INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

INTRODUCTION TO DEEP LEARNING WITH TENSORFLOW

Fabien Baradel
PhD Candidate

fabienbaradel.github.io
@fabienbaradel
fabien.baradel@insa-lyon.fr

DEEP LEARNING?

TENSORFLOW?

TensorFlow

TENSORFLOW BACKGROUND

- **6 months internship at Xerox Research Center Europe:**
  - *« Unsupervised Domain Adaptation »*
  - Image recognition & Sentimental text classification
  - Machine Learning for Services team
  - Grenoble, France

- **Phd Candidate at LIRIS - INSA Lyon since October 2016:**
  - *« Deep Learning for human understanding: gestures, poses, activities »*
  - Imagine team
  - Supervisors: Christian Wolf & Julien Mille
  - Working with videos

DEEP LEARNING FOR HUMAN UNDERSTANDING

- Action recognition => Classification
- Sequence Learning
- Supervised Learning

DEEP LEARNING FOR HUMAN UNDERSTANDING

- Microsoft Kinect - Xbox

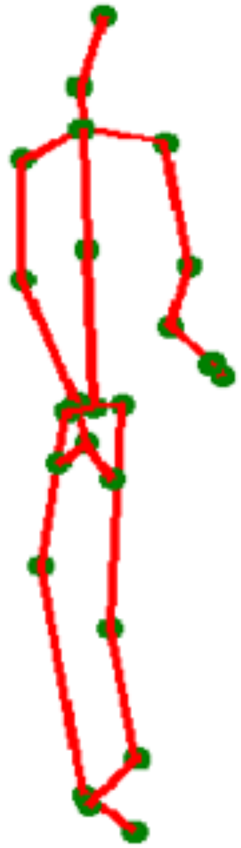
DEEP LEARNING
FOR HUMAN UNDERSTANDING

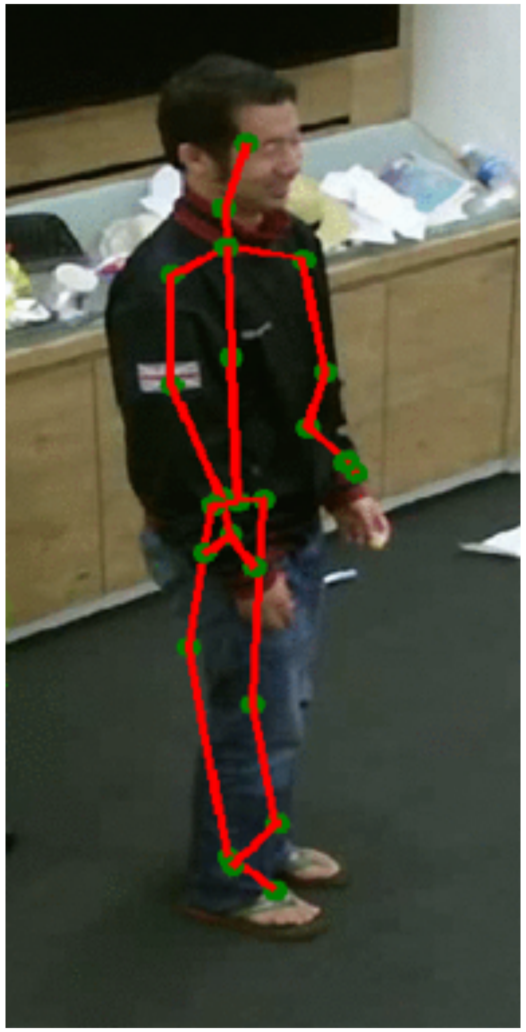
Microsoft Kinect v2:

- 3D joint location
- 25 joints

Video = sequence of frames

Skeleton not enough (looking at book, looking at smartphone)





SELF-DRIVING CARS

Tesla

- https://www.youtube.com/watch?v=CxanE_W46ts

MATERIALS

Virtual Machine: USB key

- Python 2.7
- Tensorflow
- Ubuntu 16.04
- Datasets

Seminar slides & Code corrections:

https://fabienbaradel.github.io

Introduction

- Basics of Tensorflow
- Machine Learning: analytic solution vs. gradient descent

Supervised Learning (image recognition)

- Neural networks reminder
- Convolution Networks
- Going deeper with ConvNets

Unsupervised Learning

- Autoencoder
- Generative Adversial Network

Sequence modelling

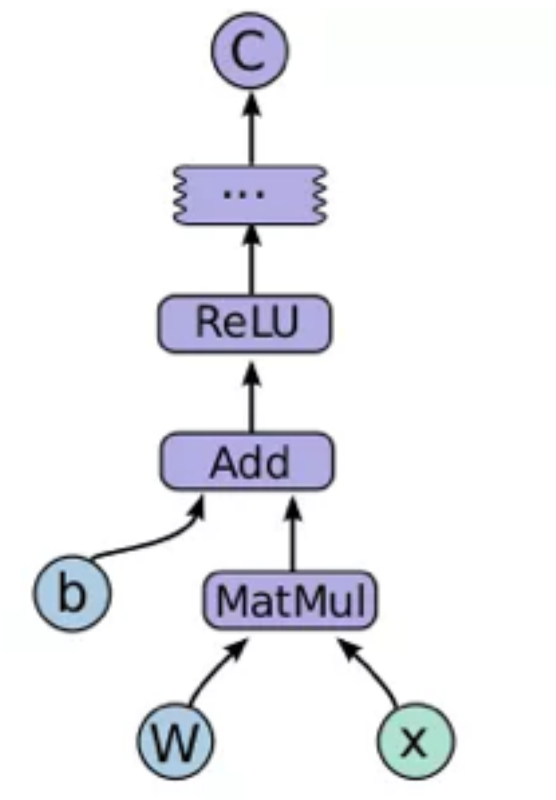
- RNN, LSTM
- Word2vec

Reinforcement Learning

- Deep Q-learning
- Frozen Lake

INTRODUCTION

WHAT IS TENSORFLOW?



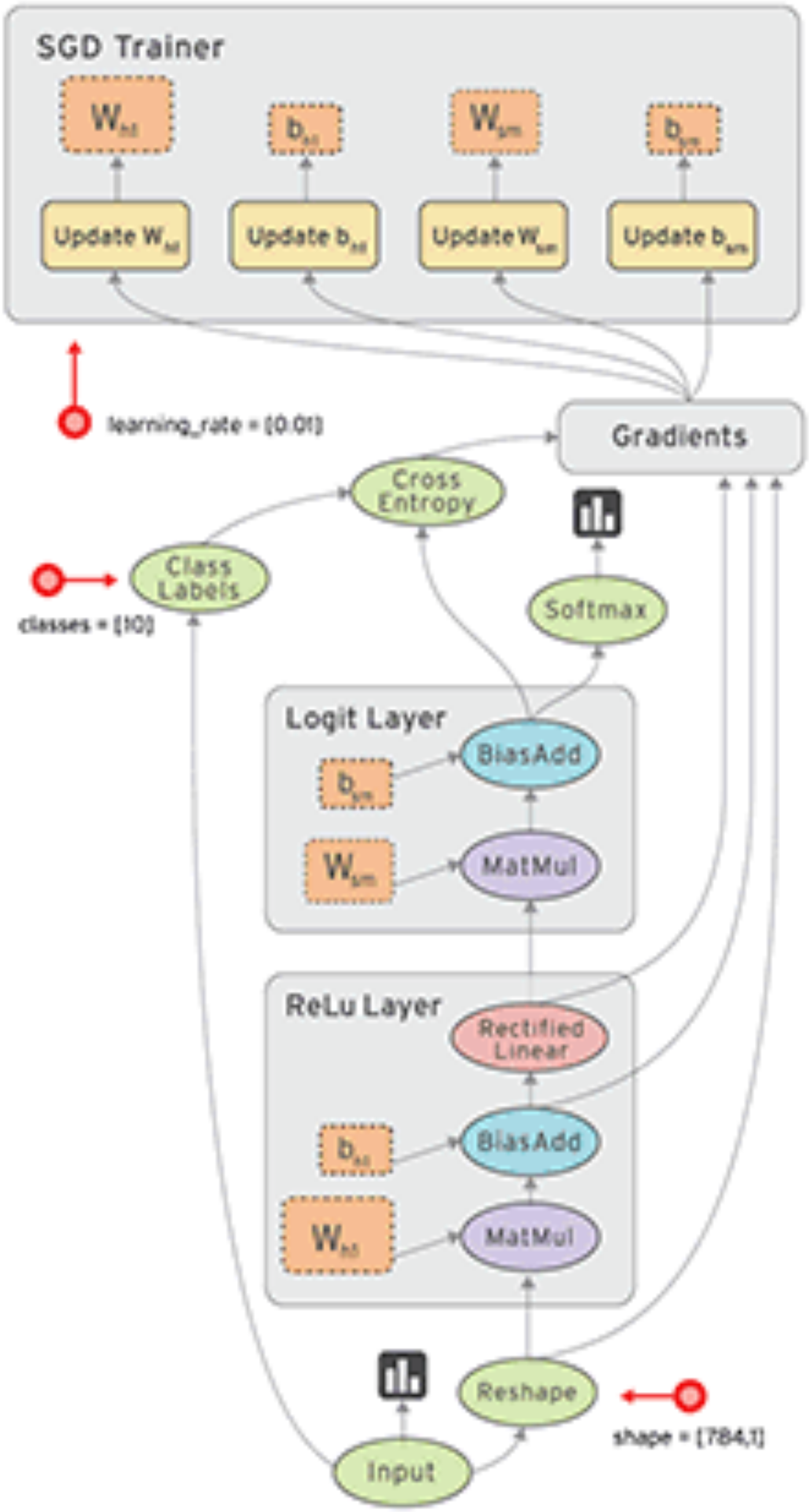
- A python library
- **pip install tensorflow**
- Google
- open-source
- library for numerical computation using data flow graphs
- CPU and GPU
- Research & Industry

```
pip import tensorflow
```

Companies using TensorFlow

ARM
quantiphi
AIRBUS DEFENCE & SPACE
CI&T
CEVA
Google
Movidius
UBER
JD.COM 京东
DeepMind
ebay
Dropbox

PRINCIPLE


SGD Trainer
Update W
Update b
Update W
Update b
learning_rate = [0.01]
Gradients
Cross Entropy
Class Labels
classes = [10]
Softmax
Logit Layer
BiasAdd
MatMul
ReLu Layer
Rectified Linear
BiasAdd
MatMul
Reshape
shape = [784,1]
Input

« HELLO WOLRD »

- **INTRODUCTION EXERCISES**

- Difference between ***constant/variable*** and ***placeholder***

- ***Constant = a fixed Variable***

- ***With placeholder you need to feed data to your graph during your session***

- Tensorflow workflow:
  - Draw your graph
  - Feed data
  - … and optimize

« BASIC MATHS OPERATIONS »

- Open « math_ops.py »
- Same thing with integer
- Mathematical operation done only using Tensorflow library (no numpy or else)
- **Draw the schema of the code**

WITH CONSTANTS

?

WITH PLACEHOLDERS

?

ANALYTIC SOLUTION IN ML

Linear regression $y = X\beta + \epsilon$

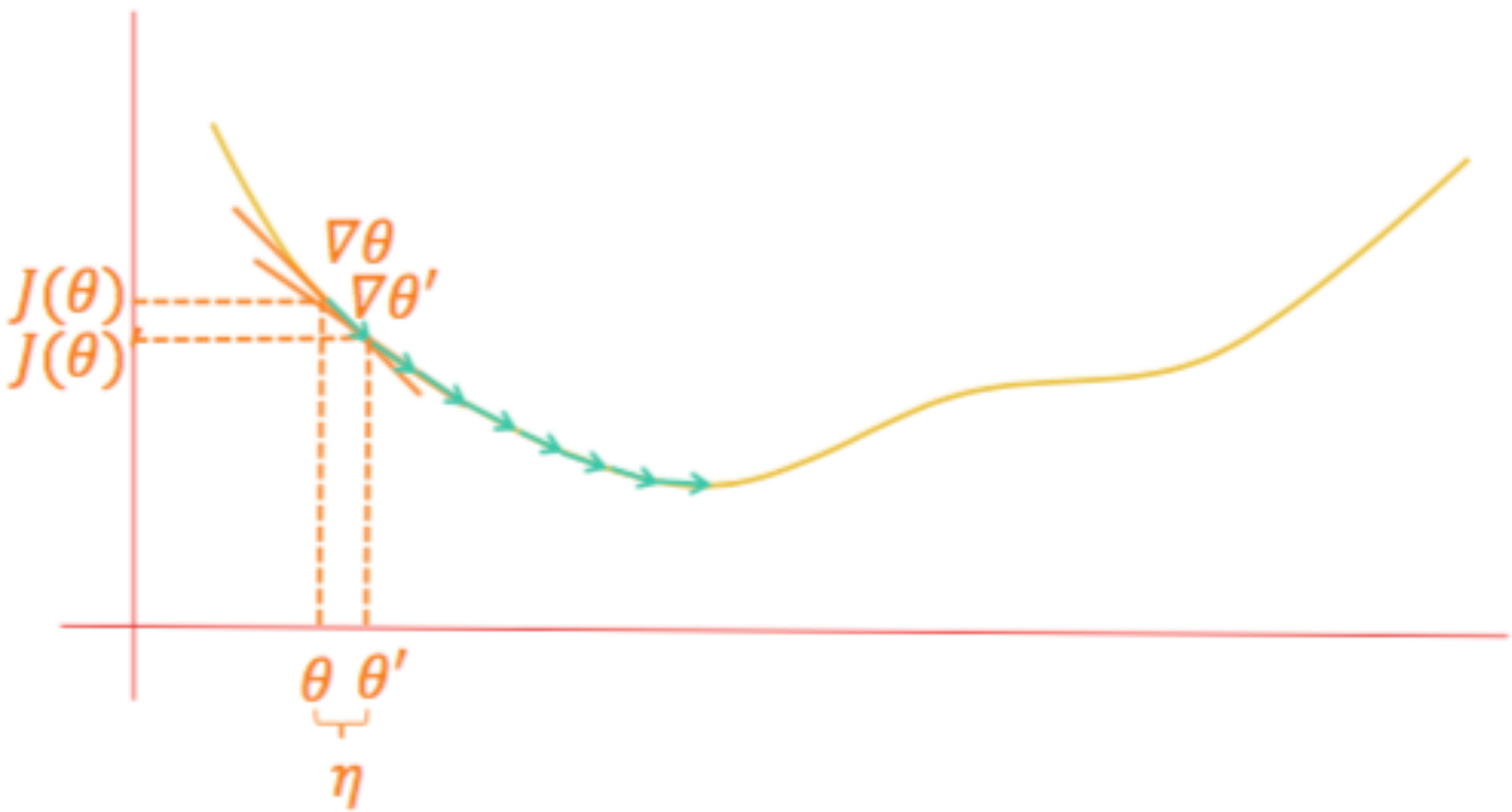
Least Squares solution

$$\hat{\beta} = \arg\min ||X\beta - y||_2$$

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Could solve the same problem by solving the optimization problem using **gradient descent**

GRADIENT DESCENT



- Goal: minimizing an function
- Random initialization of parameters
- At time t, gradient gives the slope of the function
- Iterative process
- Updating the parameters in the positive direction of the gradient according to a learning rate
- Repeat until convergence

BATCH STOCHASTIC GRADIENT DESCENT

Linear regression $y = X\beta + \epsilon$

Minimize a loss function:

$$J(\beta) = \sum_{i=1}^{N} (X_i\beta - y_i)^2$$

$$\hat{\beta} = \arg\min J(\beta)$$

- **Initialize** $\hat{\beta}_0$ **randomly**
- **Choose a learning rate** η
- **for** t **in range(training_step):**
  - **Compute the loss**

    $$J(\hat{\beta}_t) = \sum_{i=1}^{N} (X_i\hat{\beta}_t - y_i)^2$$

  - **Update parameters**

    $$\hat{\beta}_{t+1} = \hat{\beta}_t - \eta \nabla J(\hat{\beta}_t)$$

https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/#train-your-dragon

MINI-BATCH SGD

- **Initialize $\hat{\beta}_0$ randomly**
- **Choose a learning rate η**
- **Choose a batch size n**
- **for t in range(training_step):**
  - **Pick a random sample S_t^n from training data**
  - **Compute the loss function**

    $$J(\hat{\beta}_t) = \sum_{i \in S_t^n} (X_i \hat{\beta}_t - y_i)^2$$

  - **Update parameters**

    $$\hat{\beta}_{t+1} = \hat{\beta}_t - \eta \nabla J(\hat{\beta}_t)$$

SGD = stochastic gradient descent

- **Initialization is important!**
- **Learning rate too!**
- **Need a validation set to avoid overfitting**

Neural nets always trained with mini-batch SGD!

EXERCISES

- Go to the Github repo and complete the codes:
  - ✳ SGD/linear_regression_exo.py
  - ✳ SGD/binary_classif_exo.py

import ipdb; ipdb.set_trace()

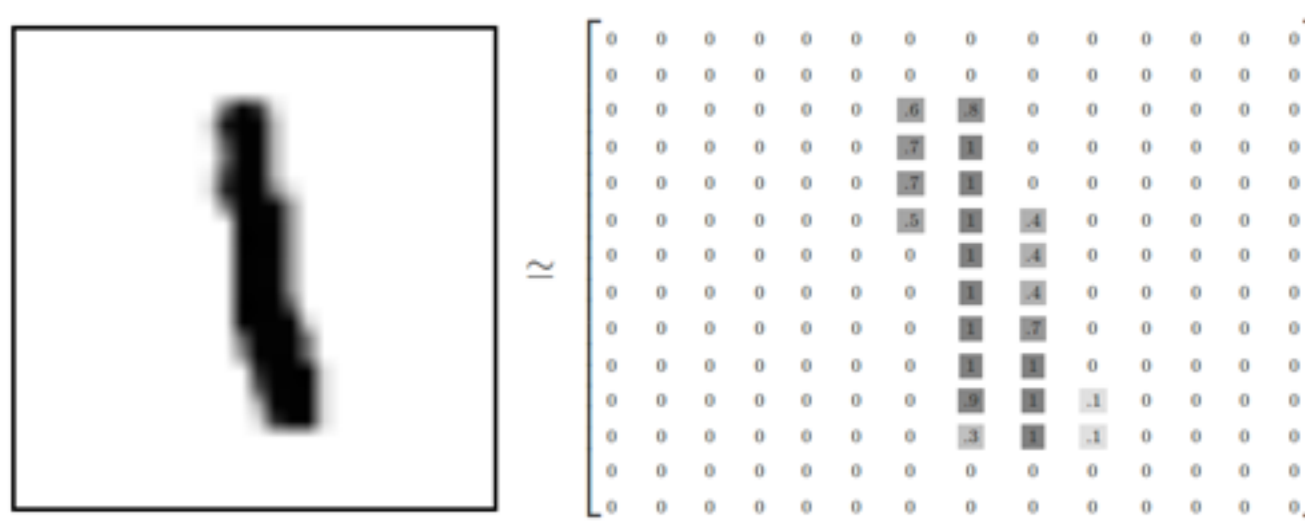
http://playground.tensorflow.org/

https://wookayin.github.io/TensorflowKR-2016-talk-debugging/

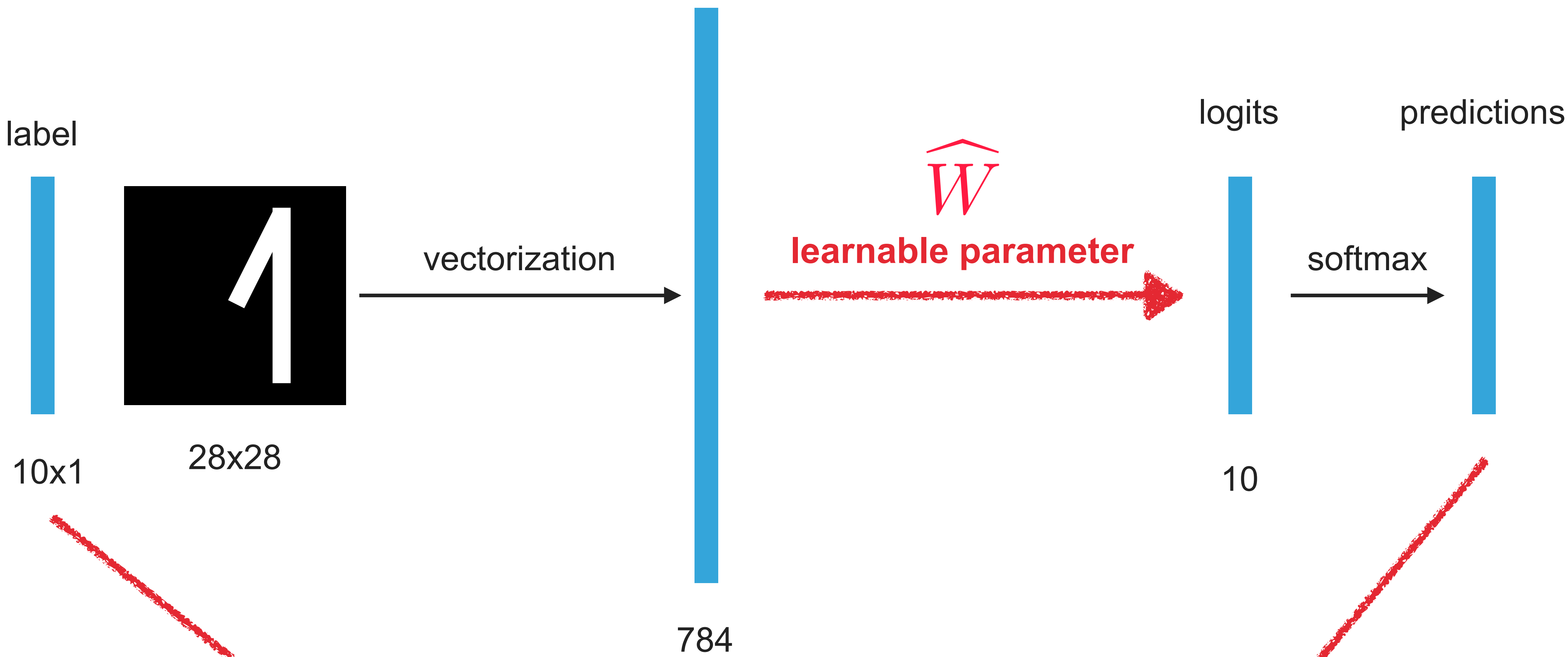
NEURAL NETWORKS

MNIST DATASET

- Handwritten digits
- 60.000 training data and 10.000 test data
- 28x28 grayscale images
- matrix of size 28x28 with value between 0 and 255
- data preprocessing = rescaling to [0,1]



MULTINOMIAL LOGISTIC REGRESSION ON MNIST: CREATE THE GRAPH



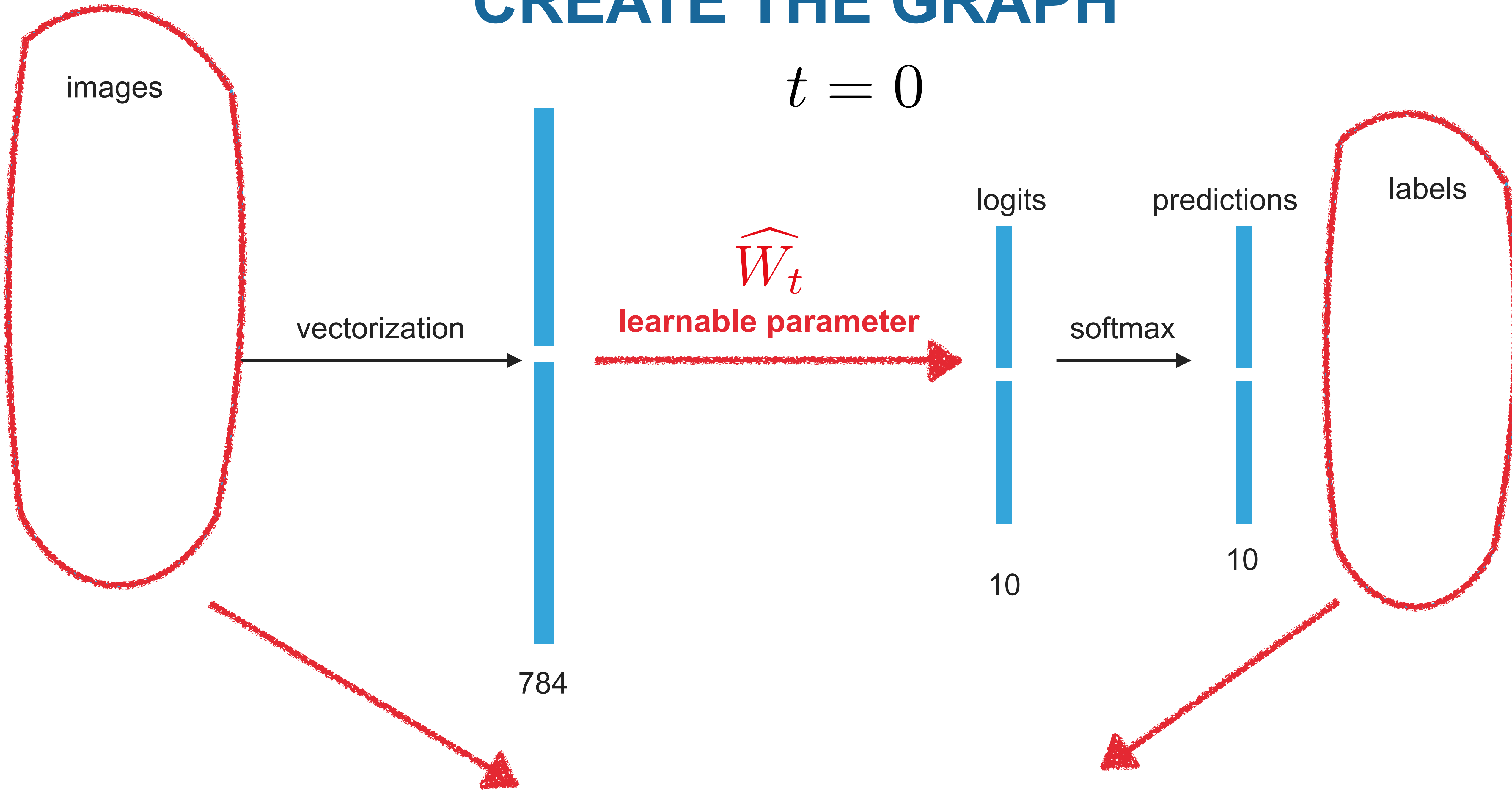
Compute cross-entropy

$$J(\widehat{W}) = -y \times log(\hat{y})$$

WHY LOG?

