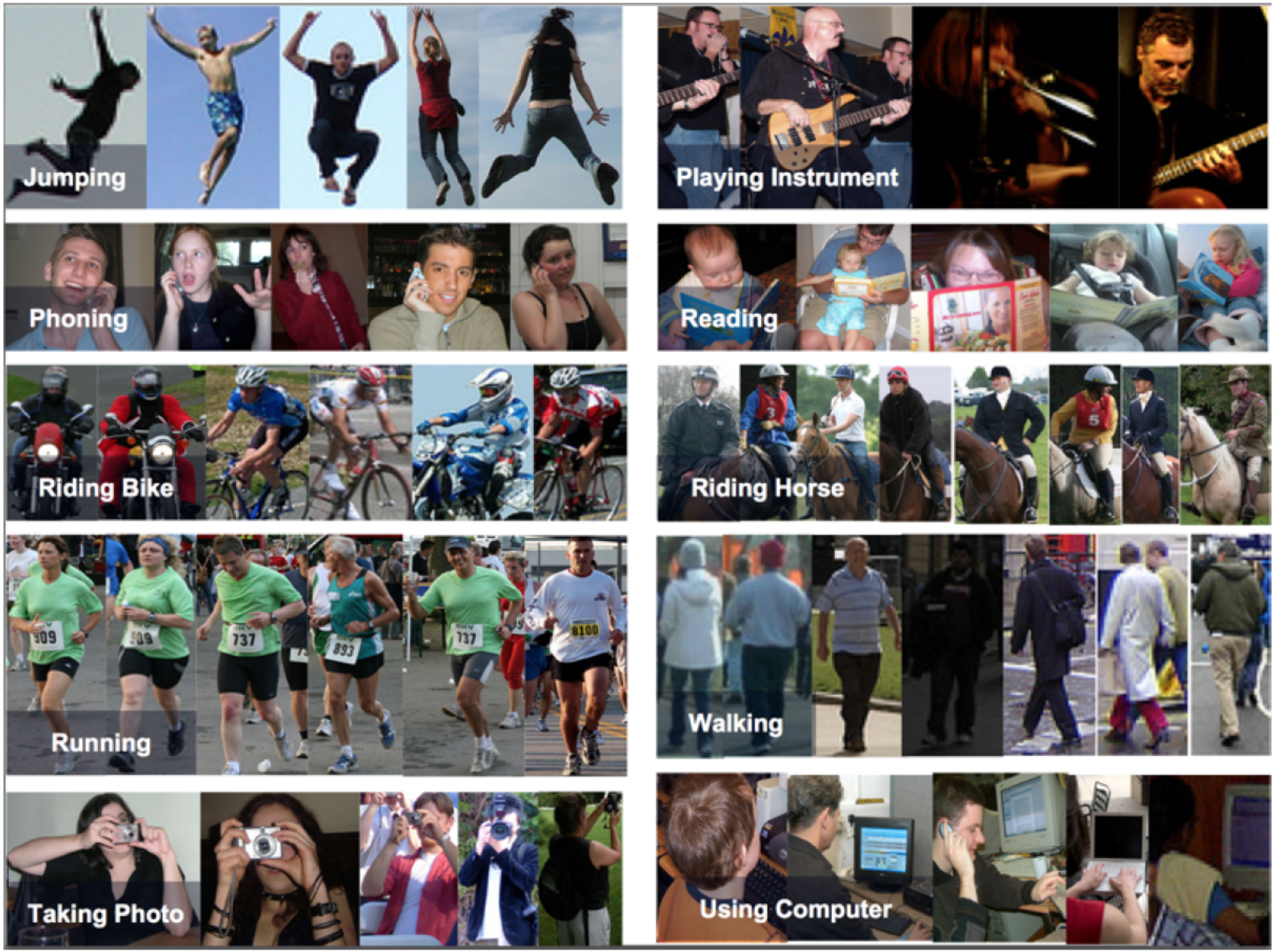
Figure: Top label is given by basic AlexNet CNN while bottom one is given by CNNTree (green color corresponds to a correct prediction)

Outline

Pose estimation - Deeppose



Action recognition



Outline

Object detection and segmentation

(a) (b)

Object detection - YOLO, SDD

More recently, YOLO (You Only Look Once) and SSD (Single Shot Detector) allow single pipeline detection that directly predicts class labels.

