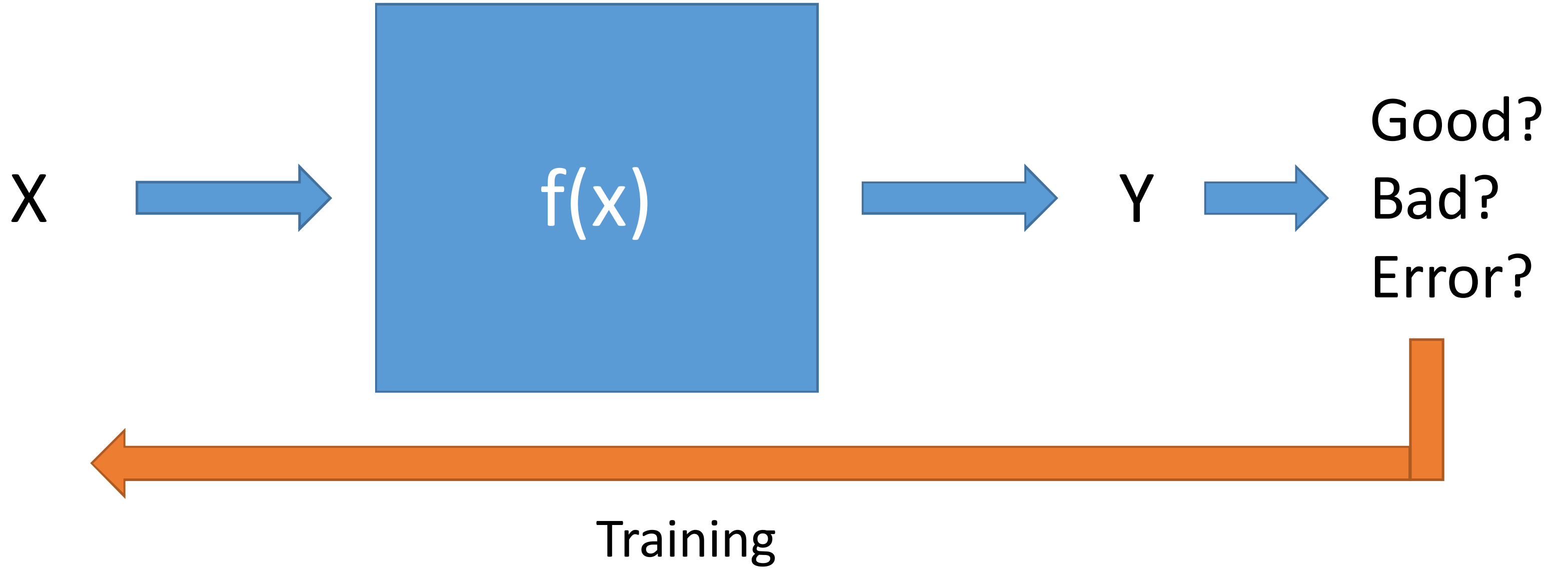
Machine Learning Theory and Frameworks

In less than two hours

Quick Recap: ML

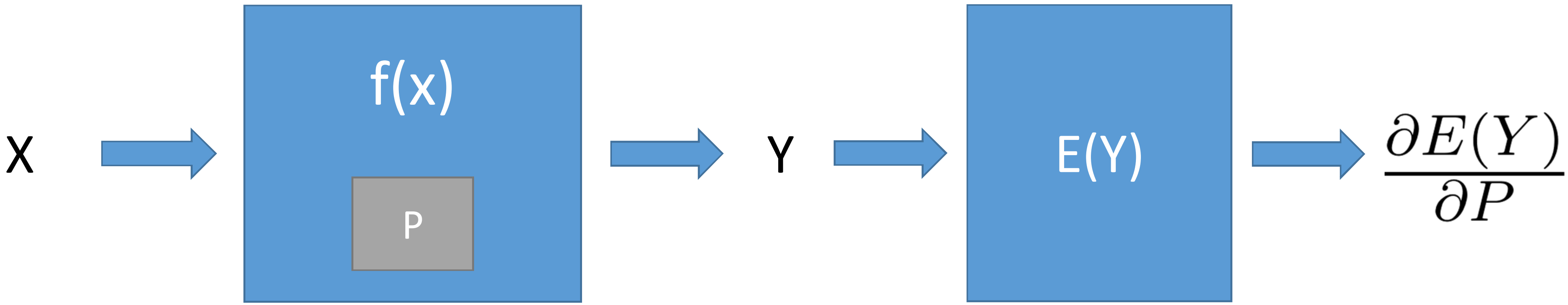
Dataset

X	Y



Quick Recap: SGD

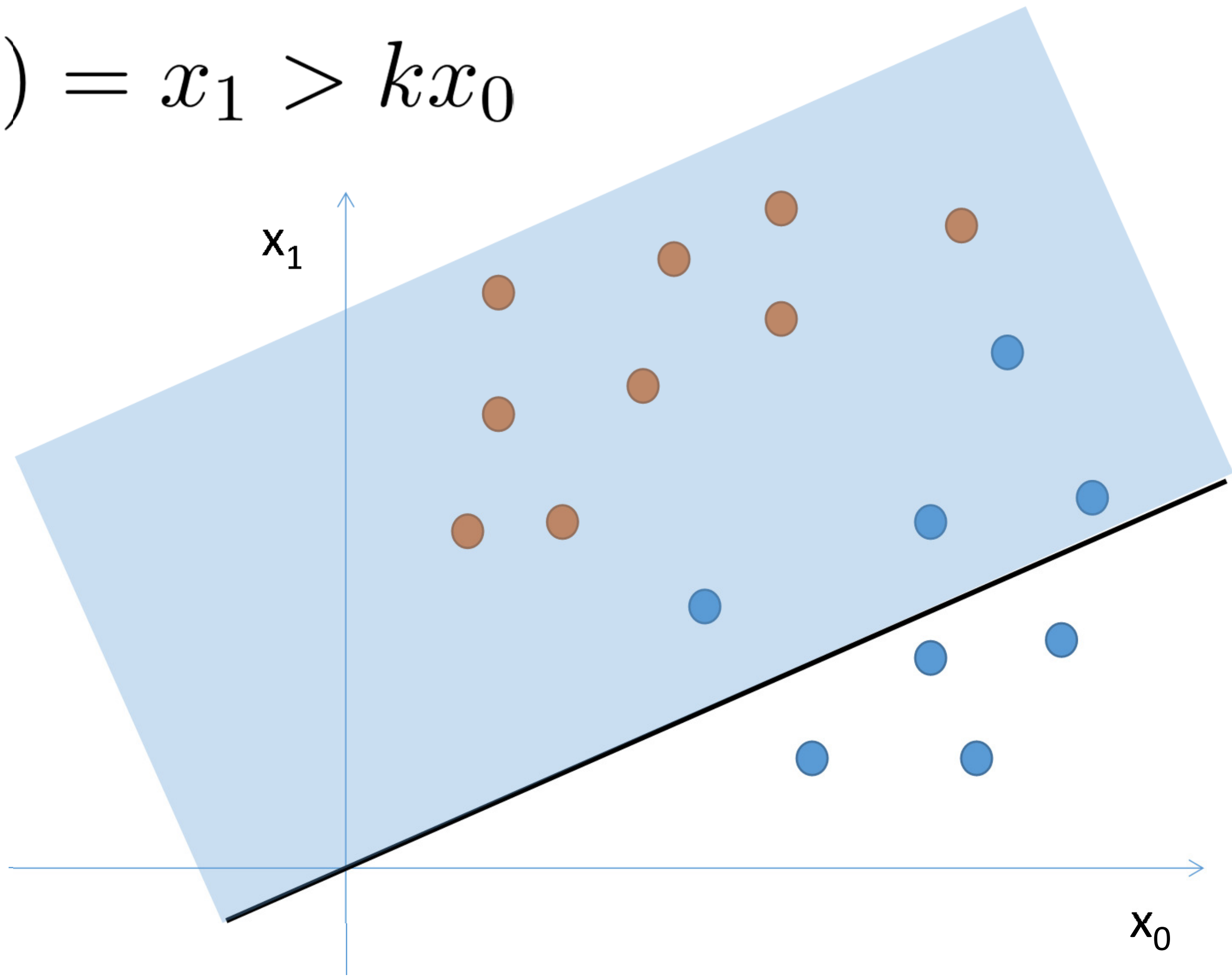
Most of the times we use SGD (Stochastic Gradient Descend) to train our ML models



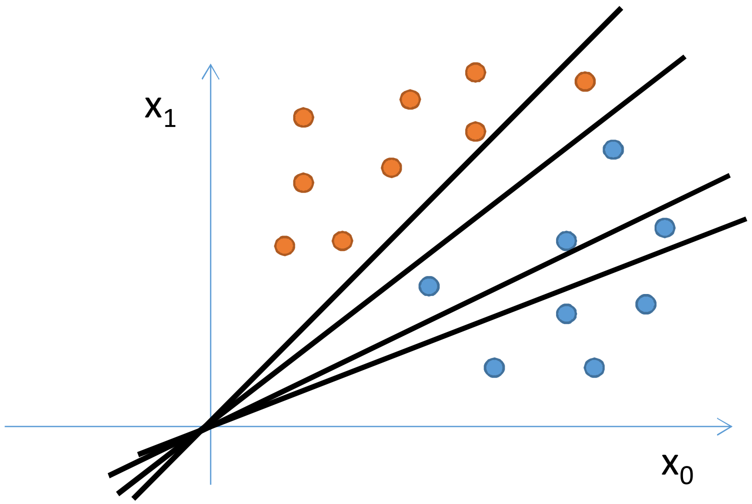
$$P = P + \alpha P\left(-\frac{\partial E(Y)}{\partial P}\right)$$