http://colah.github.io/posts/2015-09-Visual-Information/

MULTINOMIAL LOGISTIC REGRESSION ON MNIST: CREATE THE GRAPH



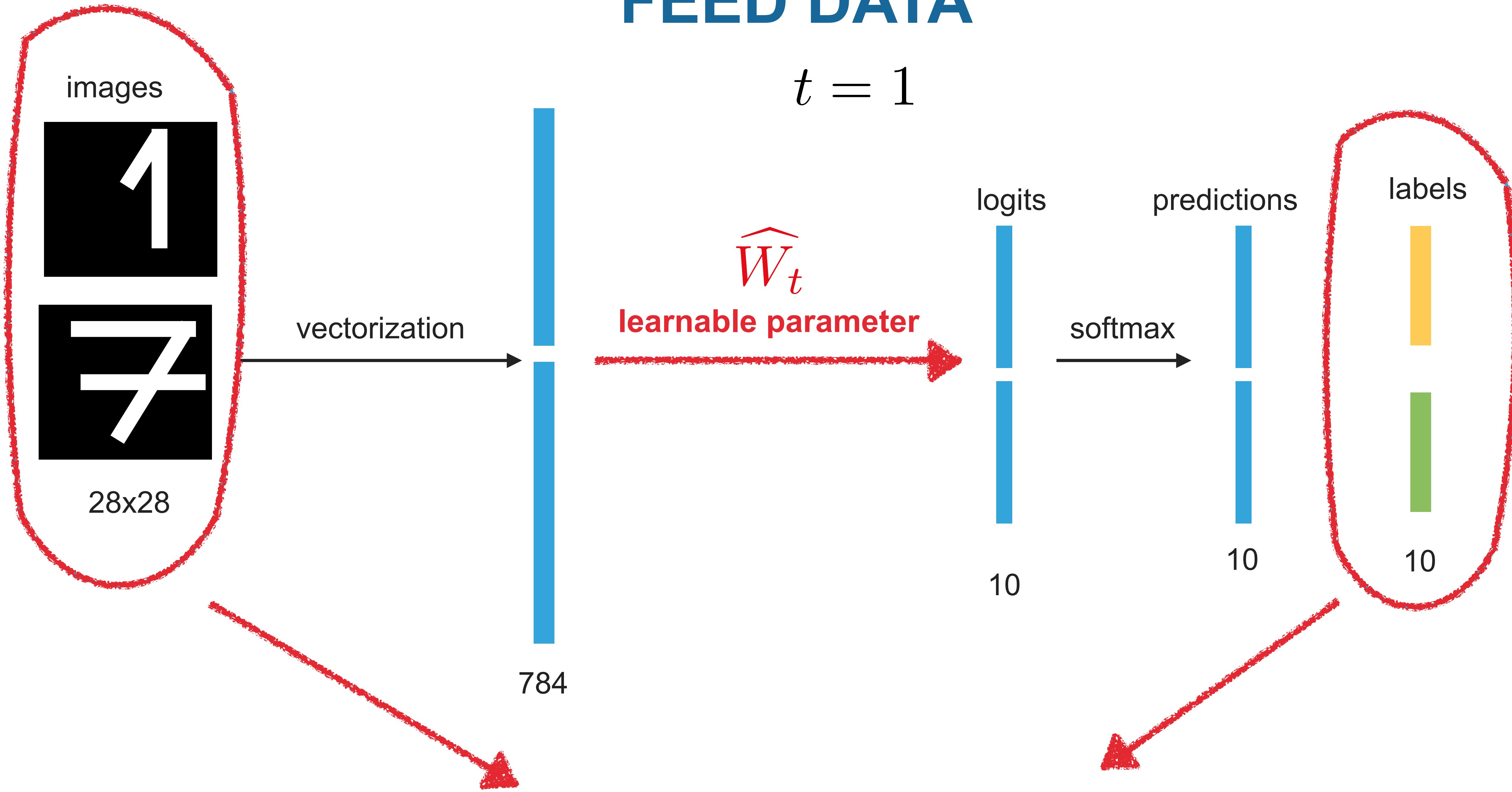
Compute cross-entropy

$$J(\widehat{W}_t) = -\sum_{i \in S_t^2} y_i \times log(\hat{y}_i)$$

$\widehat{W}_t$ updated by SGD

$$\widehat{W_{t+1}} = \widehat{W}_t - \eta \nabla J(\widehat{W}_t)$$

MULTINOMIAL LOGISTIC REGRESSION ON MNIST: FEED DATA



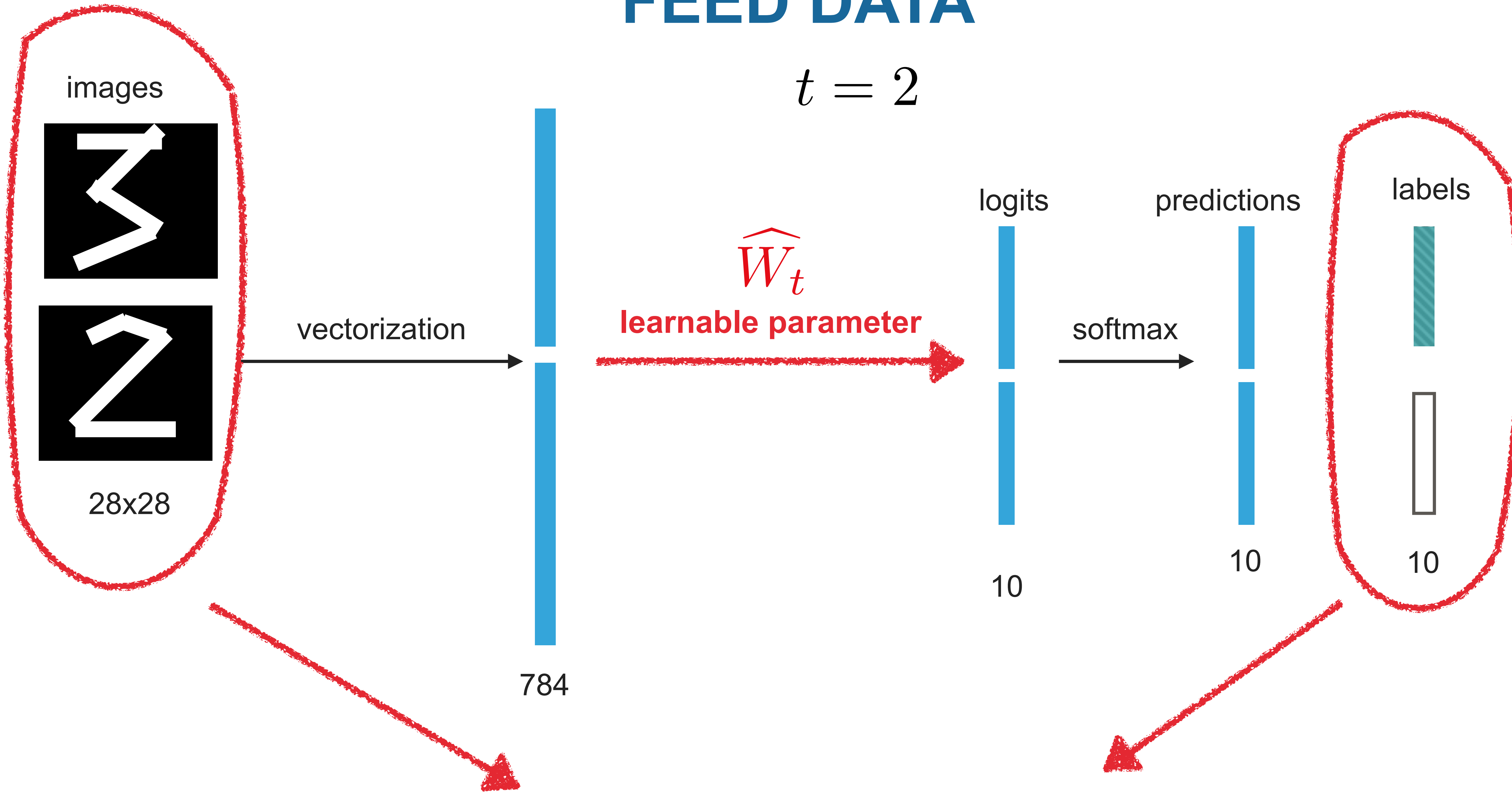
Compute cross-entropy

$$J(\widehat{W}_t) = -\sum_{i \in S_t^2} y_i \times log(\hat{y}_i)$$

$\widehat{W}_t$ updated by SGD

$$\widehat{W_{t+1}} = \widehat{W}_t - \eta \nabla J(\widehat{W}_t)$$

MULTINOMIAL LOGISTIC REGRESSION ON MNIST: FEED DATA



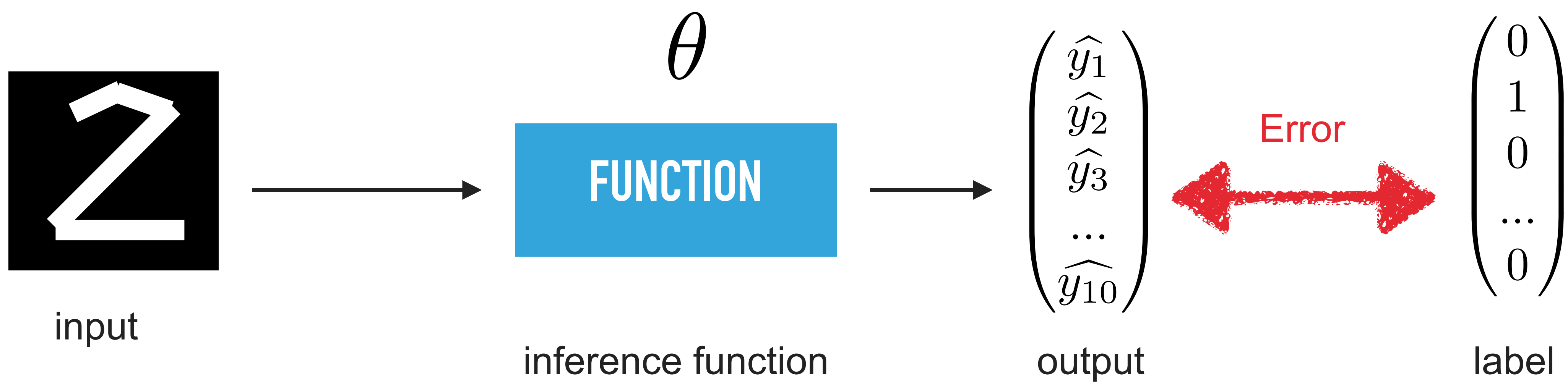
Compute cross-entropy

$$J(\widehat{W_t}) = -\sum_{i \in S_t^2} y_i \times log(\hat{y_i})$$

$\widehat{W_t}$ updated by SGD

$$\widehat{W_{t+1}} = \widehat{W_t} - \eta \nabla J(\widehat{W_t})$$

NEURAL NETWORKS



- Minimize your error on a training set
- Find the best inference function parameters
- Difference between neuralNets an deepNets: only in the inference function

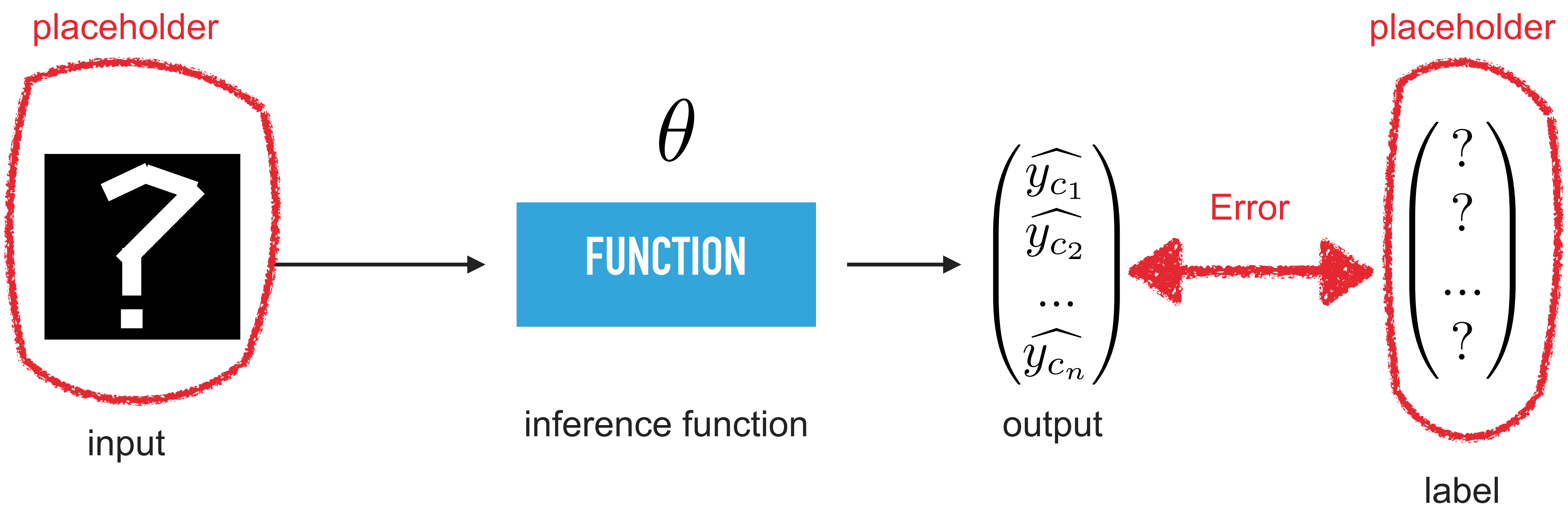
$$\hat{y} = f(\theta, x)$$

$$J(\theta) = \text{error}(\hat{y}, y) \text{ given } \theta$$

$$\hat{\theta} = \text{argmin } J(\theta)$$

And train it using mini-batch SGD!

NEURAL NETWORKS IN TENSORFLOW: GENERAL GRAPH



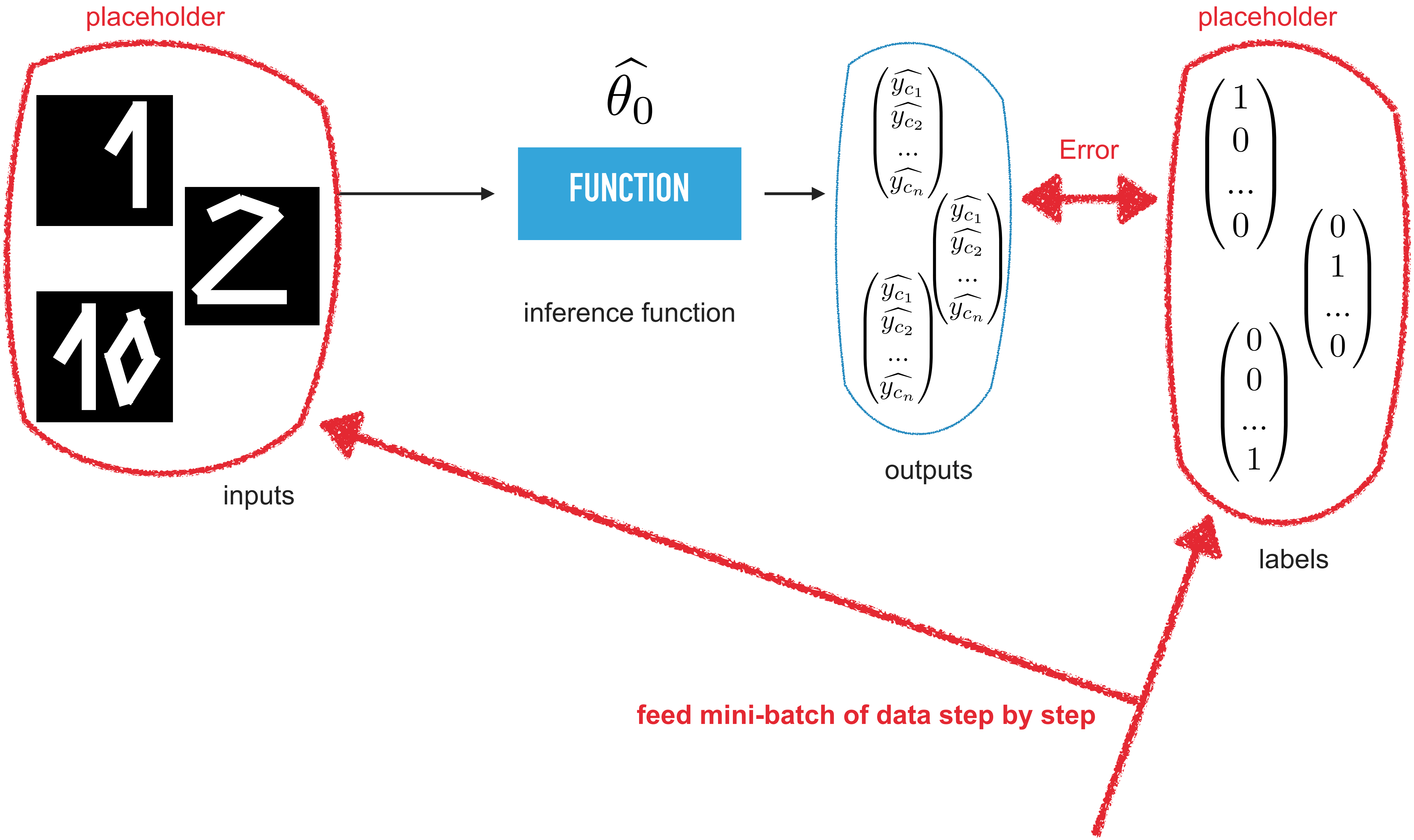
inference function => $$\hat{y} = f(\theta, x)$$

loss function => $$J(\theta) = \mathrm{error}(\hat{y}, y) \text{ given } \theta$$

optimization problem => $$\hat{\theta} = \mathrm{argmin}\ J(\theta)$$

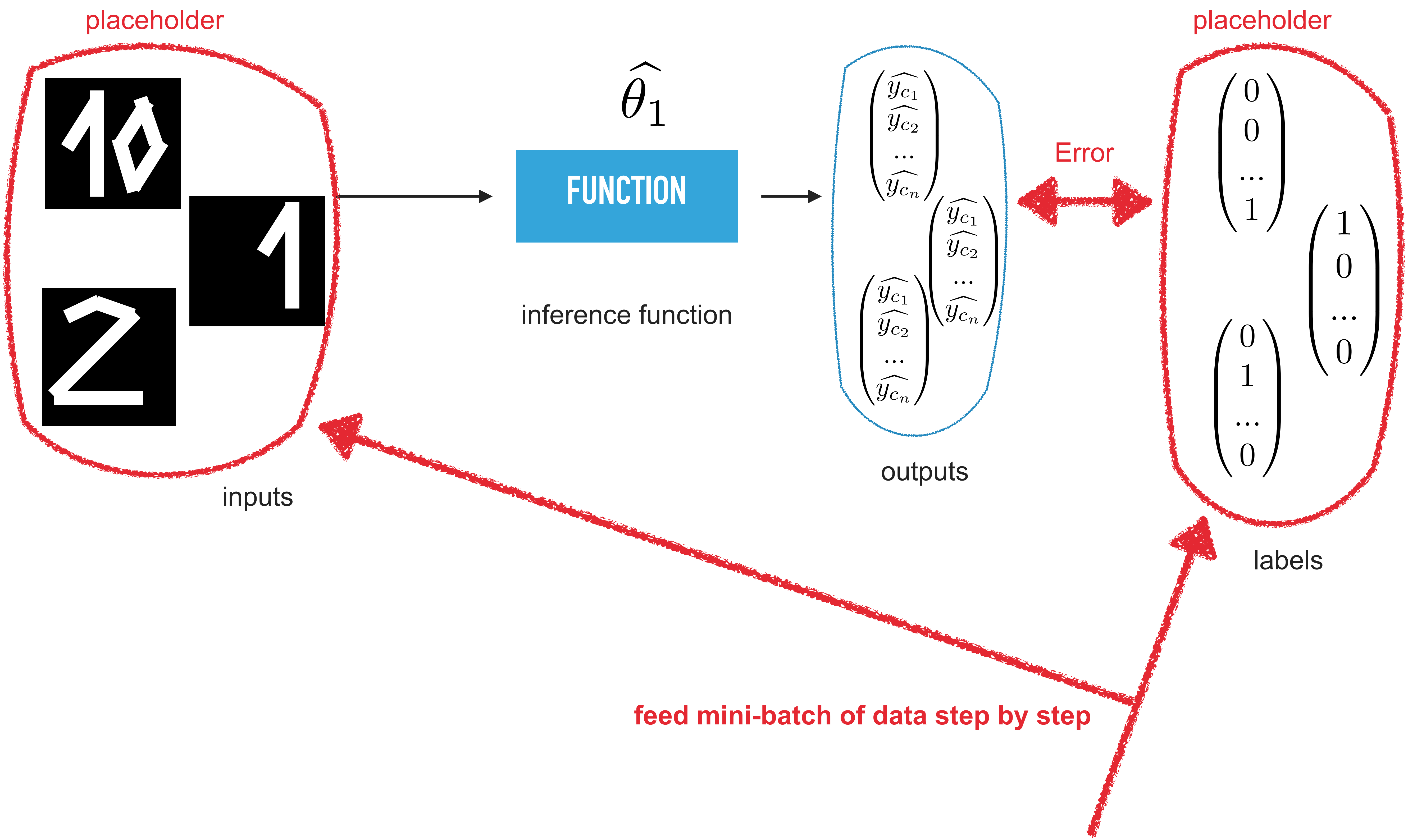
And train it using mini-batch SGD in a Tensorflow session!

NEURAL NETWORKS IN TENSORFLOW: GENERAL TRAINING



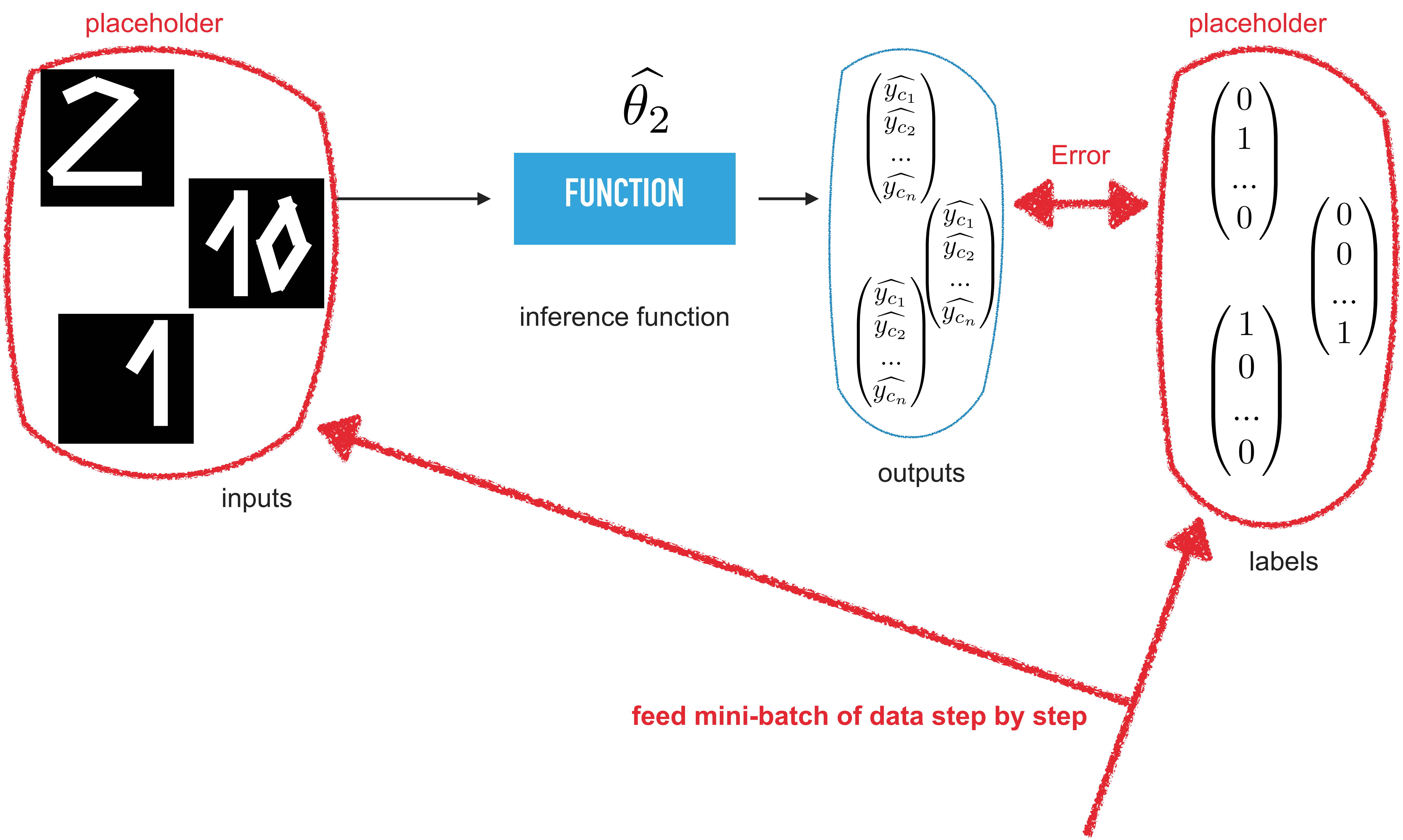
And train it using mini-batch SGD in a Tensorflow session!

NEURAL NETWORKS IN TENSORFLOW



And train it using mini-batch SGD in a Tensorflow session!

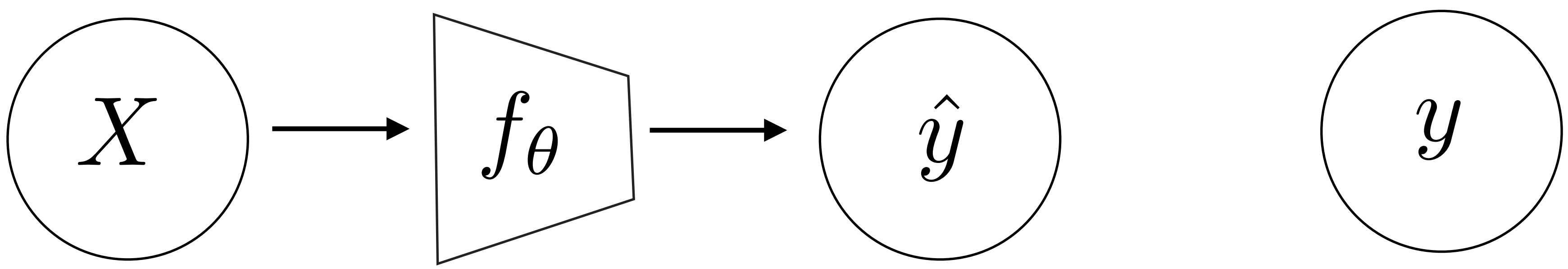
NEURAL NETWORKS IN TENSORFLOW



And train it using mini-batch SGD in a Tensorflow session!

