Deep Neural Networks
And Where to Find Them

Artem Korenev, Nikita Gryaznov

Skoltech ISP, 2018

Welcome

- Hi everybody!
- Two TAs: Artem and Nikita (2nd year MSc students)
- 6 days in-class (~20 hours)
- 40 hours for the course in total
- Other time for HW and self education
- If you don't get something – you are very welcome to ask a question right away
- Don't be shy and come after lecture if you need

What the Course is About

- Introduction in Deep Neural Networks and how to use them
- This is mostly an introductory course (your first starting point in DL)
- We aim on people outside IT track
- Bias towards practice instead of theory
- We will use Keras library (not Tensorflow, not Theano, not PyTorch)
- Currently, we are not going to even briefly touch things like GAN, RNN, Reinforcement Learning, Segmentation and Detection
- If you are an advanced student and still want to learn DL further please come forward and we will discuss what you can do individually

Syllabus*

Days:

1. ML introduction. Logistic regression. Neural networks approach.
2. Matrix representation of neural networks. Details of NNs optimization.
3. Activation functions. Loss functions. Regression problem.
4. Different optimizers. Regularization.
5. Convolutional Neural Networks basics
6. Practical tricks and consultation.

+ Daily seminars and exercises climaxing with a final homework assignment

** Subject to change depending on the pace*

Requirements for the Course

- Basic proficiency in Python
- Simple math knowledge (matrices, derivatives)
- Having a laptop

- None of this is absolutely mandatory but will help you understand many things
- Let's have a quiz everybody: https://goo.gl/ZTXf3v

Why Bother?

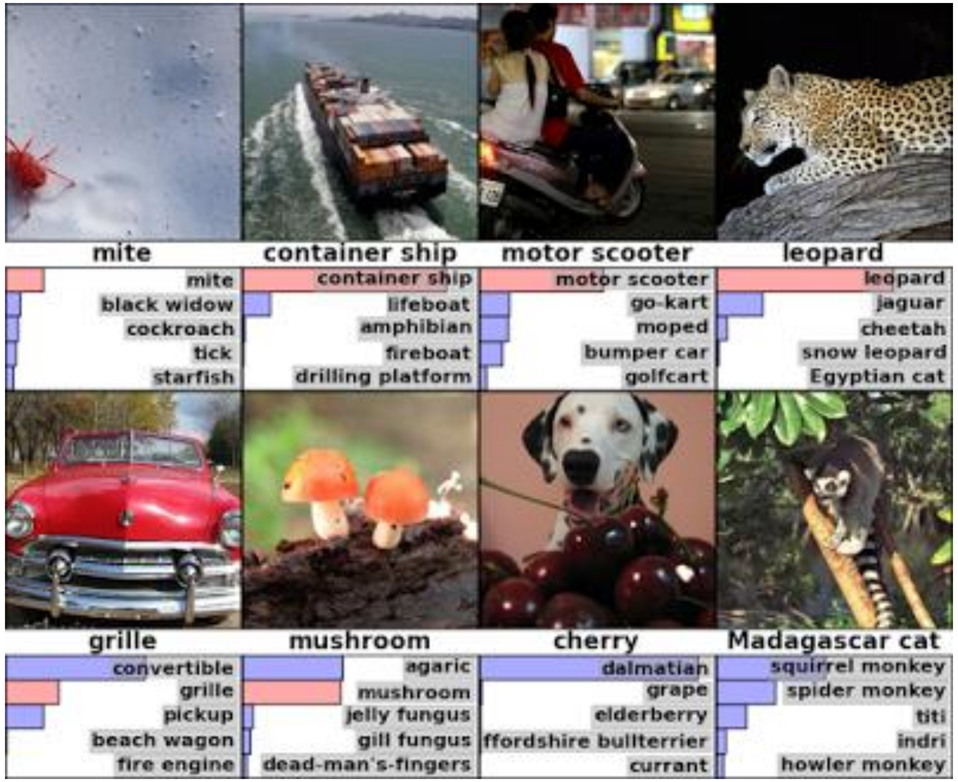
A Couple of Words About Machine Learning

Machine Learning in 2018

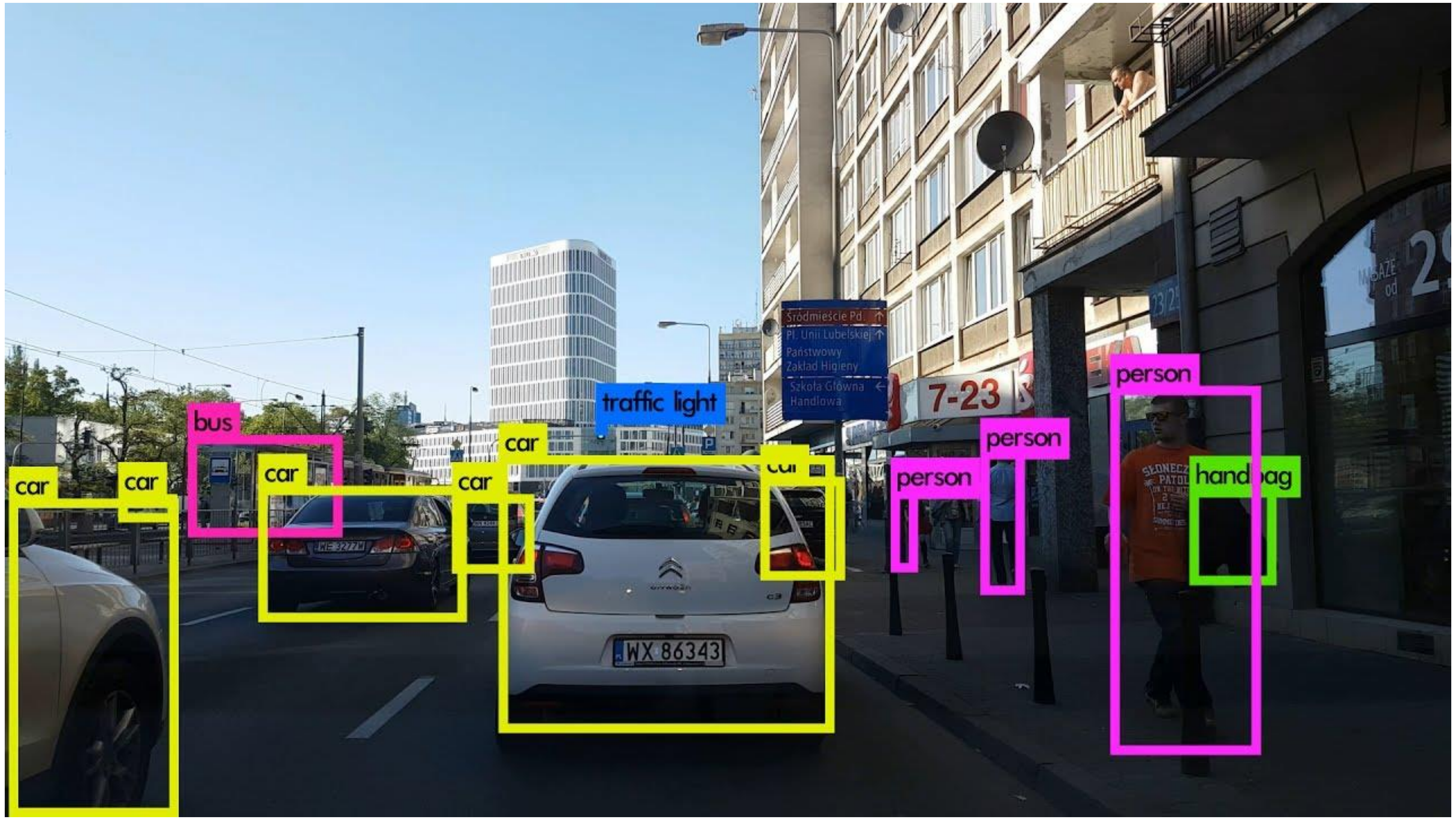
- The sexiest job of the 21st century
- Creating programs from data
- Field has a lot of algorithms (kNN, decision trees, SVM, etc.)
- Neural Networks is just one of them
- NNs are growing crazy since 2012
- NNs are everywhere
- Let’s see some examples…

Image classification

- ImageNet Challenge, 2012

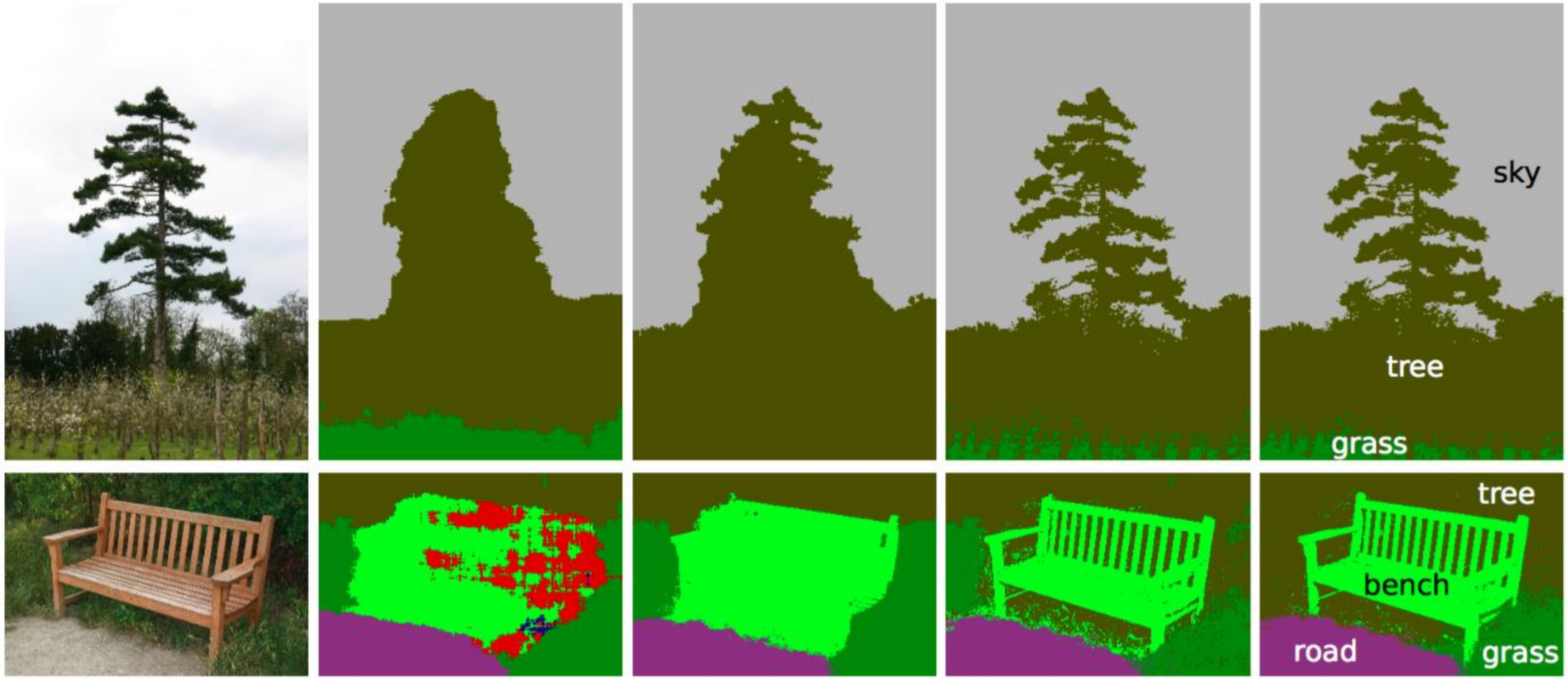


Object Detection



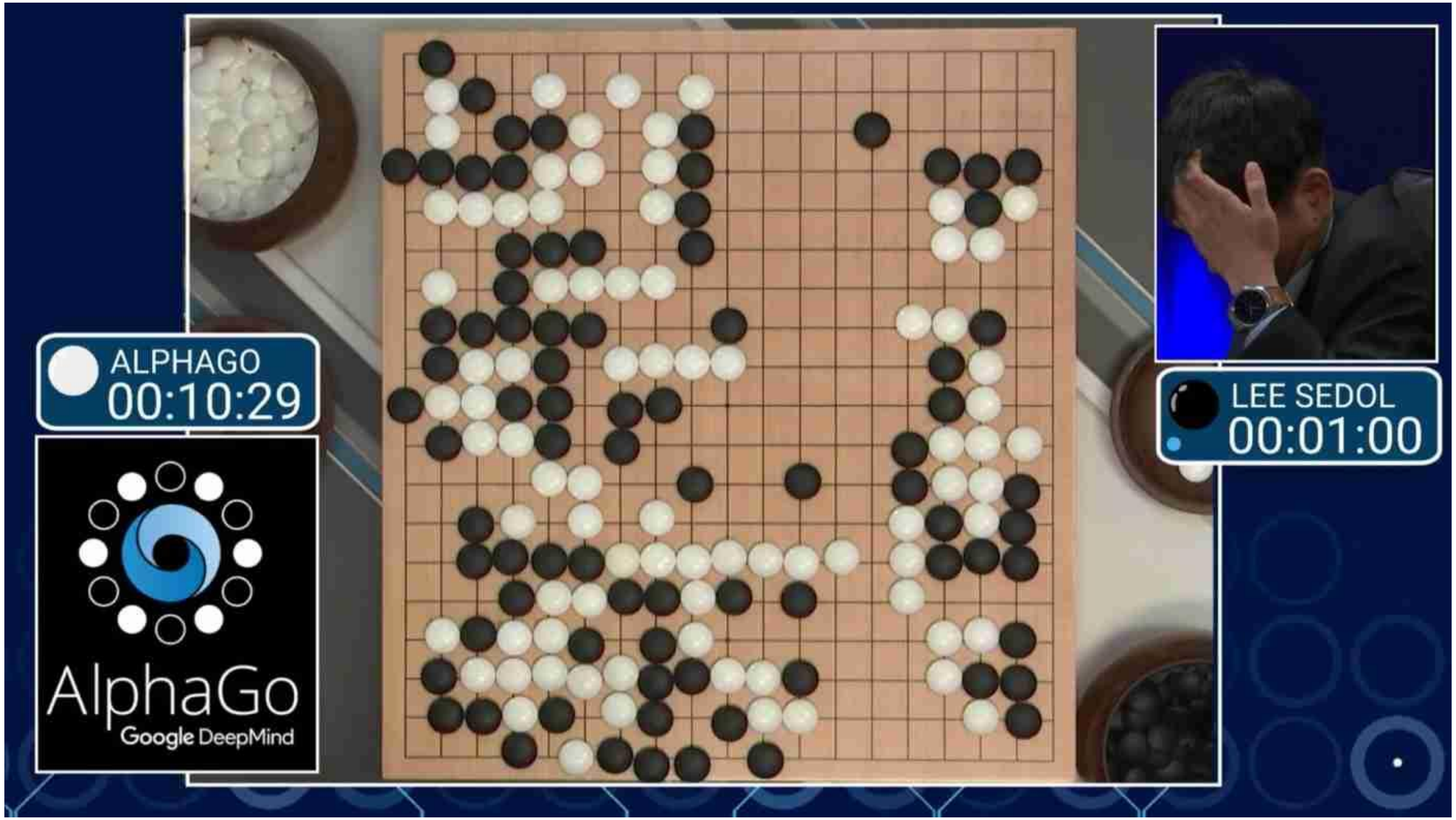
https://www.youtube.com/watch?v=VOC3huqHrss

