Basic Introduction to DL for PP

Eilam Gross

Acknowledgments: Jonathan Shlomi, Nilotpal Kakati

Sources and Recommended Reading

- PDG review of Machine Learning
  https://pdg.lbl.gov/2022/reviews/rpp2022-rev-machine-learning.pdf

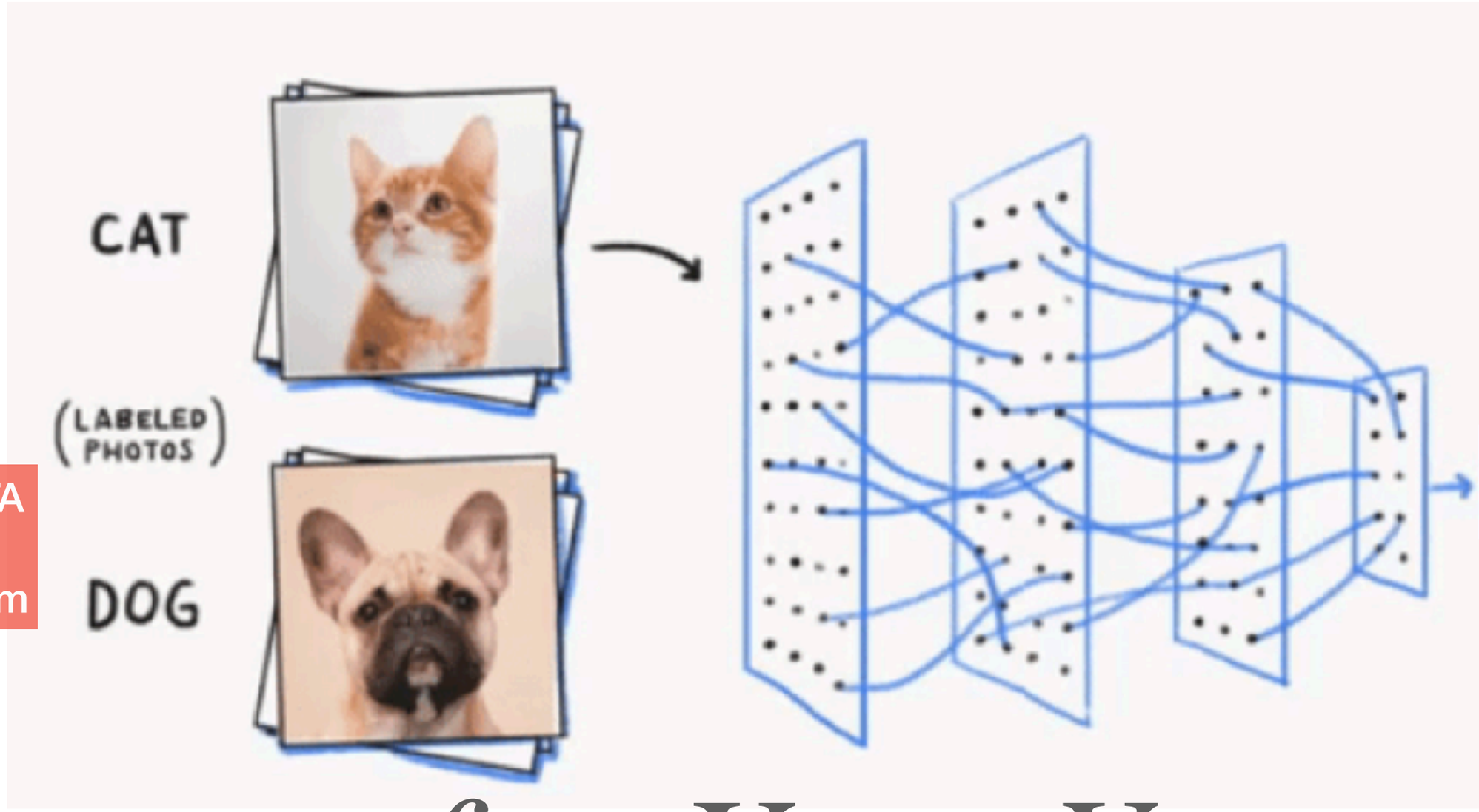
- Graph Neural Net in Particle Physics
  Jonathan Shlomi, Peter Battaglia and Jean-Roch Vlimant
  https://iopscience.iop.org/article/10.1088/2632-2153/abbf9a

Why DL is a Boom Now?

- ML started in the 1950s
- Deep Convolutional Nets since the 1990s
- Autonomous driving is now a multi-billion dollars business… TESLA (and NOT only) is already there… Why only now?
- Big DATA - Cheap Storage, easy access, and lots of Big Data
- GPUs are changing the face of computer hardware (Parallelizable tasks)
- Sophisticated software/firmware tools for Deep Learning Models implementation