BACKPROPAGATION

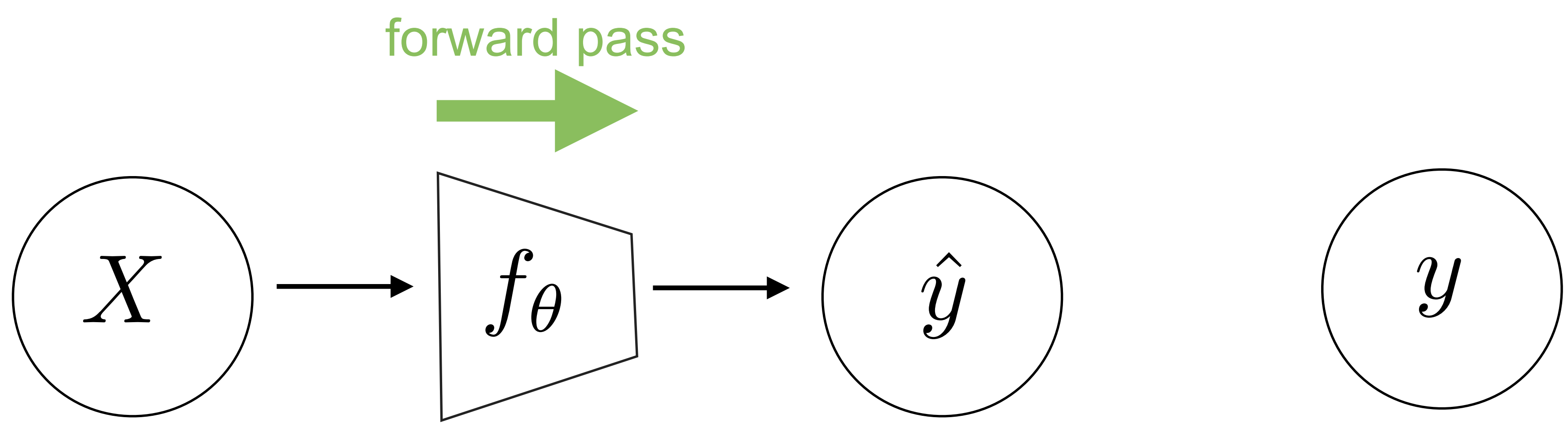
- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters



https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu

BACKPROPAGATION

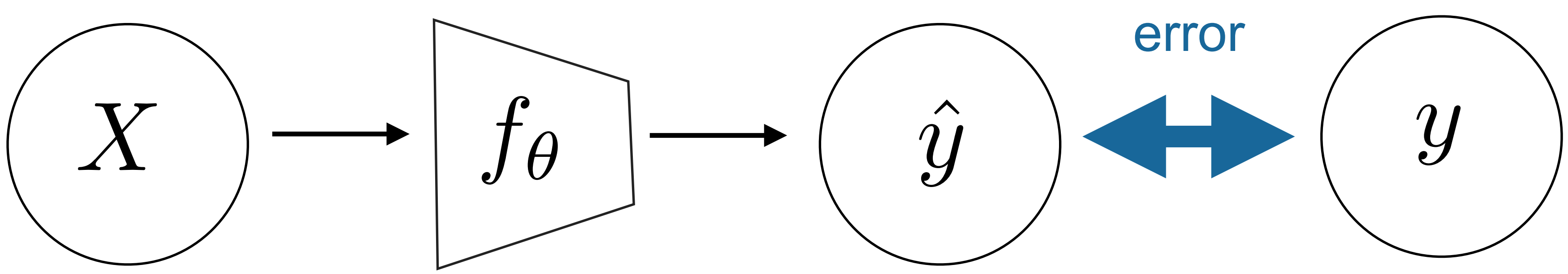
- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters



https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu

BACKPROPAGATION

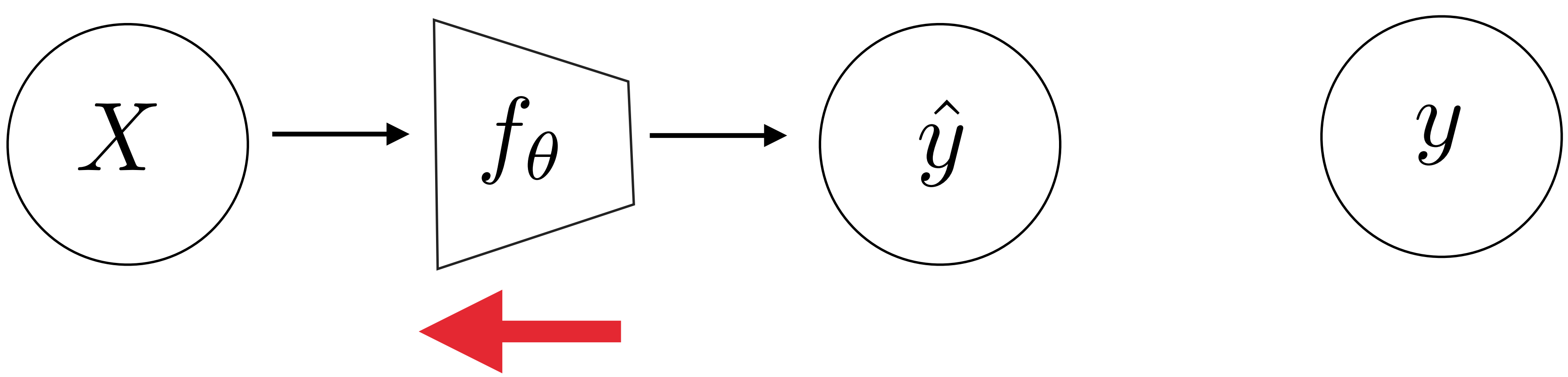
- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters



https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu

BACKPROPAGATION

- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters

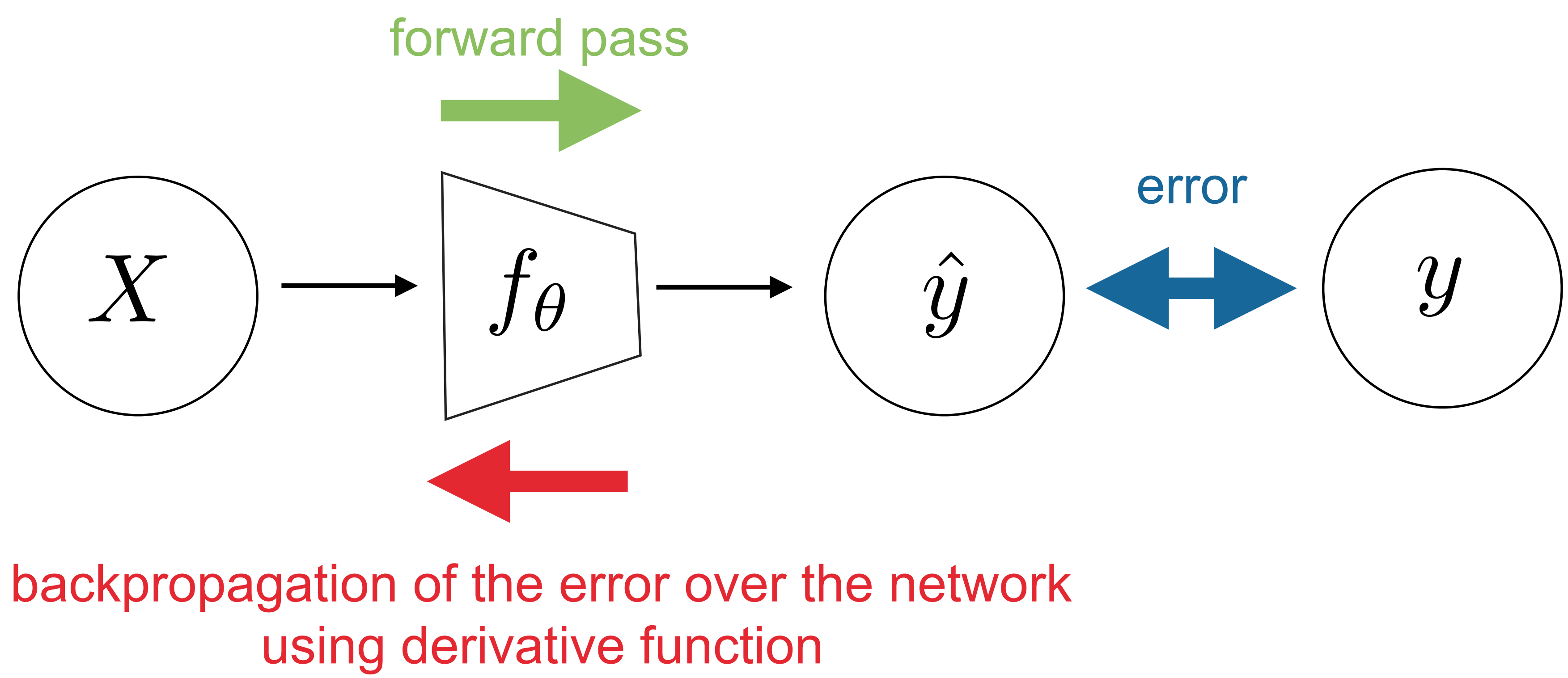


backpropagation of the error over the network
using derivative function

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu

BACKPROPAGATION

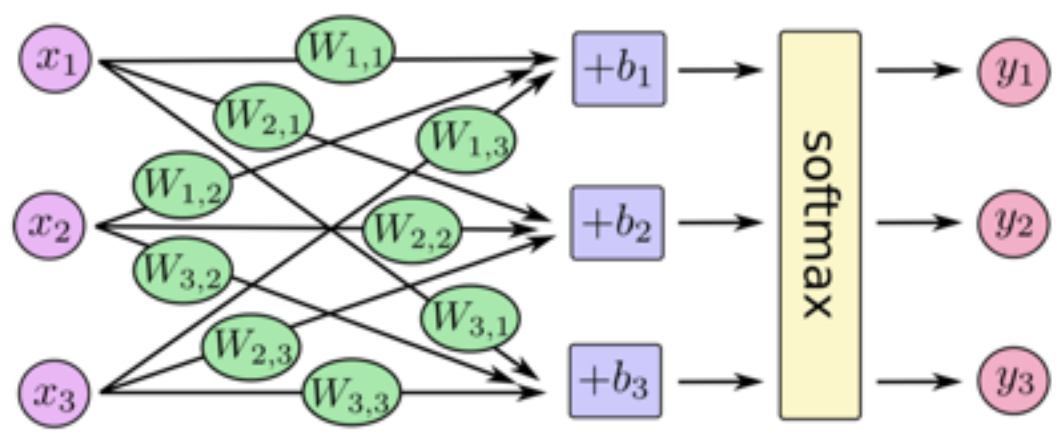
- **Forward Activation**: Predict the output
- **Compute the loss**
- **Backward Error**: And correct the parameters

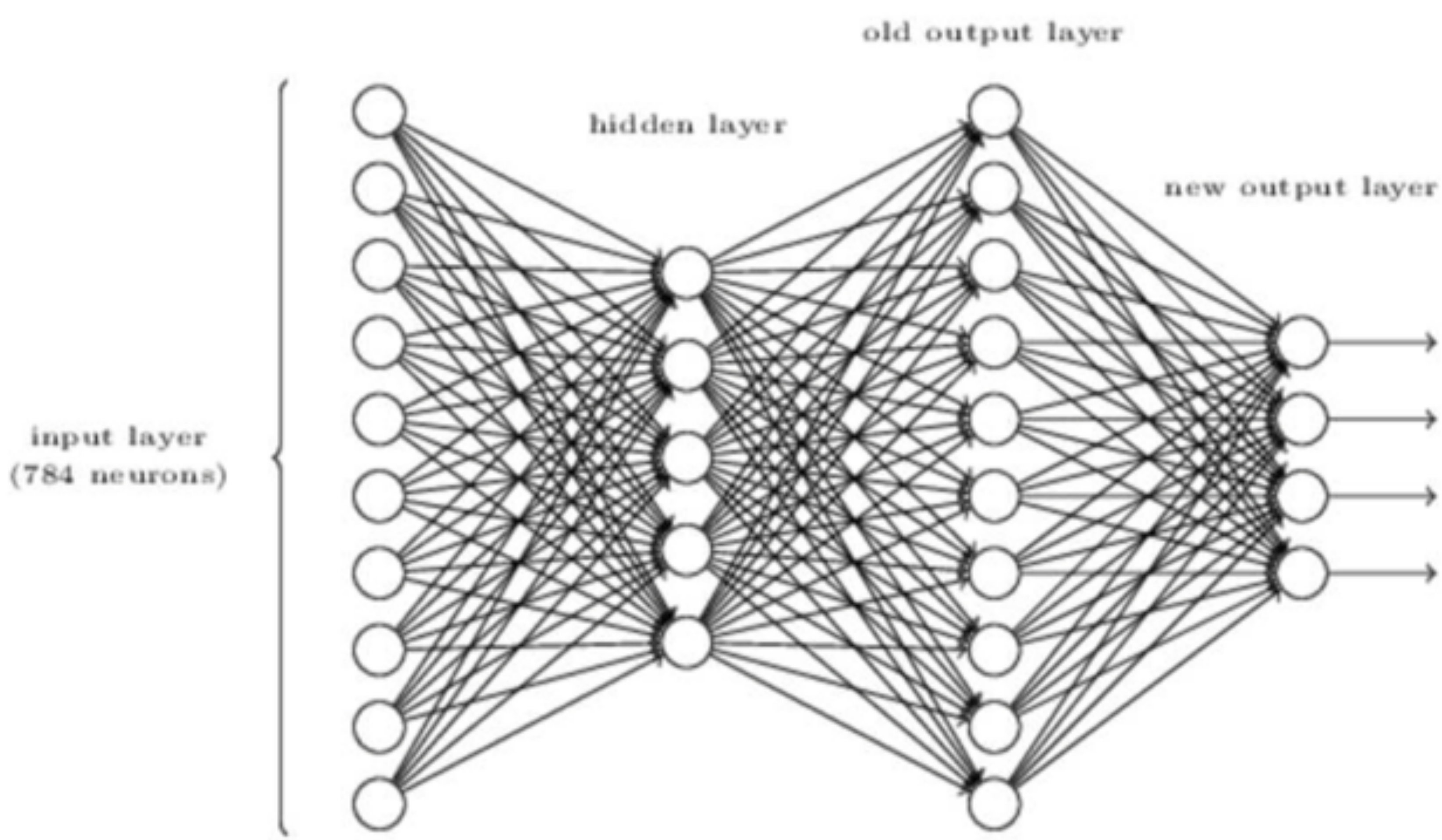


https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.l8cz02hlu

EXERCISES

- Go here: https://github.com/fabienbaradel/Tensorflow-tutorials
- And do the softmax and multilayer perceptron exercises



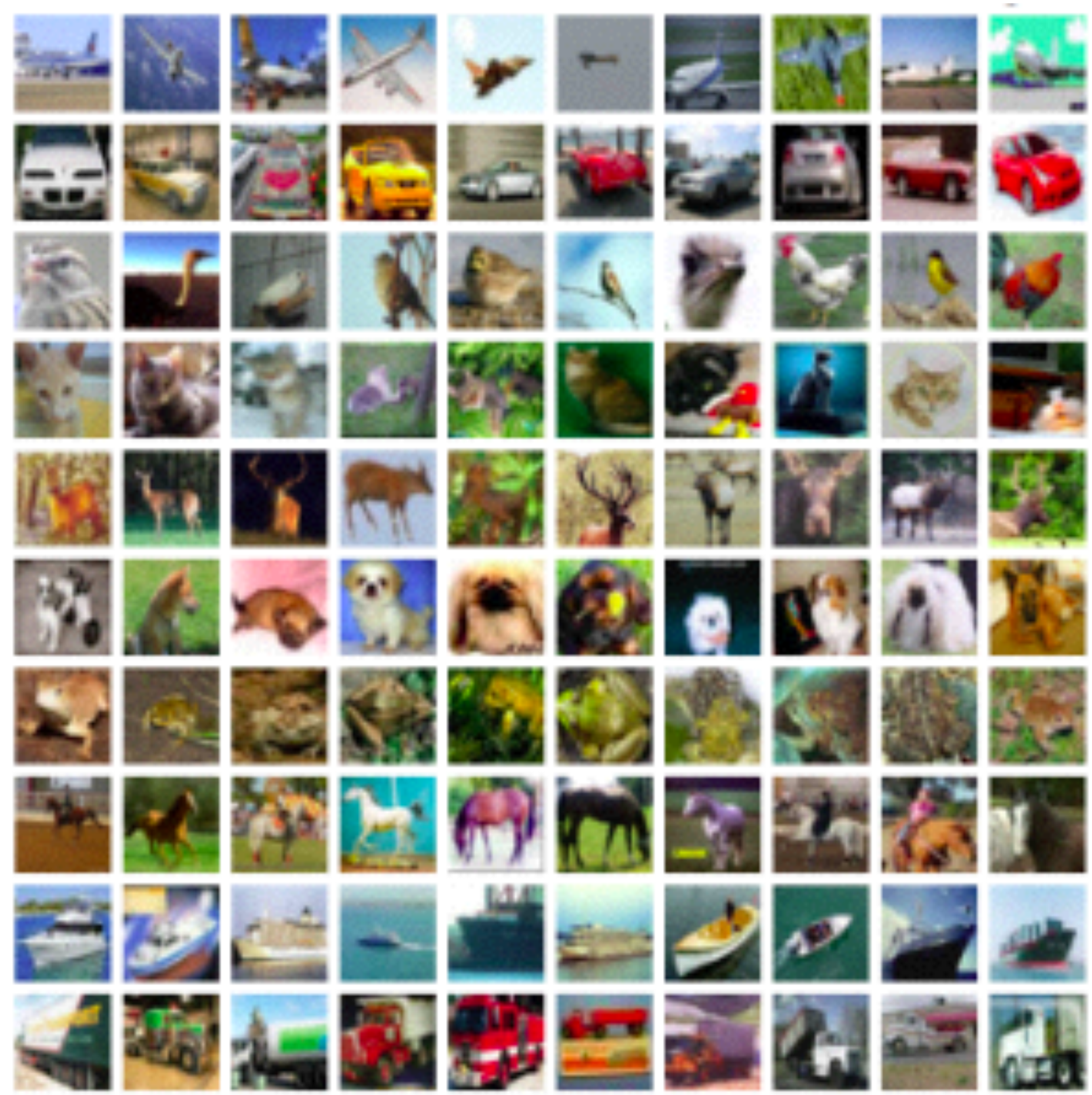


CONVOLUTIONAL NETWORKS

CONVNET

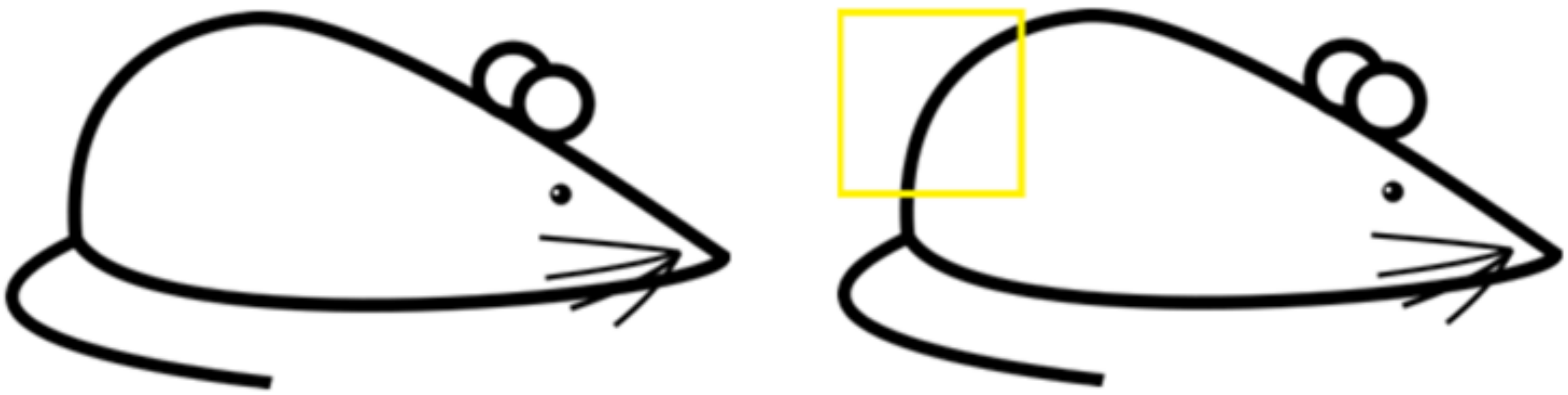
<u>« Convolutional neural networks »</u>

- Created by Yann LeCun (90's)
- Well-know since 2000
- Big acceleration with GPUs
- Computer vision
- NLP
- Artificiel Intelligence
- Convolution & Pooling



ConvNets usually evaluated on ImageNet (5 millions images, 1000 classes)

CONVOLUTION

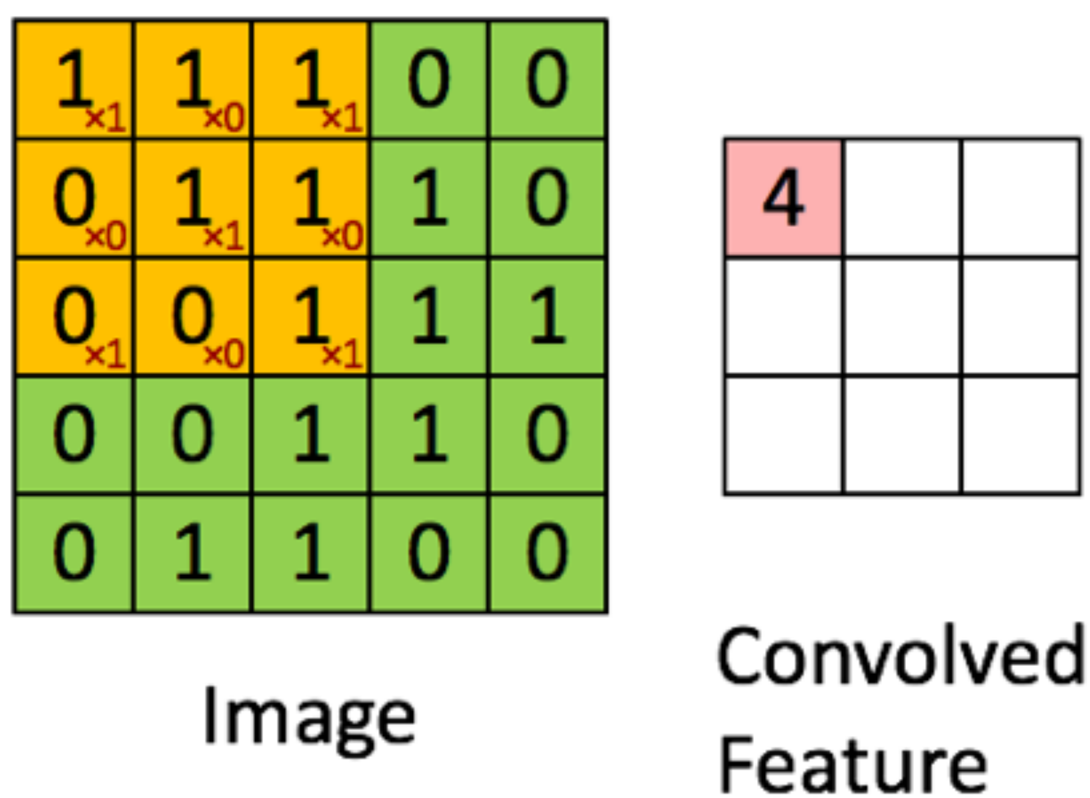


CONVOLUTION

- **Finding information in subpart of the image**
- **Local spatial correlation**
- **Mimic the biological process**
- **Less parameters than fully-connected layer**

Example: convolution on 5x5 matrix (1 filter=3x3 et stride=1)

Input: (5,5,1)