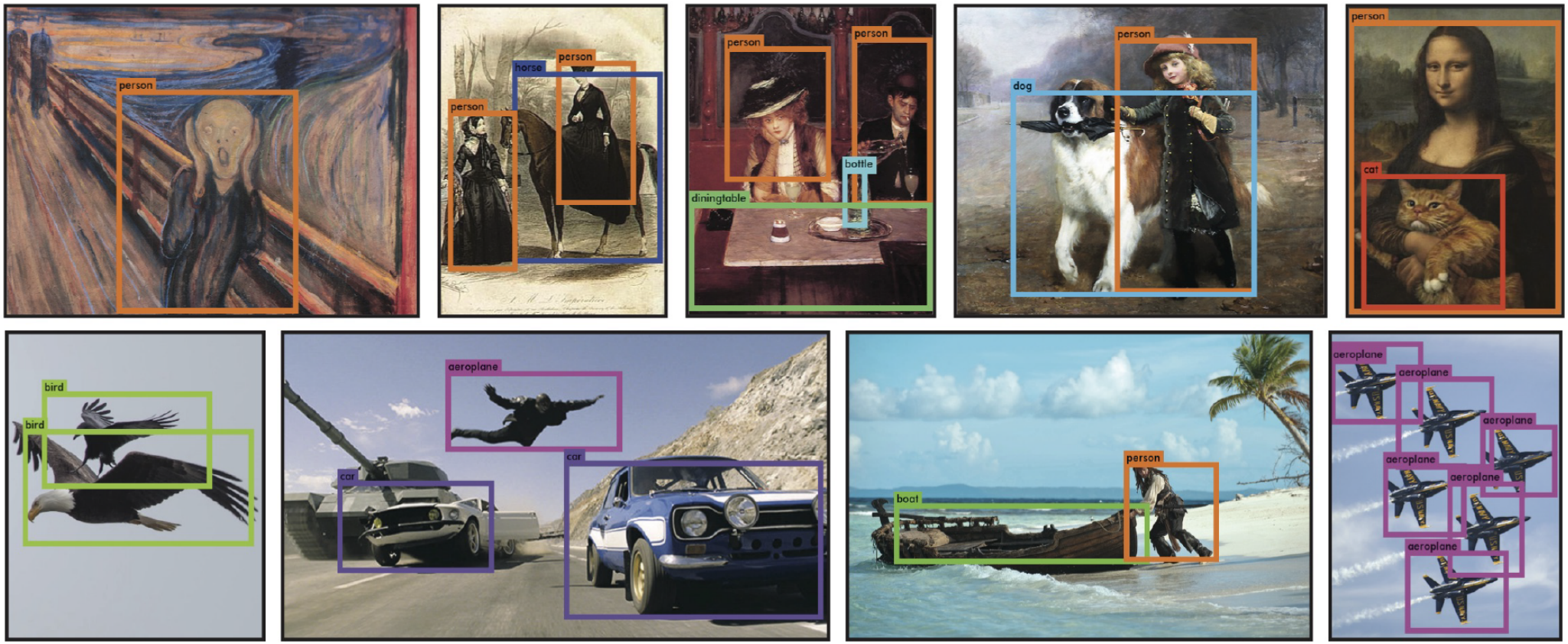
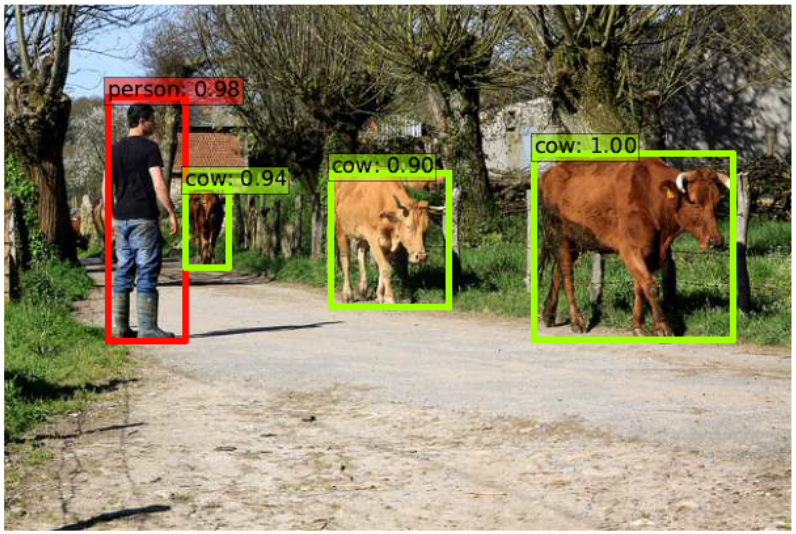
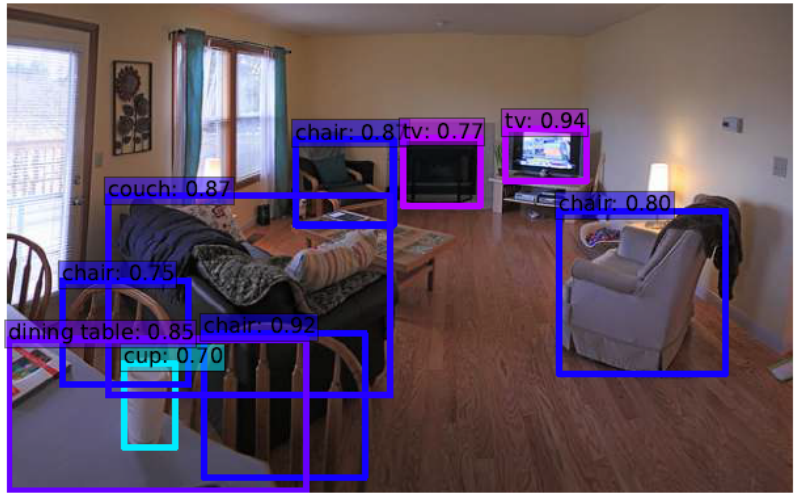
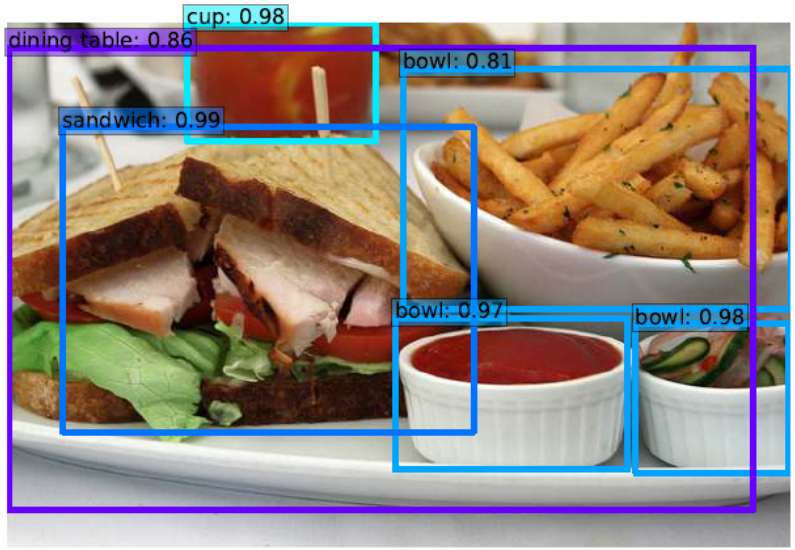