Example: Classification Dog vs Cat

Input: x



output: y

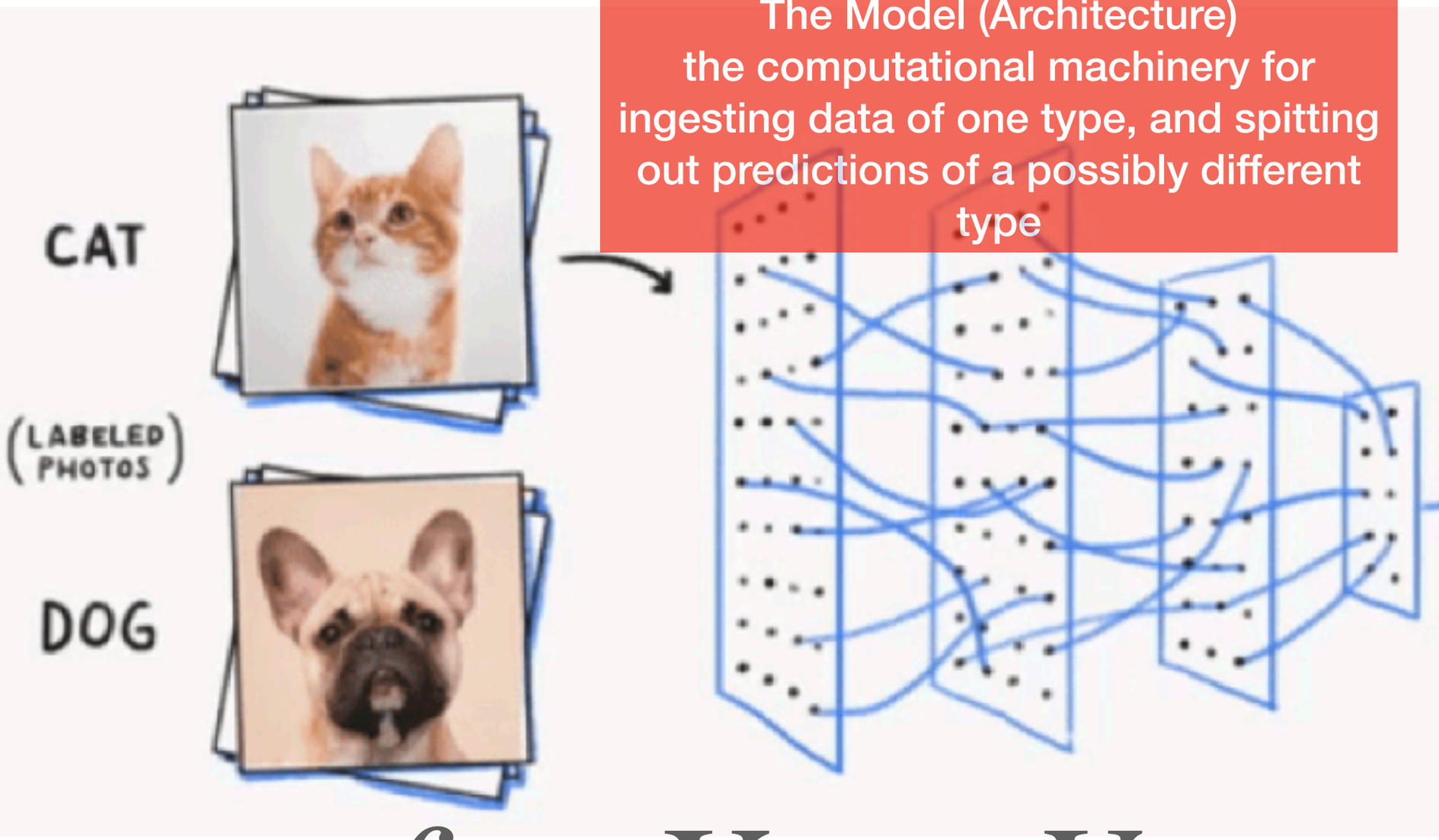
The DATA we can learn from

$$f_\phi : X \rightarrow Y$$

Example: Classification Dog vs Cat

Input: x

The Model (Architecture)
the computational machinery for ingesting data of one type, and spitting out predictions of a possibly different type



output: y

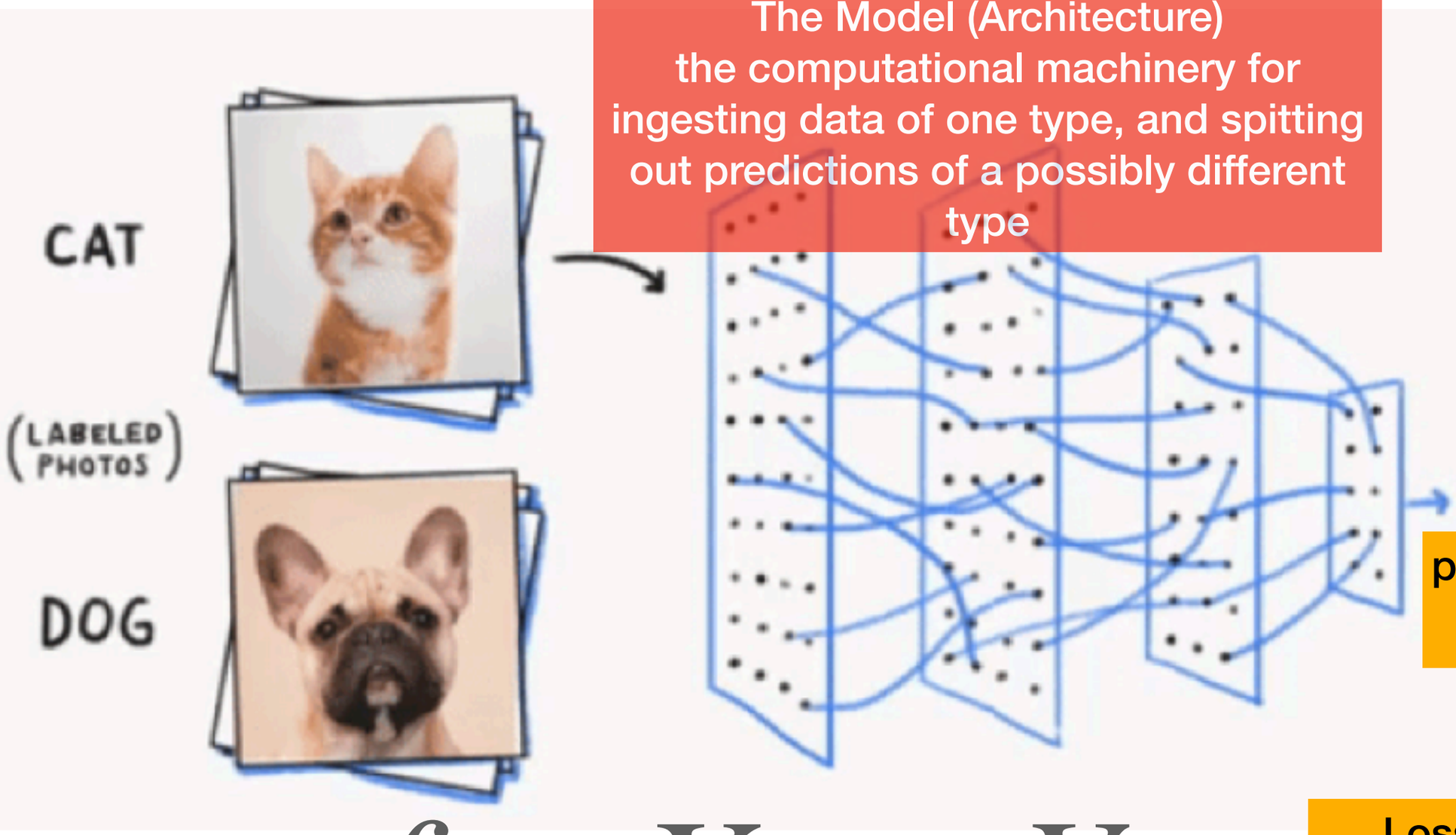
The DATA we can learn from

$$f_\phi : X \rightarrow Y$$

Example: Classification Dog vs Cat

Input: x

The Model (Architecture)
the computational machinery for ingesting data of one type, and spitting out predictions of a possibly different type



target: $y \in Y$

The DATA we can learn from

prediction: $f_\phi = \hat{y}$

$$f_\phi : X \rightarrow Y$$

Loss: $\mathscr{L}(\hat{y}, y)$

The Loss Function:
minimize error rate, i.e., the fraction of instances on which our predictions disagree with the ground truth

Example: $\mathscr{L}(\hat{y}, y) = (y - \hat{y})^2$

DATA representations

- Much of our knowledge is subjective and intuitive

- Computers need to capture this knowledge in order to make intelligent decisions