Image Segmentation

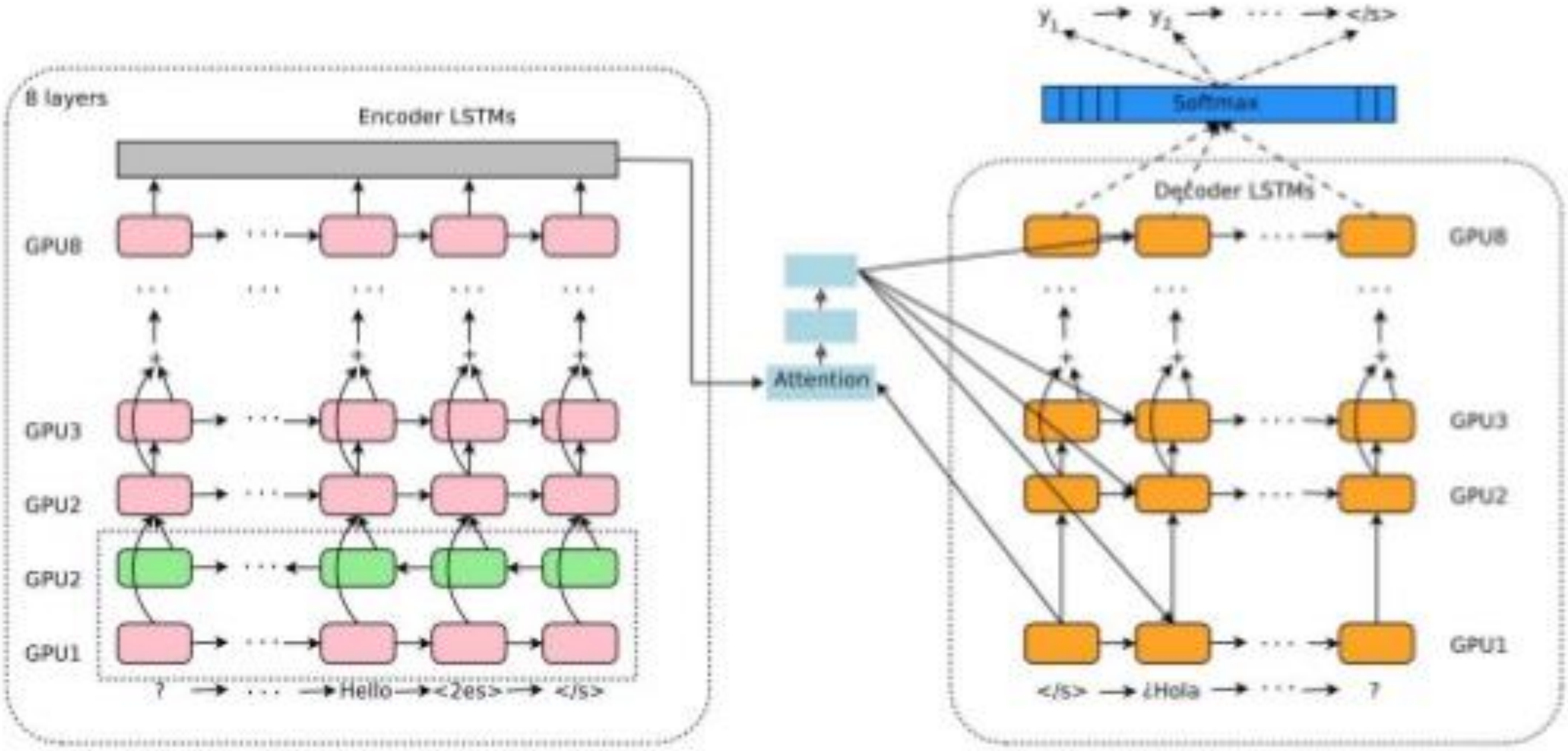


(a) Image (b) Unary classifiers (c) Robust P^n CRF (d) Fully connected CRF, MCMC inference, 36 hrs (e) Fully connected CRF, our approach, 0.2 seconds

AlphaGo



Machine Translation



Prisma App

Demos

- http://make.girls.moe/
- https://www.youtube.com/watch?v=9reHvktowLY
- https://affinelayer.com/pixsrv/
- https://www.youtube.com/watch?v=DgPaCWJL7XI
- https://google.github.io/tacotron/publications/tacotron2/index.html
- http://www.aiva.ai/

- And many more

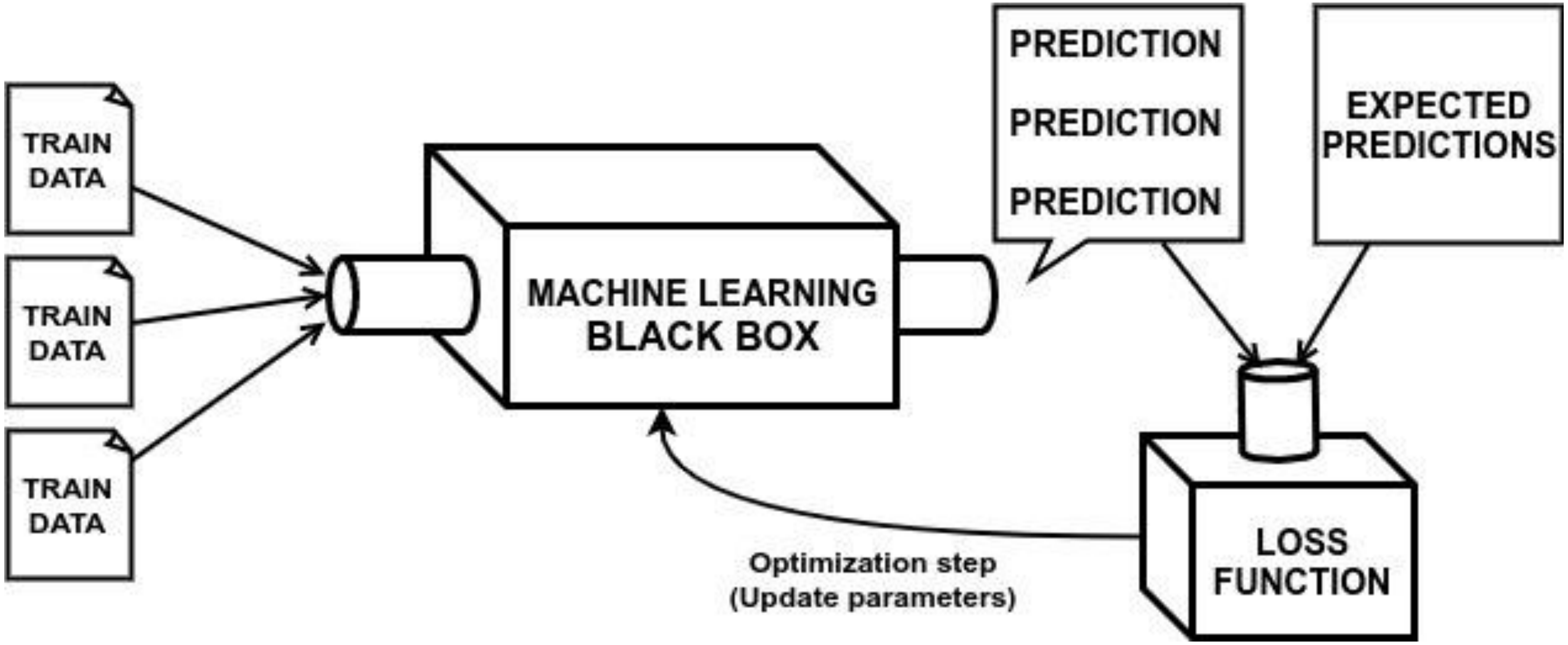
Machine Learning Algorithms

How they (usually) work

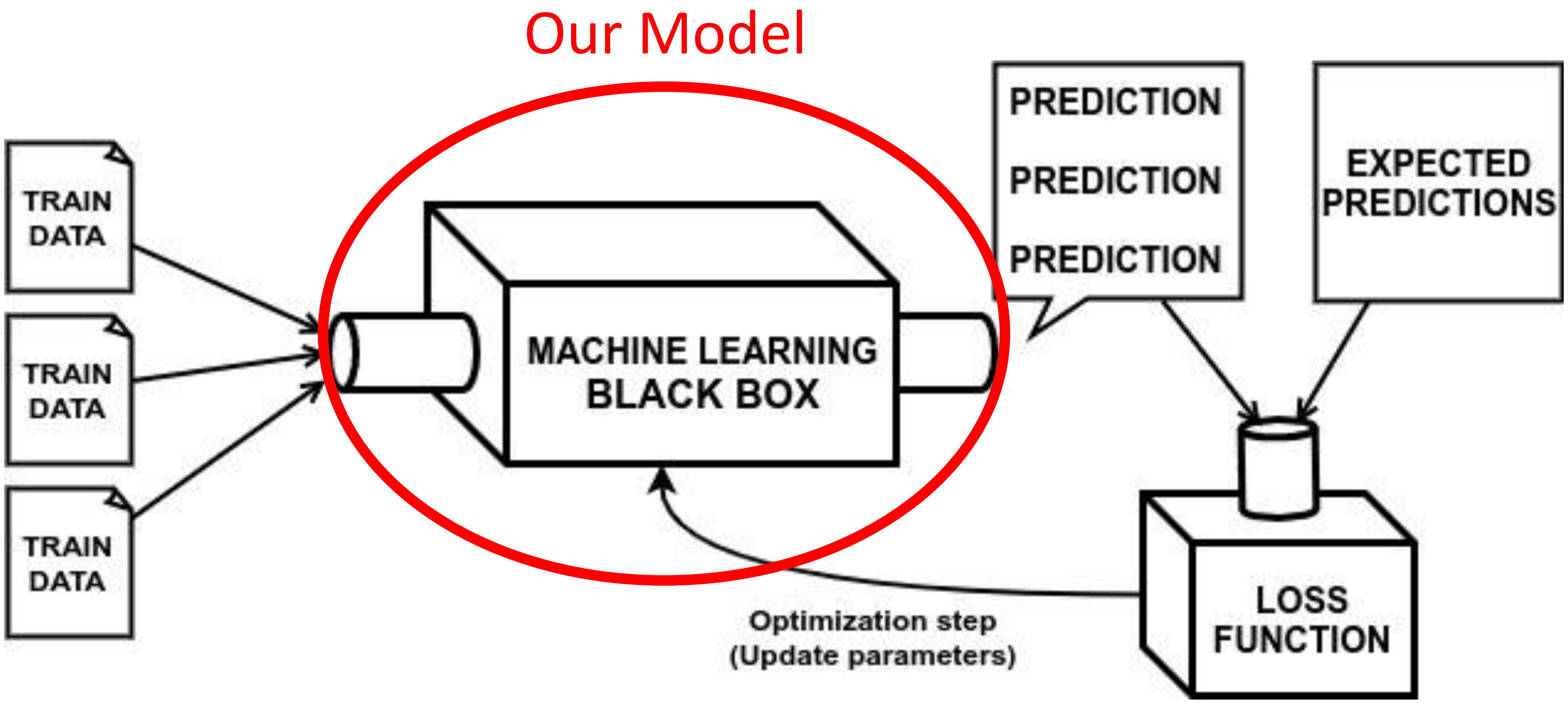
Machine Learning Algorithms

- We create a box that has needed input and output (called *model*)
- For classification problem, for example, an image and the number of a class
- We design a function that measures badness of this black box (called *loss function*)
- Given data we try to tweak or *model* to achieve the minimal score for *loss function*