Figure: YOLO results

SSD


person: 0.98
cow: 0.94
cow: 0.90
cow: 1.00


chair: 0.8
tv: 0.77
tv: 0.94
couch: 0.87
chair: 0.80
chair: 0.75
dining table: 0.85
chair: 0.92
cup: 0.70

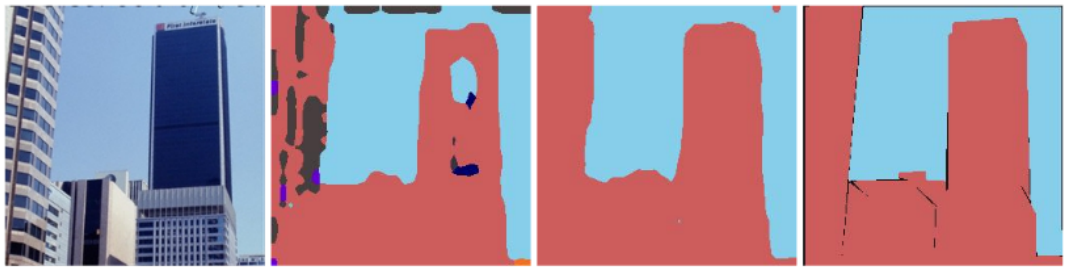

cup: 0.98
dining table: 0.86
bowl: 0.81
sandwich: 0.99
bowl: 0.97
bowl: 0.98


person: 1.00
backpack: 0.82
motorcycle: 0.99

Outline

Scene labeling - DAG-RNN

Input Image CNN DAG-RNN Ground Truth



Input Image CNN DAG-RNN Ground Truth

Outline

Object tracking

Object Detection in a video:

- YOLO `https://pjreddie.com/darknet/yolo/`
- YOLO - James Bond `https://www.youtube.com/watch?v=VOC3huqHrss`

Deep Learning - Many many other applications!

1. Image and video (super-resolution, 3D, Image captioning), medical applications...
2. Self Driving cars (scene segmentation, etc.)
3. In NLP (language, text), automatic translation, voice recognition, text generation, next word completion ... (Virtual assistants)
4. Recommendation systems...
5. For time series, complex datasets too...

But also beyond the supervised settings: using adversarial networks (GANS) for generation of images, faces, voice, etc.

`http://www.whichfaceisreal.com/`

Limits and problems

Limits of Deep Learning: → Wooclap!

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory!

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory!
- Ethical and practical concerns

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory!
- Ethical and practical concerns
- Privacy

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory!
- Ethical and practical concerns
- Privacy
- Source tracking, Ownership, IP

Limits and problems

Limits of Deep Learning: → Wooclap!

- Interpretability
- Vulnerability, failures, biases
- Size of the models: Applicability, Ethical, Energy & Ecological concerns (Ai has both a positive and a huge negative carbon footprint)!
- Theory!
- Ethical and practical concerns
- Privacy
- Source tracking, Ownership, IP
- Alignment

Deep Learning: attacks

CNN are not always “robust" to adversarial attacks!