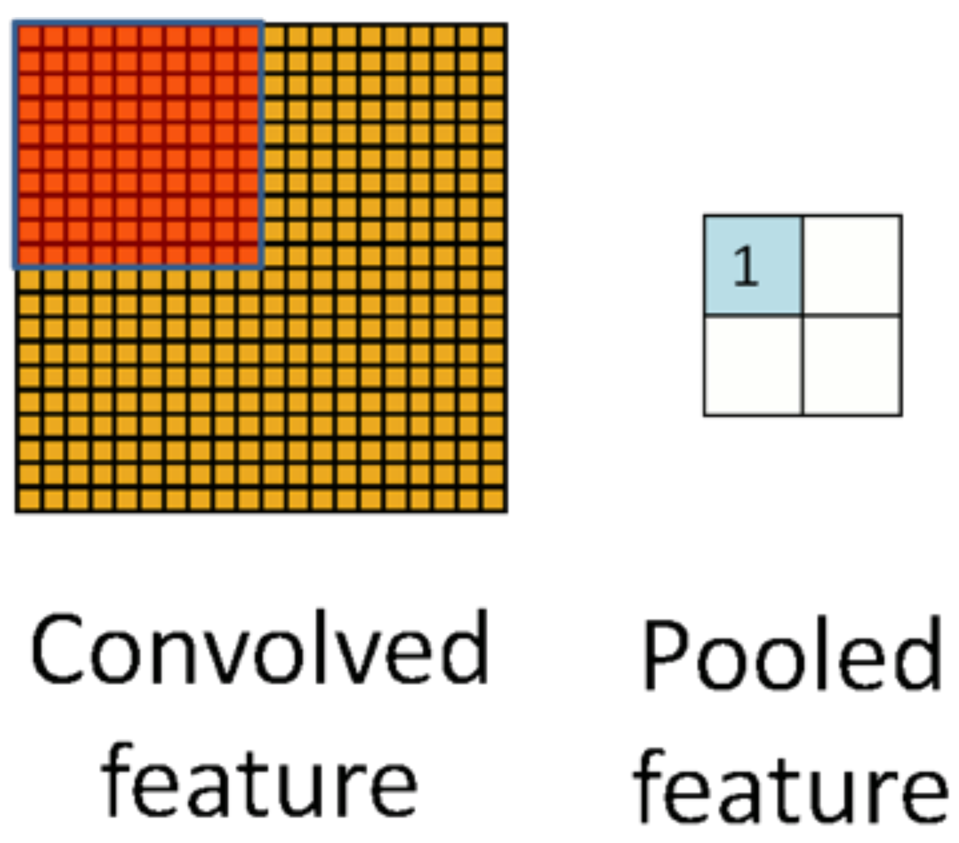
Ouput: (**nb_filter**,3,3)

http://dl.heeere.com/convolution3/

POOLING

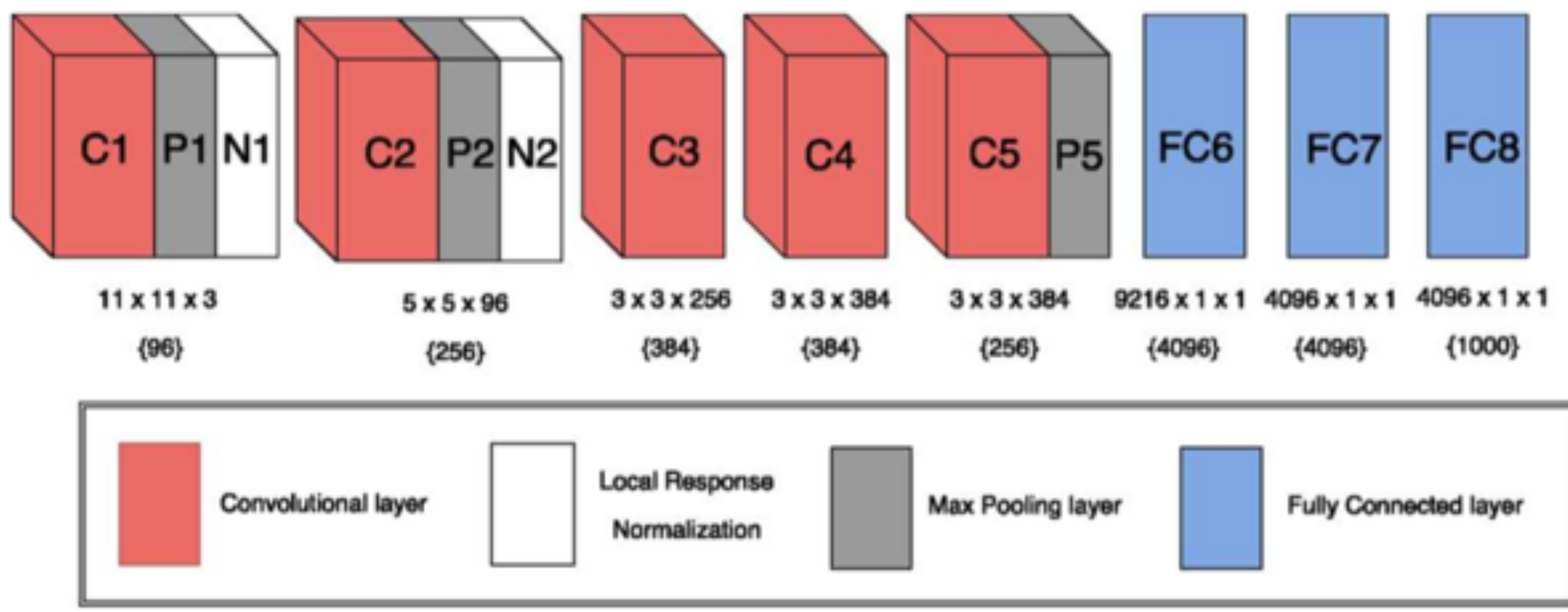
- **Sampling over a matrix**
- **Dimension reduction**
- **Reduce number of parameters of further layers**
- **No learnable parameters!**

Example: pooling over a 20x20 matrix (filter=10x10 et stride=10)



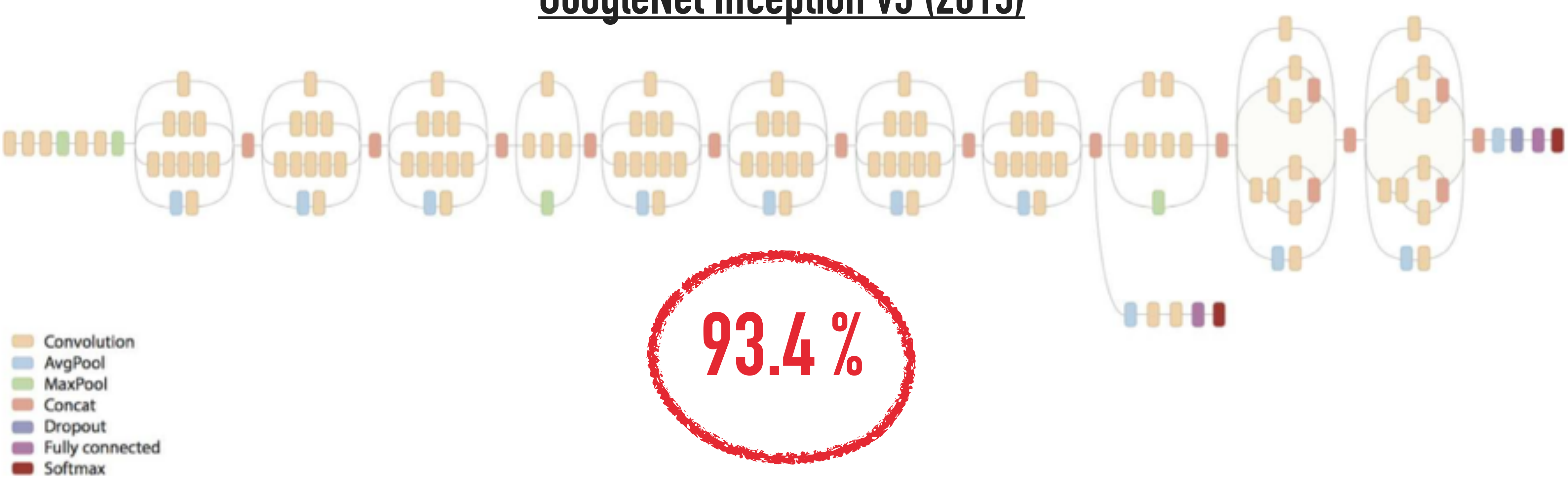
CONVNETS

AlexNet (2012)



80.1 %

GoogleNet Inception v3 (2015)



93.4 %

CONVNETS

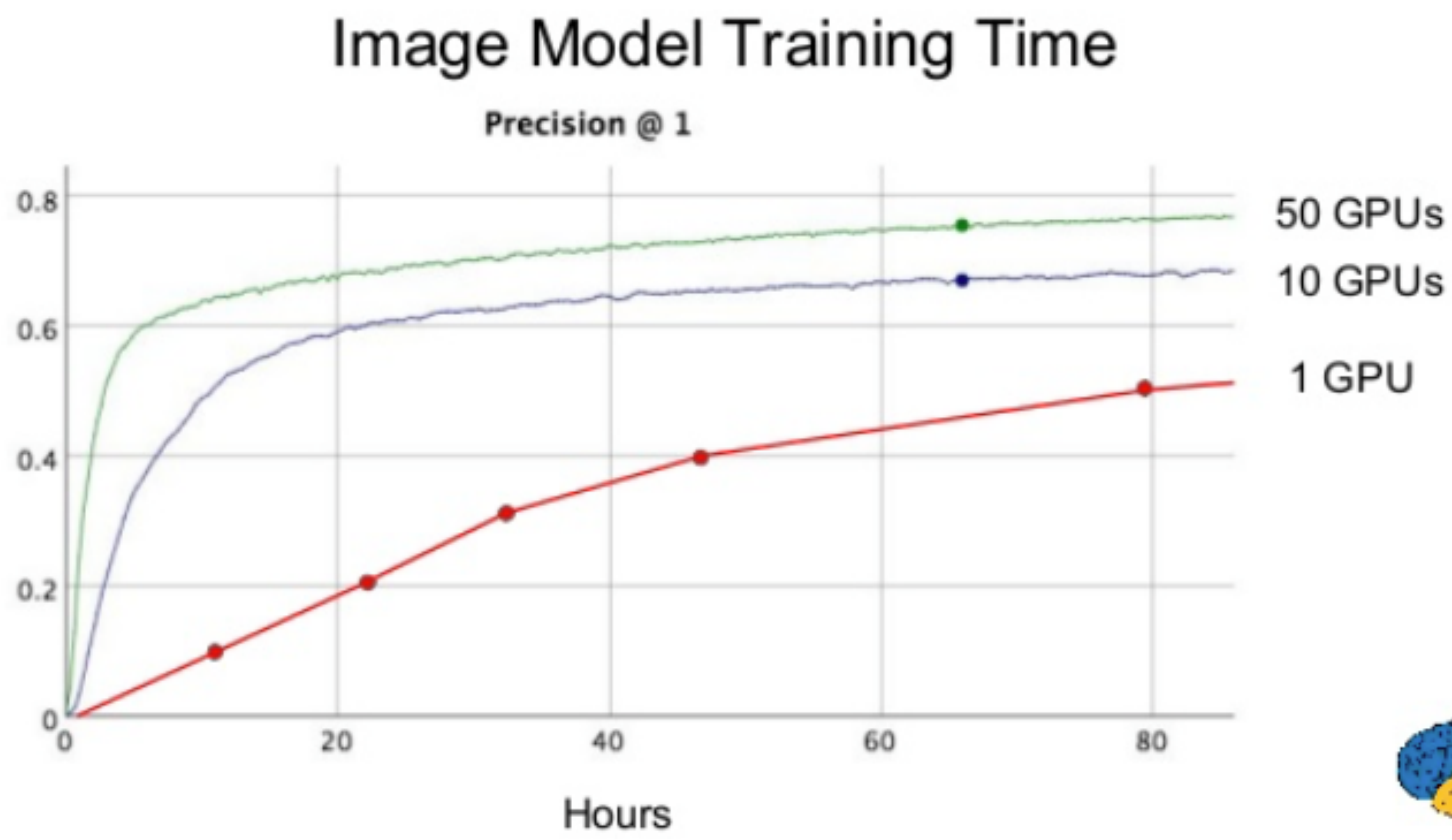


Image Model Training Time
Precision @ 1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
Hours
50 GPUs
10 GPUs
1 GPU

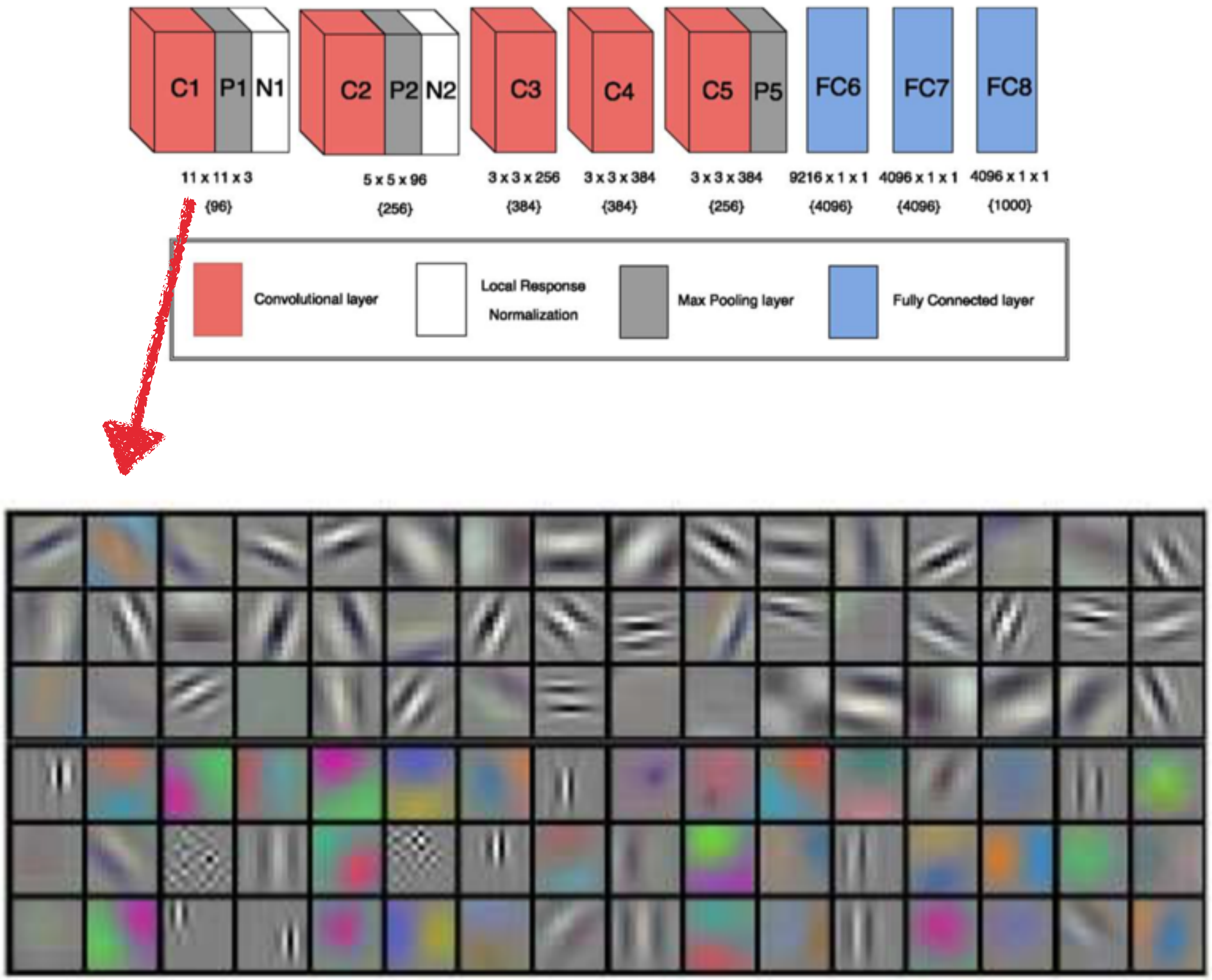
EXERCISES

- Complete the exercises:
  - ✳ **One Conv + Max Pool**
  - ✳ **LeNet**

HAVE A LOOK TO TF.SLIM TO MAKE YOUR LIFE EASIER

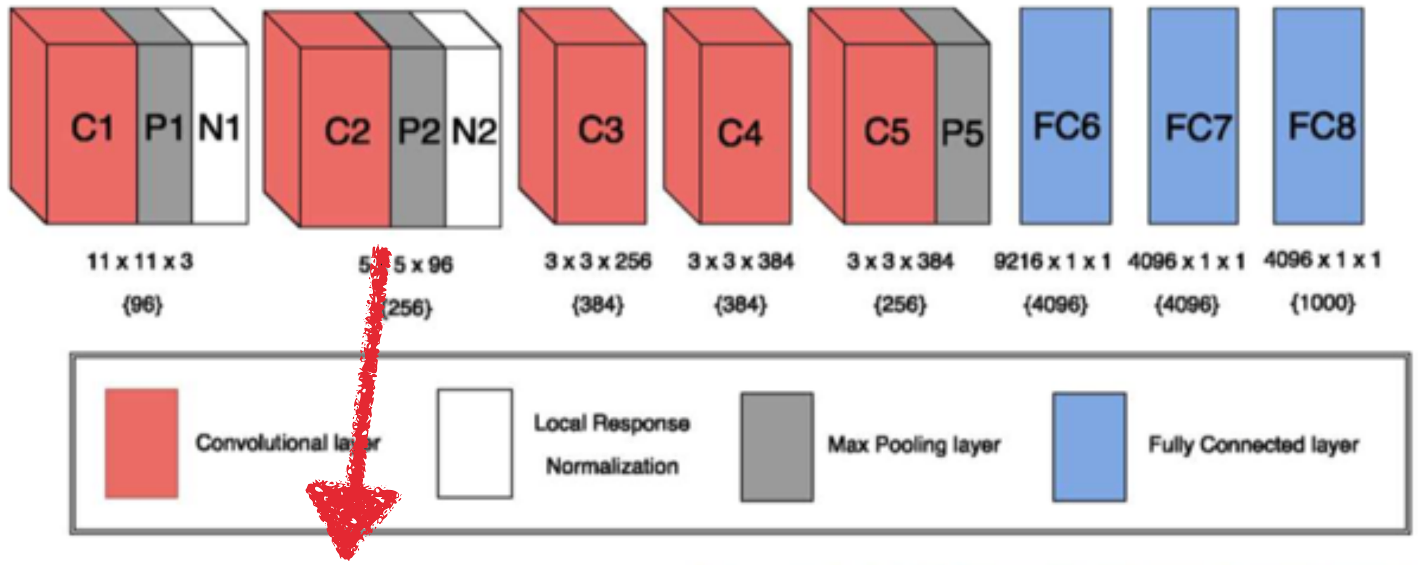
WHY CONVOLUTION WORKS?