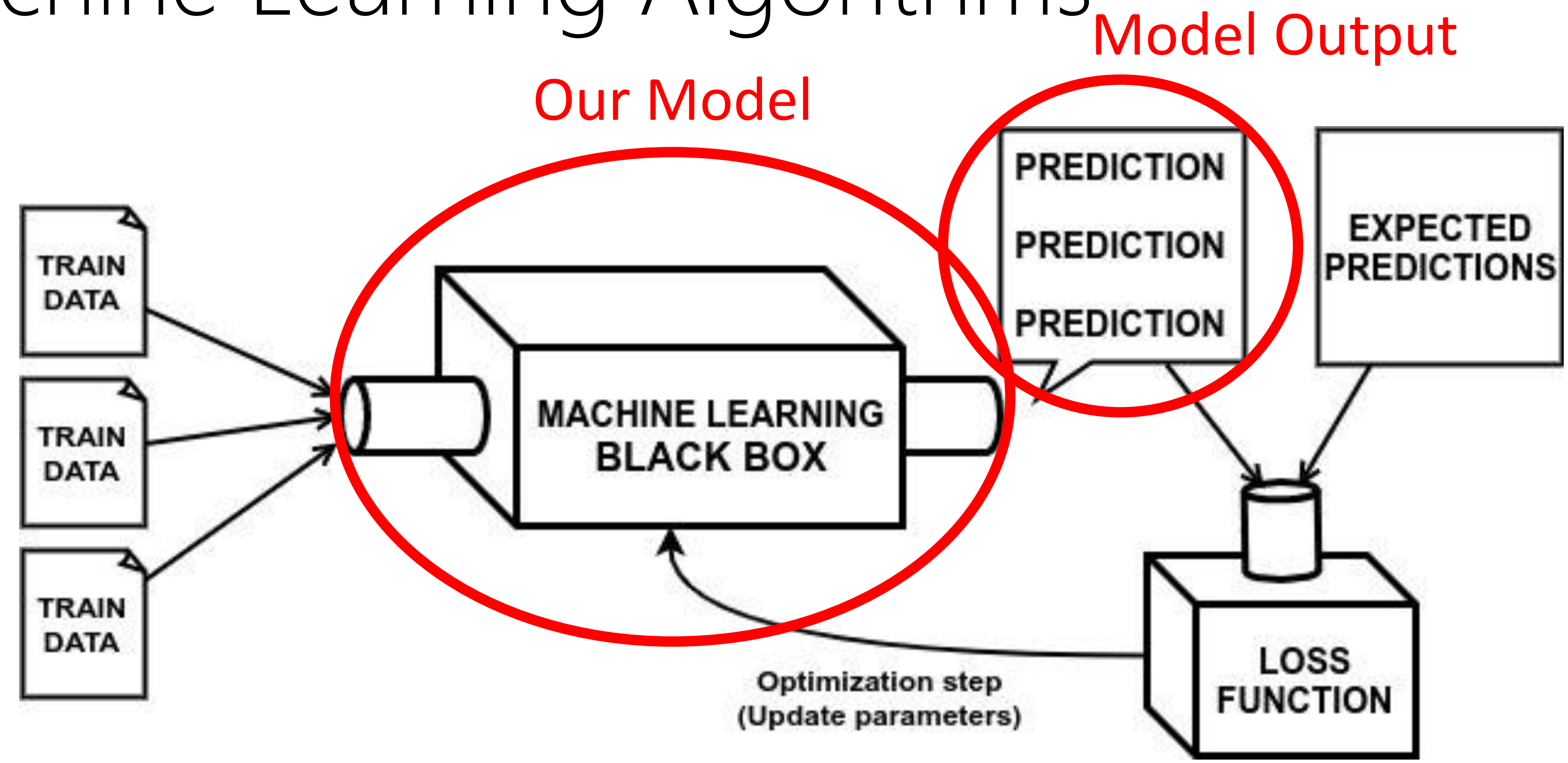
Machine Learning Algorithms



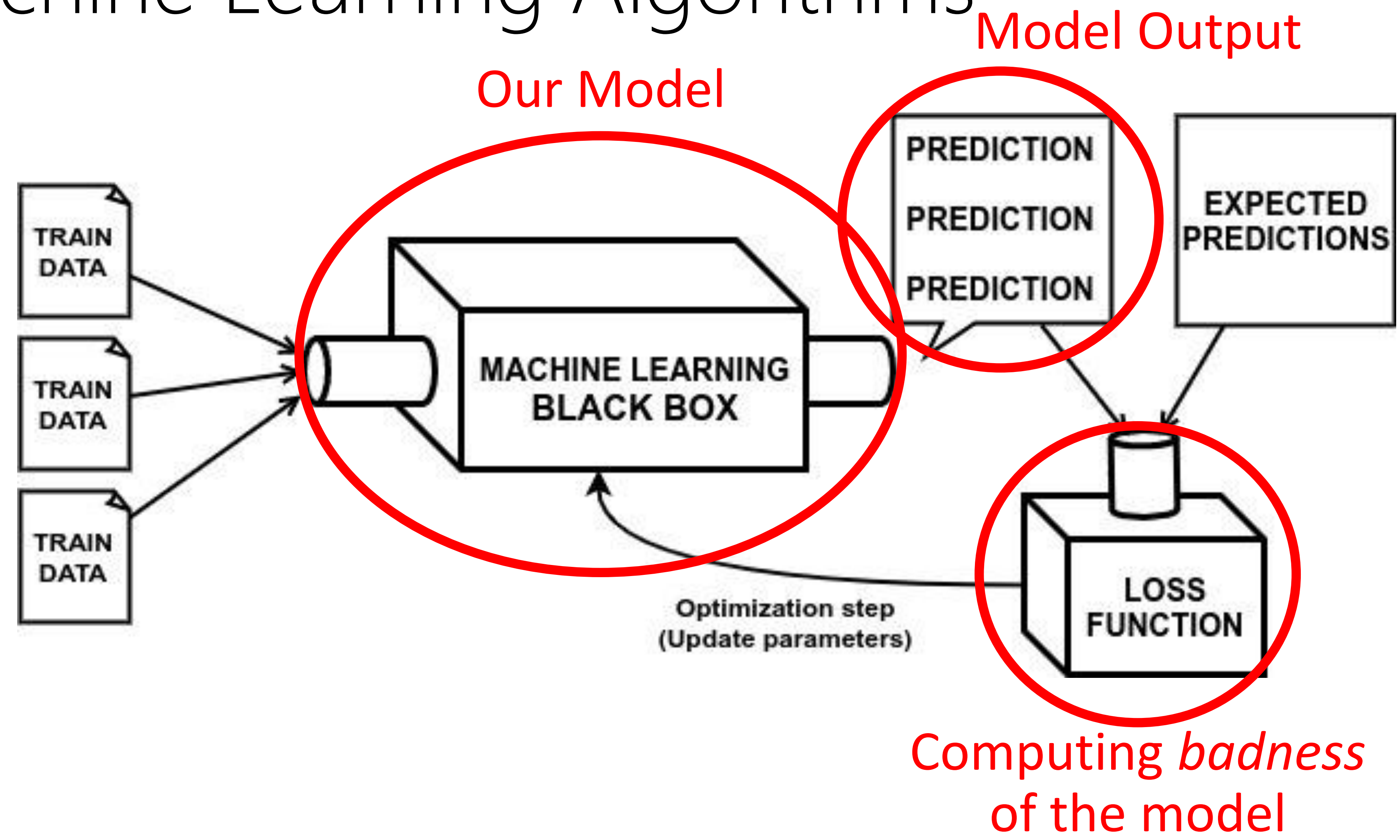
Machine Learning Algorithms



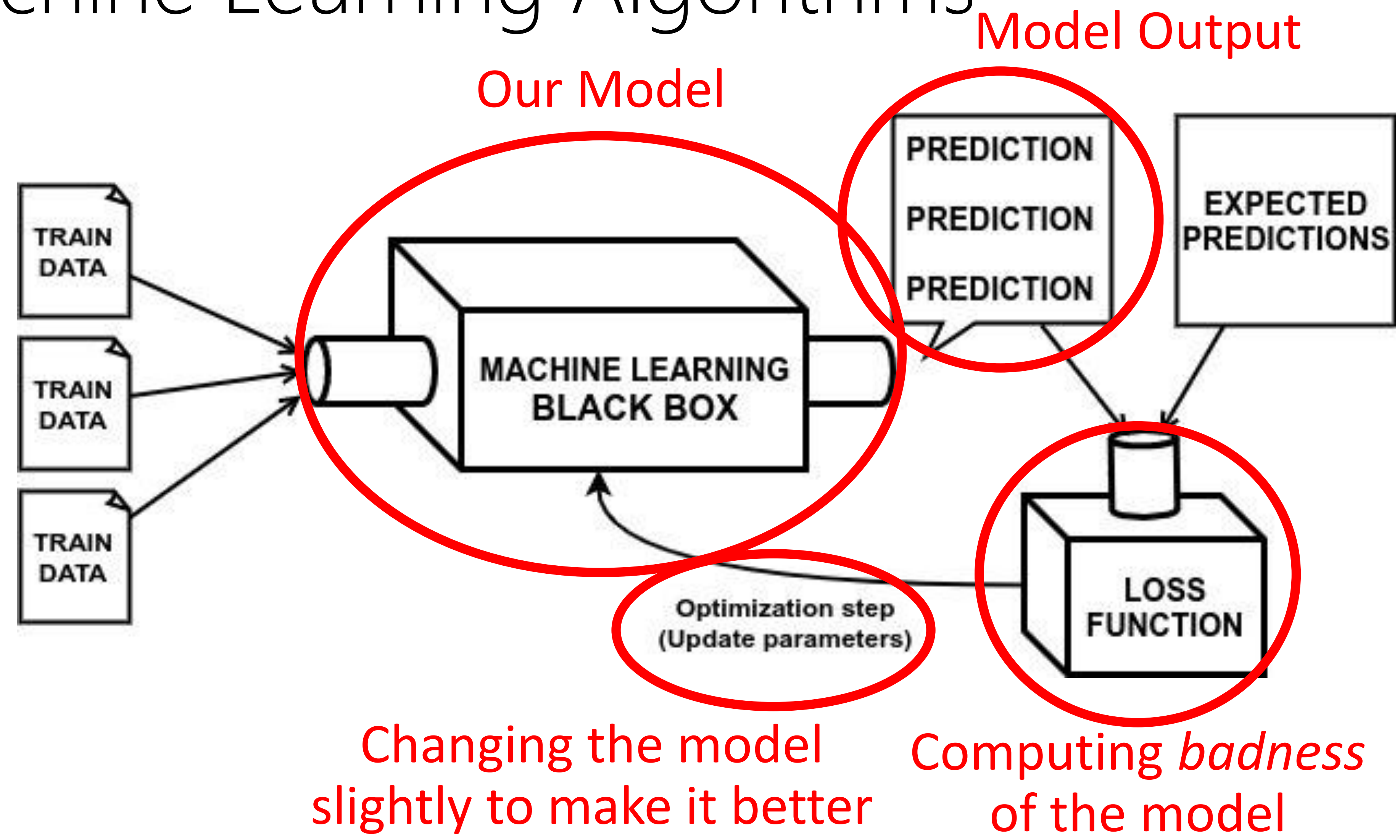
Machine Learning Algorithms



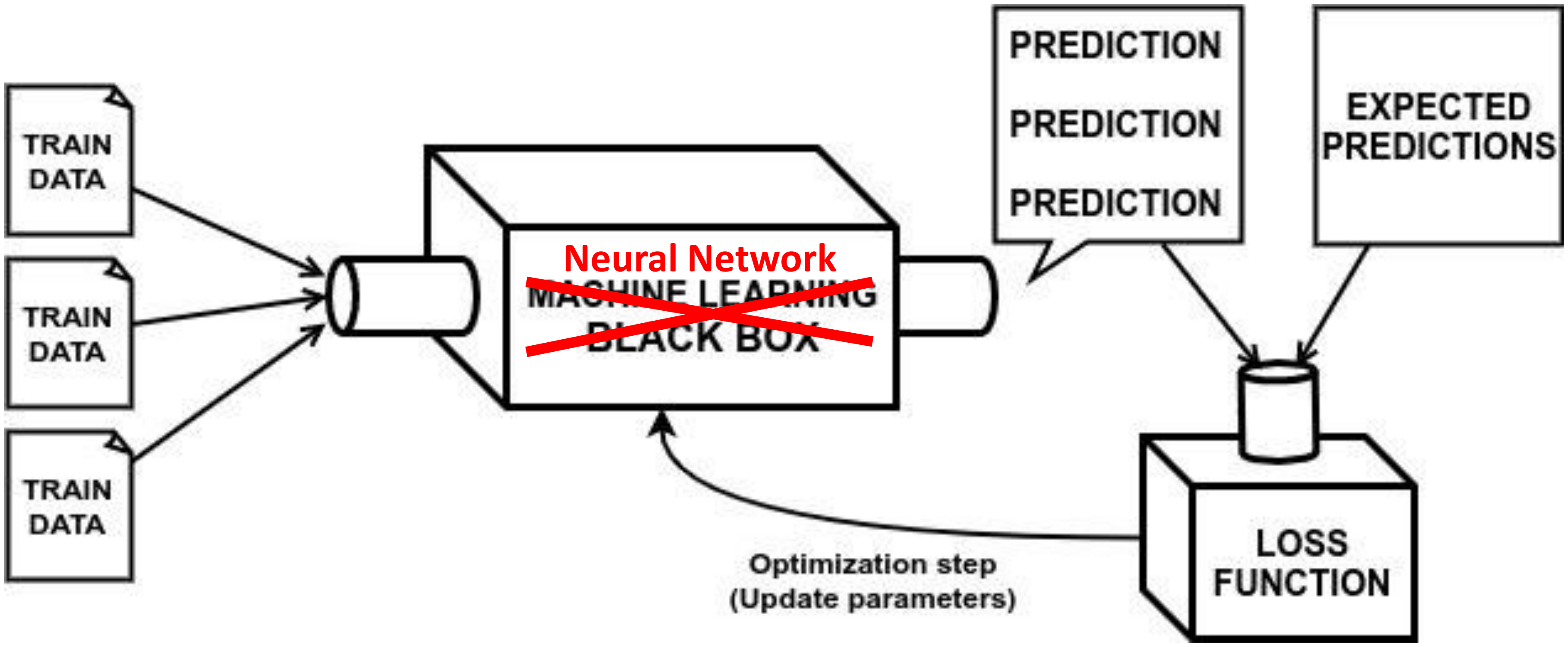
Machine Learning Algorithms



Machine Learning Algorithms

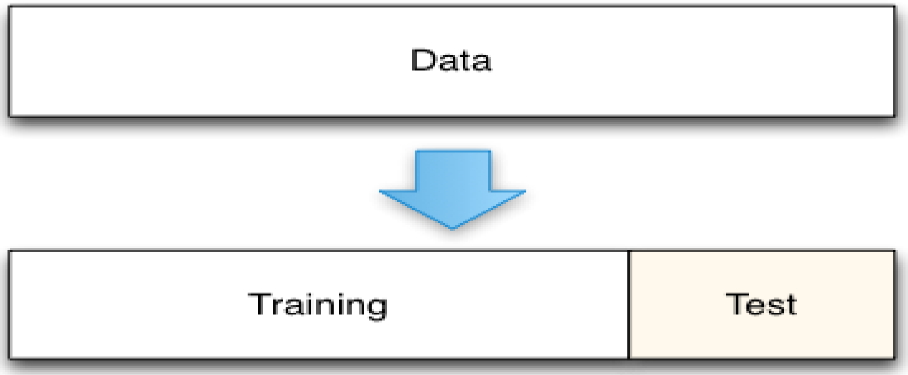


Neural Network as a ML Black Box

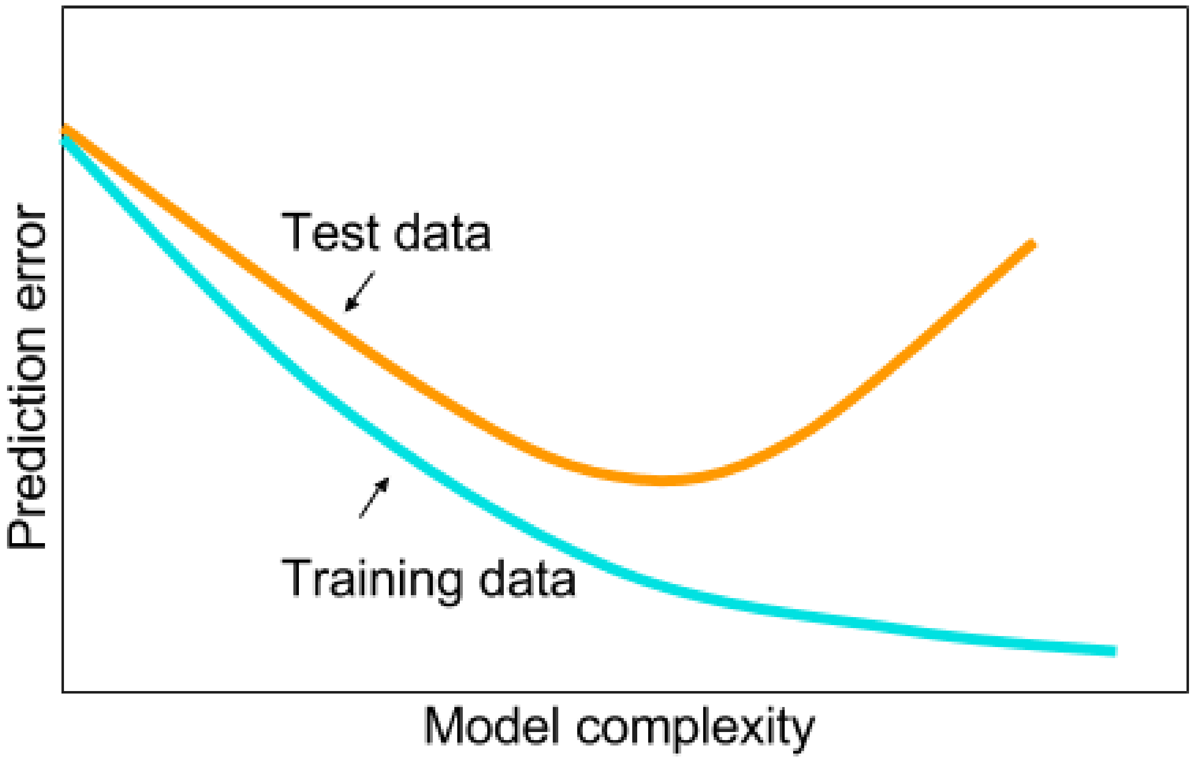


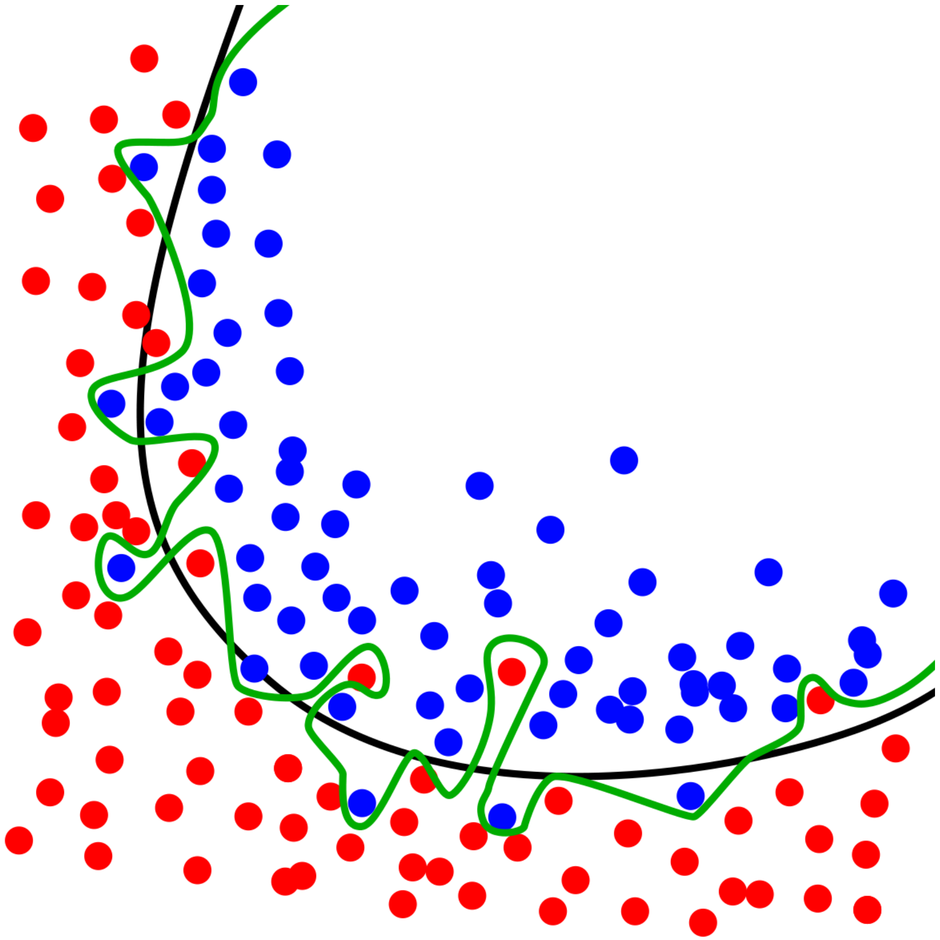
A Bit About Data

- To train a model we use data
- Models sometimes become really complex so they just memorize dataset instead of generalizing the problem
- To spot this we perform splitting data to *training* and *testing* data
- We train model using *training data* and measure score on *testing data*



Overfitting





What Types of Problems Can NNs Solve?

Most popular ones:

- Classification (output one of the predefined classes $\{0, 1, \ldots, n\}$)
- Regression (output real value) (e.g. predict size of something)

- Many tasks are just combinations of these two
- There are exceptions (but we are not going that deep)

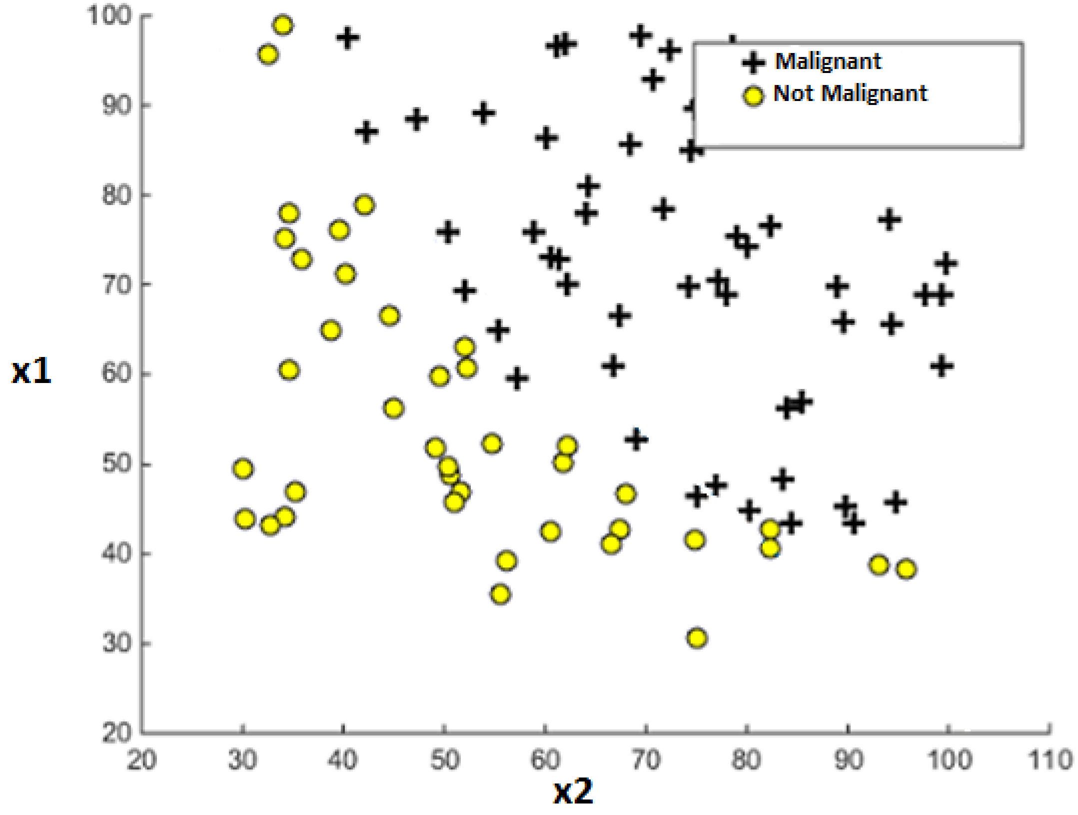
Logistic Regression Algorithm

Logistic Regression

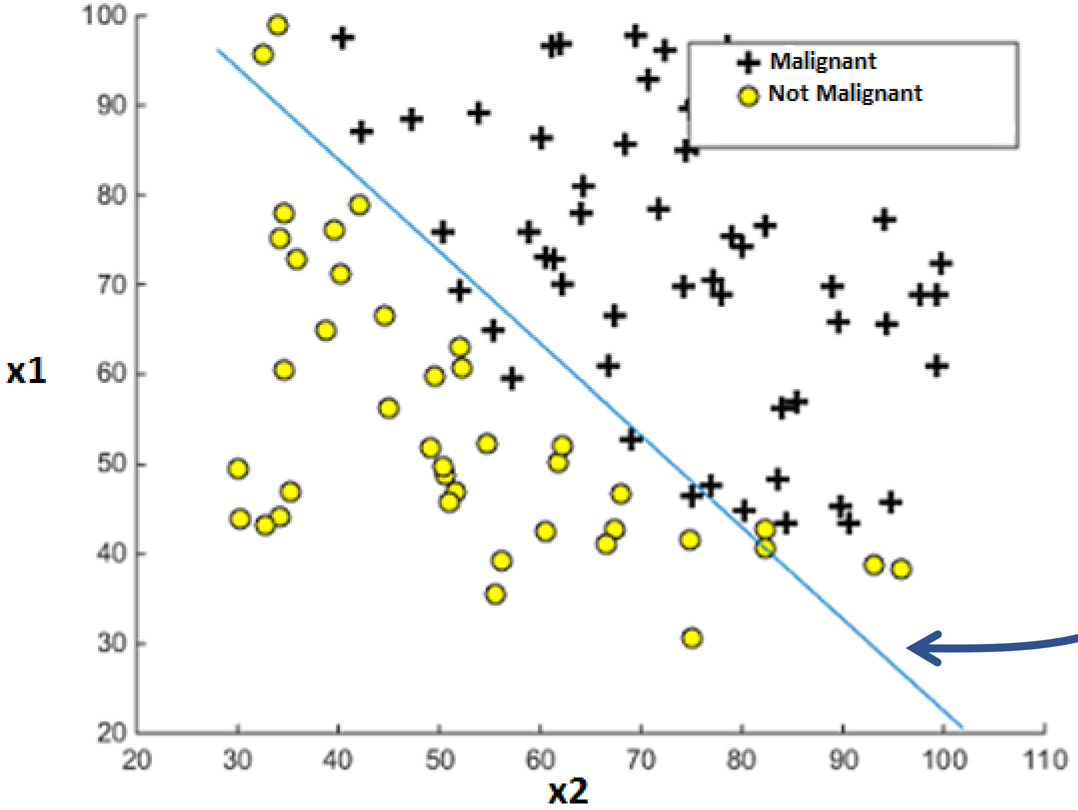
- A simple binary classification algorithm
- Output: 0 or 1.
- Input: data with n parameters

- Example: prediction of tumor malignancy
- Output: *not malignant (0)* or *malignant (1)*
- Input: data with parameters: *tumor size*, *age*, *tumor age*…

Linear Data Separation



Linear Data Separation



Separating data with a line

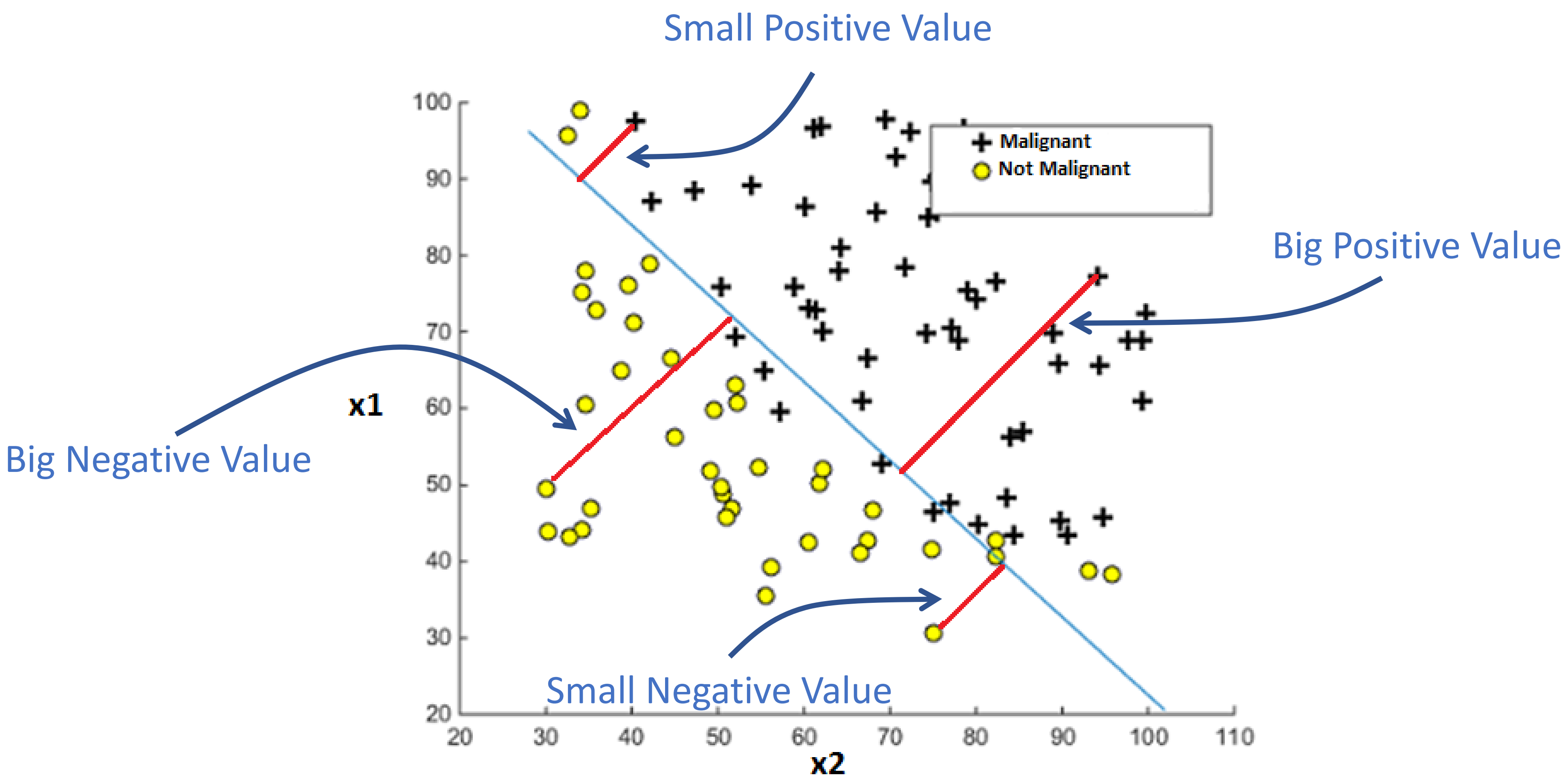
Linear Data Separation

- We create equation for the line:
$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b = 0$$