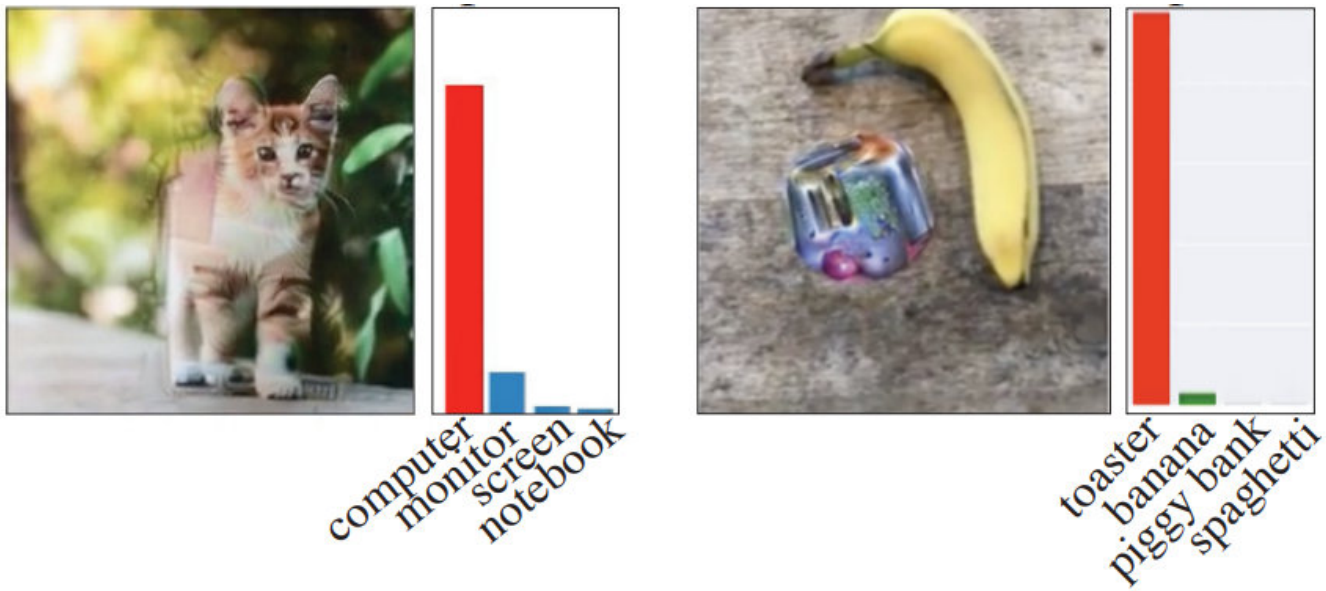
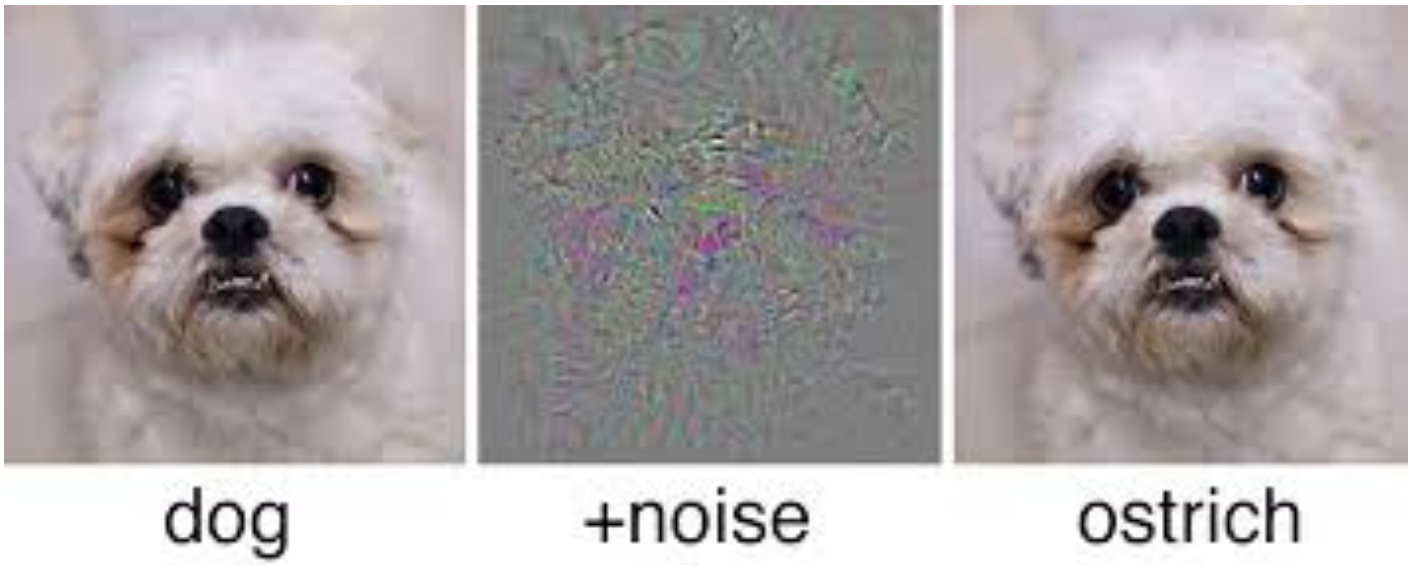


Figure: What happened here??

Deep Learning: attacks

CNN are not always “robust" to adversarial attacks!



Adding a small well chosen noise can completely fool a CNN!

↬ Can we trust deep networks on cars, medical applications, planes...?

Deep Learning: failures

1. Biases in Learning sets that are transferred
    1. Chatbot becomes racist
    2. Google apologises for Photos app's racist blunder

Deep Learning: failures

1. Biases in Learning sets that are transferred
   1. Chatbot becomes racist
   2. Google apologises for Photos app's racist blunder
2. Privacy leaks:
   1. The size of the models often enable memorization of the training sets.
   2. Need for guarantees and regulation

Deep Learning: failures

1. Biases in Learning sets that are transferred
   1. Chatbot becomes racist
   2. Google apologises for Photos app's racist blunder
2. Privacy leaks:
   1. The size of the models often enable memorization of the training sets.
   2. Need for guarantees and regulation
3. False information/prediction, hallucinations...

Deep Learning: failures

1. Biases in Learning sets that are transferred
   1. Chatbot becomes racist
   2. Google apologises for Photos app's racist blunder
2. Privacy leaks:
   1. The size of the models often enable memorization of the training sets.
   2. Need for guarantees and regulation
3. False information/prediction, hallucinations...
4. Overall, lack of control on the model behavior

PIXELS • VIE PRIVÉE

Nouvelle plainte contre OpenAI pour infraction au RGPD

ChatGPT enfreint le Règlement général sur la protection des données, estime l'association autrichienne None of Your Business (NOYB), car il peut donner de fausses informations concernant un individu, sans possibilité de les faire corriger ou supprimer.