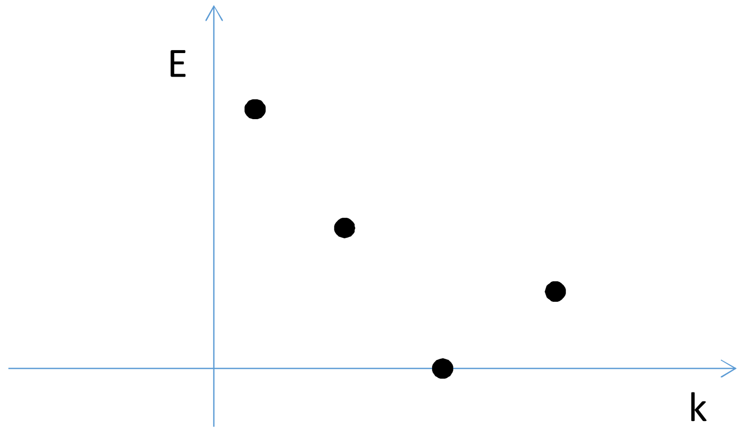
Quick Recap: SGD

$$f(x_0, x_1) = x_1 > kx_0$$

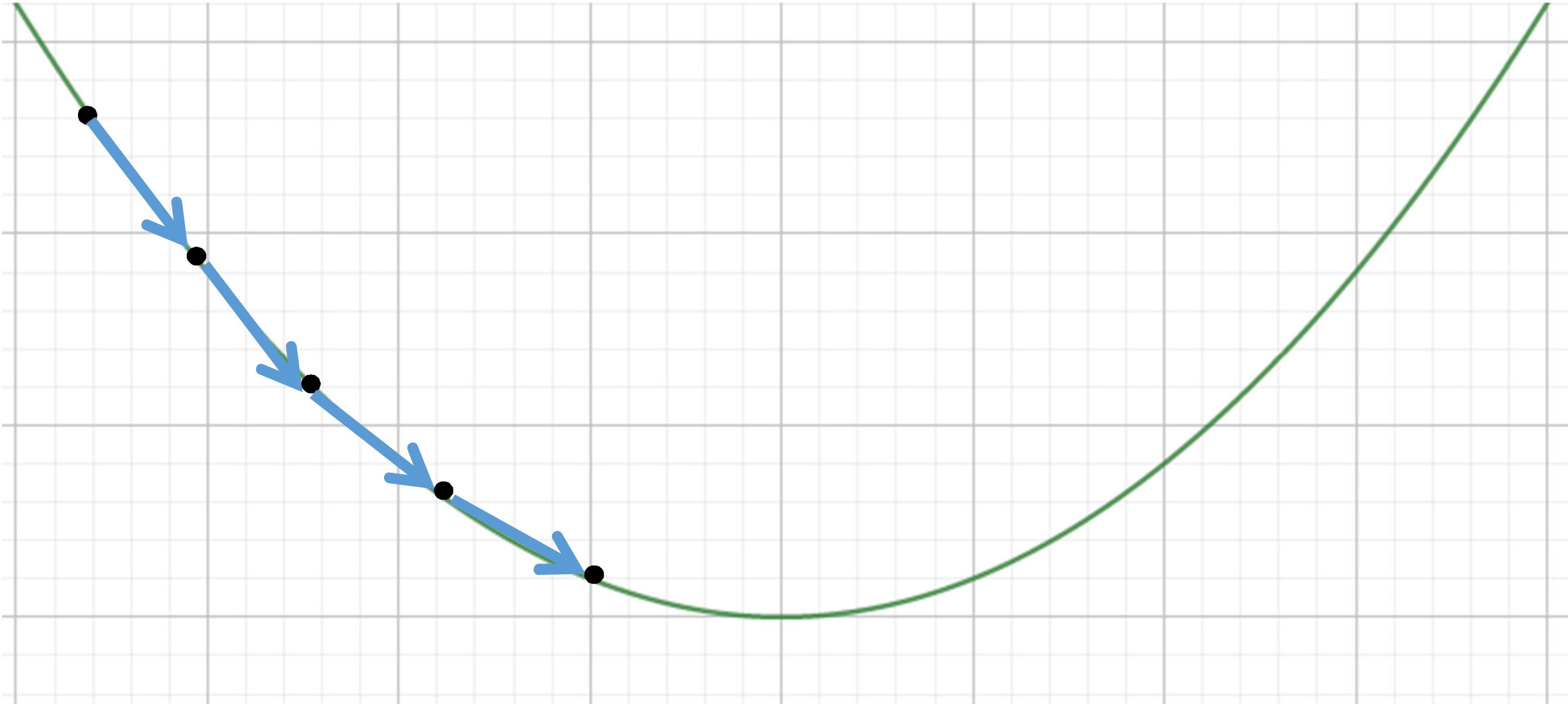


Quick Recap: SGD



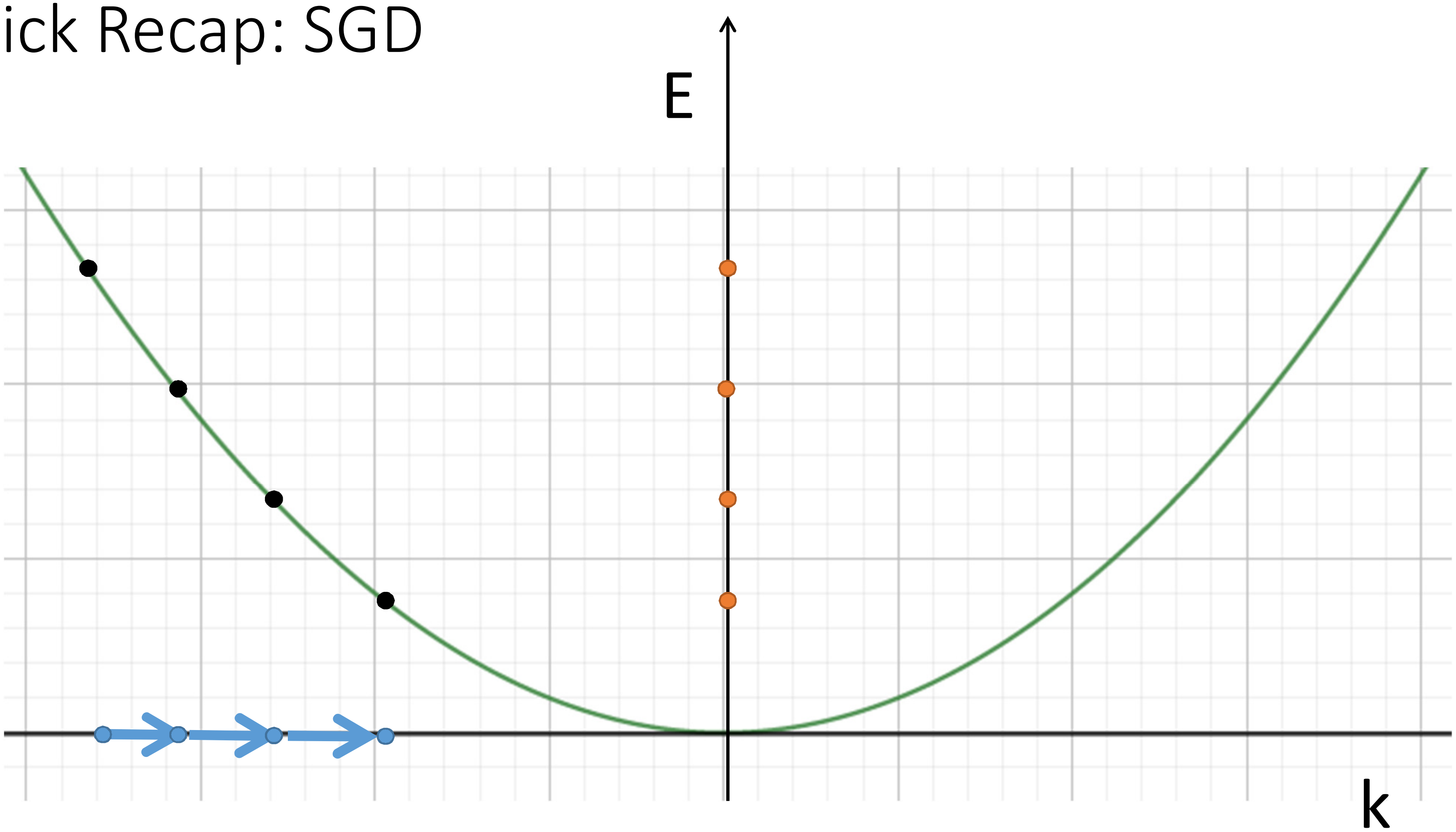


Quick Recap: SGD



Wrong!