- To measure distance from this line we can use
$$W(x) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

- Sign of $W(x)$ would say what class the data point has

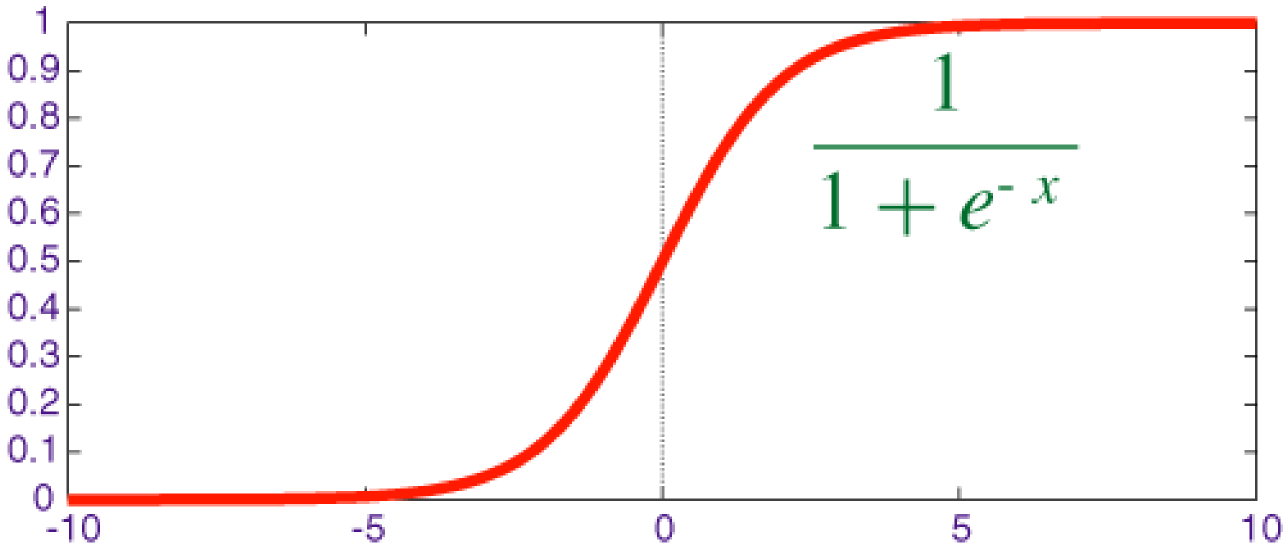
Linear Data Separation



Linear Data Separation

- We can transform distance from the line to *probability* of a data point having a positive class
- We can use *sigmoid* function to do that

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Logistic Regression

More strict way

- Input: $x = x_1, x_2, \dots, x_n$ (just n parameters for an object)
- Output: $f(x) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) \in [0; 1]$
- Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
- For final decision $\{0, 1\}$ we can just round the output:
  - 0 if $f(x) < 0.5$
  - 1 if$f(x) \geq 0.5$

Logistic Regression – Closer Look

Input Data Parameters (*tumor size*, *age*, …)

Sigmoid Function

$$f(x) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)$$

Learned parameters of the models (so called *weights*)

This special weight is called *bias*

How Do Measure Badness?

- We need an objective function (loss function)
- Cross Entropy:

$$L(y, f(x)) = -\frac{1}{n}\sum_{i}^{n}[y_i \log\big(f(x)\big) + (1 - y_i)\log\big(1 - f(x)\big)]$$

- (Derived from Maximum Likelihood Estimation)

Looking Closer on the Objective

$$L(y, f(x)) = -\frac{1}{n}\sum_i^n [y_i \log(f(x)) + (1 - y_i)\log(1 - f(x))]$$

and $f(x) \in [0; 1]$

$f(x)$ better be close to 1

If $y_i = 1$

$$\overbrace{y_i}^{1} \log(f(x)) + \underbrace{(1 - y_i)}_{0} \log(1 - f(x))$$

If $y_i = 0$

$$\underbrace{y_i}_{0} \log(f(x)) + \underbrace{(1 - y_i)}_{1} \log(1 - f(x))$$

$f(x)$ better be close to 0

How Do We Optimize This?

- We have an objective $L(y, f(x))$ where
$$f(x) = \sigma(w_1 x_1 + \cdots + w_n x_n + b)$$

- We can compute derivative of this function with respect to model parameters: $L'\big(y, f(x)\big)_{w,b}$
- Computed derivatives called *gradient*

Computing Derivatives

- Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

$$\sigma'(x) = \sigma(x)\big(1-\sigma(x)\big)x'$$

Thus

$$\sigma'_{w_1}(\dots) = \sigma(\dots)\big(1-\sigma(\dots)\big)w_1$$
$$\sigma'_{b}(\dots) = \sigma(\dots)\big(1-\sigma(\dots)\big)$$

- Logarithm function: $\log(x)$

$$\log{}'(x) = \frac{1}{x}x'$$

Optimization Procedure

- Okay, now we have derivatives with respect to parameters depending on our training data
- Let's make an arbitrary model (i.e. initialize *weights* randomly)
- Let's compute our loss $L(y, f(x))$
- Then the derivative $L'\big(y, f(x)\big)_{w,b}$
- Then for each weight let's tweak it a little bit so our loss function becomes slightly better

$$w_i \coloneqq w_i - \alpha L'\big(y, f(x)\big)_{w_i}$$

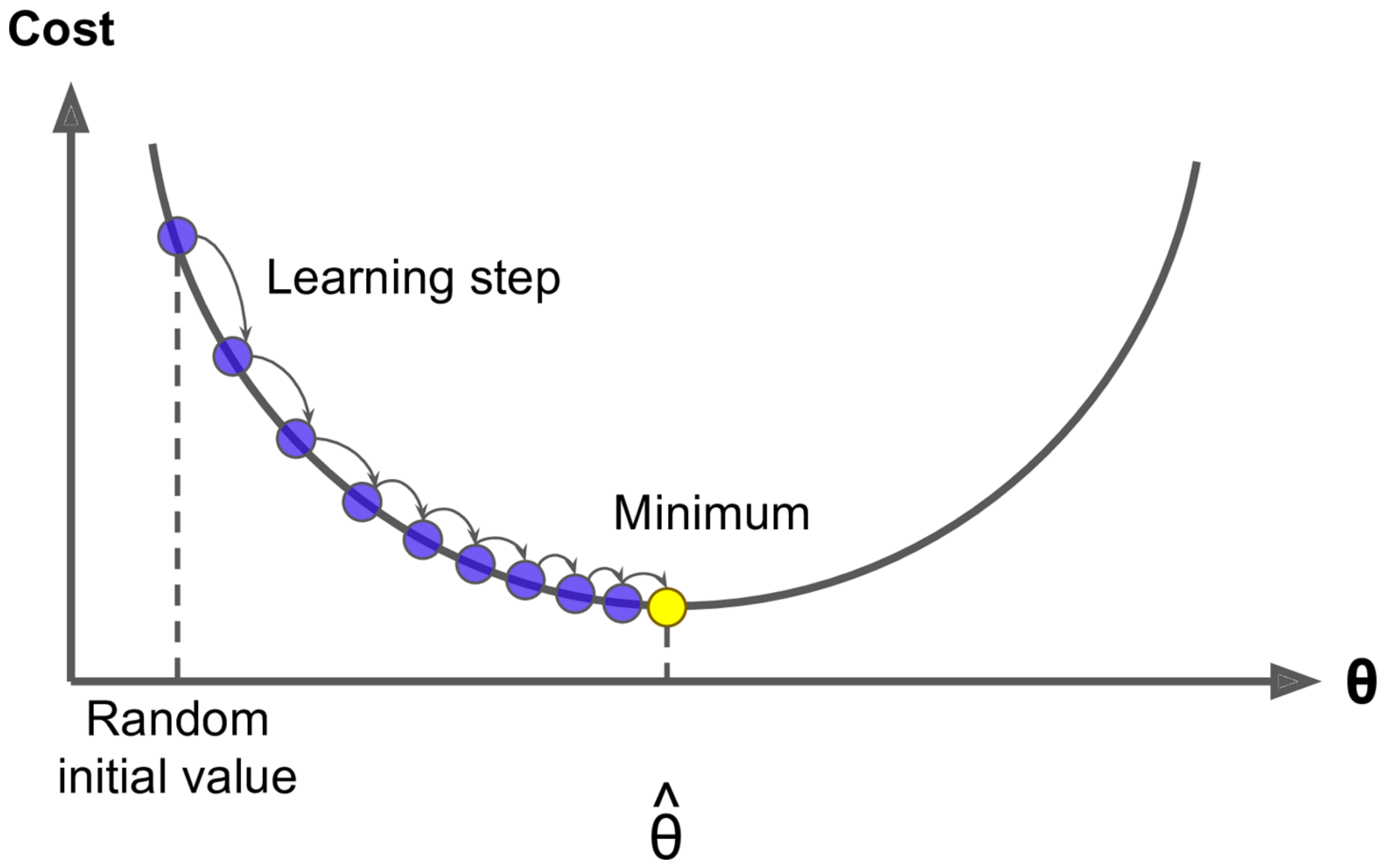
Learning rate – how strong should be an optimization step

Optimization Procedure

1. Compute loss
2. Compute the gradient for this loss w.r.t. the model parameters
3. Update the model a little bit towards gradients
4. Go to 1

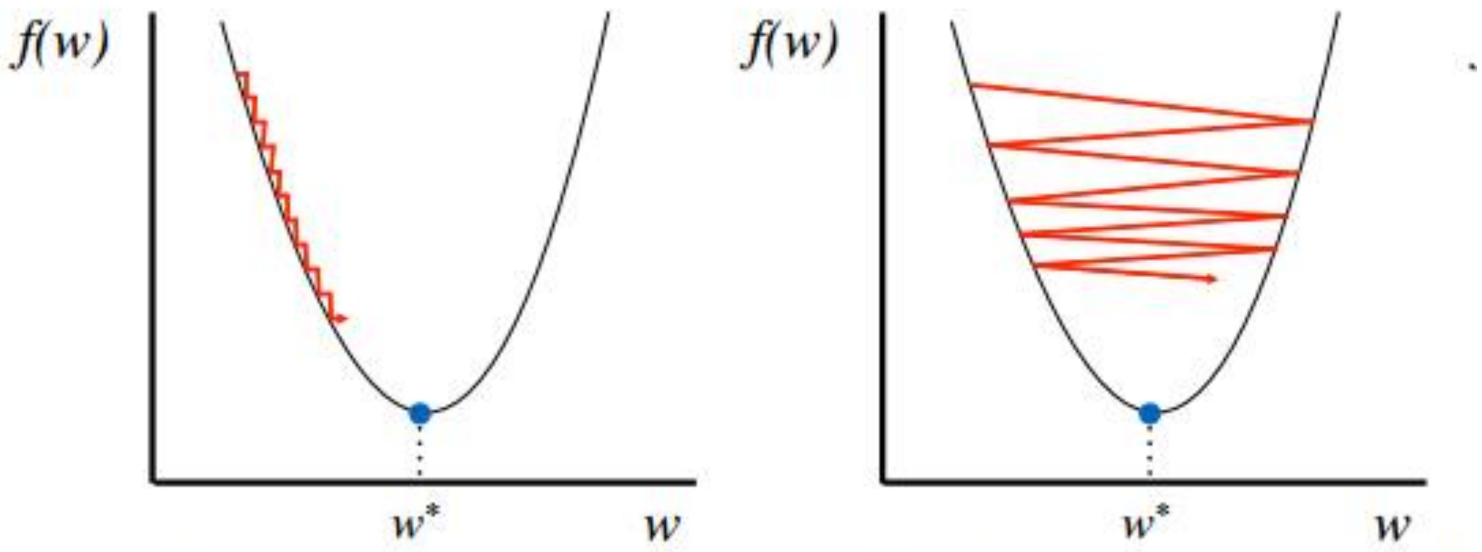
- This is called *gradient descent*

Gradient Descent Intuition



Learning Rate

- *Learning rate* is a parameter you need to adjust wisely



Too small: converge very slowly

Too big: overshoot and even diverge

Logistic Regression Recap

- Model output (0.5 value is a decision boundary)

$$f(x) = \sigma(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)$$

- Loss function (how bad things are)