- The capability to acquire knowledge by extracting patterns from raw DATA is what Machine Learning is all about

label: DOG

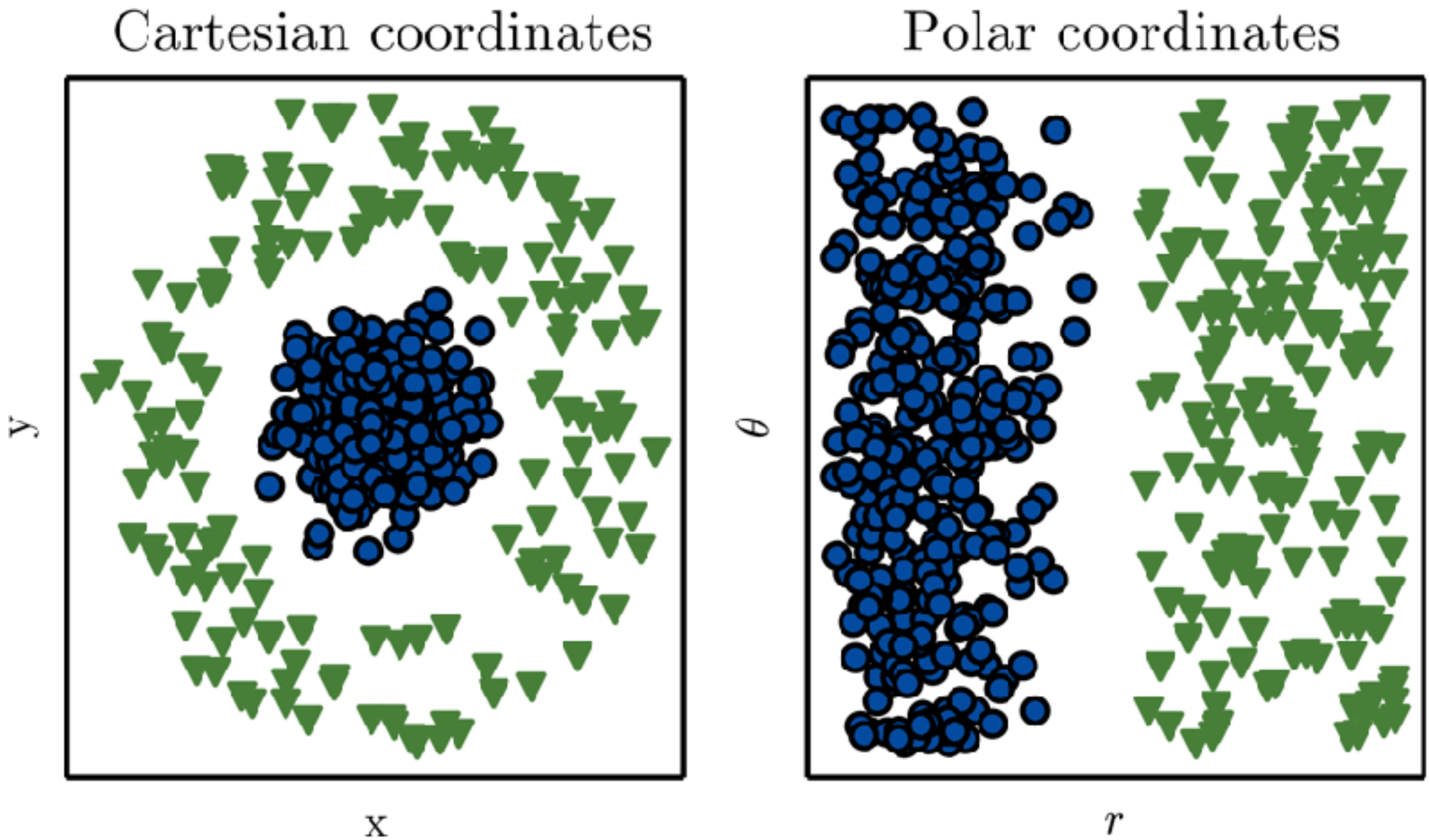
label: CAT

- The performance of a ML algorithm depends heavily on the **representation** of the DATA they are given.

- Each piece of information given in the representation is called a **feature**.

DATA representations

- Example:



- Choosing the right set of features can make a huge difference in solving a task

Machine Learning Example

- Map inputs (such as images) to targets (such as labels: Cat, Dog, Woman)

- BUT let your mind flies by

Layer transforms DATA into a useful representation

Its all about finding appropriate representations

This is done via training rather than programming

Transformation is controlled by internal parameters (weights and biases)

Gentle Introduction (PDG)

- Data: Feature vector x=$(E, P, \phi, \eta, \ldots.)$
- $X \in R^d$
- Say, using MC we know if x originated from an electron or a photon

- Task: Find a function $f : x \rightarrow y$ that can accurately predict the label for the DATA
- Supervised Learning—> create pairs $\{x_i, y_i\}_{i=1,2,\ldots,n}$ where y=0 for electrons, 1 for photons
- Use Neural Net to provide a family of functions $f_\phi : R^d \rightarrow R$
  ϕ are the internal parameters of the NN (weights and biases)