Quick Recap: SGD



Quick Recap: SGD

You are computing a **gradient**

$X \in R^N$	OK!
$Y \in R^M$	Sounds fair!
$f(X) : R^N \Rightarrow R^M$	Of course!
$P \in R^{10^6}$	No problem!
$E(Y) : R^M \Rightarrow R$	Never forget this!

Quick Recap: SGD

You are computing a **gradient**

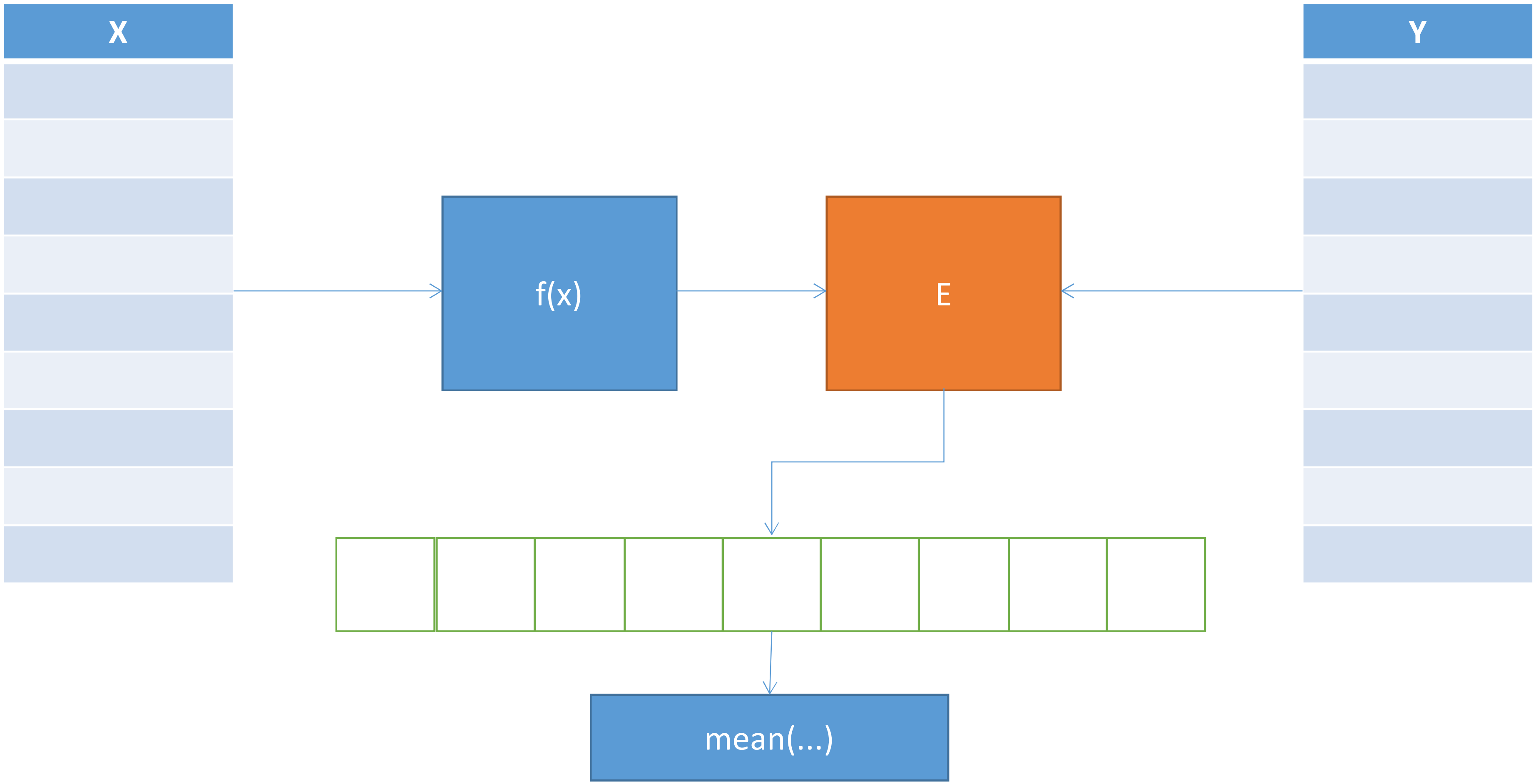
$$E(y_0, y_1, \cdots, y_{M-1}) : R^M \Rightarrow R$$

$$Y = f(X|P)$$

$$E(Y|p_0, p_1, \cdots, p_P) : R^M \Rightarrow R$$

$$\nabla_P E(Y|P) = \frac{\partial E(Y|P)}{\partial P} = \begin{bmatrix} \frac{\partial E(Y|P)}{\partial p_0} \\ \frac{\partial E(Y|P)}{\partial p_1} \\ \vdots \\ \frac{\partial E(Y|P)}{\partial p_P} \end{bmatrix}$$