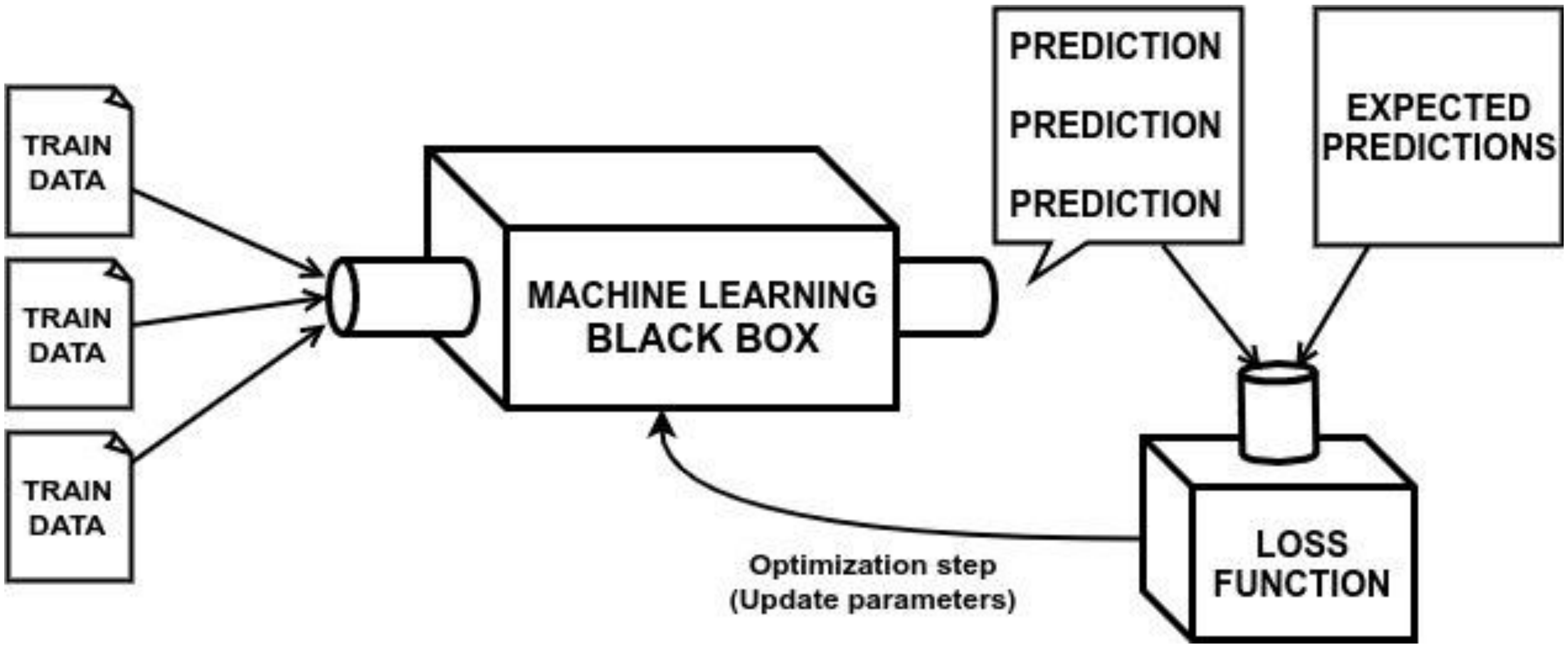
$$L(y, f(x)) = -\frac{1}{n}\sum_i^n [y_i \log\big(f(x)\big) + (1 - y_i) \log\big(1 - f(x)\big)]$$

- We update parameters computing gradient and slightly changing the model

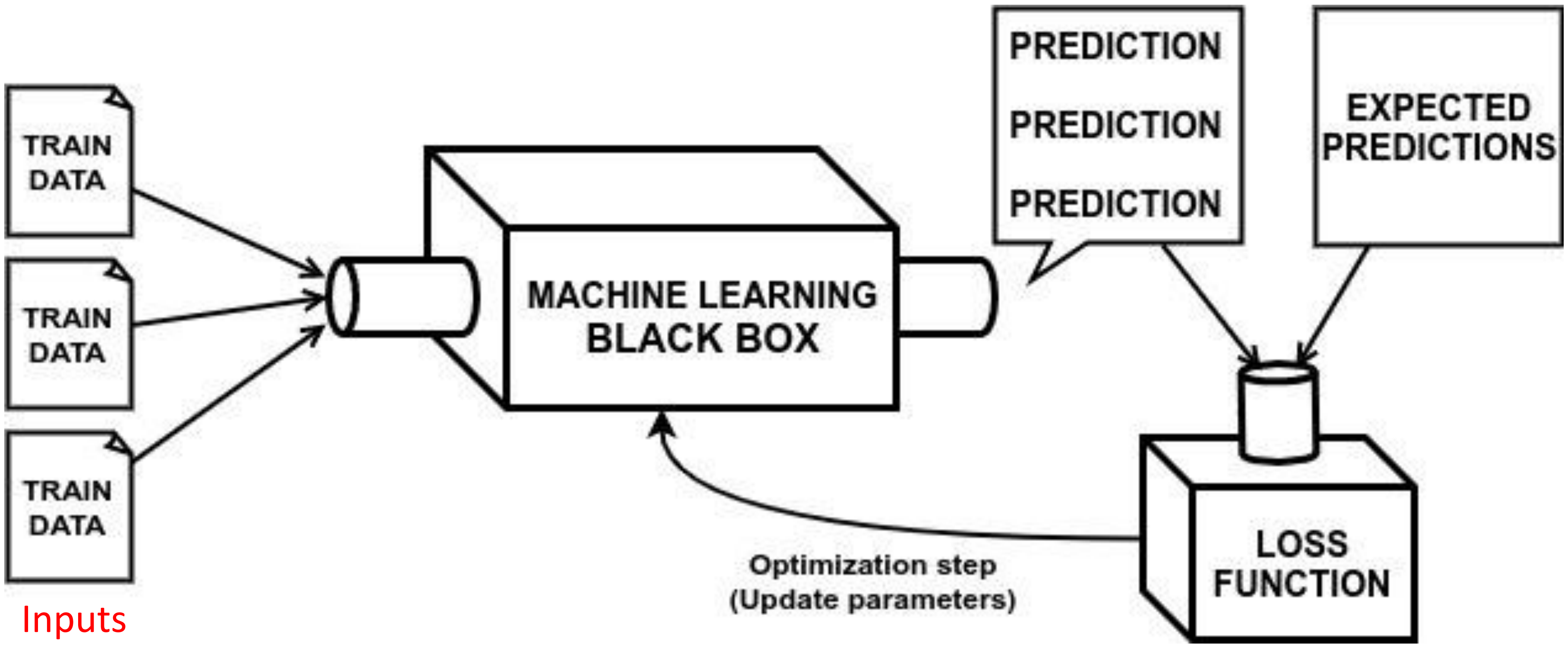
$$w_i \coloneqq w_i - \alpha L'\big(y, f(x)\big)_{w_i}$$

- Hoping the model will optimize itself in the best possible way

Our Scheme


TRAIN DATA
TRAIN DATA
TRAIN DATA
MACHINE LEARNING
BLACK BOX
PREDICTION
PREDICTION
PREDICTION
EXPECTED PREDICTIONS
LOSS FUNCTION
Optimization step
(Update parameters)