$$f_\phi : R^d \rightarrow R$$

- Training Procedure: Find values ϕ that provide the best predictions for y

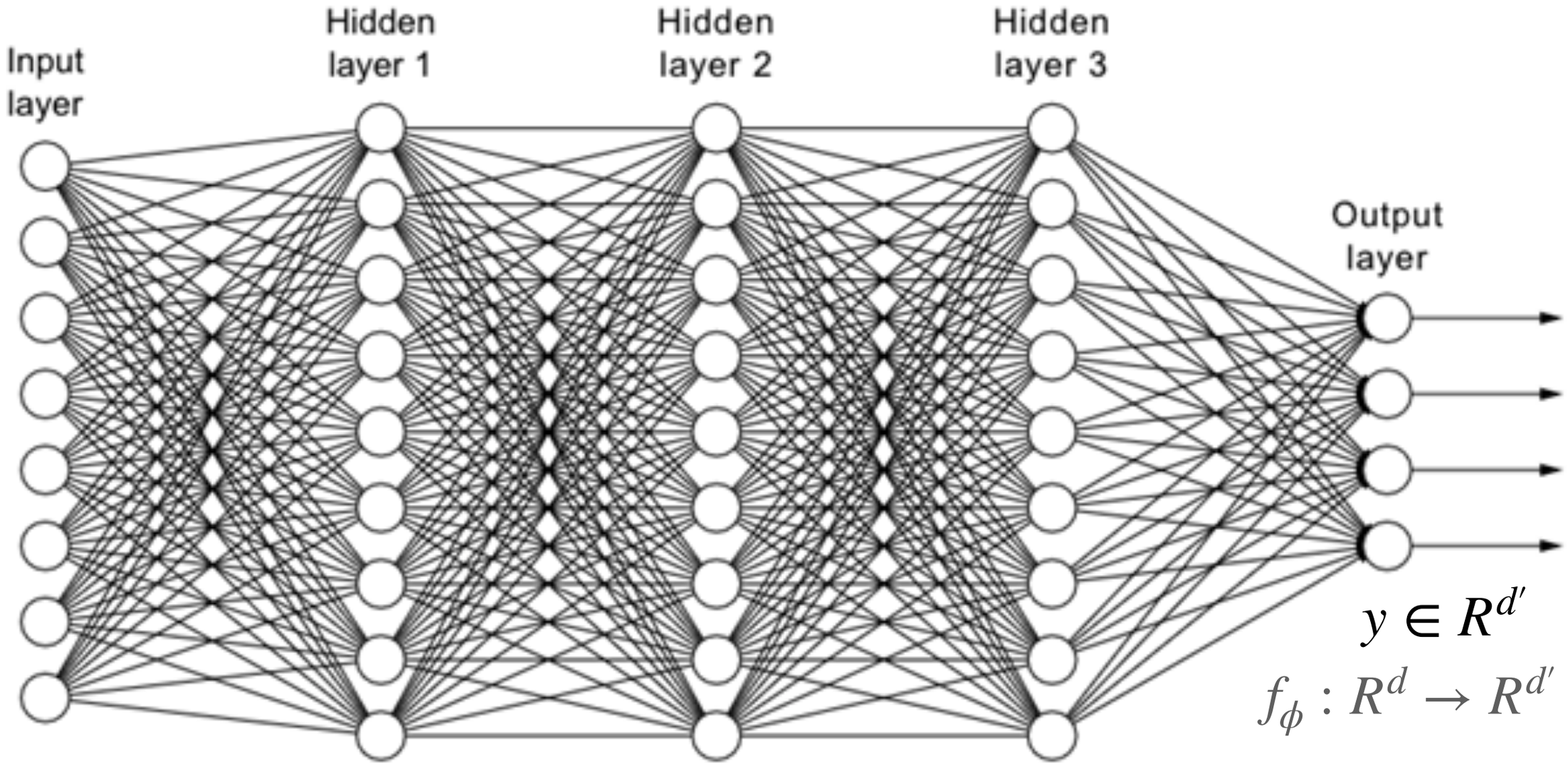
- Training is done by **minimizing** a loss Function $\mathscr{L}\left(f_\phi(x), y\right)$, for example:

$$\mathscr{L}_{\text{MSE}}\left(f_\phi(x), y\right) = \left(f_\phi(x) - y\right)^2$$

- Universal Approximation Theorem (simplified): **Neural Networks can approximate any function**