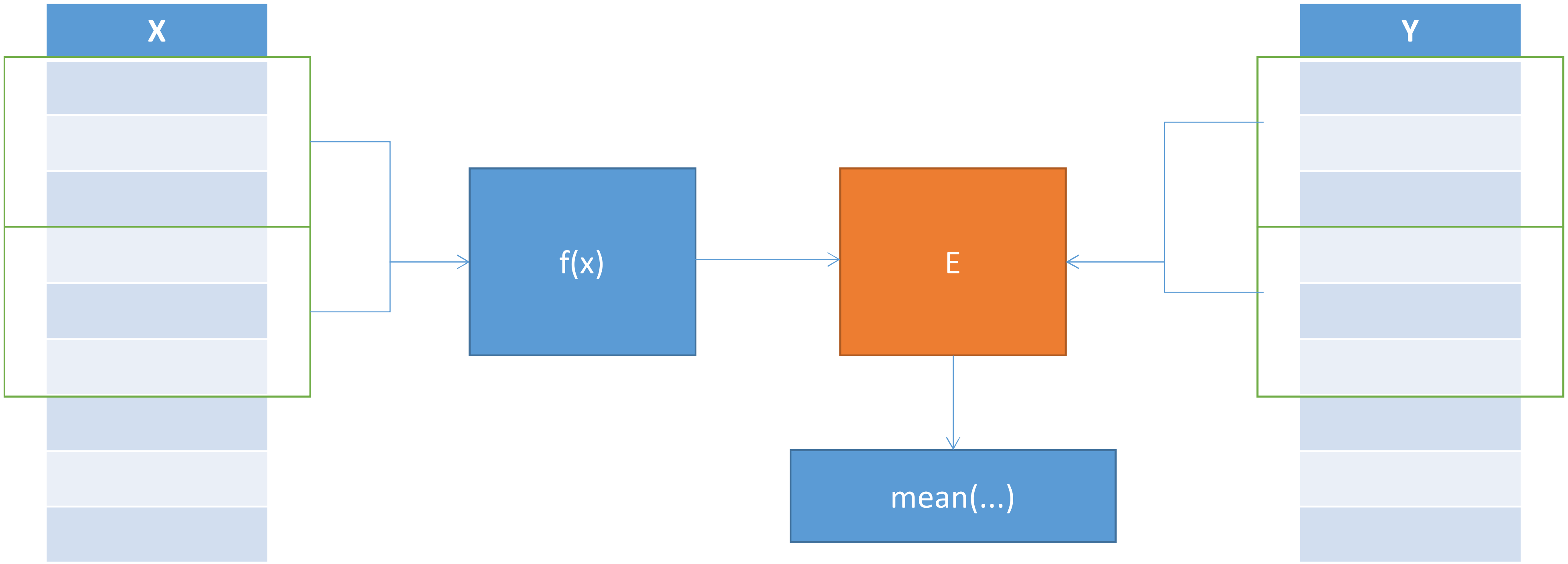
What is actually going on?



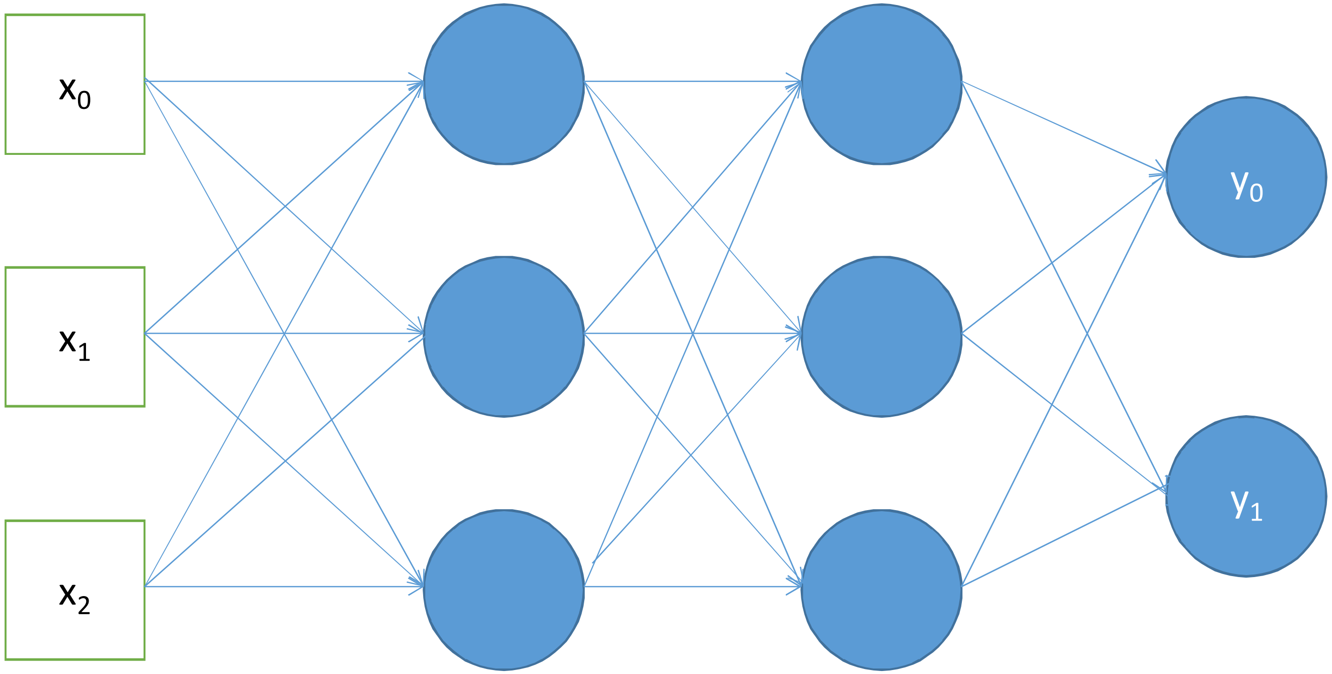
Why is stochastic?