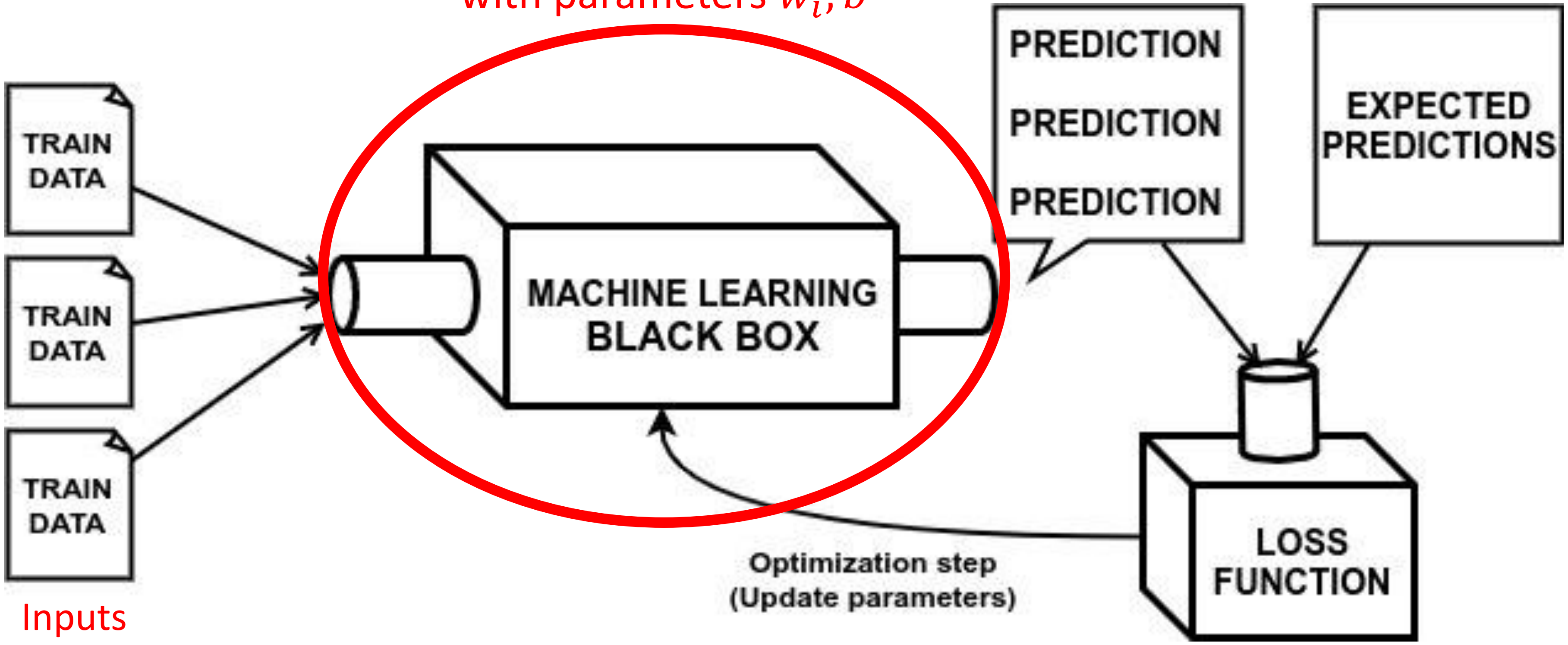
Our Scheme



Inputs
x_i

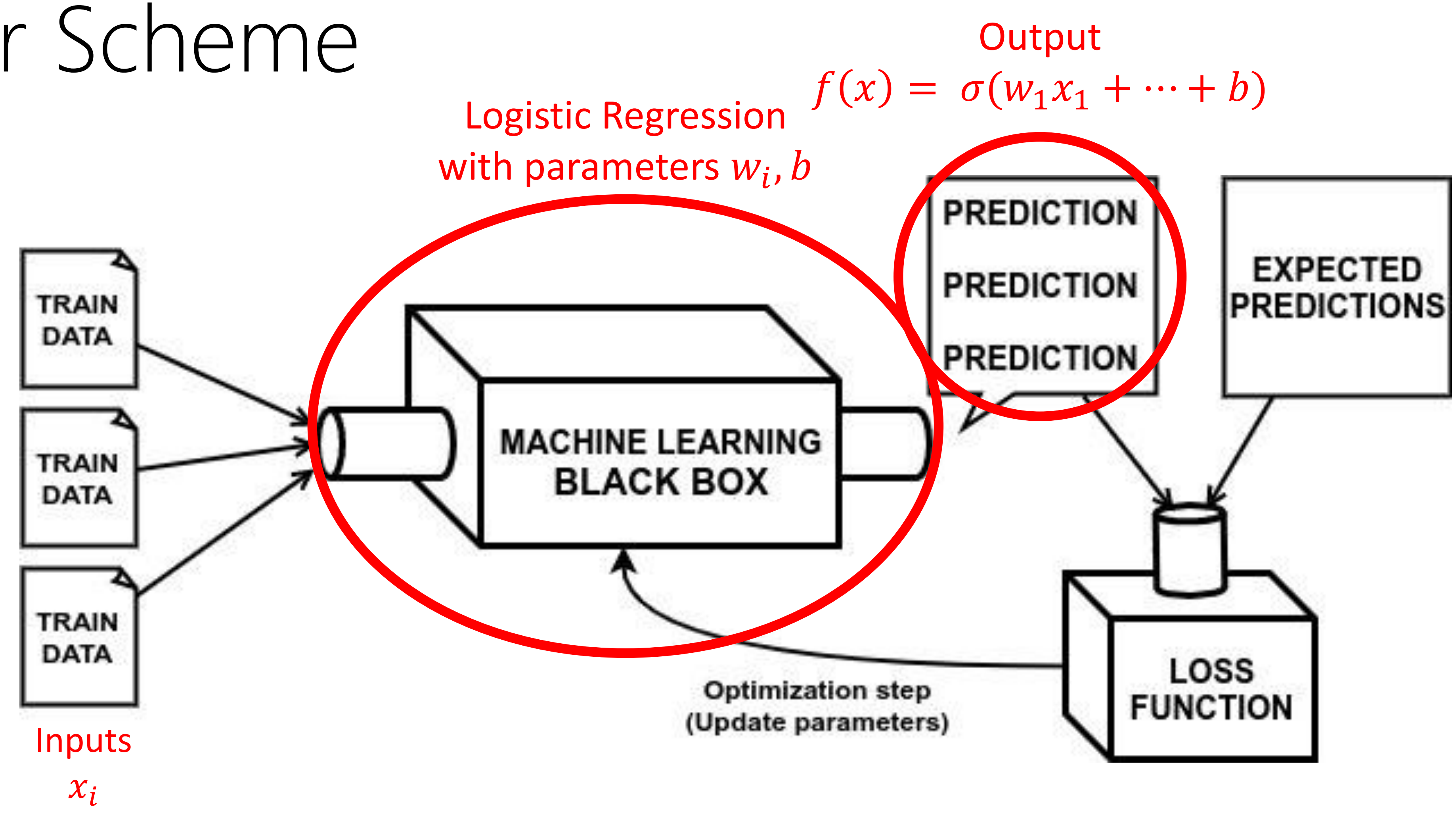
Our Scheme

Logistic Regression
with parameters w_i, b

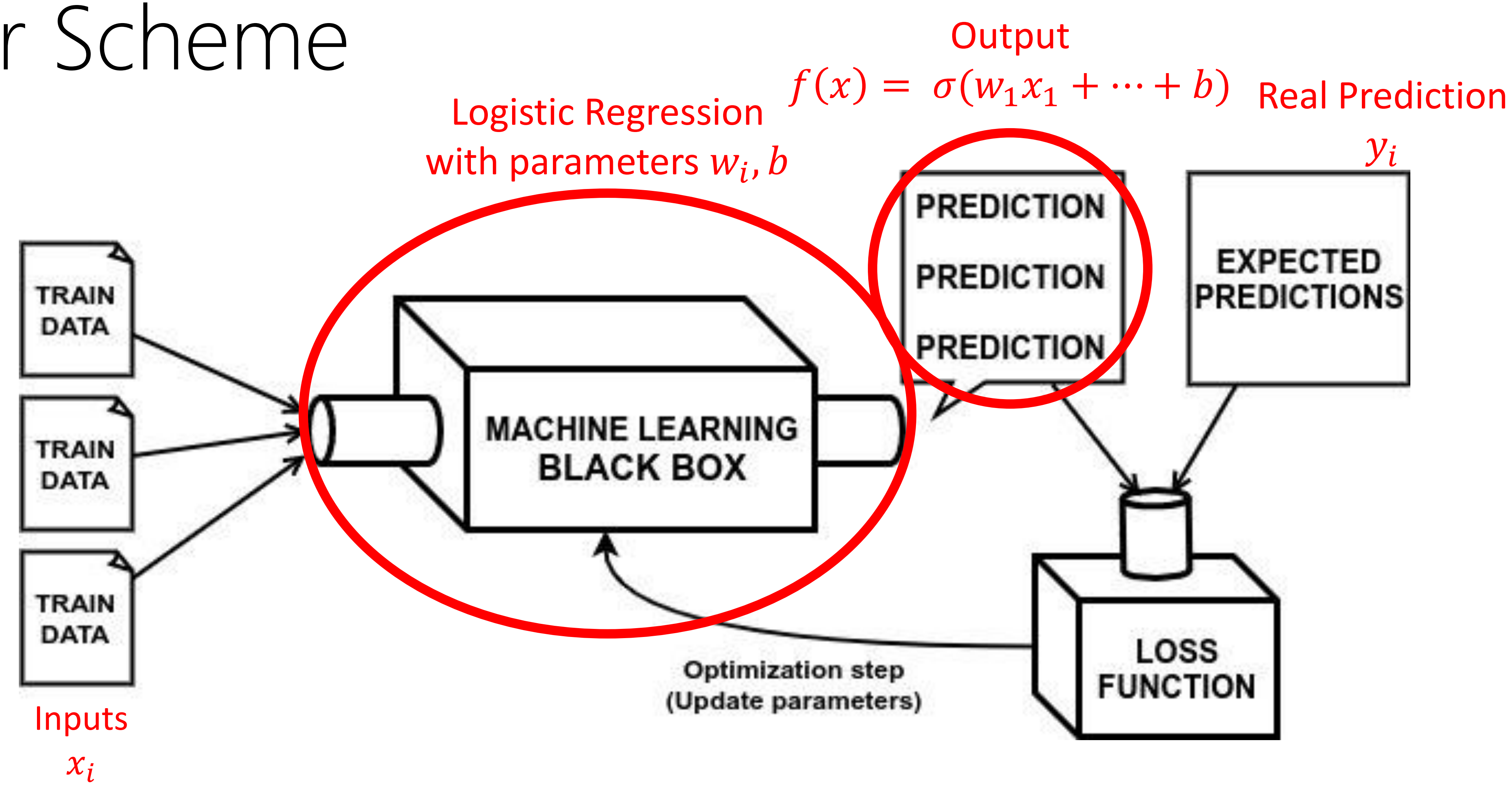


Inputs
x_i

Our Scheme

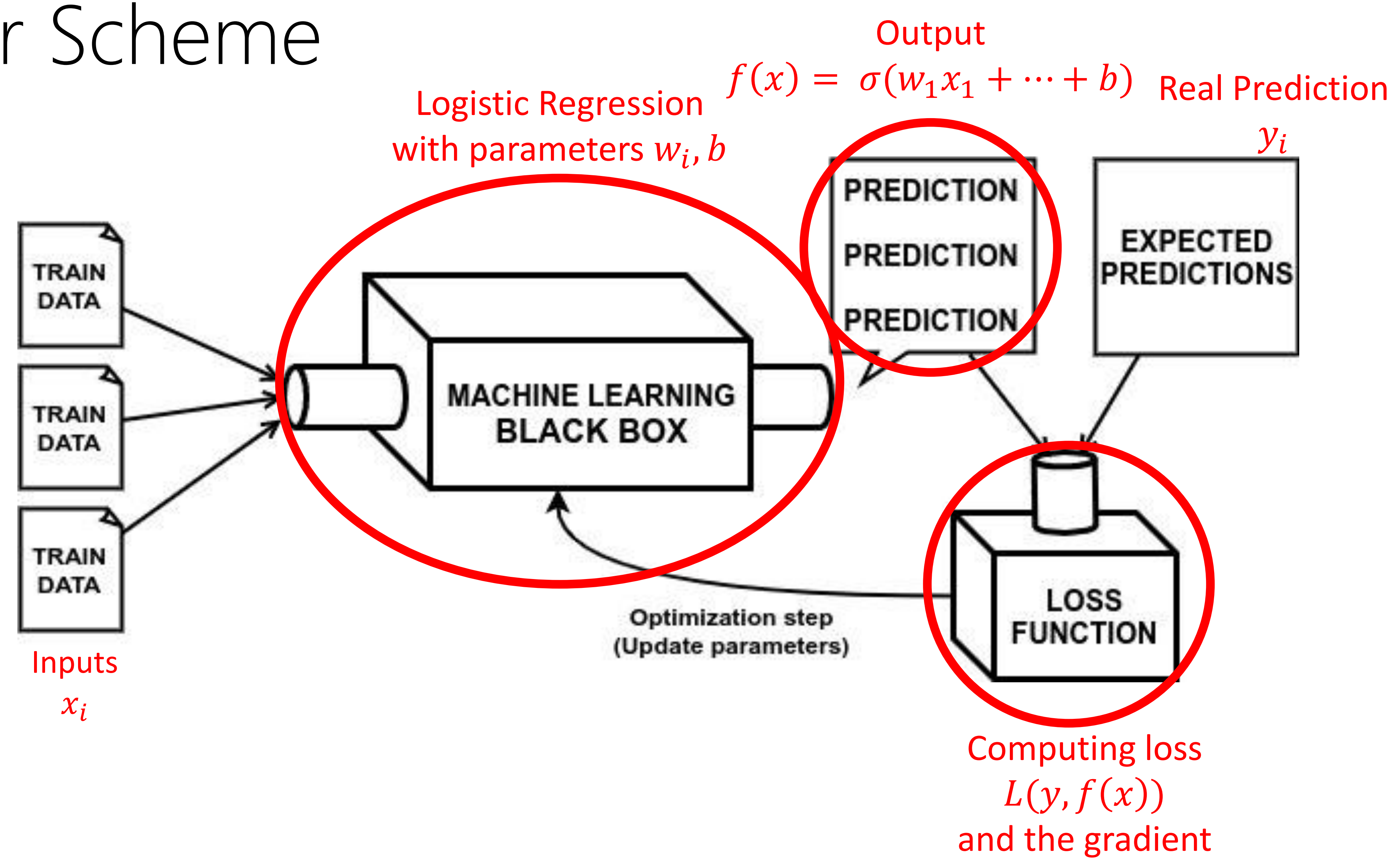

Output
$f(x) = \sigma(w_1 x_1 + \cdots + b)$
Logistic Regression
with parameters w_i, b
TRAIN DATA
TRAIN DATA
TRAIN DATA
MACHINE LEARNING BLACK BOX
PREDICTION
PREDICTION
PREDICTION
EXPECTED PREDICTIONS
LOSS FUNCTION
Optimization step
(Update parameters)
Inputs
x_i

Our Scheme



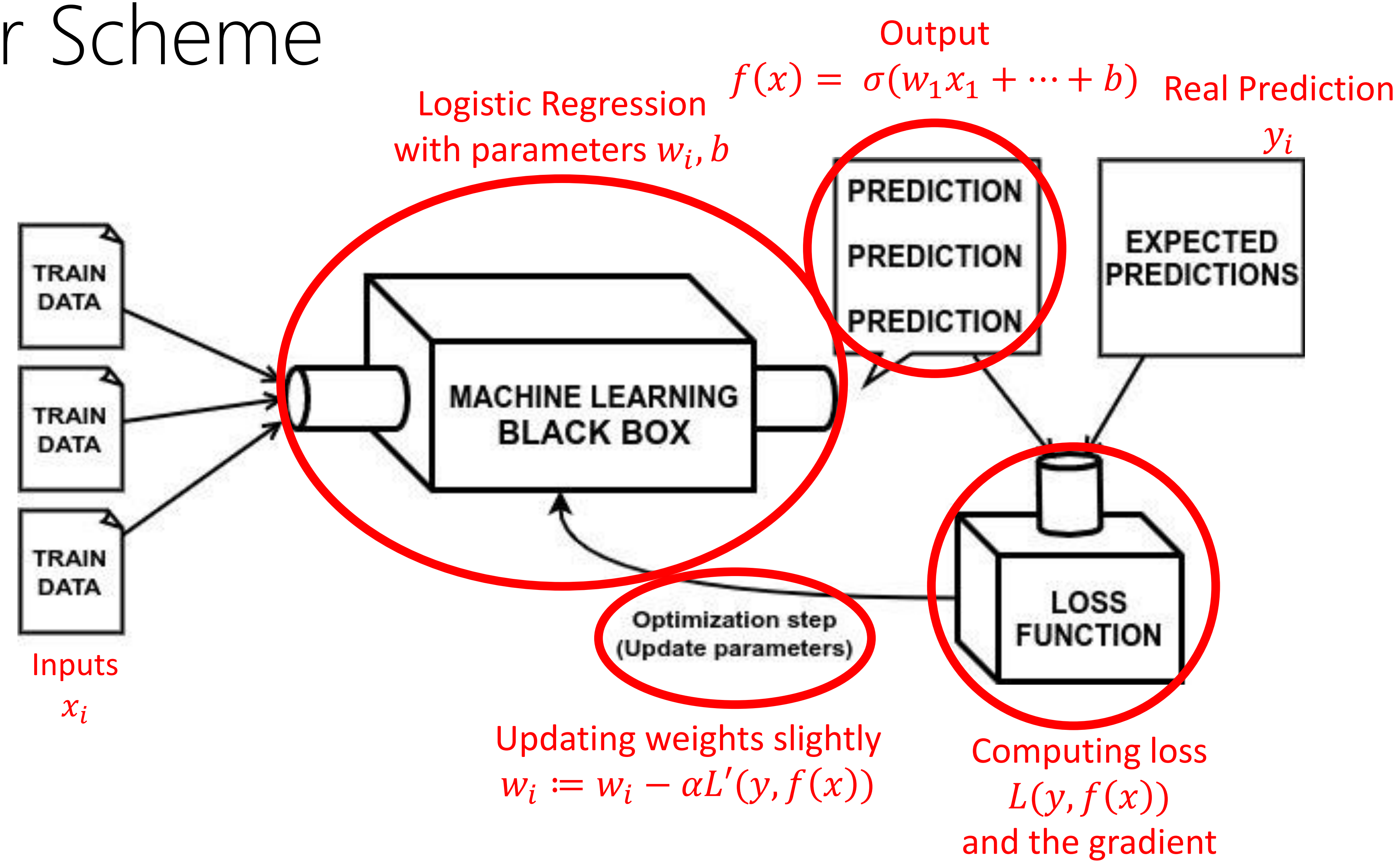
Output
$f(x) = \sigma(w_1 x_1 + \cdots + b)$
Real Prediction
y_i
Logistic Regression
with parameters w_i, b
PREDICTION
PREDICTION
PREDICTION
EXPECTED PREDICTIONS
TRAIN DATA
TRAIN DATA
TRAIN DATA
MACHINE LEARNING BLACK BOX
LOSS FUNCTION
Optimization step
(Update parameters)
Inputs
x_i

Our Scheme



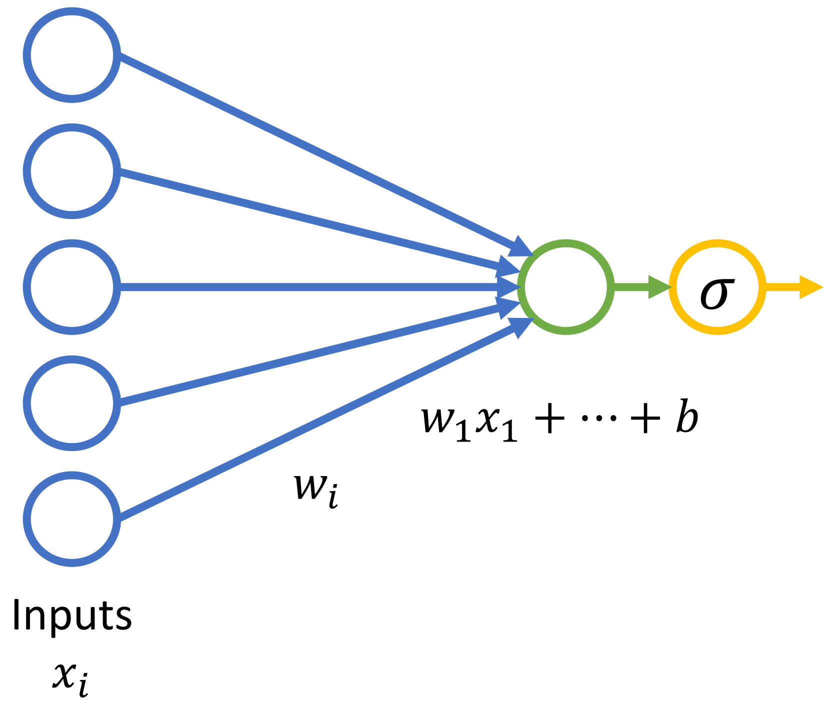
Output
$f(x) = \sigma(w_1 x_1 + \cdots + b)$
Real Prediction
y_i
Logistic Regression
with parameters w_i, b
PREDICTION
PREDICTION
PREDICTION
EXPECTED PREDICTIONS
TRAIN DATA
TRAIN DATA
TRAIN DATA
MACHINE LEARNING BLACK BOX
LOSS FUNCTION
Optimization step
(Update parameters)
Inputs
x_i
Computing loss
$L(y, f(x))$
and the gradient