Neural Nets

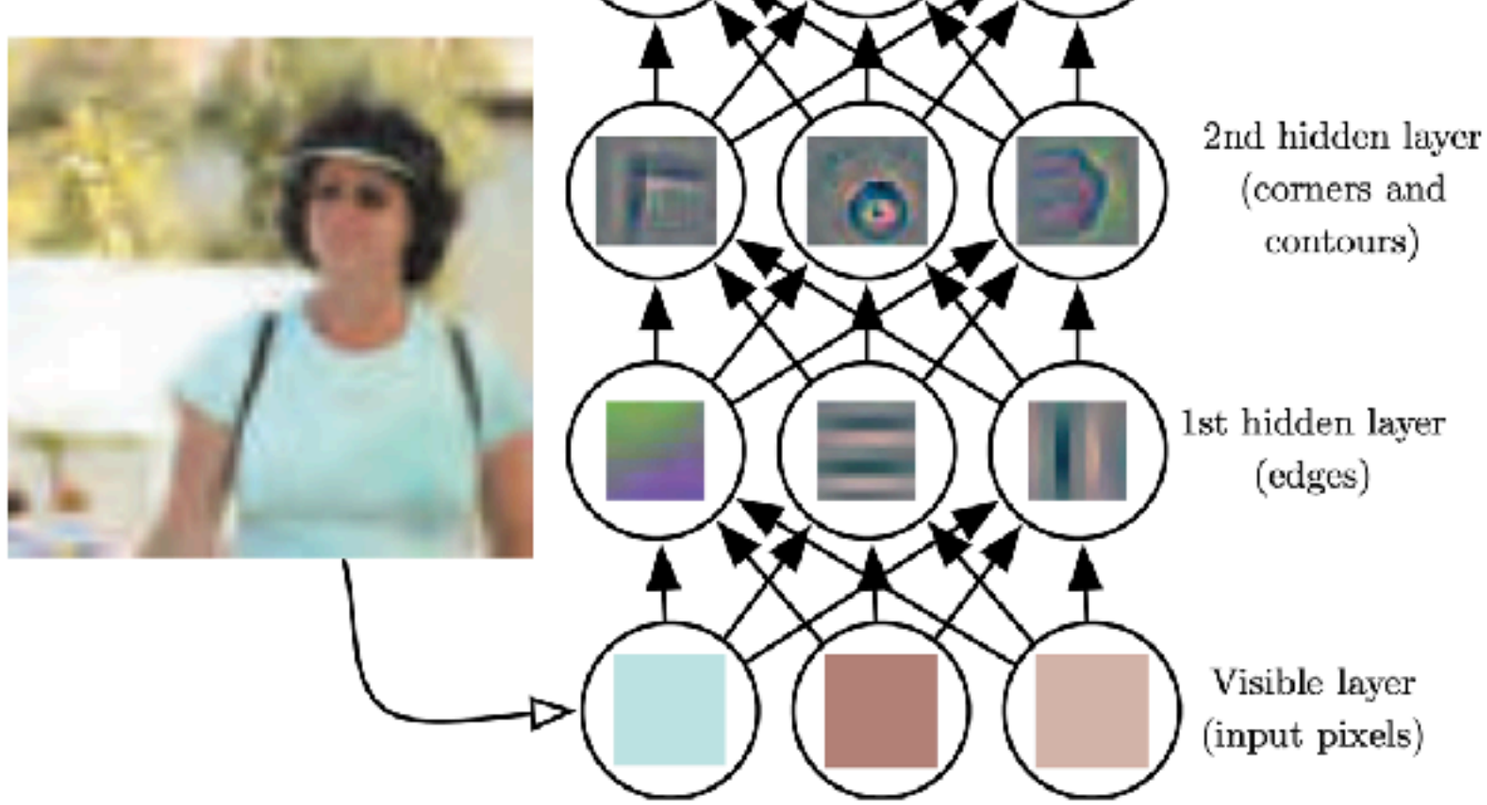
$f_\phi : R^d \rightarrow R^{d'}$



$$\mathscr{L}\left(f_\phi(x), y\right)$$

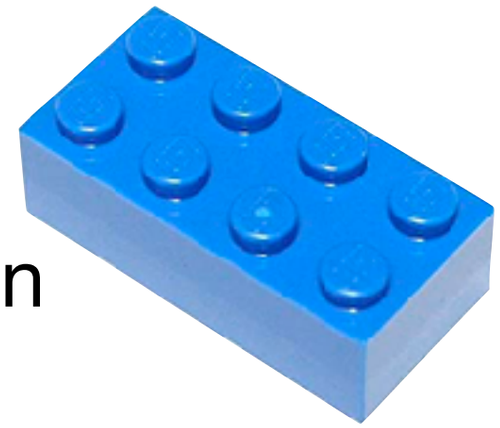
Deep “representation” Learning

- Break the complex presentation into a series of simpler nested ones

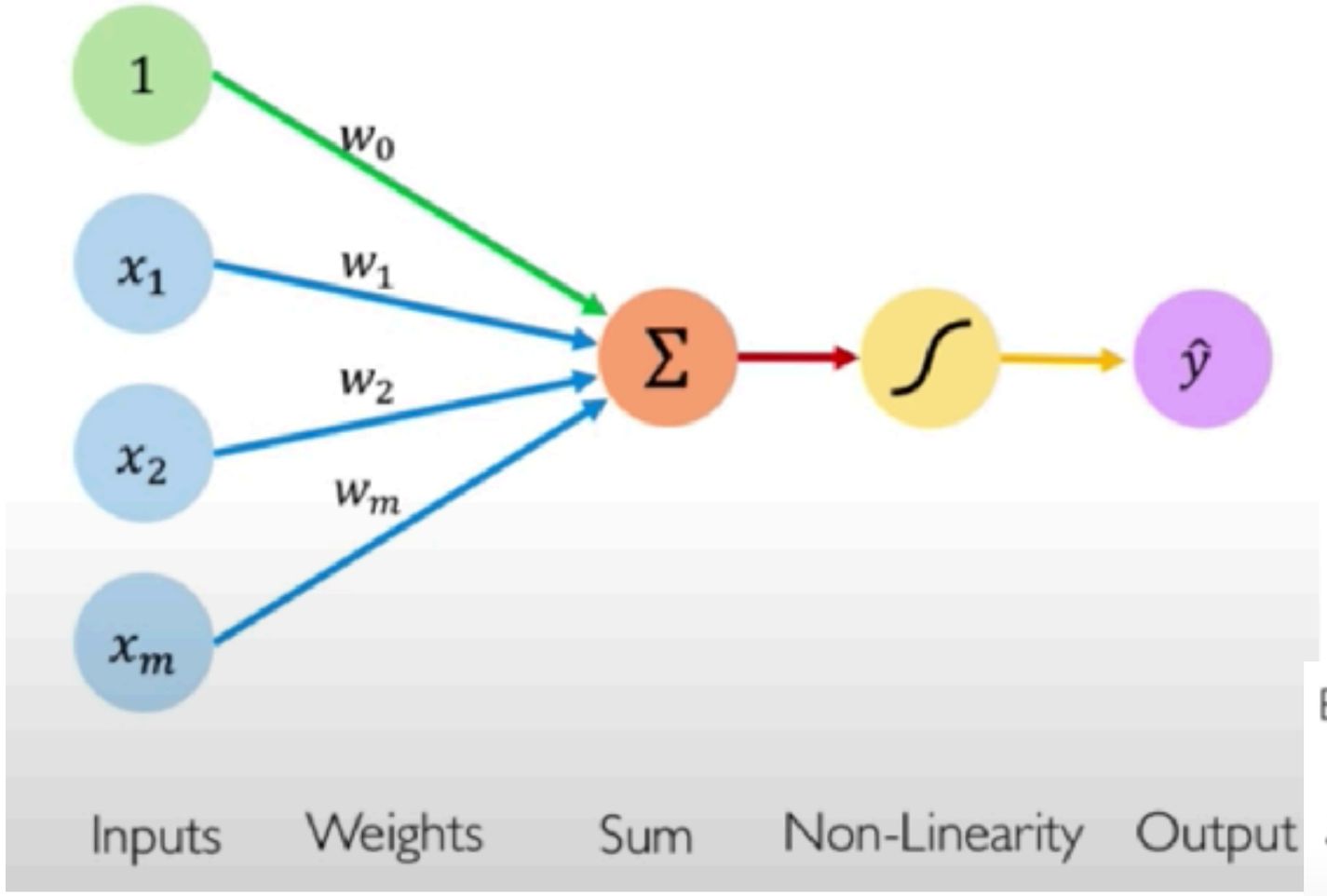


Based on Zeiler and Fergus, 2014

Perceptron



- The basic unit of Deep Learning is the Perceptron



output $\hat{y}$

Bias Term w_0

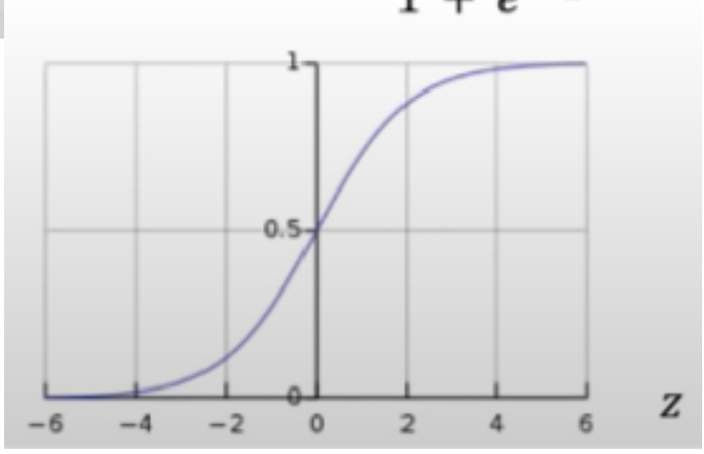
Lin Comb of Weights

$$\hat{y} = g\left(w_0 + \Sigma_{i=1}^{m} w_i x_i\right)$$

Nonlinear Activation Function g