Using all the dataset is impossibile

So we use little chunks called batches



Artificial Neural Networks: Topological view



$$\sigma(\sum_{i=0}^{N} w_i x_i + b)$$

Artificial Neural Networks: Mathematical View

$$x \in R^N$$

$$W \in R^{N \times M}$$

$$b \in R^M$$

$$\sigma : R^M \Rightarrow R^M$$

$$y = \sigma(xW + b)$$

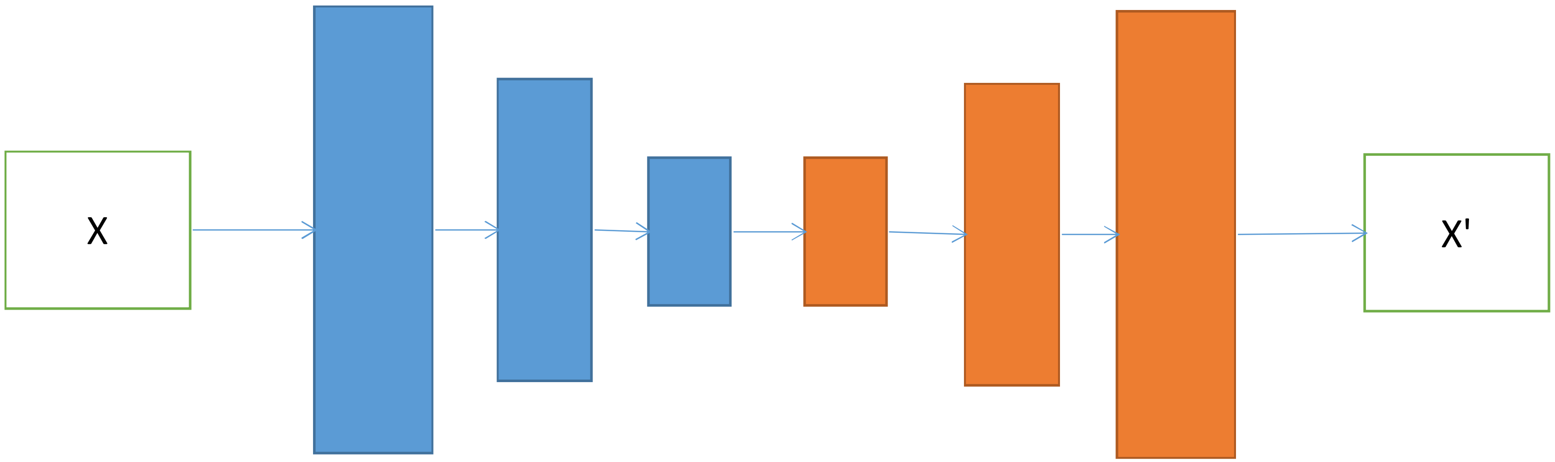
$$y_1 = \sigma(xW_0 + b_0)$$

$$y_2 = \sigma(y_1 W_1 + b_1)$$

$$y_3 = \sigma(y_2 W_2 + b_2)$$

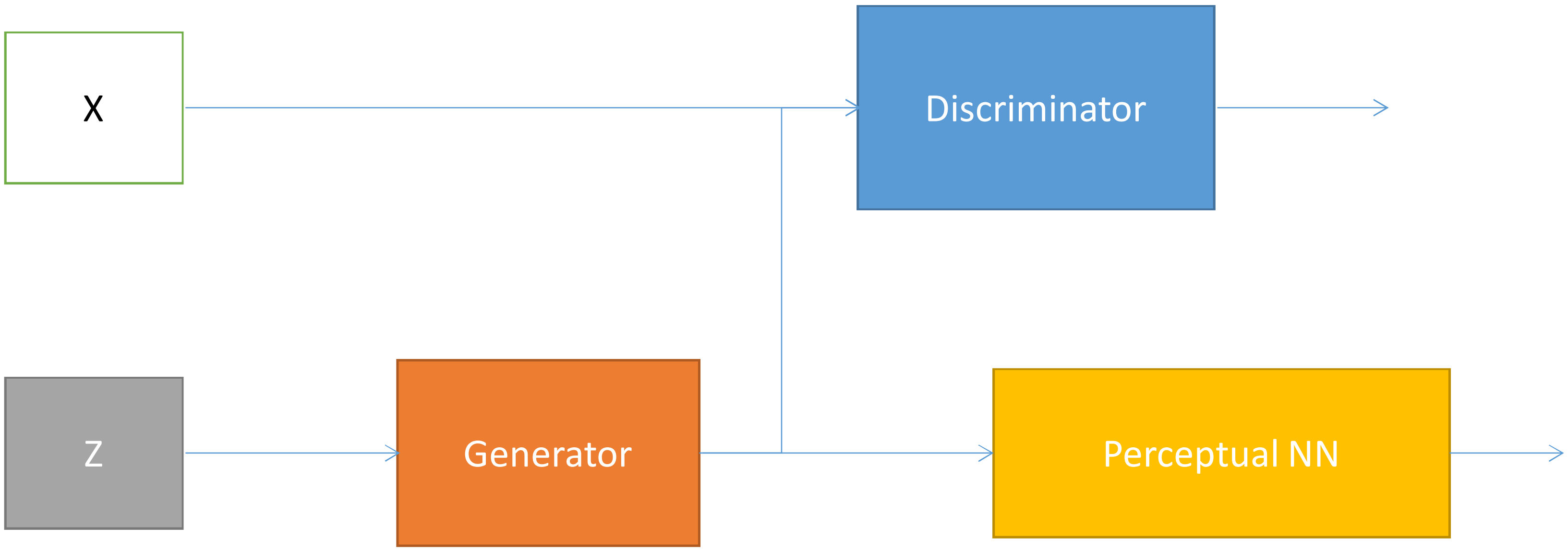
NN Architectures can be wild!

Auto Encoders



NN Architectures can be wild!