Le Monde avec AFP
Publié hier à 10h44 • Lecture 1 min.

Figure: Link

Alignment


FAST COMPANY
CO.DESIGN TECH WORK LIFE NEWS IMPACT PODCASTS VIDEO RECOMMENDER INNOVATION FESTI
11-06-17 | PLATFORM WARS
Netflix CEO Reed Hastings: Sleep Is Our Competition
For Netflix, the battle for domination goes far beyond which TV remote to pick up.
CEO Reed Hastings [Photo: Mondileinchen/Wikimedia Commons]

Deep Learning: societal risk and questionable applications

Are some applications just bad (they all already exist :( )?

- Deep Fakes: insert someone in a video.
- Underground exploration: finding new fossil resources.
- Voice imitation: what if I cannot check who is calling me.
- Military applications + automatic weapons.
- Mass surveillance.

The likelihood of each of those applications being largely deployed increases every year.

Deep Learning: societal risk and questionable applications

Are some applications just bad (they all already exist :( )?

- Deep Fakes: insert someone in a video.
- Underground exploration: finding new fossil resources.
- Voice imitation: what if I cannot check who is calling me.
- Military applications + automatic weapons.
- Mass surveillance.

The likelihood of each of those applications being largely deployed increases every year.

Deep Learning: questionable applications

A challenging problem: chain of responsibility. The same tools are used for positive and negative applications, e.g.:

- mass surveillance
- tumor recognition

are both based on image recognition, and the same ML techniques.

Deep Learning: questionable applications

A challenging problem: chain of responsibility. The same tools are used for positive and negative applications, e.g.:

- mass surveillance
- tumor recognition

are both based on image recognition, and the same ML techniques.

Good news! the research community is very aware of the situation.
Some references:

- Asilomar AI Principles Ai principles
- AI Act
- European guidelines
- AI for Good

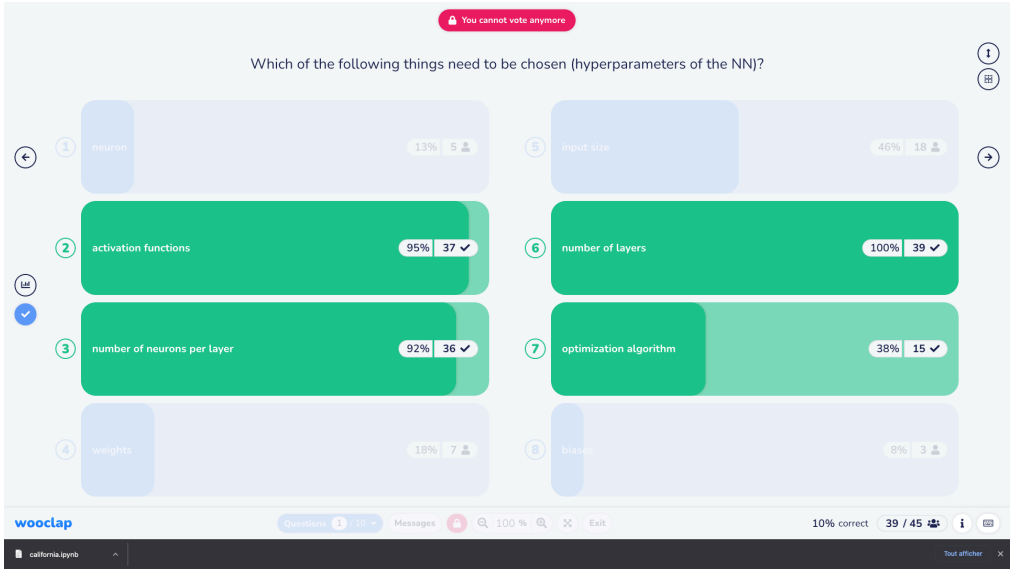
We need you to be aware and active on those challenges!!