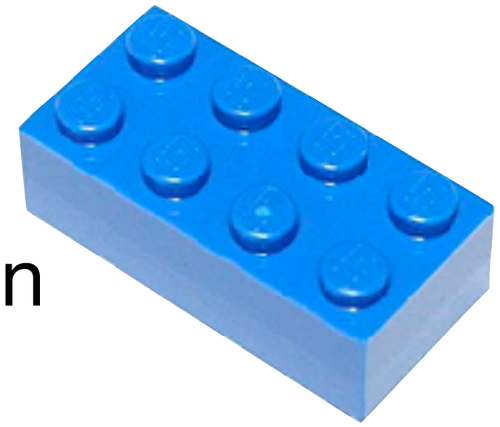
Example: sigmoid

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

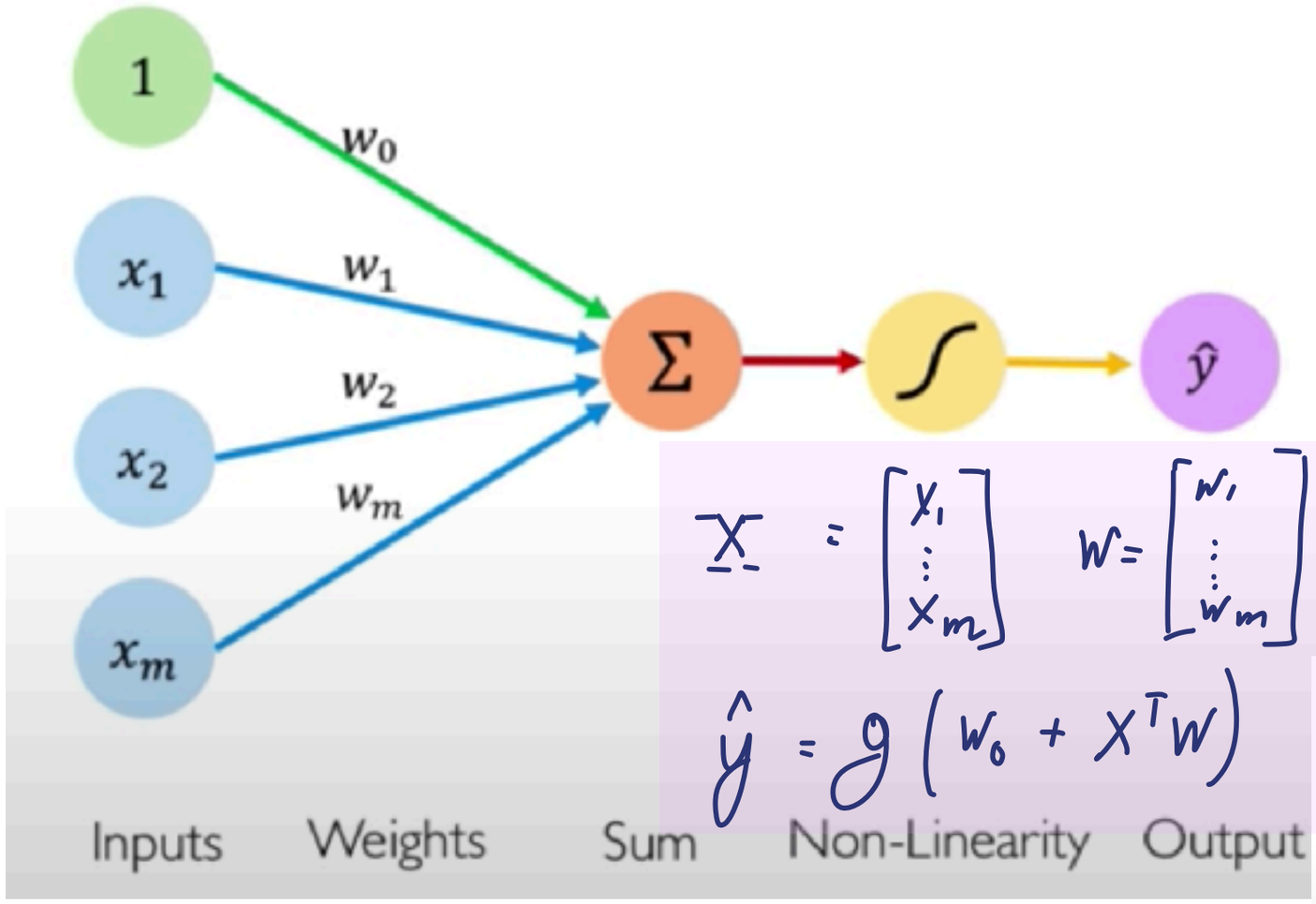


http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Perceptron



- The basic unit of Deep Learning is the Perceptron



output $\hat{y}$

Bias Term w_0

Lin Comb of Weights

$$\hat{y} = g\left(w_0 + \Sigma_{i=1}^{m} w_i x_i\right)$$

$$\hat{y} = g(w_0 + x^T w)$$

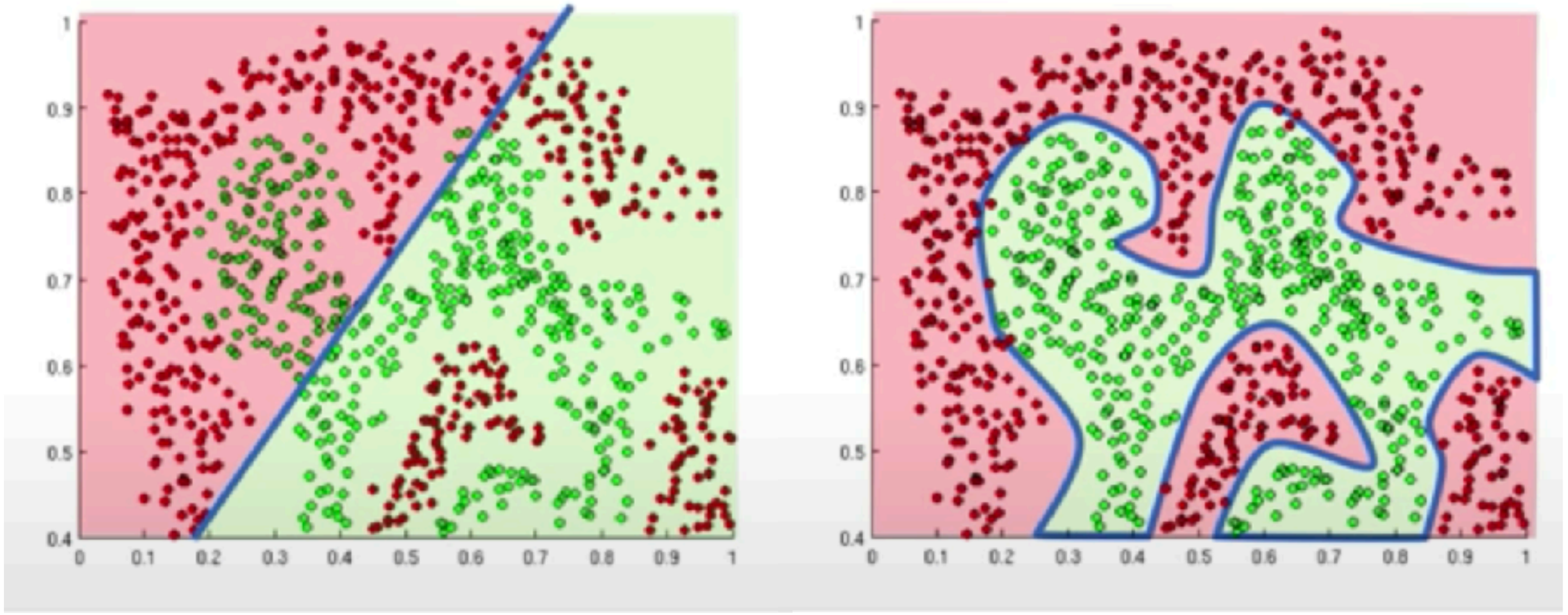
Nonlinear Activation Function g

Example: sigmoid

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Why non-linearity?