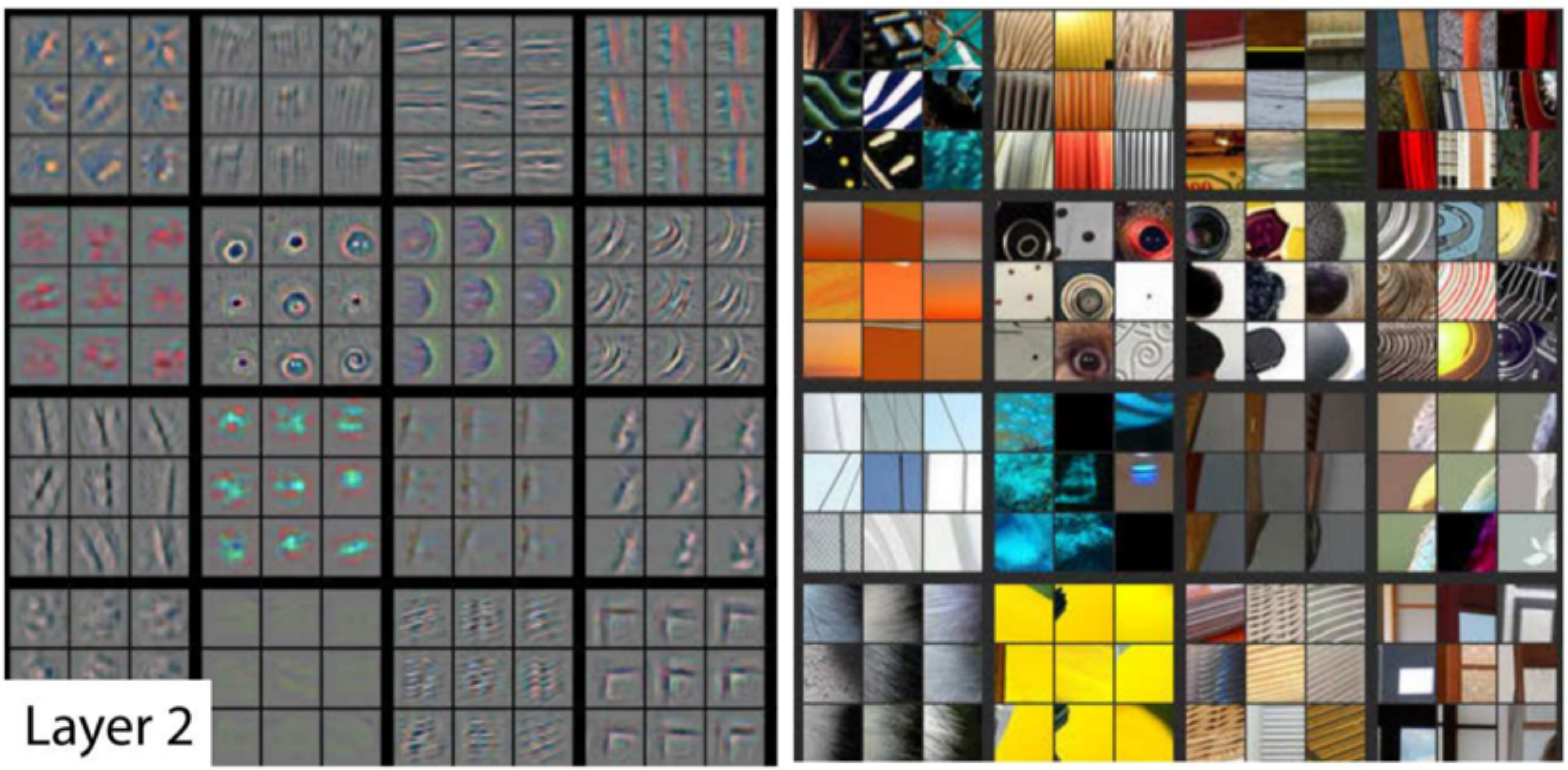
FEATURE MAPS



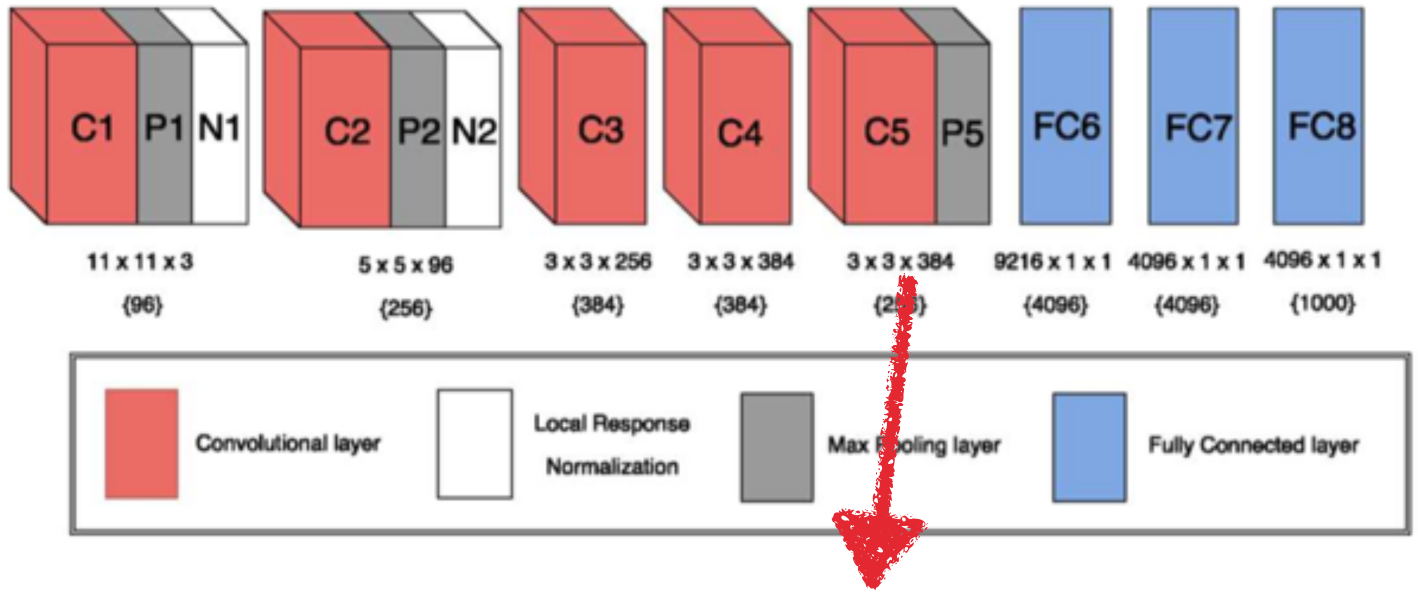
Layer 1: ~ Gabor filters

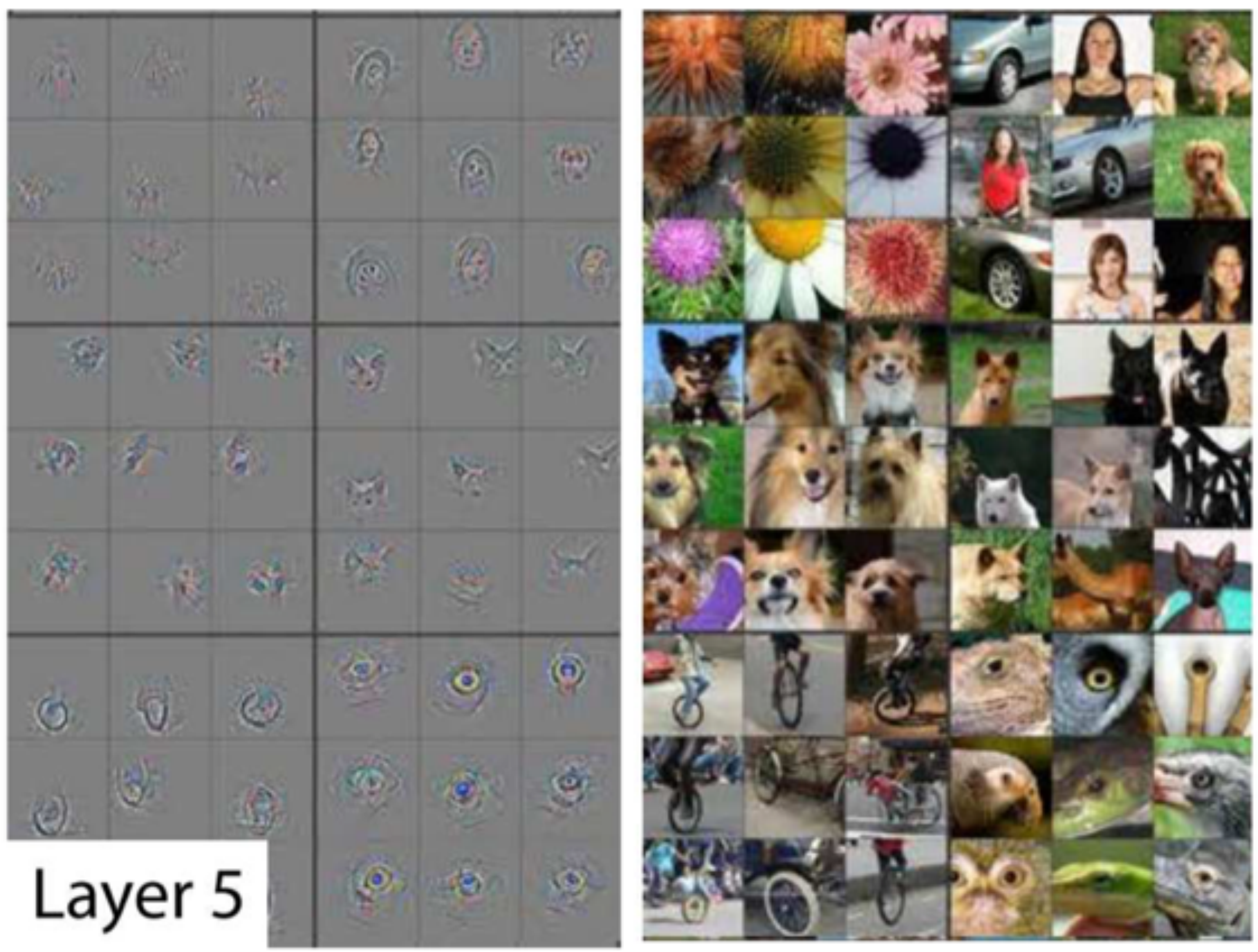
FEATURE MAPS



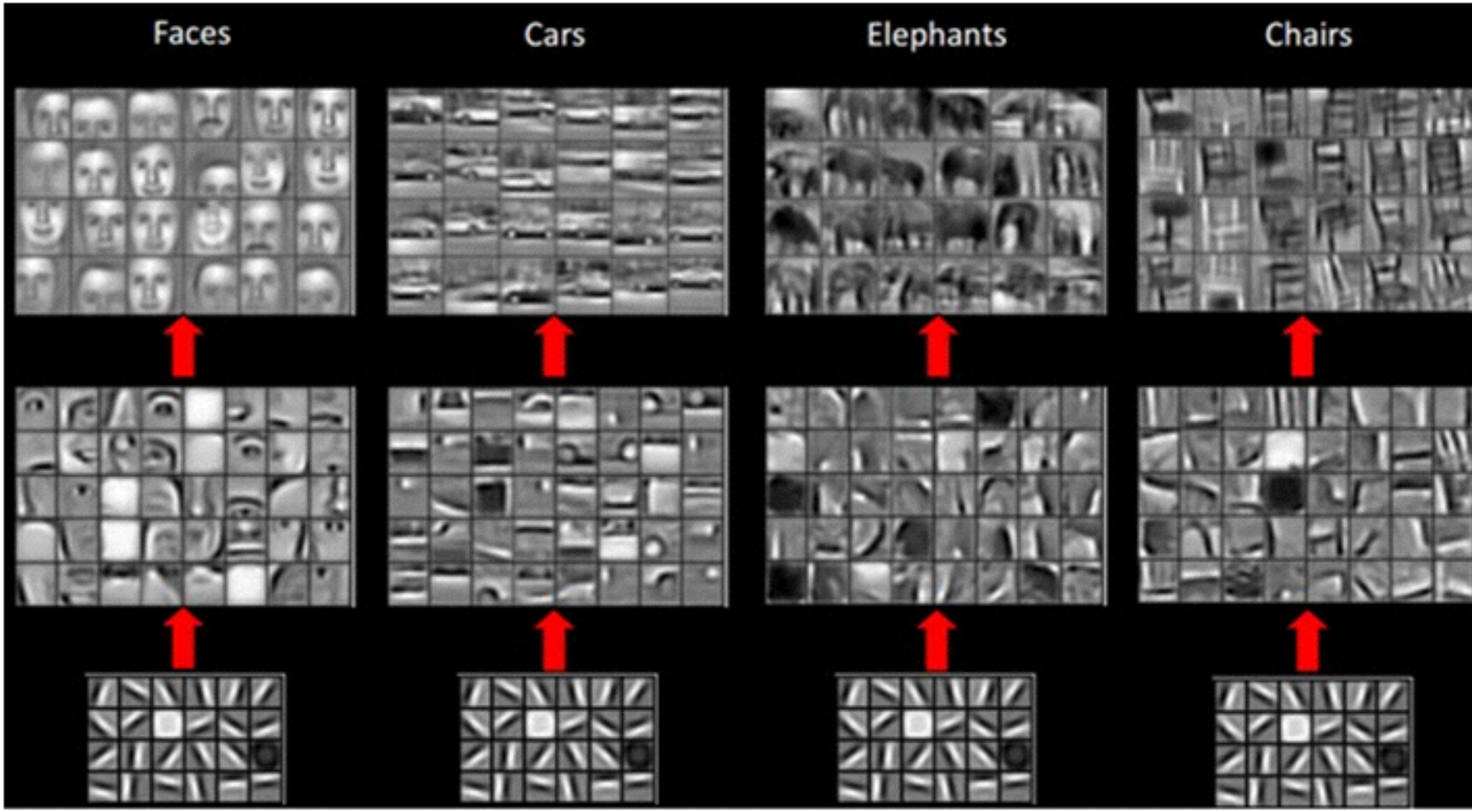


FEATURE MAPS



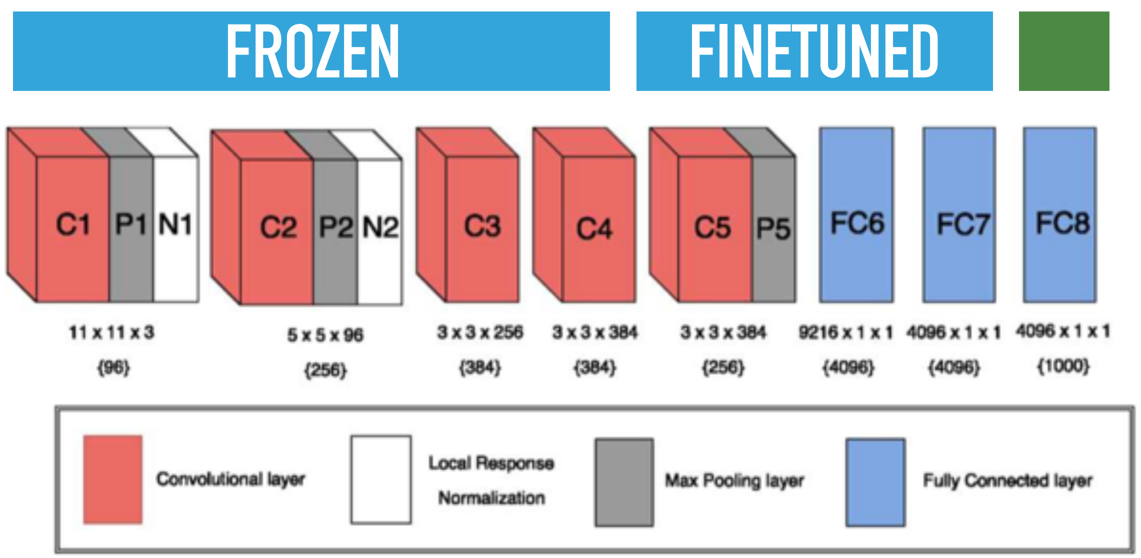


FILTERS



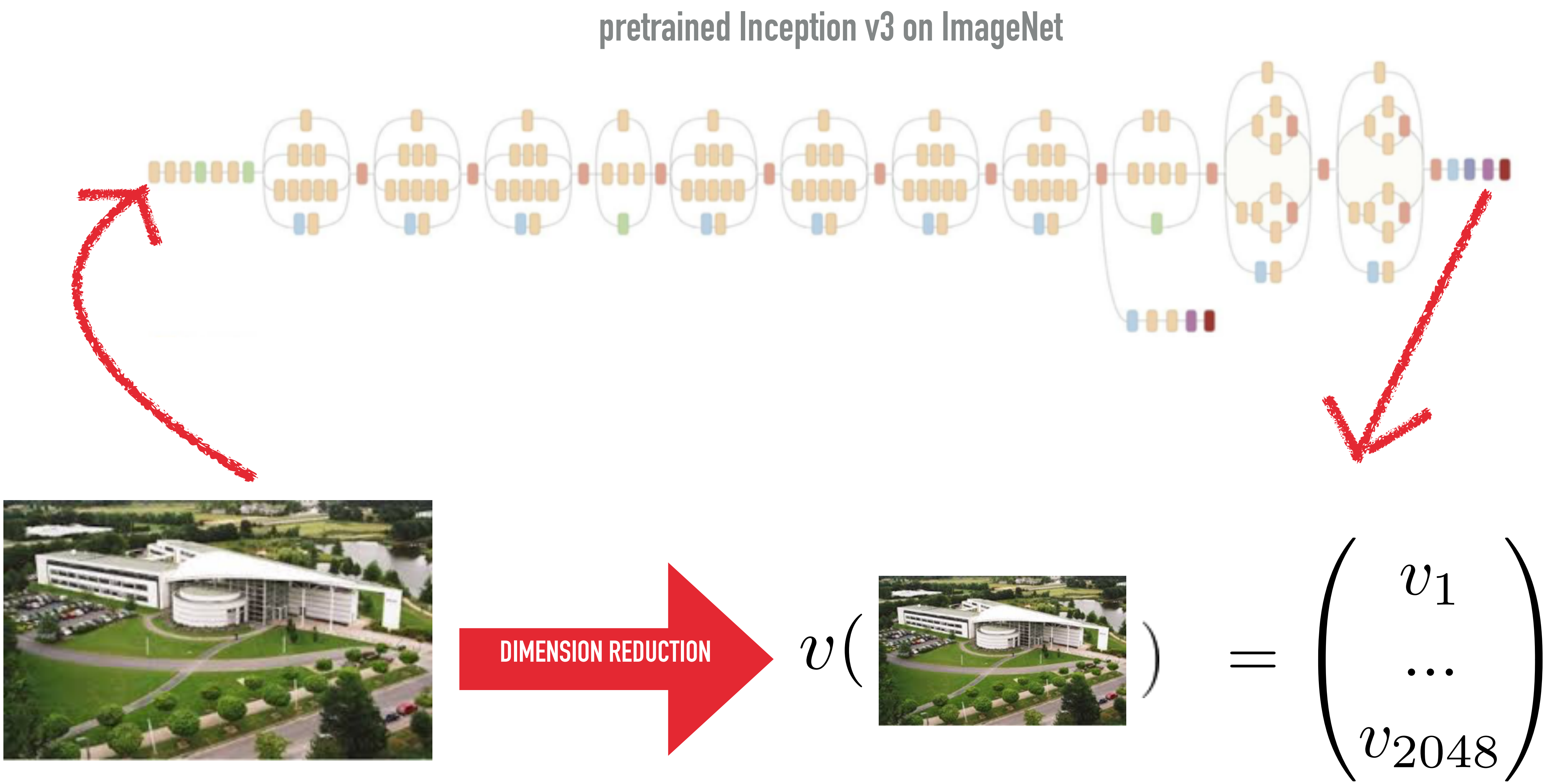
https://www.youtube.com/watch?v=AgkfIQ4IGaM

FINE-TUNING



- Filters after first convolutional layer are generic (Gabor filters)
- Deeper you go in network and more task specific are your filters

FINE-TUNING


pretrained Inception v3 on ImageNet
DIMENSION REDUCTION
$v($ $)$ $=$
v_1
$\dots$
v_{2048}

EXERCISE

- Wanna win $150’000? YES YOU CAN!
- Go to the Github repo and do the « Classification from DeepFeatures » exercise
- And submit your .csv in Kaggle (and cross your fingers)

WHAT IS YOUR SCORE?

…

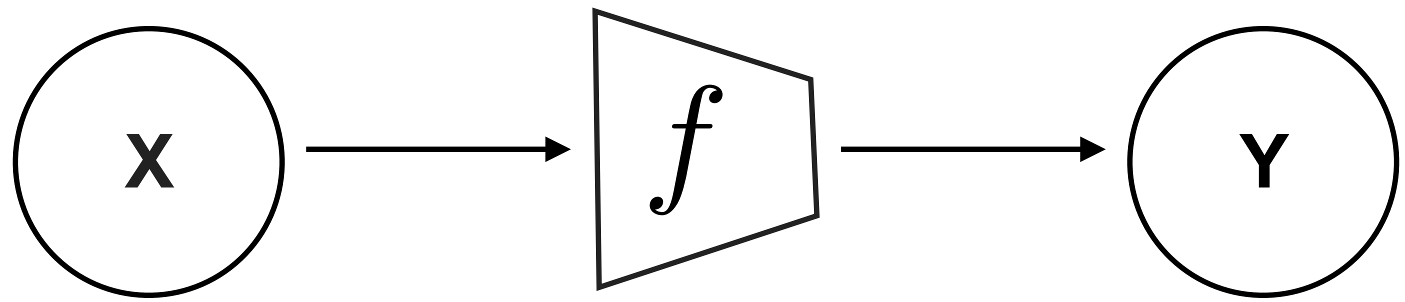
Want to add convolution?

Reshape your vector to a 3D matrix…

*2048 = 32*32*3*

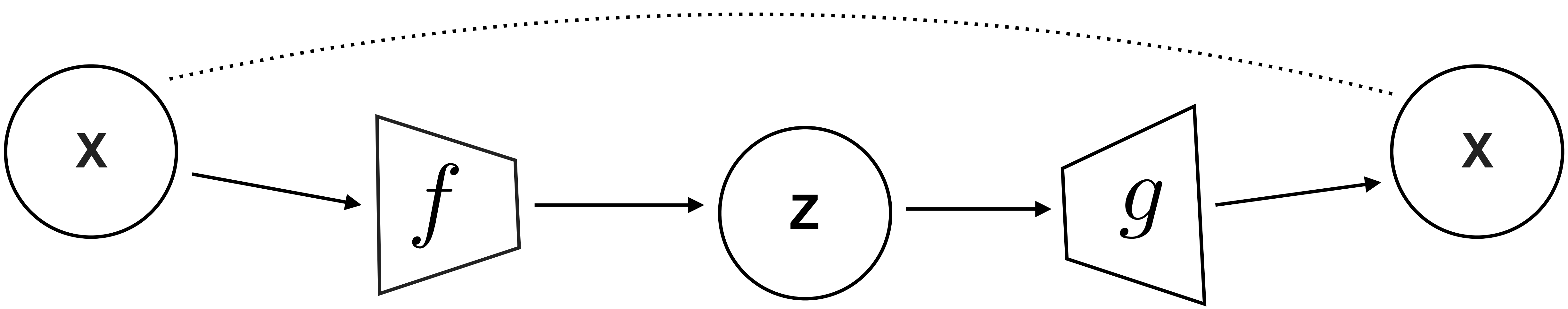
AUTOENCODER

NEURAL NETWORK LEARNING



Supervised learning

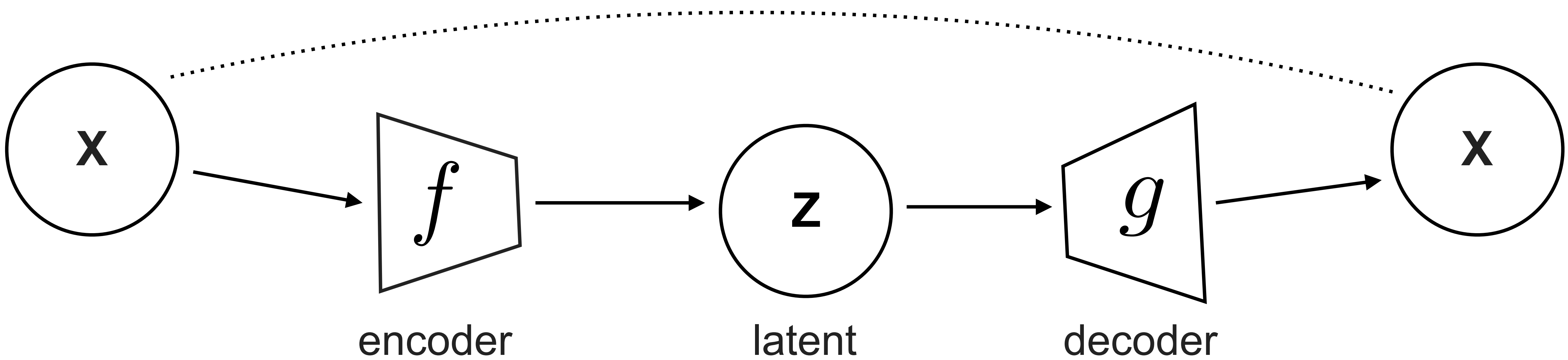
- y are given !



Unsupervised learning

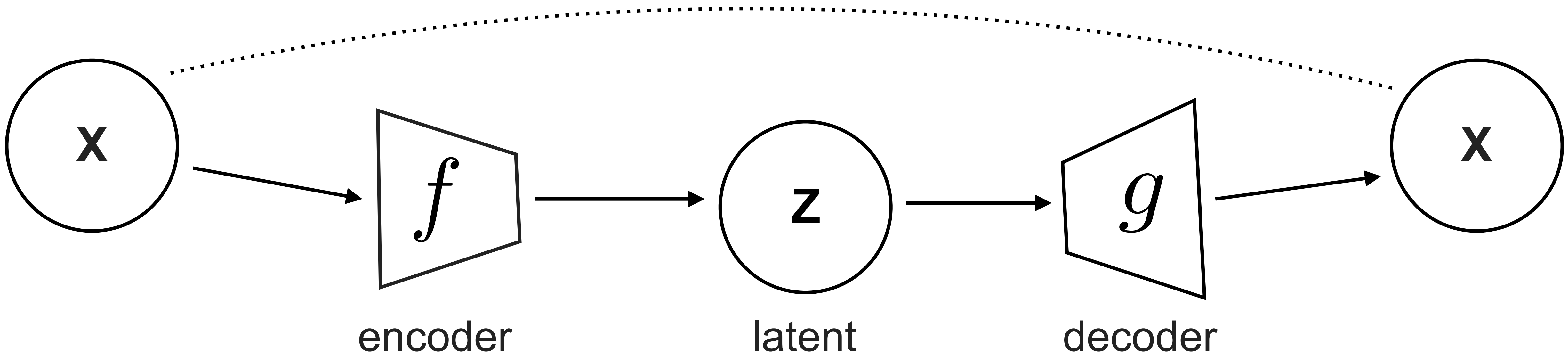
- y is no longer needed

AUTOENCODER



- <u>Learning a compact data representation</u>
- Encode input to smaller latent space
- Decode from the latent space to the input
- **Predict input from input**
- Loss function = mean square error
- **f** and **g** are neural networks
- SGD as usual

AUTOENCODER

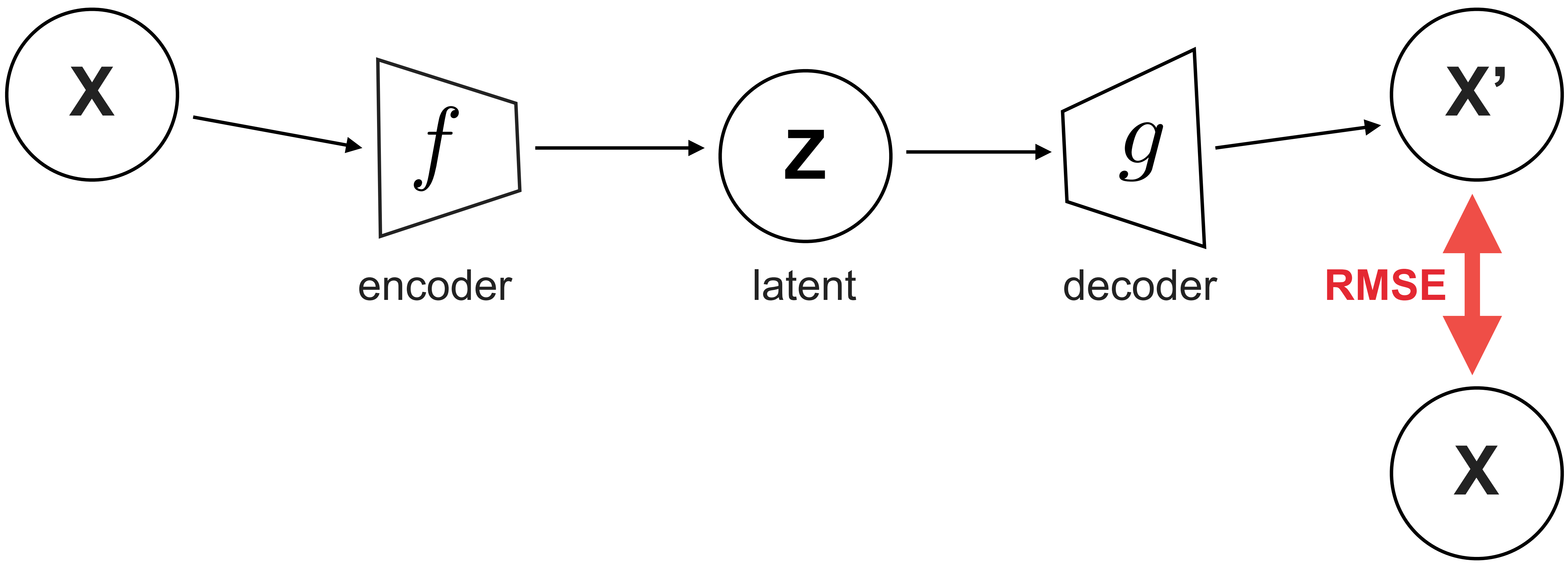


f and g are <u>linear without hidden layer</u>

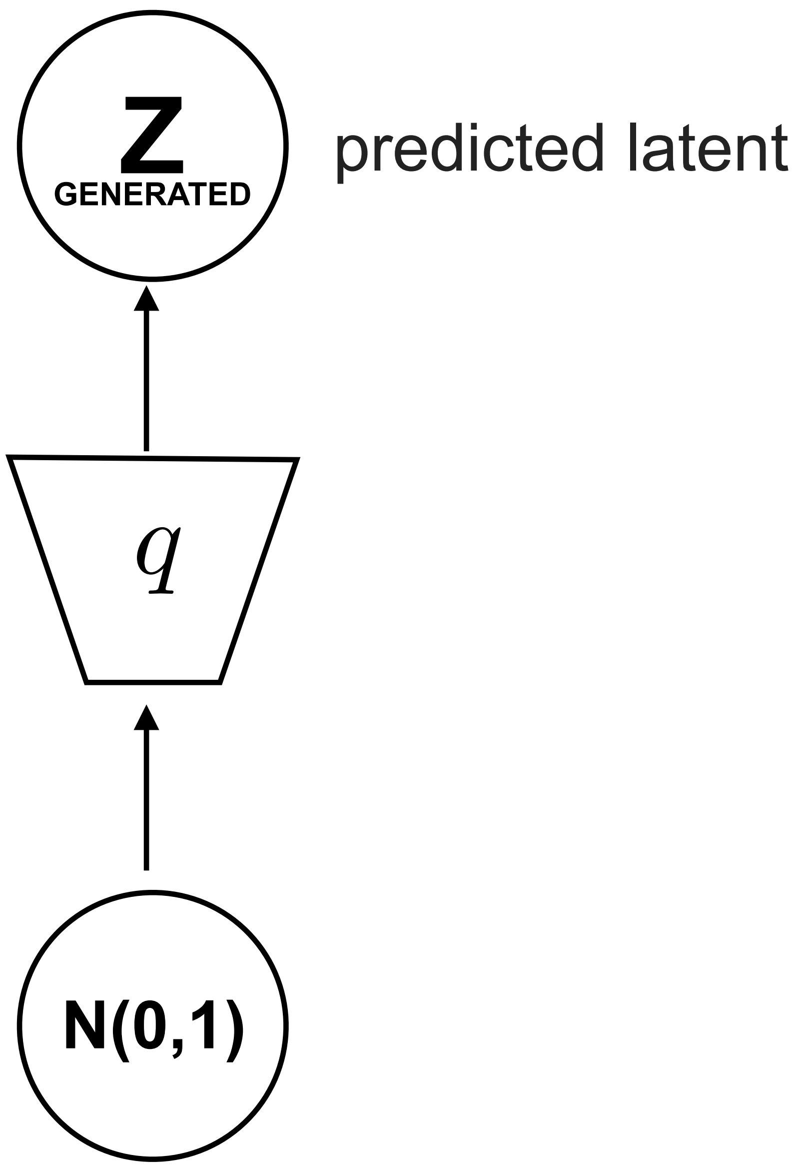
=> your solution is an approximation of a PCA