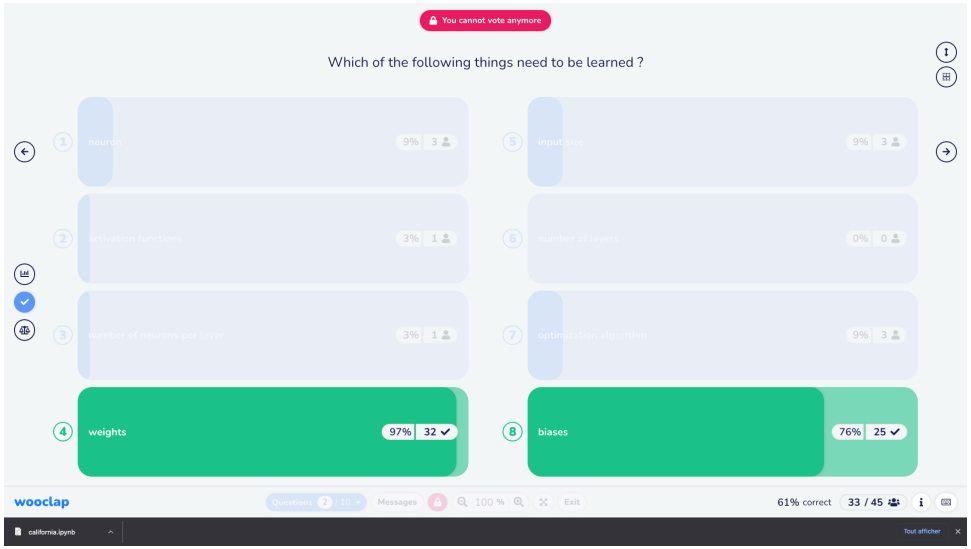
Conclusion: Deep Learning in one slide

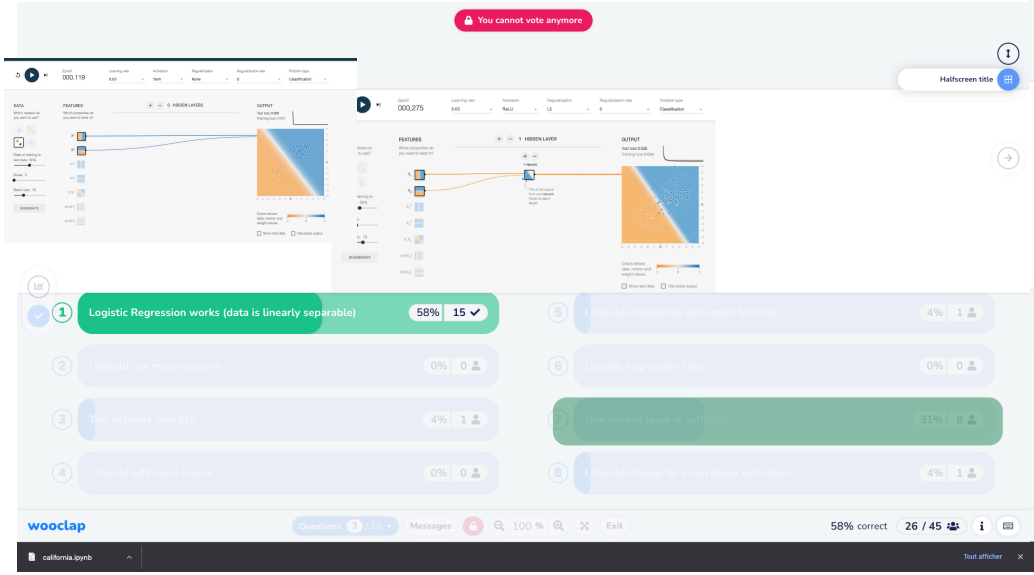
- **How does it work:**
  - Automatically learn representations of observations
  - Learn highly non-linear models.
- **What does it require:**
  - Large datasets with structure
  - Computational power
- **Why now:**
  - Combination of the 2 points above
  - investment!
- **Some Applications**
  - Image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

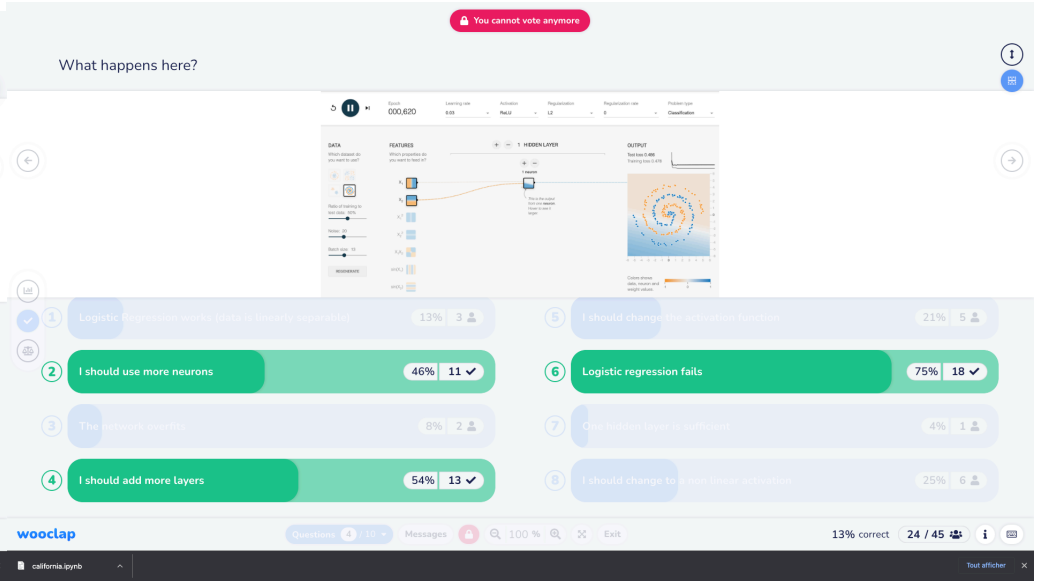
Goals: Understanding core concepts of Deep Learning.