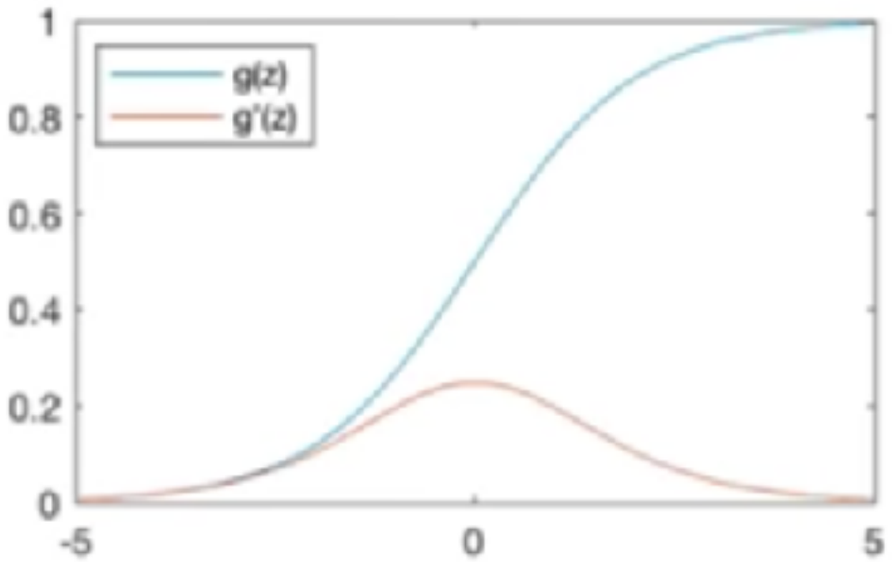


http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

- Linear —> Linear decision boundary

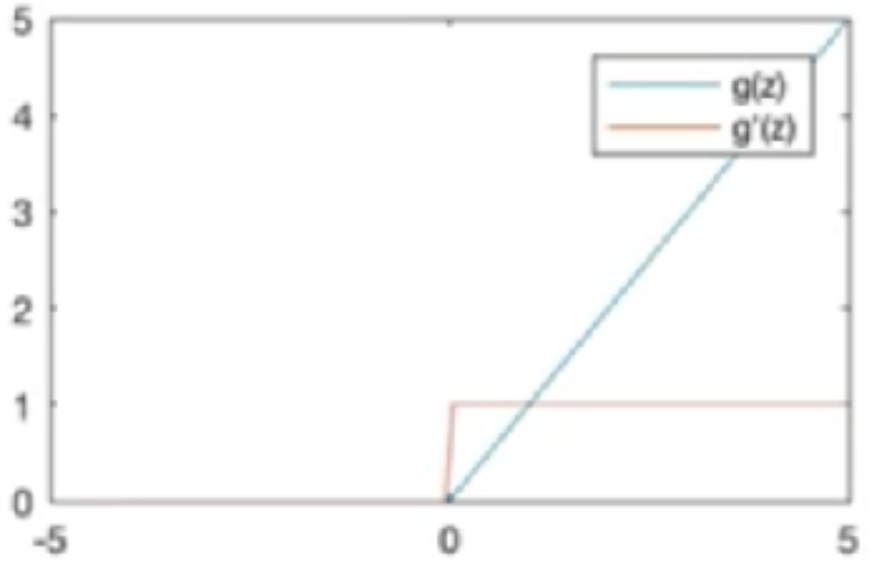
- Non-linear—>Non-linear (complex) decision boundary

Non-Linear Activation Functions



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

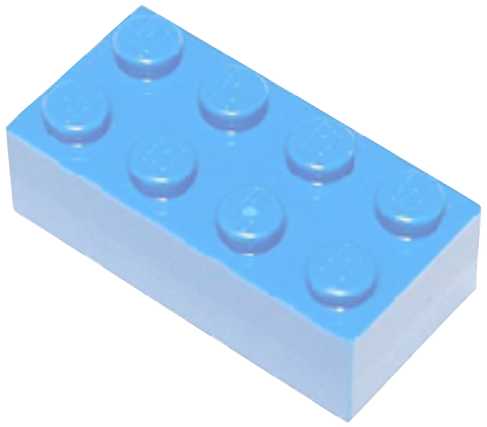
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