GENERATIVE MODELS

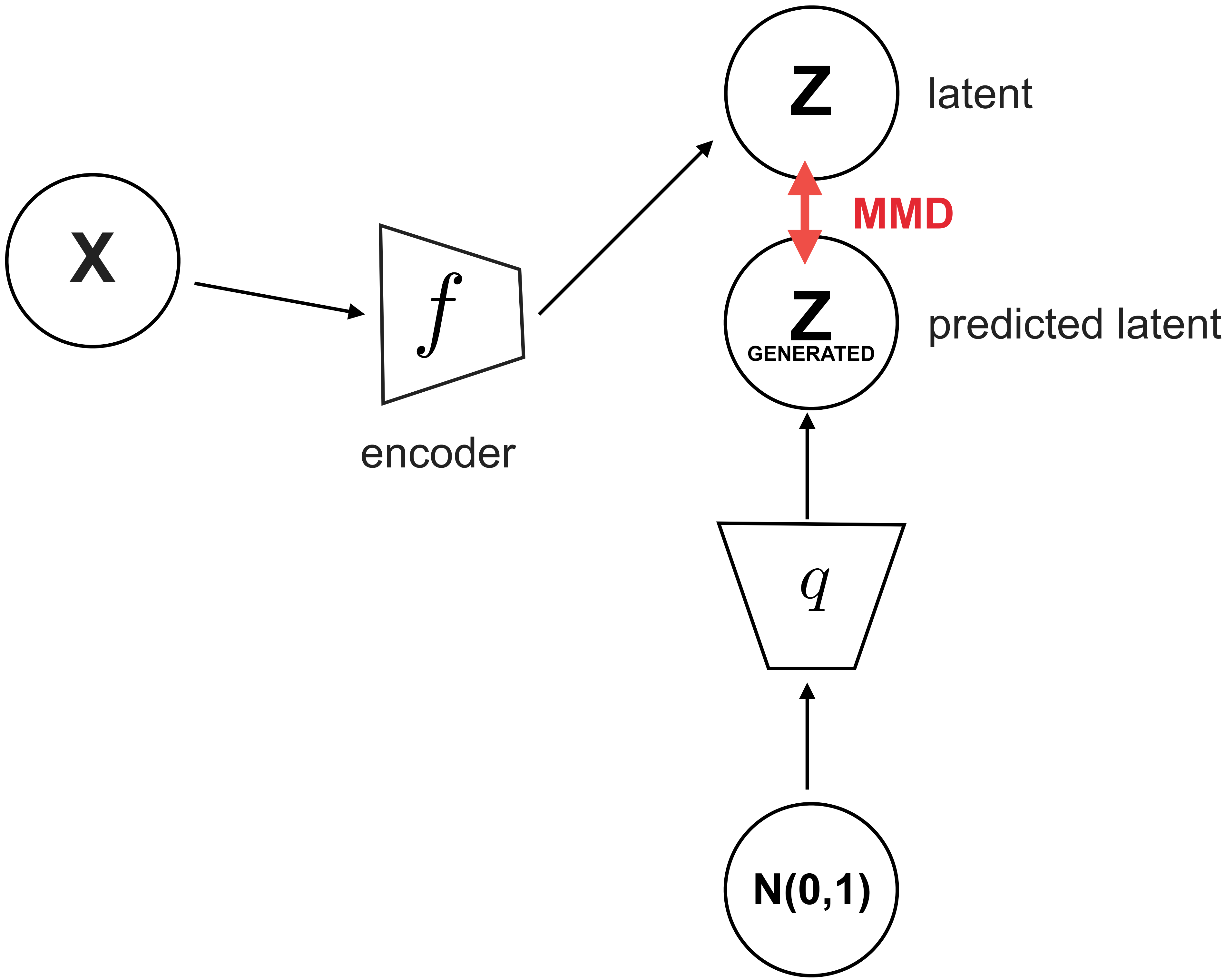
GENERATIVE MOMENT MATCHING NETWORKS



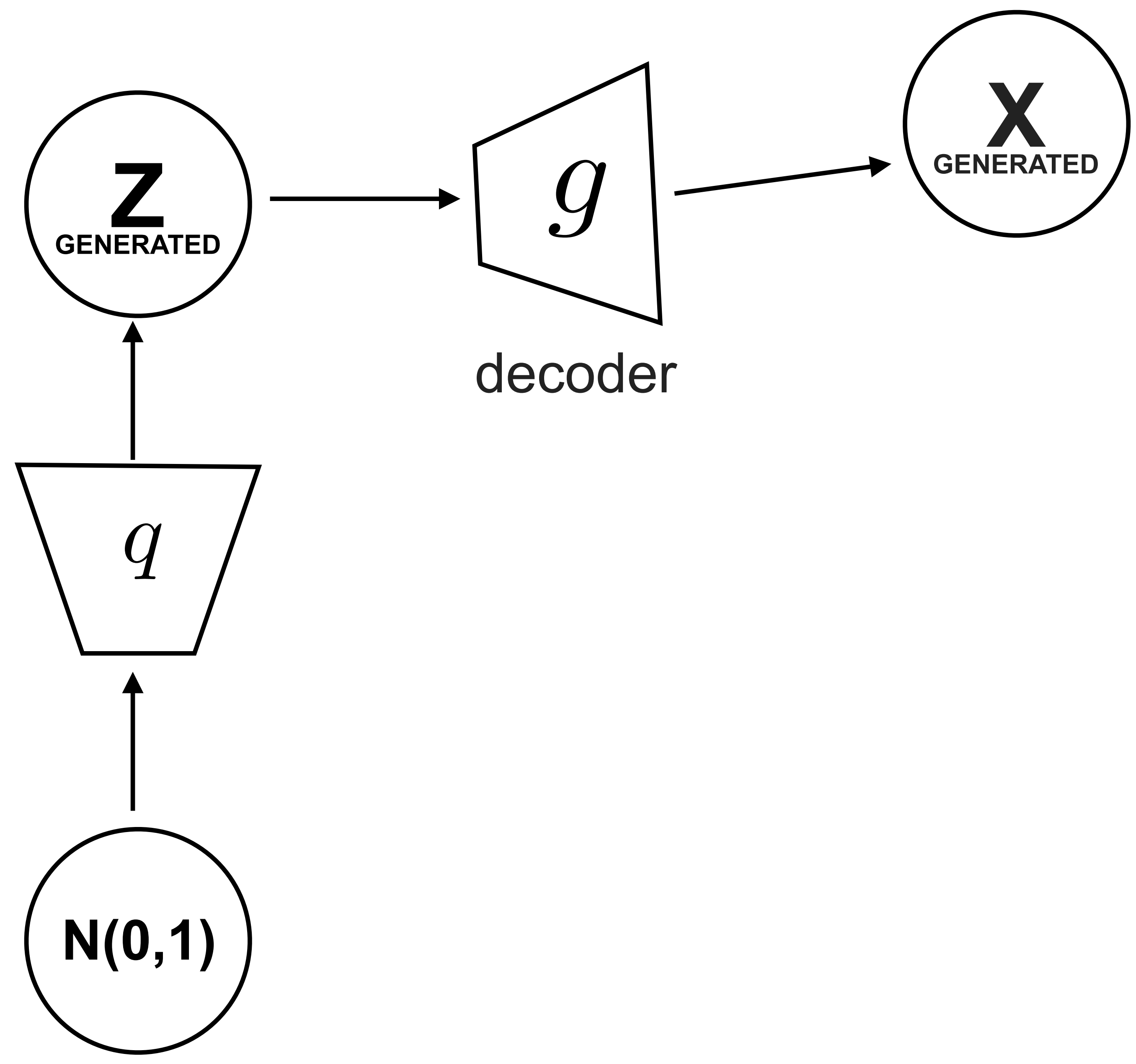
GENERATIVE MOMENT MATCHING NETWORKS



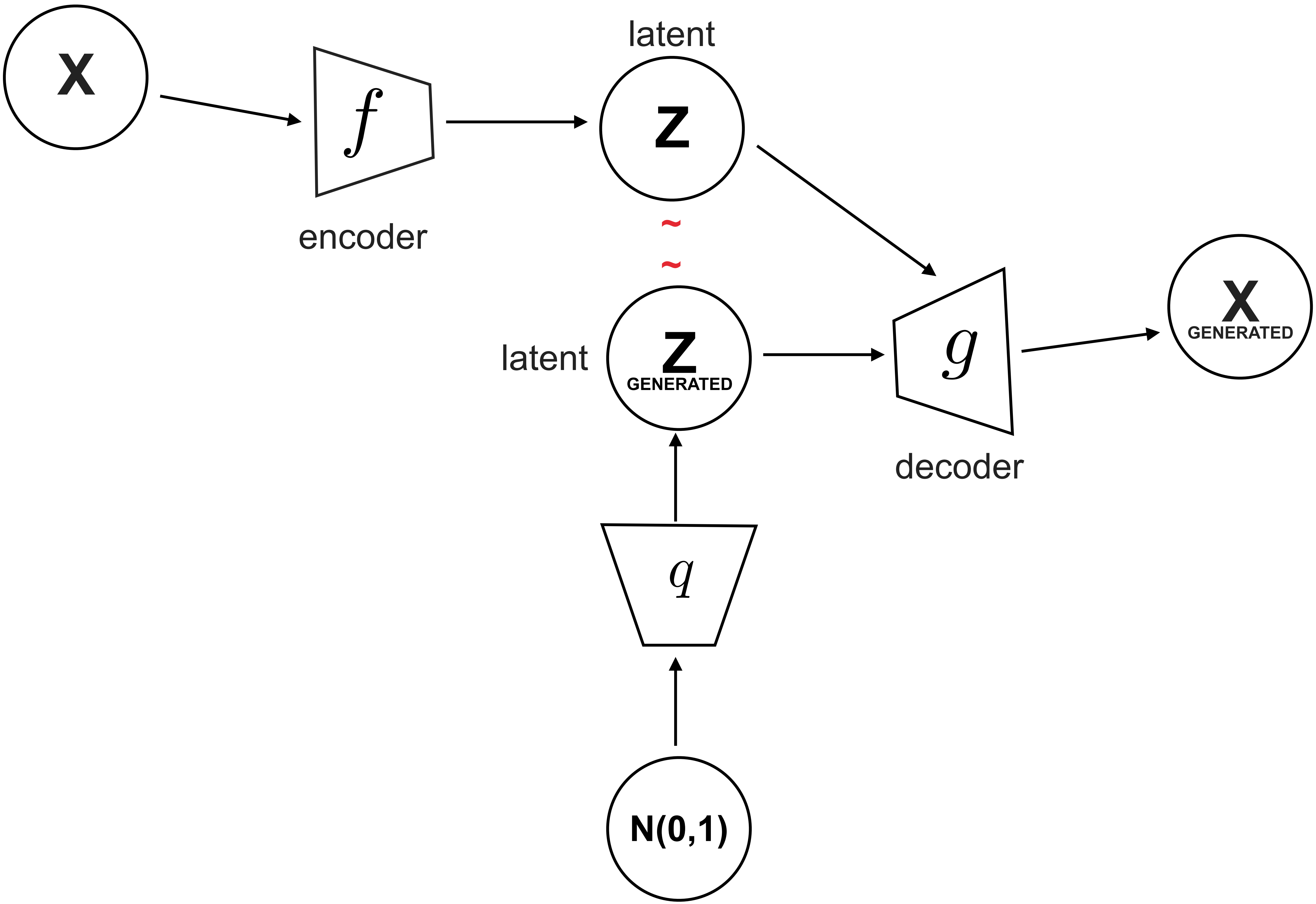
GENERATIVE MOMENT MATCHING NETWORKS



GENERATIVE MOMENT MATCHING NETWORKS



GENERATIVE MOMENT MATCHING NETWORKS



GENERATIVE ADVERSIAL NETWORKS

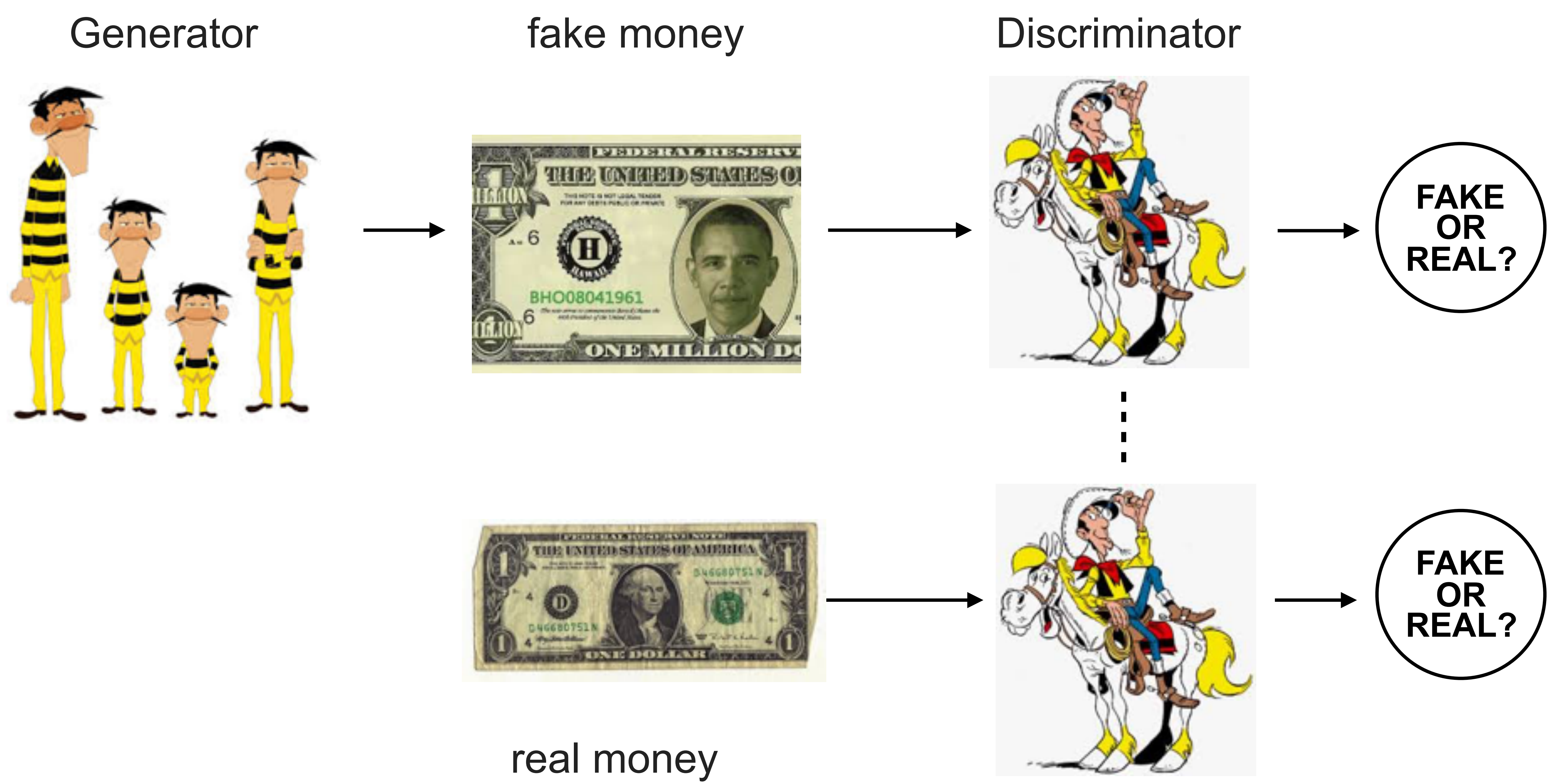
Intuition

Generator

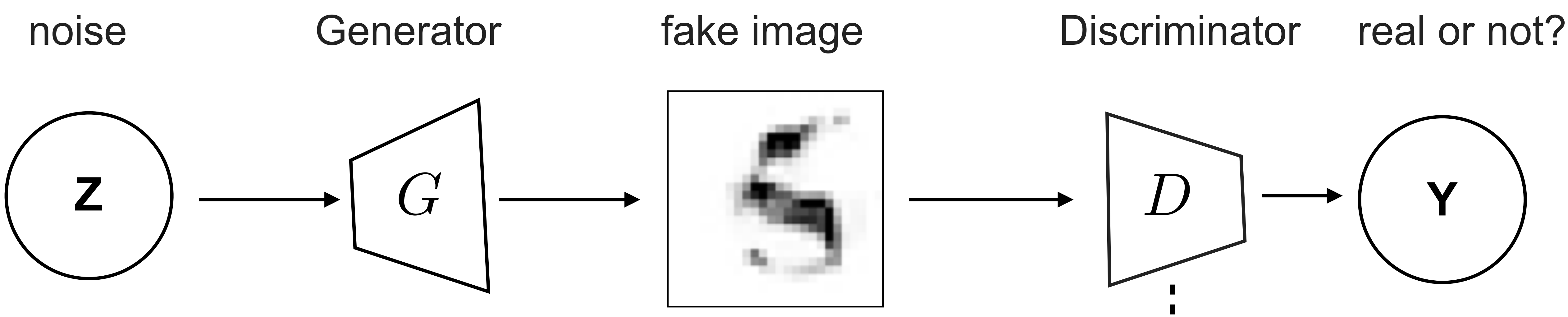
fake money

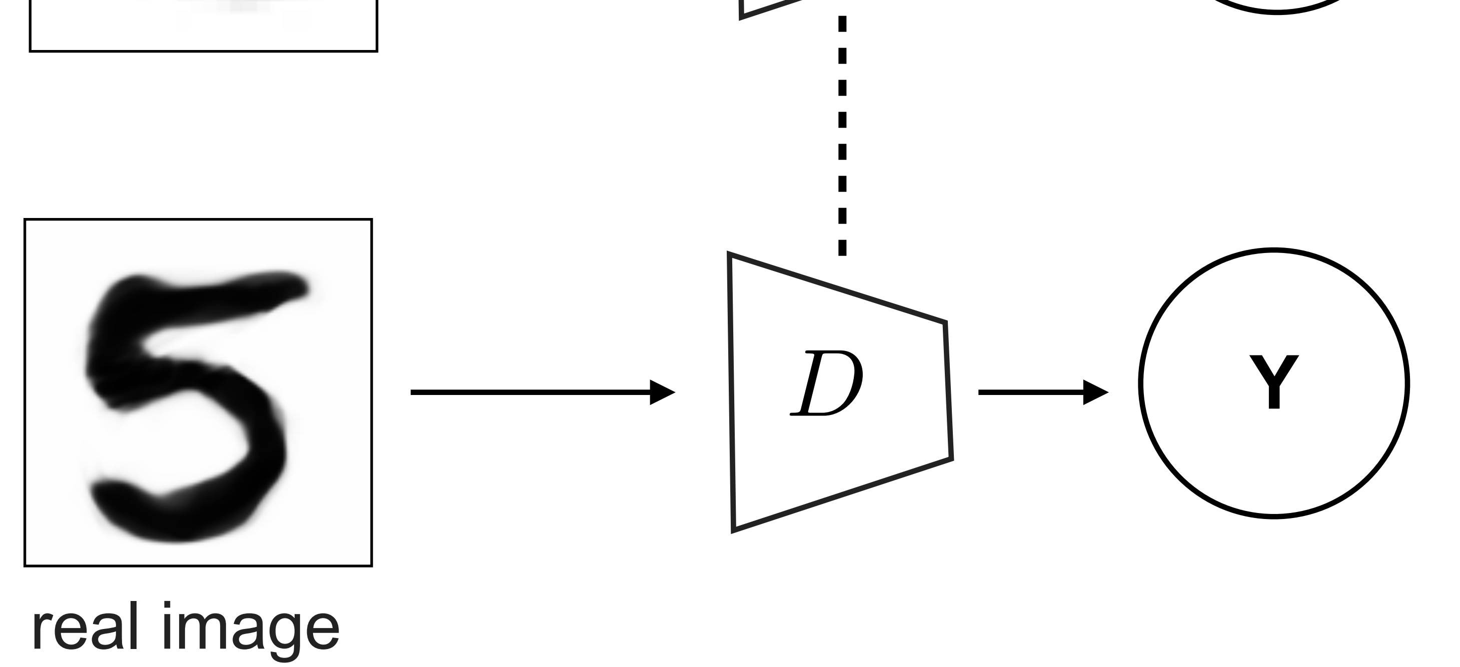
GENERATIVE ADVERSIAL NETWORKS

Intuition



GENERATIVE ADVERSIAL NETWORKS





- G and D are neural networks
- **Find a G that minimizes the accuracy of the best D**
- **Alternate optimization of G and D**

$$\min_G \max_D V(D,G) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z})))]}.$$

log prob of D predicting that real-world data is genuine

log prob of D predicting that G's generated data is not genuine

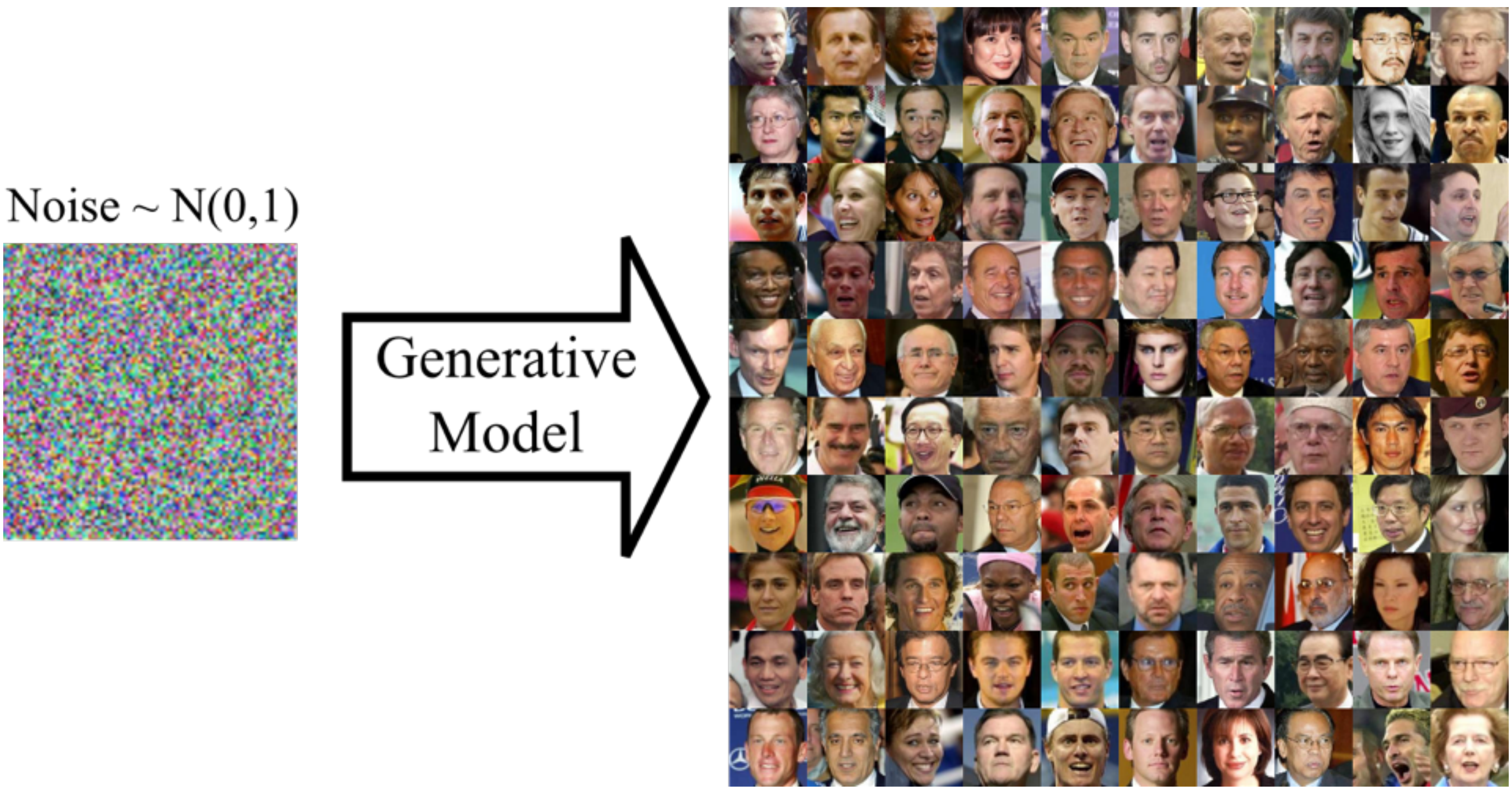
http://blog.aylien.com/introduction-generative-adversarial-networks-code-tensorflow/

GAN: EXAMPLES

- DCGAN (Deep Convolutional GAN), Radford et al., 2015/2016

Groundtruth MNIST GAN DCGAN (ours)

GAN: EXAMPLES



GAN: EXAMPLES

Ongoing topic…

EXERCISE

- Complete the exercises:
  - « Autoencoder_exo »
  - « Conv-Deconv Autoencoder_exo »
  - And GMMN if you are fast enough!

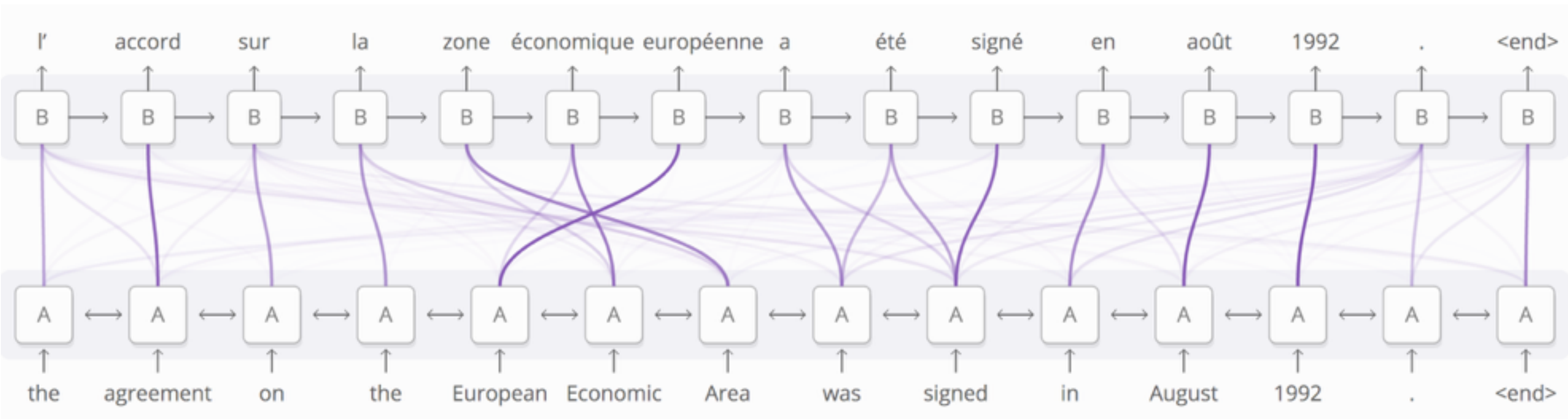
SEQUENCE MODELING

WHAT ABOUT SEQUENCE?

Image = Static
(almost) Solved

Vidéo = Sequence of images
not solved at all…

WHAT ABOUT SEQUENCE?