GANs

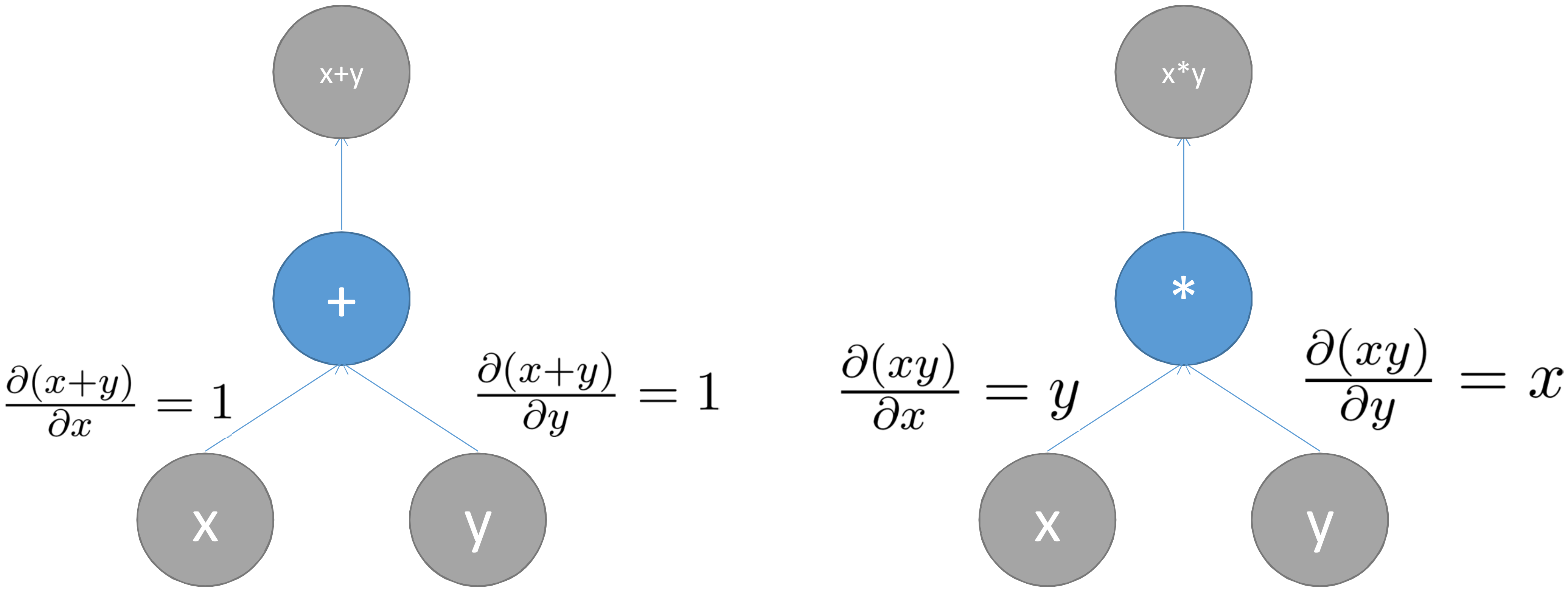


Do I need to compute the gradients by hand?