Our Scheme

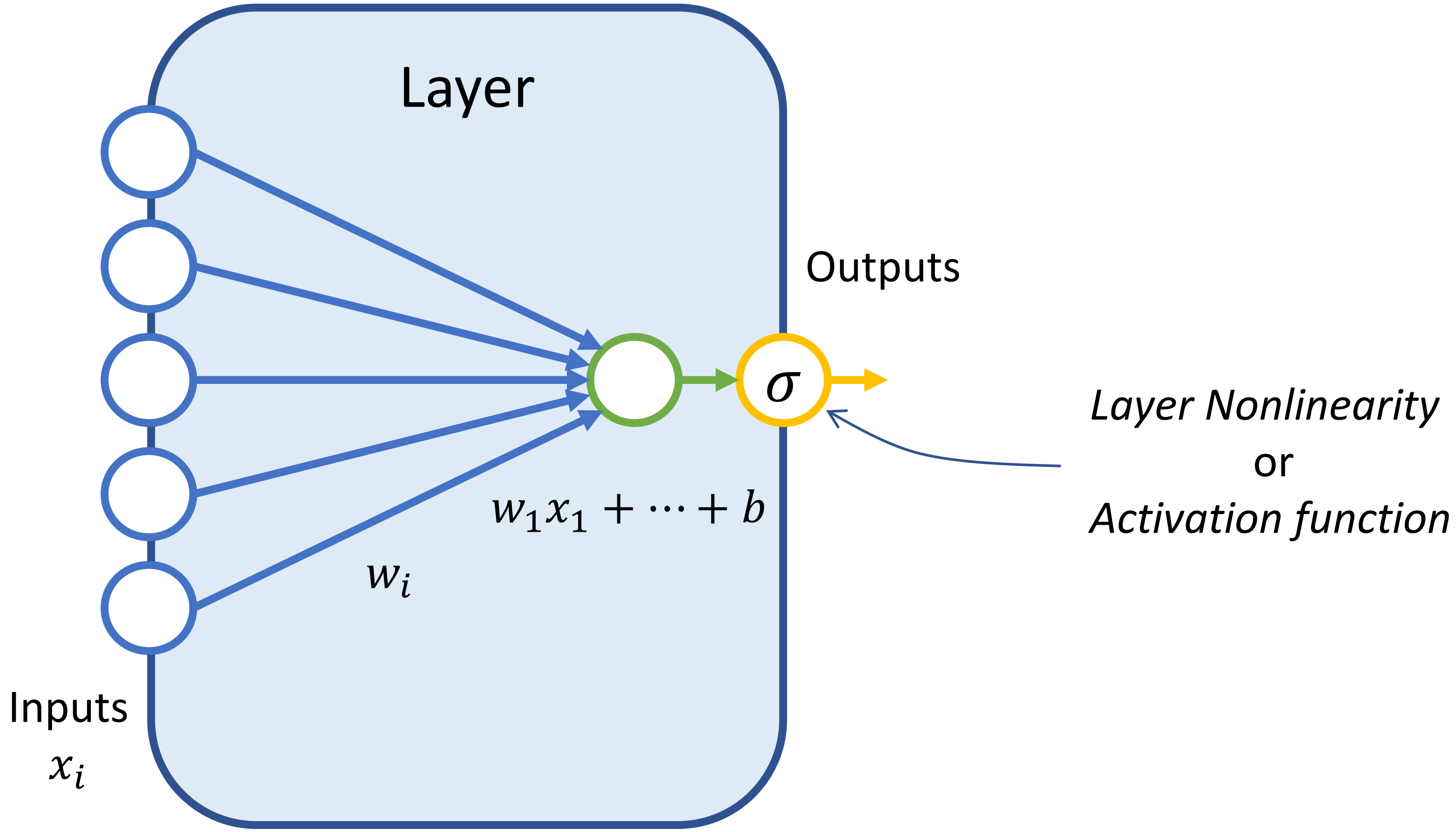


Output
$f(x) = \sigma(w_1 x_1 + \dots + b)$
Real Prediction
y_i
Logistic Regression
with parameters w_i, b
PREDICTION
PREDICTION
PREDICTION
EXPECTED PREDICTIONS
TRAIN DATA
TRAIN DATA
TRAIN DATA
MACHINE LEARNING BLACK BOX
LOSS FUNCTION
Optimization step
(Update parameters)
Inputs
x_i
Updating weights slightly
$w_i := w_i - \alpha L'(y, f(x))$
Computing loss
$L(y, f(x))$
and the gradient

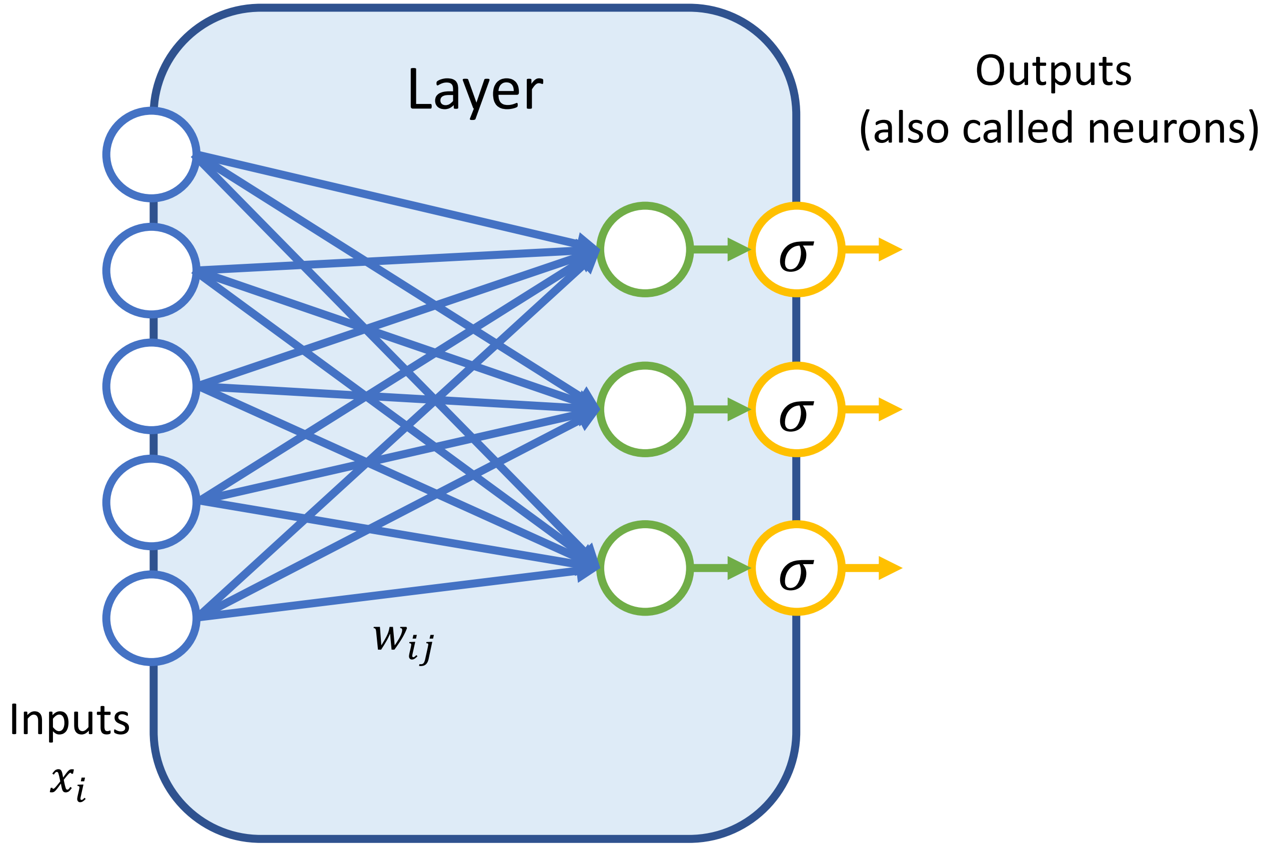
Logistic Regression Scheme



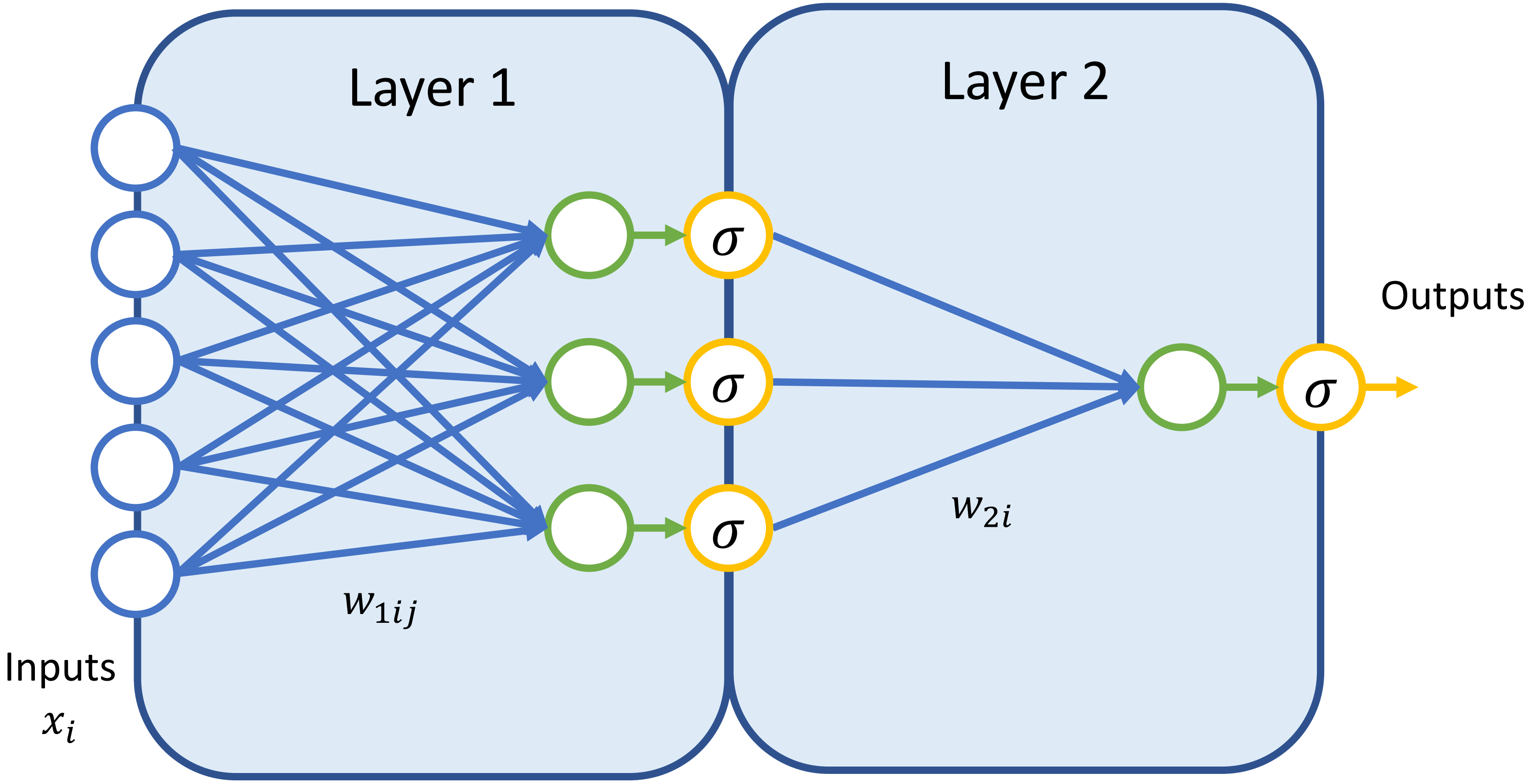
Logistic Regression as a 1-layer NN



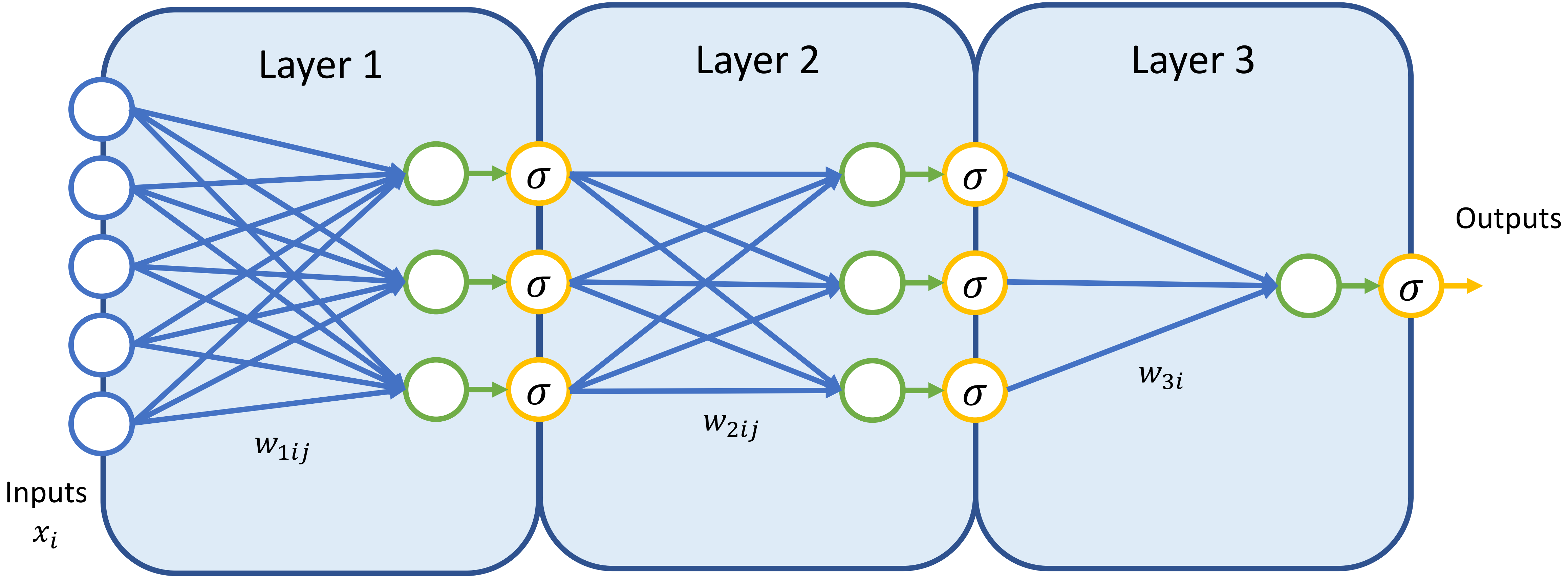
Multi-output Layer



Layers as Blocks



Layers as Blocks



Neural Networks

- You can stack many layers with various size of outputs together to achieve more complex models
- Don’t forget about *non-linearity* at the end of the layers (this is crucial)
- The deeper network – the more complex features *layers* can learn

Lecture Recap

- ML is really powerful nowadays and grows rapidly
- Splitting data into *train* and *validation* subsets to control overfitting
- Logistic Regression learns to separate data with a line (or a plane)
- Optimization of Logistic Regression with *gradient descent*
- Picking *learning rate* carefully
- Logistic Regression is a 1-layer Neural Network
- You can stack multiple layers to get deeper and more effective NNs

This is it for the first lecture