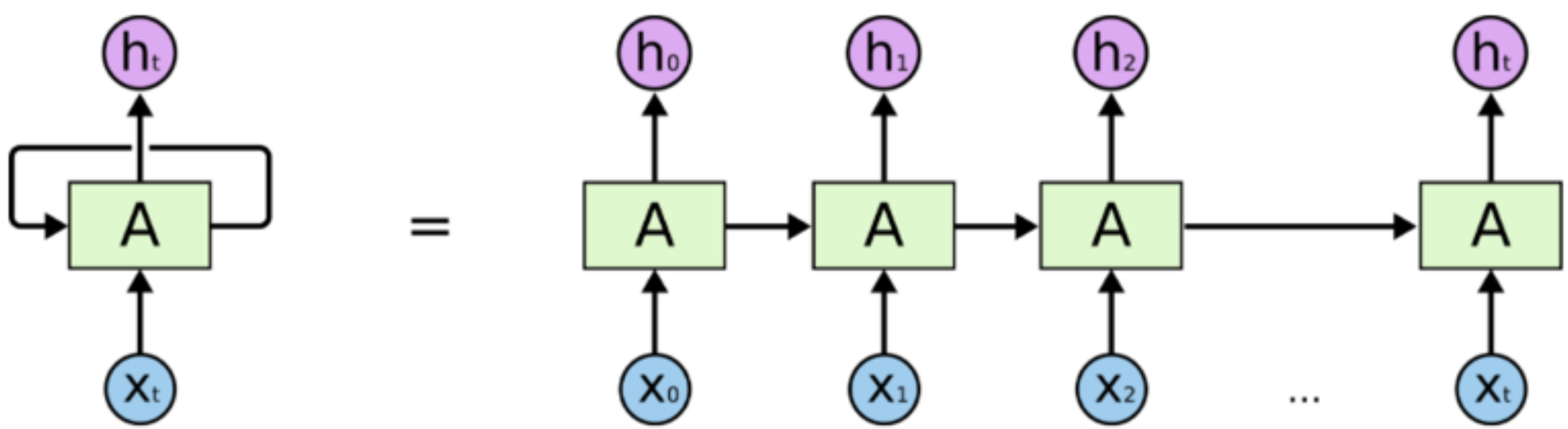


Sequence to sequence:

Machine Translation

RECURRENT NEURAL NETWORK

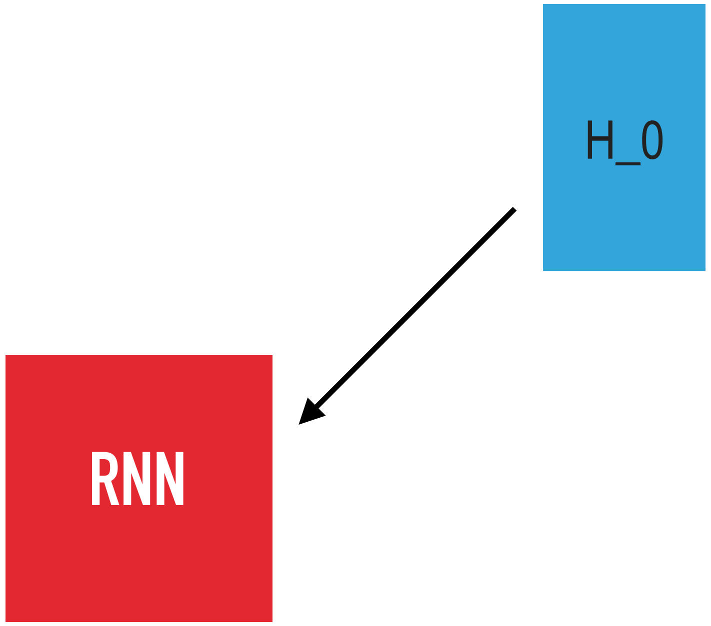
- Imagine **X** as a time series : **(x1, x2, …, xn)**
- **h** is the hidden state of the RNN
- Initialized at (1,1,…,1) at t=0
- And **h is modified after each timestep**



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

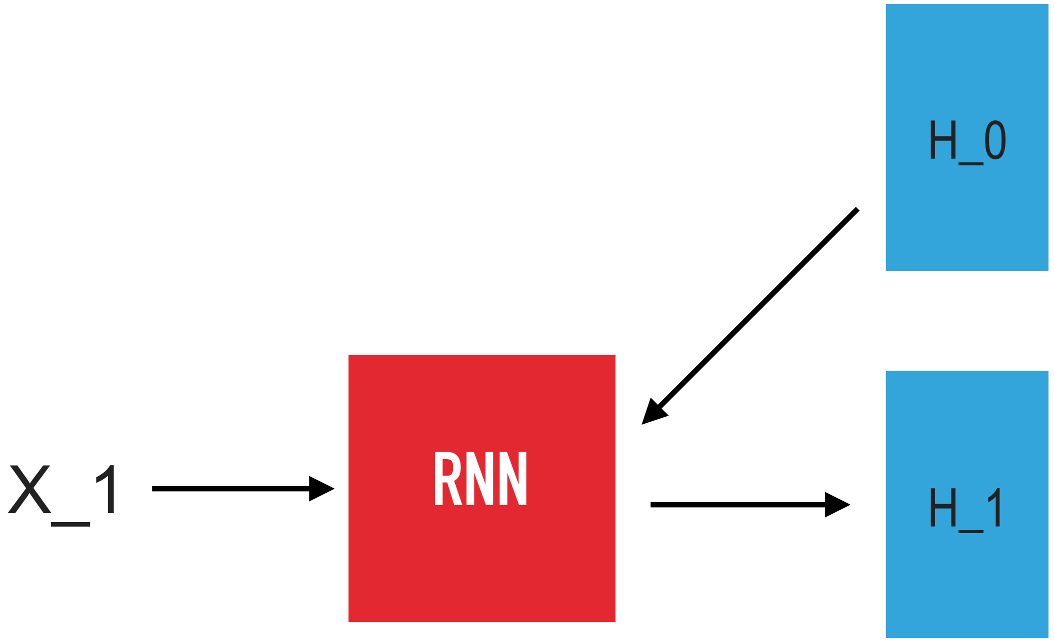
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN AND CLASSIFICATION

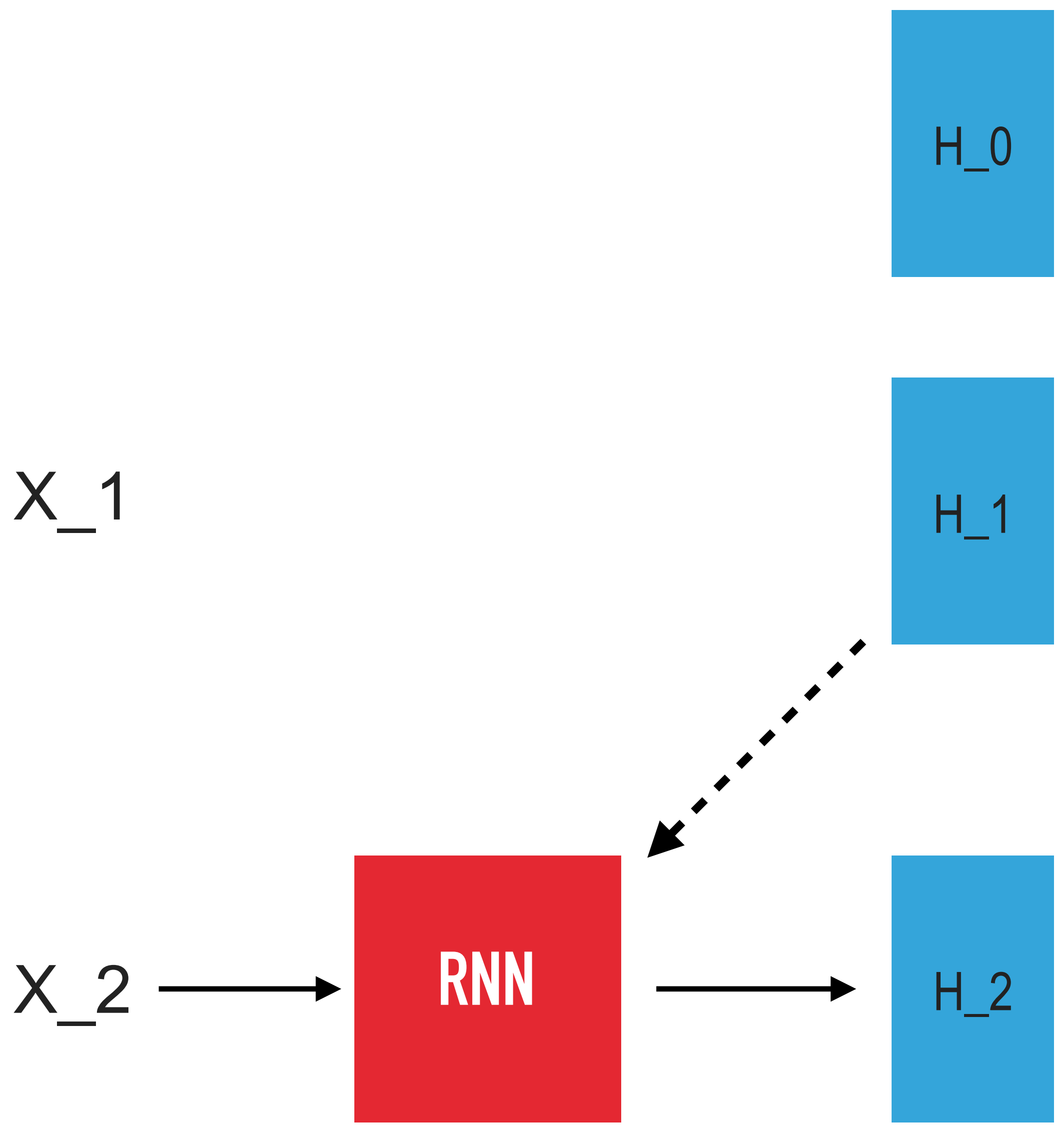


Initialize randomly

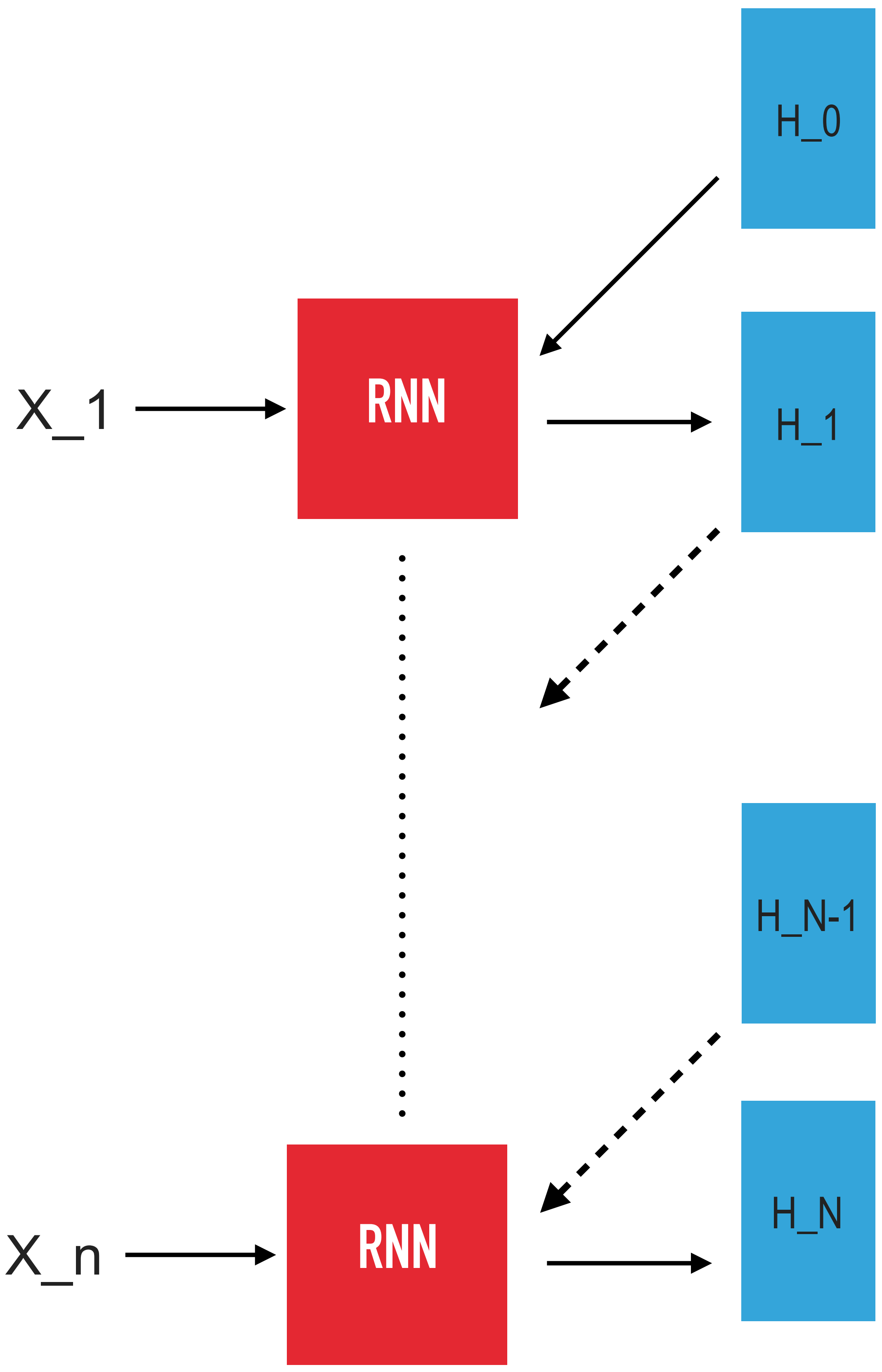
RNN AND CLASSIFICATION



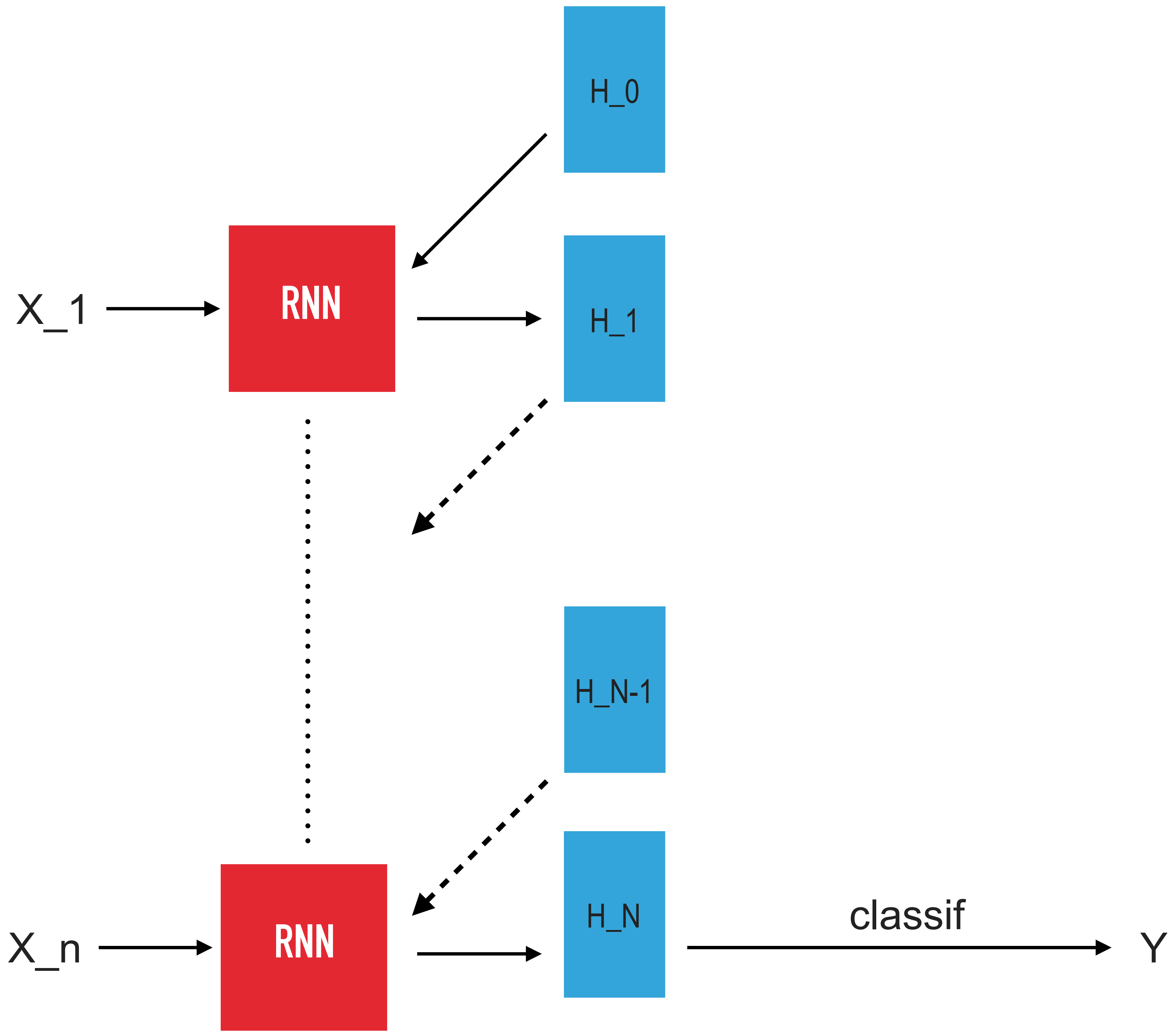
RNN AND CLASSIFICATION



RNN AND CLASSIFICATION



RNN AND CLASSIFICATION



WORD2VEC

How to represent a word as a vector?

~~*TF-IDF?*~~

=> Learning word embedding

$$\text{Italy} = (5.12, 7.21, ..., 0.78) \in \mathbb{R}^{100}$$

Beautiful word2vec relationships:

king – man + woman = queen
Tokyo – Japan + France = Paris
best – good + strong = strongest

And of course some mistakes:

England – London + Baghdad = ?

WORD2VEC

How to represent a word as a vector?

~~*TF-IDF?*~~

=> Learning word embedding

$$\text{Italy} = (5.12, 7.21, ..., 0.78) \in \mathbb{R}^{100}$$

Beautiful word2vec relationships:

king – man + woman = queen
Tokyo – Japan + France = Paris
best – good + strong = strongest

And of course some mistakes:

England – London + Baghdad = Mosul ?

WORD2VEC

How to represent a word as a vector?

~~*TF-IDF?*~~

=> Learning word embedding

$$\text{Italy} = (5.12, 7.21, ..., 0.78) \in \mathbb{R}^{100}$$

Beautiful word2vec relationships:

king – man + woman = queen
Tokyo – Japan + France = Paris
best – good + strong = strongest

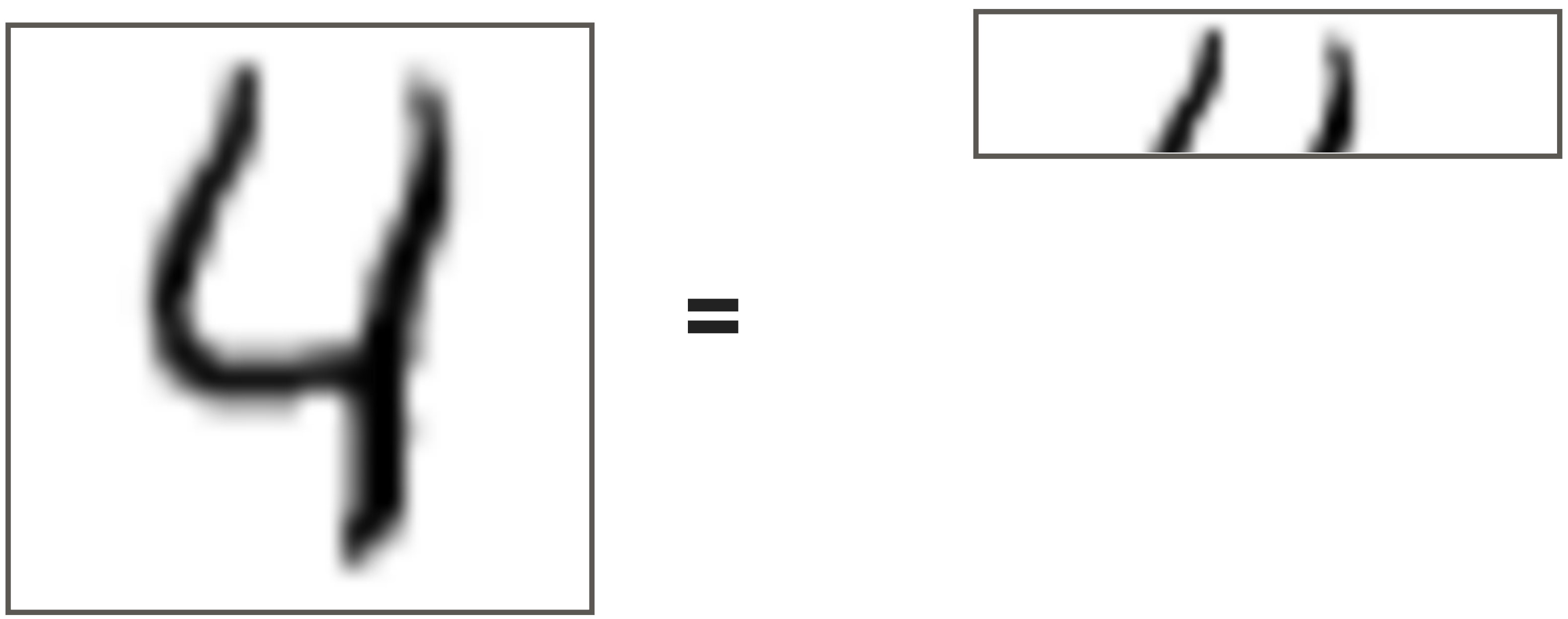
And of course some mistakes:

England – London + Baghdad = ~~Mosul~~ Iraq

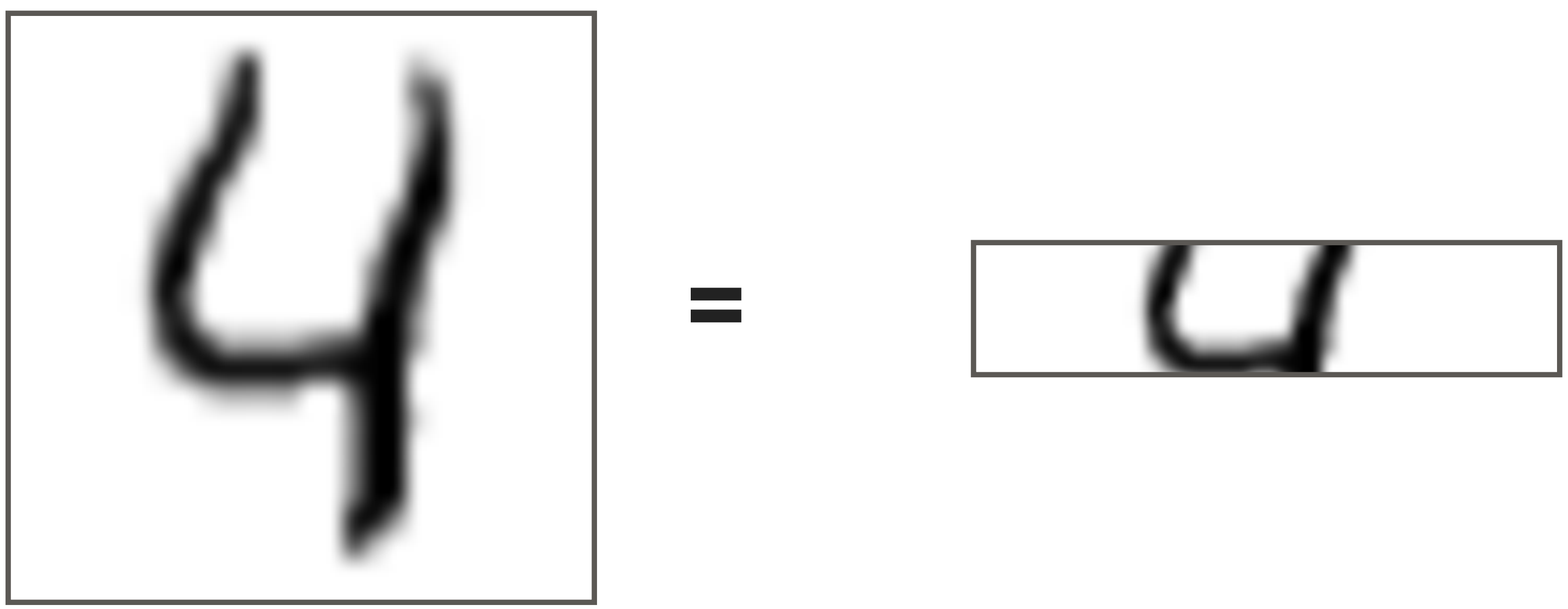
RNN ON MNIST

=

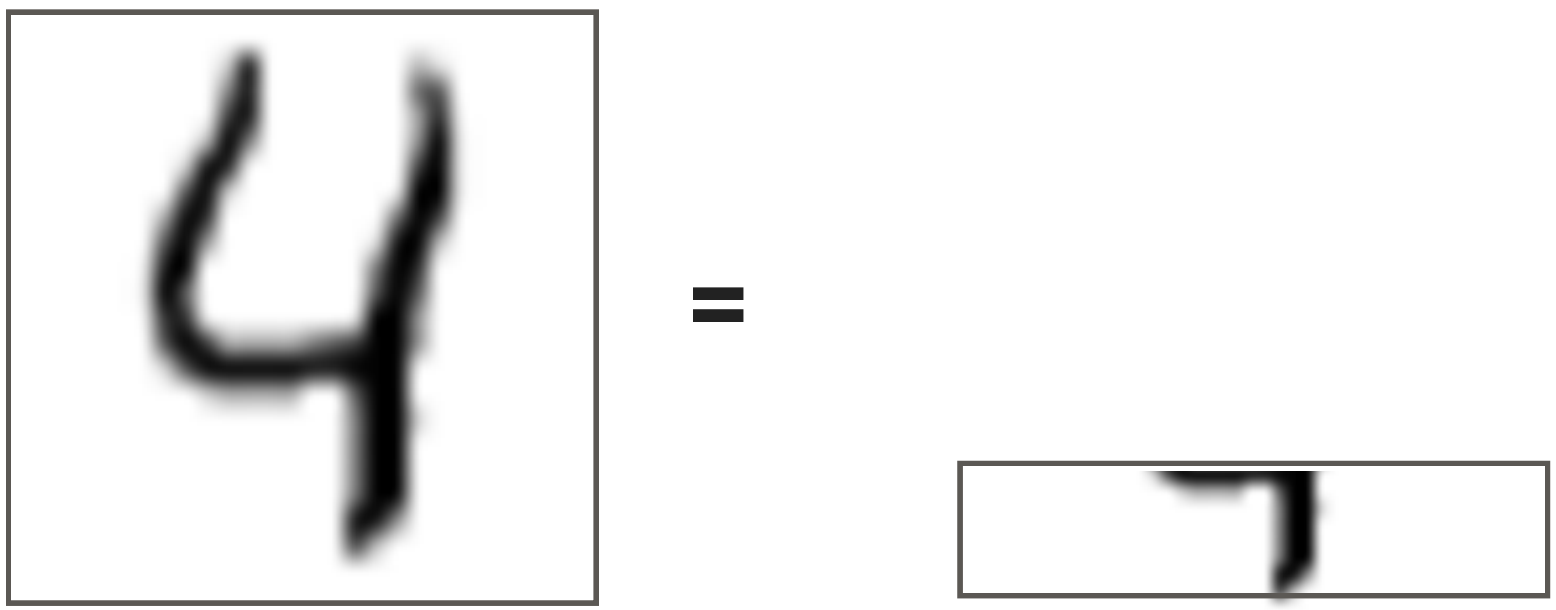
RNN ON MNIST



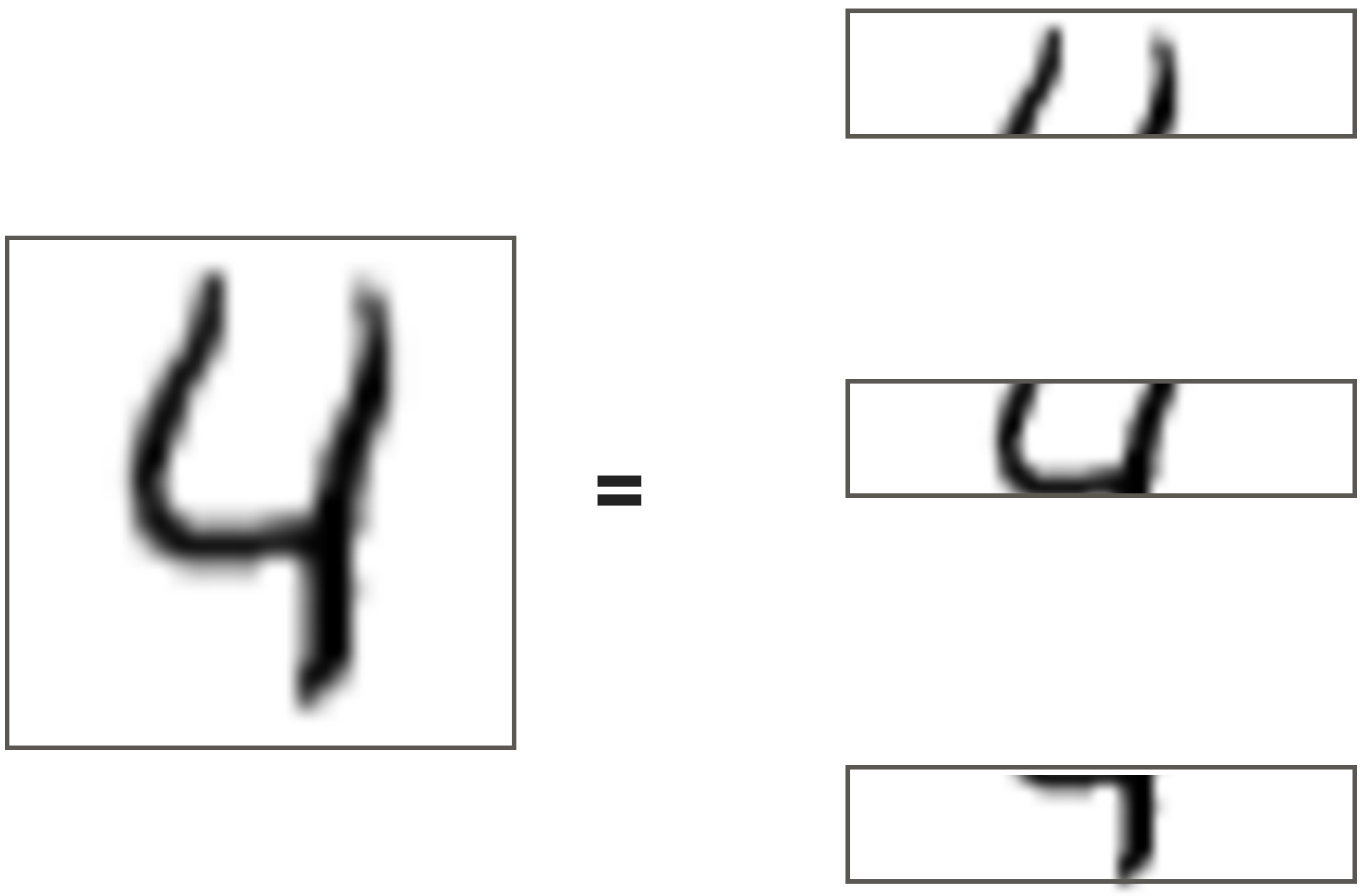
RNN ON MNIST



RNN ON MNIST



RNN ON MNIST

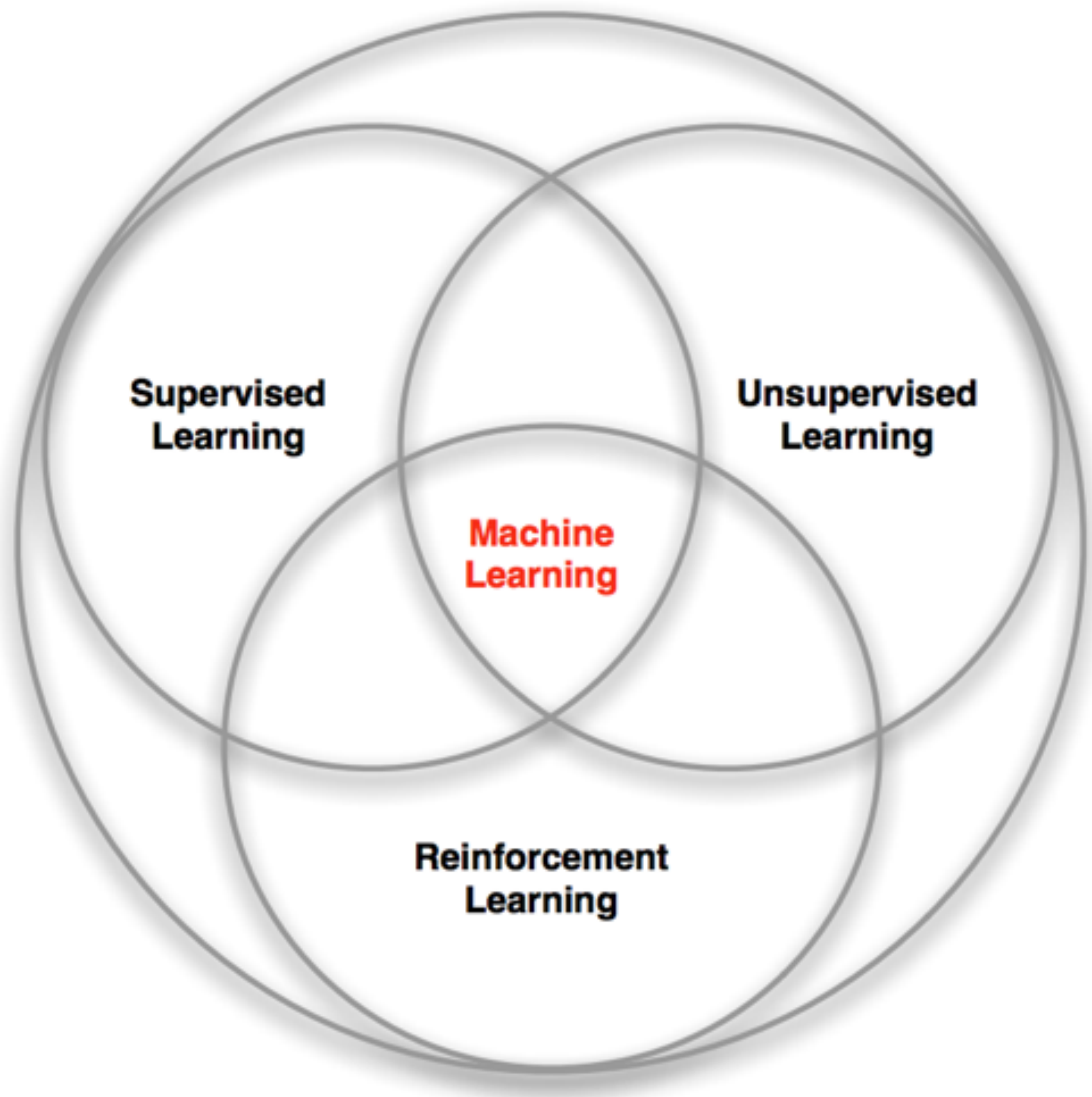


[28,28] = sequence of 28 vectors of size 28

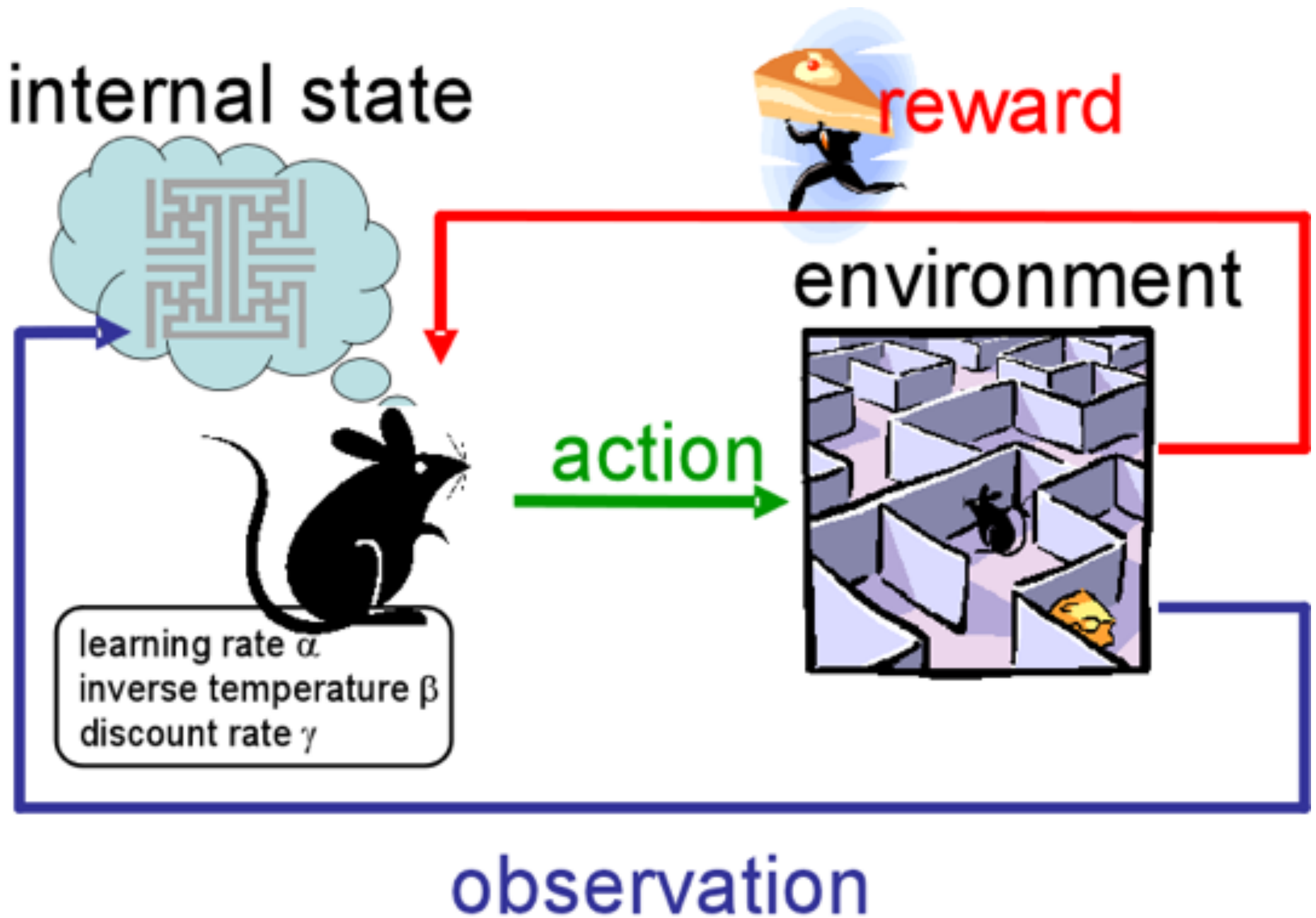
RNN ON MNIST

- Complete the exo « **rnn_exo** »

MACHINE LEARNING

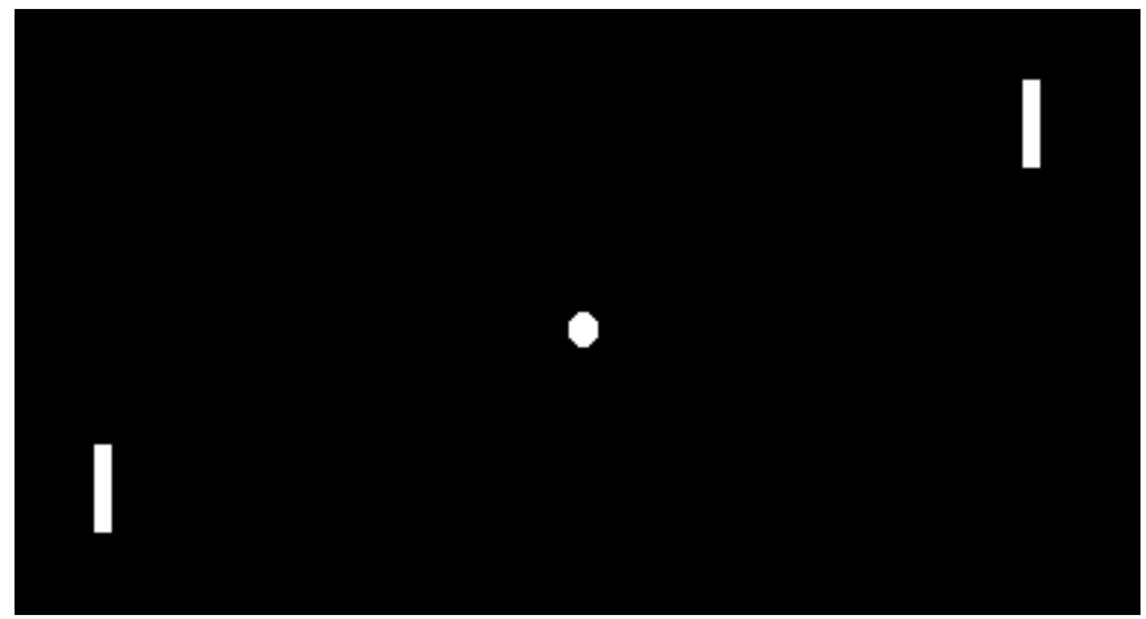


REINFORCEMENT LEARNING



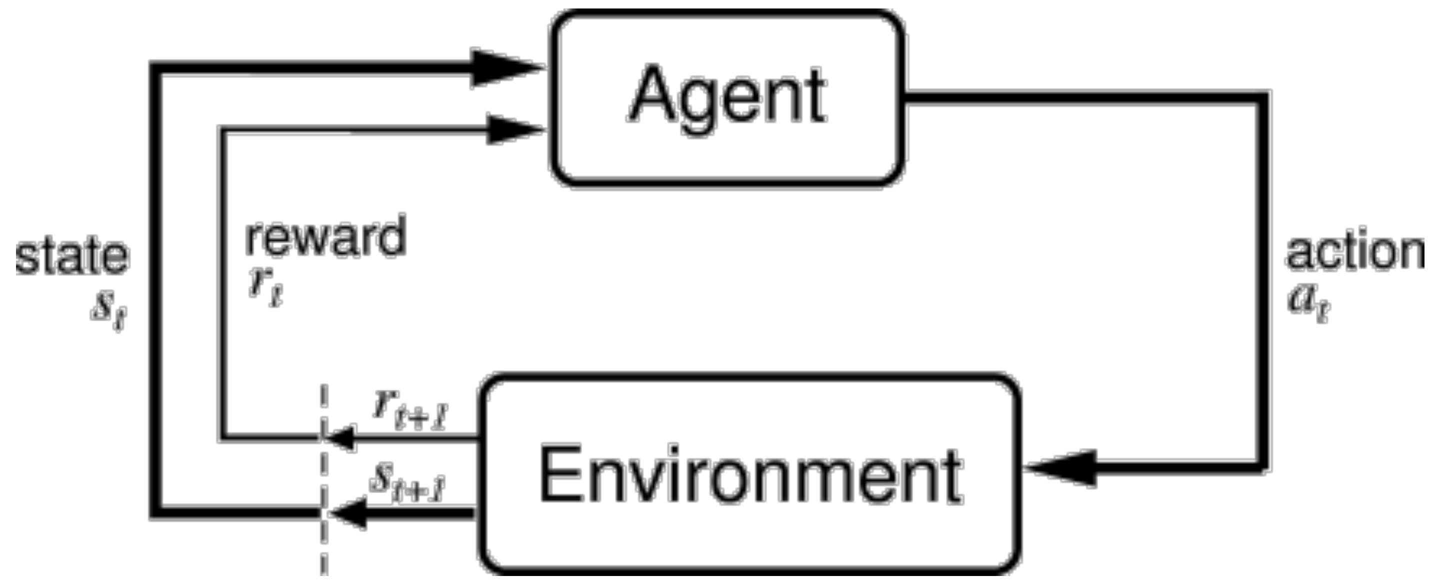
RL = Reinforcement Learning

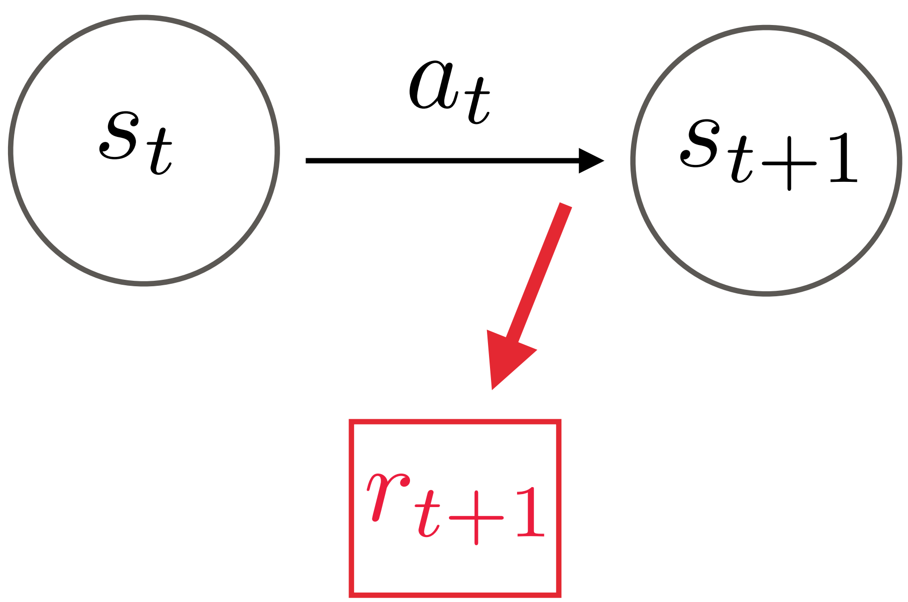
RL: FEW EXAMPLES



https://www.youtube.com/watch?v=V1eYniJ0Rnk

RL IN A FINITE STATE SPACE





$$s_t, s_{t+1} \in S$$

$$a_t \in A(s_t)$$

$$t = 0, 1, 2, \ldots$$

FROZEN LAKE EXAMPLE

$$\begin{bmatrix} S & F & F & F \\ F & H & F & H \\ F & F & F & H \\ H & F & F & G \end{bmatrix}$$

F = frozen surface, safe

G = goal, where the frisbee is located

S = starting point, safe

H = hole, fall to your doom

- **Possible actions:**
  - Up
  - Down
  - Left
  - Right

Ice is slippery: you won't always move in the direction you intend

RL: DEFINITIONS

The agent learns to assign values to state-action pairs

Discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + \gamma^{T-1} r_T$$

where $\gamma \in [0;1]$ *is the discount rate*

Action - value function for policy π :

$$Q^\pi(s,a) = E_\pi\{R_t | s_t = s, a_t = a\}$$

« How good an action is for the future given a certain state? »

FORMULATION OF THE Q-FUNCTION

Actions

$$Q^{\pi} = \begin{bmatrix} Q(0,\mathrm{Up}) & Q(0,\mathrm{Down}) & Q(0,\mathrm{Left}) & Q(0,\mathrm{Right}) \\ \dots & & & \\ Q(s_t,\mathrm{Up}) & Q(s_t,\mathrm{Down}) & Q(s_t,\mathrm{Left}) & Q(s_t,\mathrm{Right}) \\ \dots & & & \\ Q(16,\mathrm{Up}) & Q(16,\mathrm{Down}) & Q(16,\mathrm{Left}) & Q(16,\mathrm{Right}) \end{bmatrix} \quad \textbf{States}$$

Optimal value function unrolled recursively

$$Q^*(s_t, a_t) = E_{s_{t+1}}\{r_{t+1} + \gamma \times \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})\}$$

optimal action-value function

immediate reward

discount factor

the best possible value of the next state

Express state-value function by neural network with parameters θ :

$$Q(s, a, \theta) \approx Q^{\pi}(s, a)$$

But what is our target vector to compute the loss function???

DEEP Q-LEARNING

Q value function

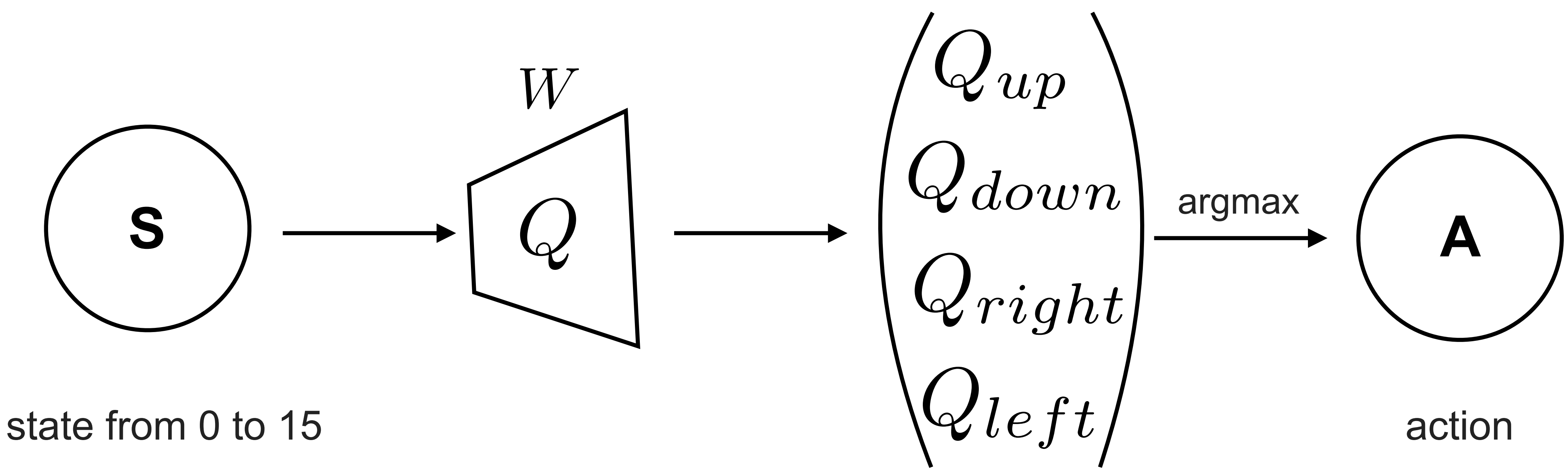
$$Q_{\theta_t} = \begin{bmatrix} Q(0, \mathrm{Up}, \theta_t) & Q(0, \mathrm{Down}, \theta_t) & Q(0, \mathrm{Left}, \theta_t) & Q(0, \mathrm{Right}, \theta_t) \\ Q(1, \mathrm{Up}, \theta_t) & Q(1, \mathrm{Down}, \theta_t) & Q(1, \mathrm{Left}, \theta_t) & Q(1, \mathrm{Right}, \theta_t) \\ \dots & \dots & \dots & \dots \\ Q(15, \mathrm{Up}, \theta_t) & Q(15, \mathrm{Down}, \theta_t) & Q(15, \mathrm{Left}, \theta_t) & Q(15, \mathrm{Right}, \theta_t) \end{bmatrix}$$

Loss function

$$J(\theta_t) = \sum \Big(\underbrace{Q(s_t, a_t, \theta_t)}_{\text{value at this state}} - \underbrace{r_{t+1} + \gamma \times \underbrace{\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \theta_t)}_{\text{best possible value at next state}}}_{\text{Q-target}} \Big)^2$$

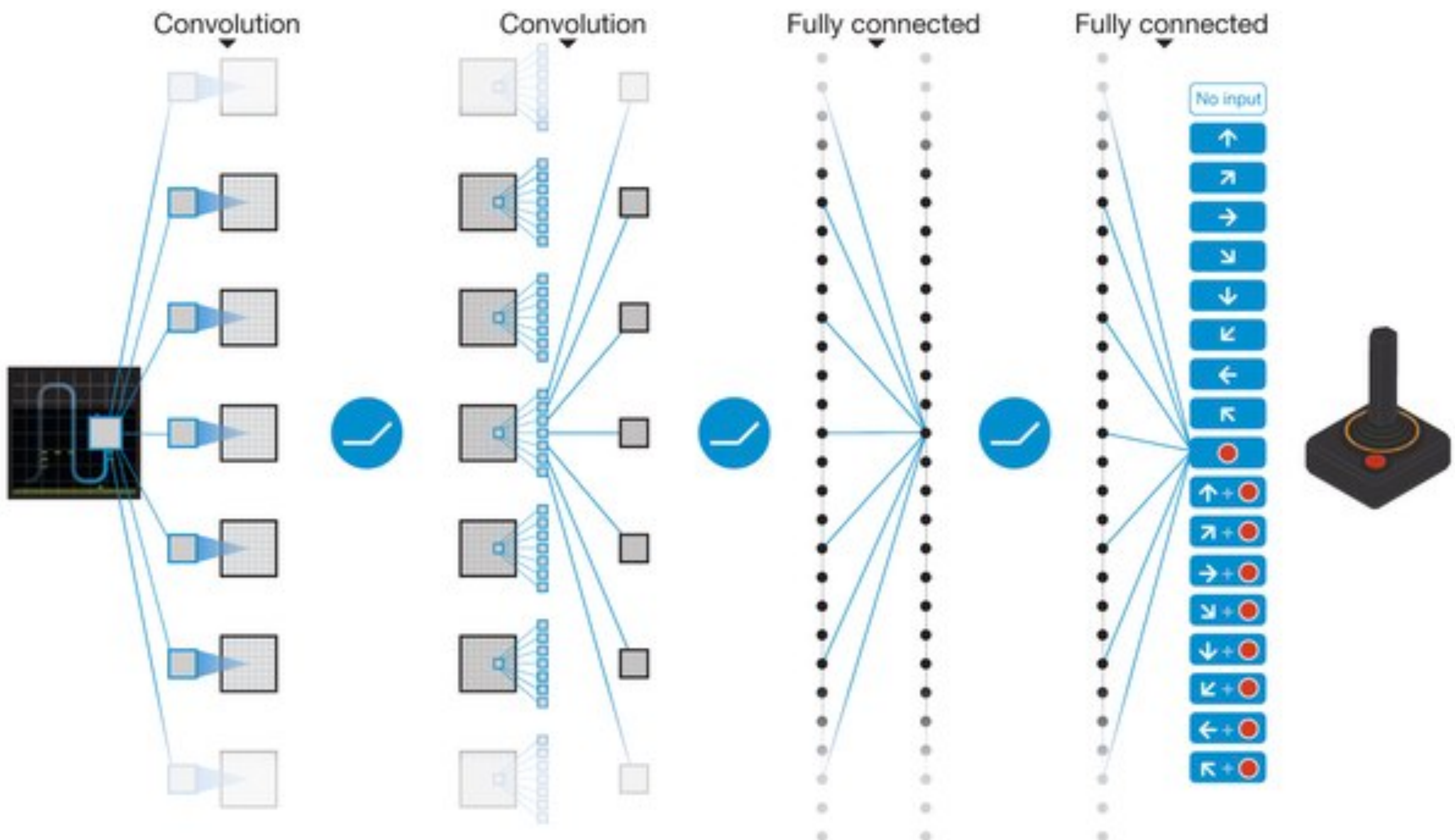
Train it using SGD!

NN Q-FUNCTION FOR FROZEN LAKE



DEEP Q-FUNCTION FOR REAL GAMES

ConvNets … here we go again!



EXERCISE

- Complete the exo « **q_learning_frozen_lake_exo** »

Or go back to the fish classification if you want ;)

WHAT ABOUT YOUR FIRST EXPERIENCE WITH TENSORFLOW?