Nope

All ML frameworks use Computational Graphs

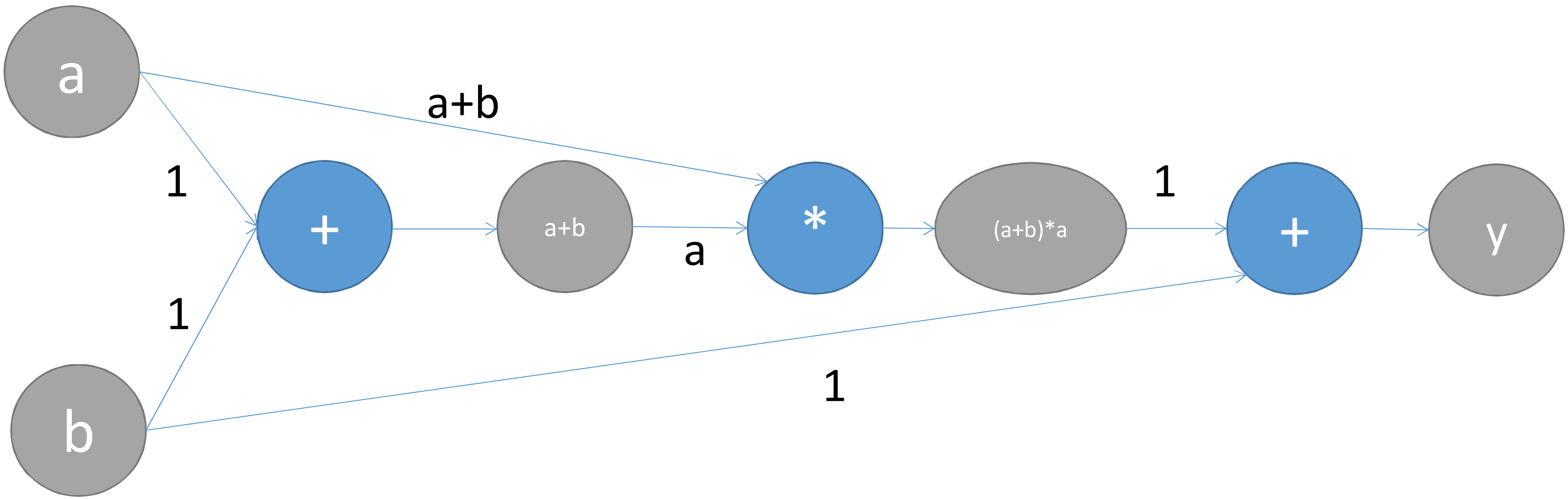
Computational Graphs



Computational Graphs

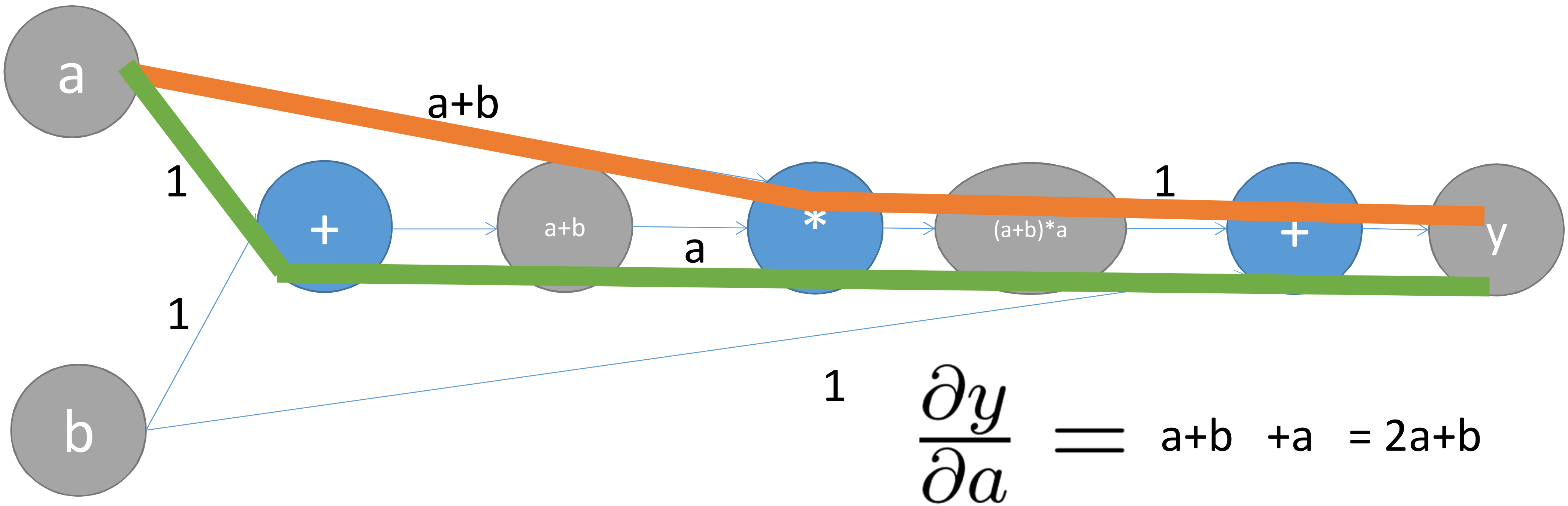
$$y = b + (a + b) \cdot a$$

$$\frac{\partial y}{\partial a} = ? \quad \frac{\partial y}{\partial b} = ?$$



Computational Graphs

$$\frac{\partial y}{\partial a} =$$ For all the paths from y to a: multiply over the edges.
Sum all the results.



$$\frac{\partial y}{\partial a} = \text{a+b} \quad \text{+a} \quad = 2a+b$$

Pytorch

- Pytorch is a Computational-Graph based ML framwork
  - Python interface, C++/Cuda implementation
  - Object Oriented Interface
  - Supports CPU/GPU/Multi-GPU transparently
  - Very well documented
  - Almost everything you need is implemented in it

pip install torch

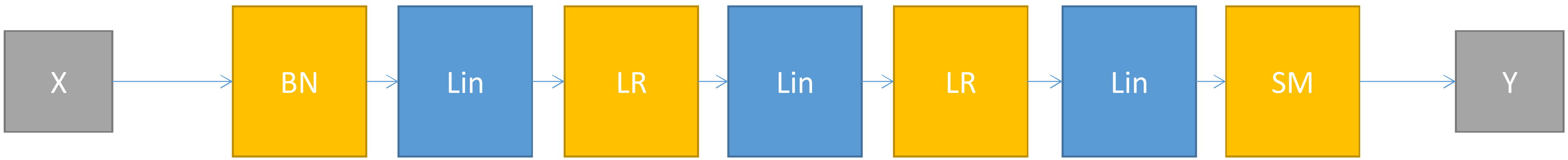
Pytorch: torch.nn.Module

Building block for a ML architecture

- forward(self, x) *[not implemented]*
  - The forward logic of your module
- parameters(self) *[implemented]*
  - Collection of paremters of all the class attributes
- to(self, device) *[implemented]*
  - Move all the parameters to device

Pytorch: torch.nn.Sequential

```
net = torch.nn.Sequential(
    torch.nn.BatchNorm1d(784),
    torch.nn.Linear(784, hidden_size),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(hidden_size, hidden_size),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(hidden_size, 10),
    torch.nn.Softmax(dim=1)
)
```



Pytorch: torch.utils.data.Dataset

Abstract dataset representation

- __getitem__(self, i) *[not implemented]*
  - You have to return a tuple with
    - i-th input element of the dataset
    - i-th output element of the dataset
- __len__(self) *[not implemented]*
  - You have to return the lenght of the dataset

Pytorch: torch.utils.data.DataLoader

Creates batches from a Dataset object

- Positional arguments
  - Dataset object
- Keywords arguments
  - batch_size: size of the batches
  - shuffle: shuffle the dataset?

```
torch.utils.data.DataLoader(
    dataset,
    batch_size=batch_size, shuffle=True
)
```

Pytorch: Other useful stuff

- torch.nn
  - package with every possibile NN building block
- torch.optim
  - package with dozens of SGD algorithms
- torchvision
  - package with dozens of vision datasets (MNIST, CIFAR, ImageNet,...)