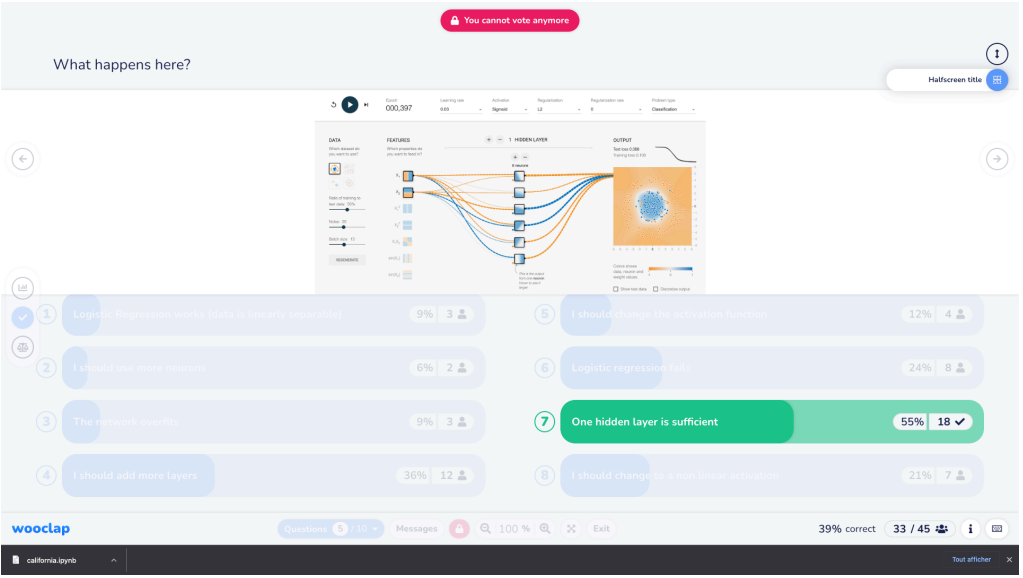
- When and how it works on paper (data, models, architecture)
- How to implement a simple neural network with Python.
- Overview of some of the main applications and challenges.

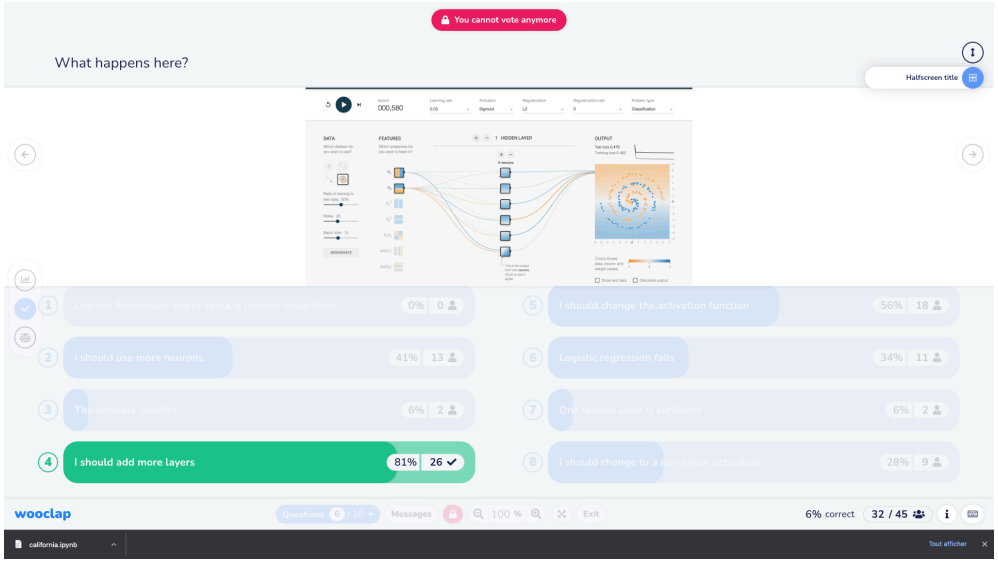

You cannot vote anymore
Which of the following things need to be chosen (hyperparameters of the NN)?
1 neuron 13% 5
2 activation functions 95% 37
3 number of neurons per layer 92% 36
4 weights 18% 7
5 input size 46% 18
6 number of layers 100% 39
7 optimization algorithm 38% 15
8 biases 8% 3
wooclap
10% correct 39 / 45

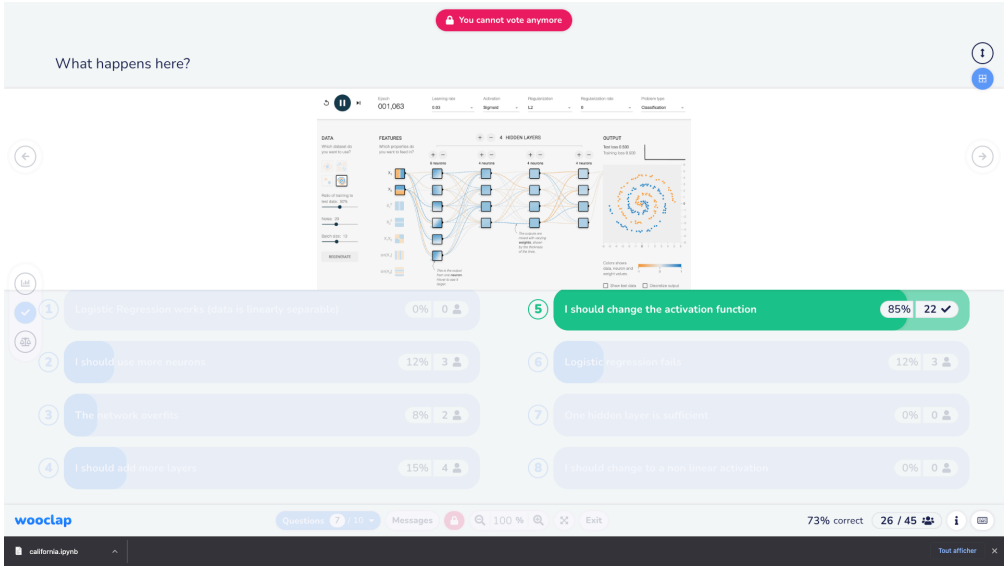

You cannot vote anymore
Which of the following things need to be learned ?
4 weights 97% 32
8 biases 76% 25
wooclap
61% correct 33 / 45

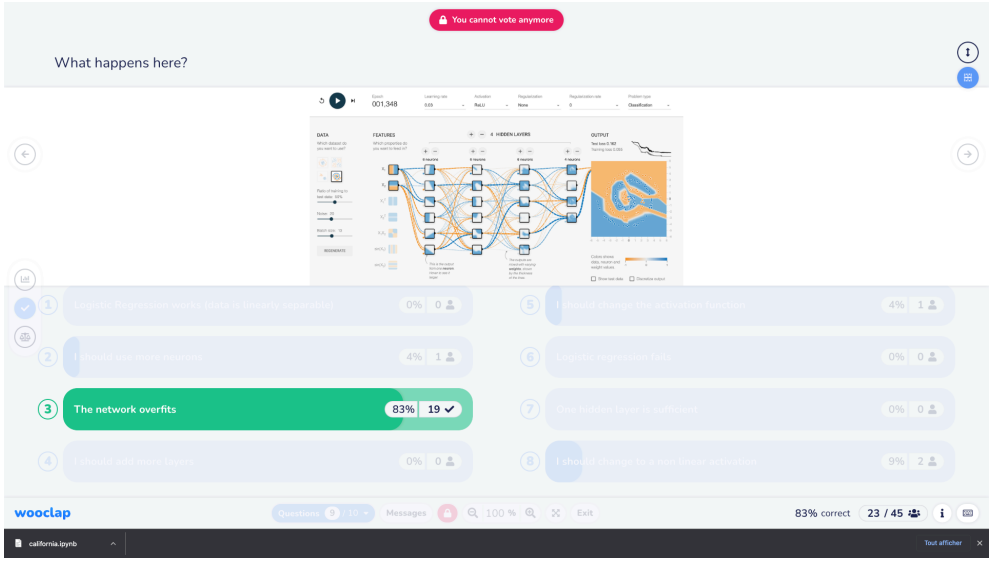

You cannot vote anymore
Halfscreen title
1 Logistic Regression works (data is linearly separable) 58% 15
wooclap
58% correct 26 / 45

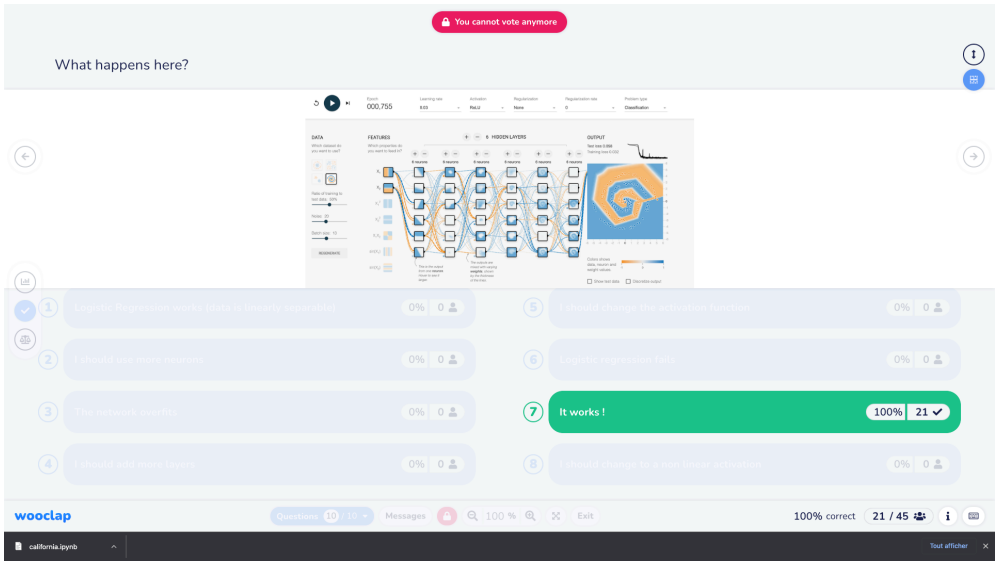

You cannot vote anymore
What happens here?
2 I should use more neurons 46% 11
4 I should add more layers 54% 13
6 Logistic regression fails 75% 18
wooclap
13% correct 24 / 45


You cannot vote anymore
What happens here?
Halfscreen title
1 Logistic Regression works (data is linearly separable) 9% 3
2 I should use more neurons 6% 2
3 The network overfits 9% 3
4 I should add more layers 36% 12
5 I should change the activation function 12% 4
6 Logistic regression fails 24% 8
7 One hidden layer is sufficient 55% 18
8 I should change to a non linear activation 21% 7
wooclap
39% correct
33 / 45
california.ipynb
Tout afficher


You cannot vote anymore
What happens here?
Halfscreen title
1 Logistic Regression works (data is linearly separable) 0% 0
2 I should use more neurons 41% 13
3 The network overfits 6% 2
4 I should add more layers 81% 26
5 I should change the activation function 56% 18
6 Logistic regression fails 34% 11
7 One hidden layer is sufficient 6% 2
8 I should change to a non linear activation 28% 9
wooclap
6% correct
32 / 45
california.ipynb
Tout afficher


You cannot vote anymore
What happens here?
5 I should change the activation function 85% 22
wooclap
73% correct 26 / 45


You cannot vote anymore
What happens here?
3 The network overfits 83% 19
wooclap
83% correct 23 / 45


You cannot vote anymore
What happens here?
7 It works ! 100% 21
wooclap
100% correct 21 / 45