In every **scalar** tensor you can call .backward() to compute the gradient

Let us build a simple NN

Tensorboard

- Traking and visualization framework for:
  - Tracking scalars (loss, accuracy, etc...)
  - Traking histograms (weights, gradients, etc..)
  - Traking Computational Graphs
  - Displaying text, images, etc...
- Part of the Tensorflow framework but can be used as standalone

tensorboardX (tb wrapper)

- writer = SummaryWriter()
- writer.add_scalar(“<name>”, <value>, <iter>)
  - Log a scalar values (loss, accuracy, ...)
- writer.add_image(“<name>”, <value>, <iter>)
  - Log an image (PIL, np.matrix, ...)
- writer.add_histogram(“<name>”, <value>, <iter>)
  - Log an histogram(weights, gradients, etc...)
- writer.add_audio
- writer.add_text
- ....

pip install tensorboardx tensorboard==1.13.0 tensorflow==1.13.1

Let us integrate it into our NN

What if I told you...

That exists a framework that does:

- Training, testing and validation
- Early stopping managing
- Tensorboard logging
- Checkpointing and retraining
- ...

With minimal coding required

pytorch-lightning

- Define
  - Model
  - Dataset
  - Training/testing/validation step

That's all

LightningModule interface

- forward(self, x):
  - The forward logic (as in pytorch)
- *_step(self, batch, batch_nb):
  - The *_step logic
- *_end(self, outputs):
  - The * phase end logic
  - outputs are all the intermediate returns of the *_step function calls
- *_loader(self):
  - Must returns the DataLoader for the * phase
- configure_optimizers(self).
  - Must return a list of optimizer to use in training

Where * can be “train” “test” or “validation”

pytorch-lightning

- The train_step() function must return a dictionary with the “loss” key
- Every function that returns a dictionary with the key
  - log
    - every key gets logged in tensorboard as scalar
  - progress_bar
    - every key get prompted in the * phase progress bar
- Training/Testing
  - Instanciate your model
  - Create a Trainer() object
  - trainer.fit(model)
  - trainer.test(model)

Let us integrate our model in pytorch-lightning

Ray

Ray is massive framework for scaling ML training

We are going to see just its “tune”

ray.tune is a framework for hyperparameters tuning at any scale (from a single computer to clusters of GPUs)

Let us do an hyperparameter search with ray