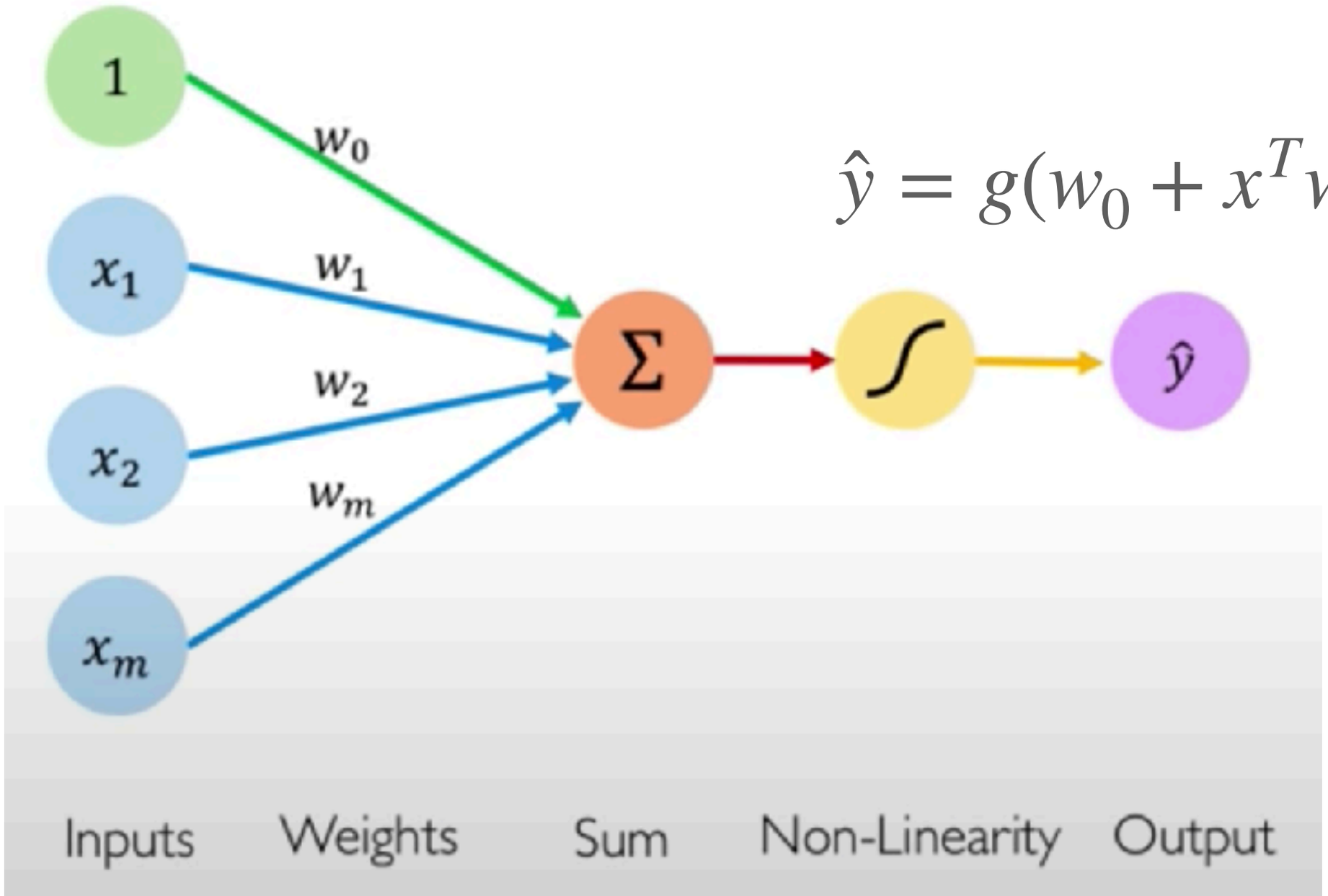
Sigmoid Saturates and kill Gradients

ReLu- Rectified Linear units

Many times you use ReLu **for all Hidden Layers and** Sigmoid **in the final layer as to output a probability (a score between 0 and 1)**

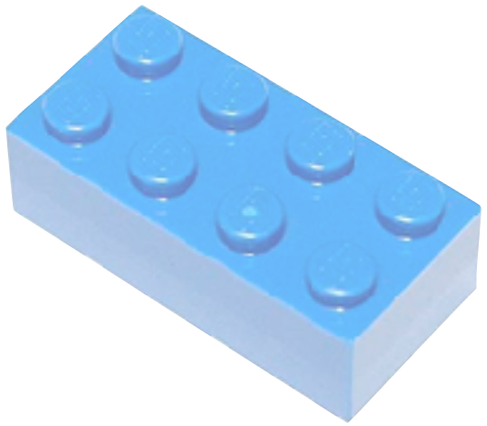
Perceptron



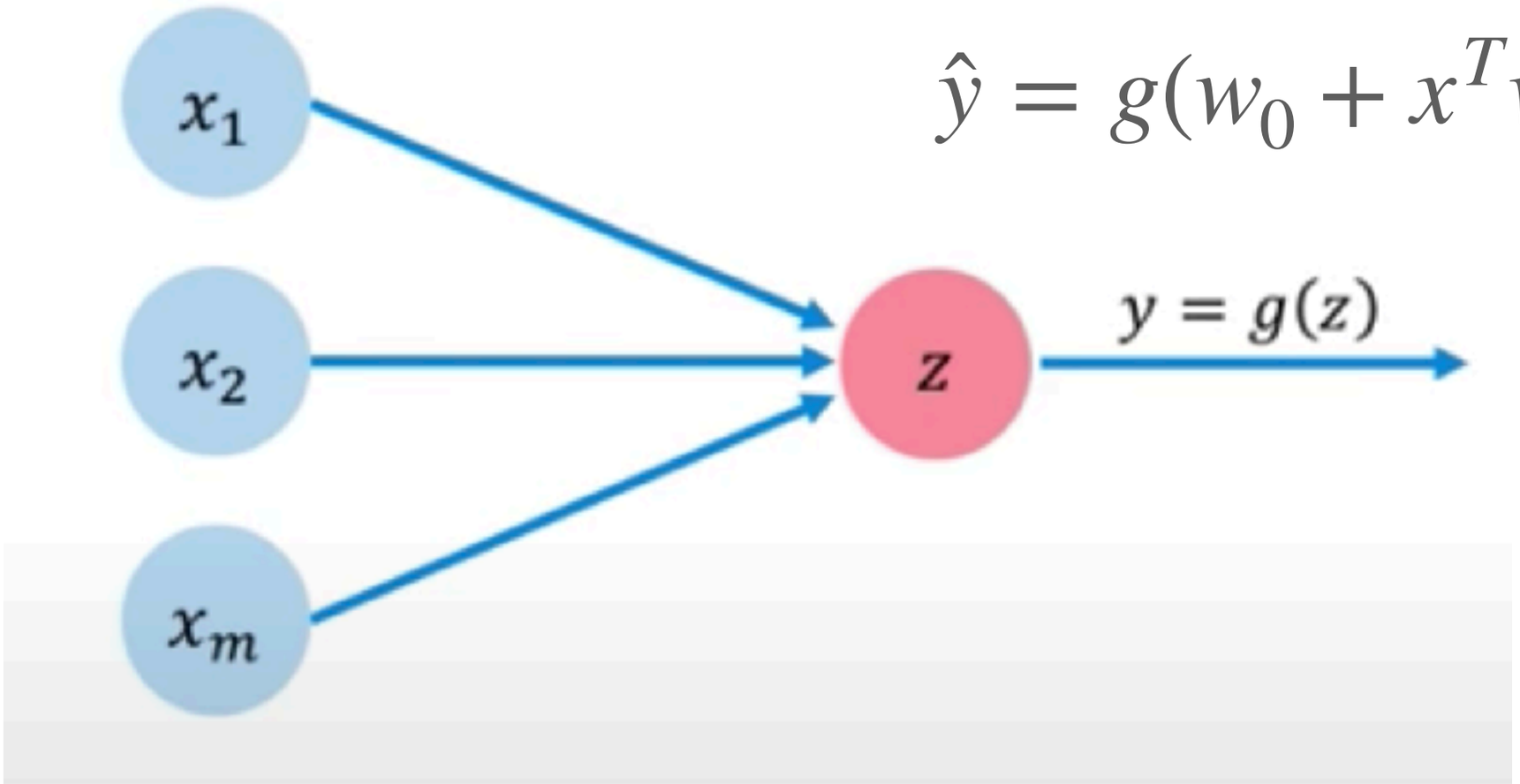


$$\hat{y} = g(w_0 + x^T w)$$

Simplified Perceptron

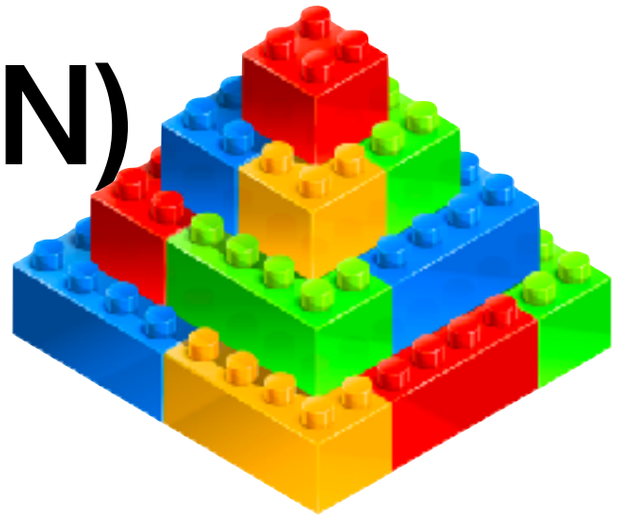


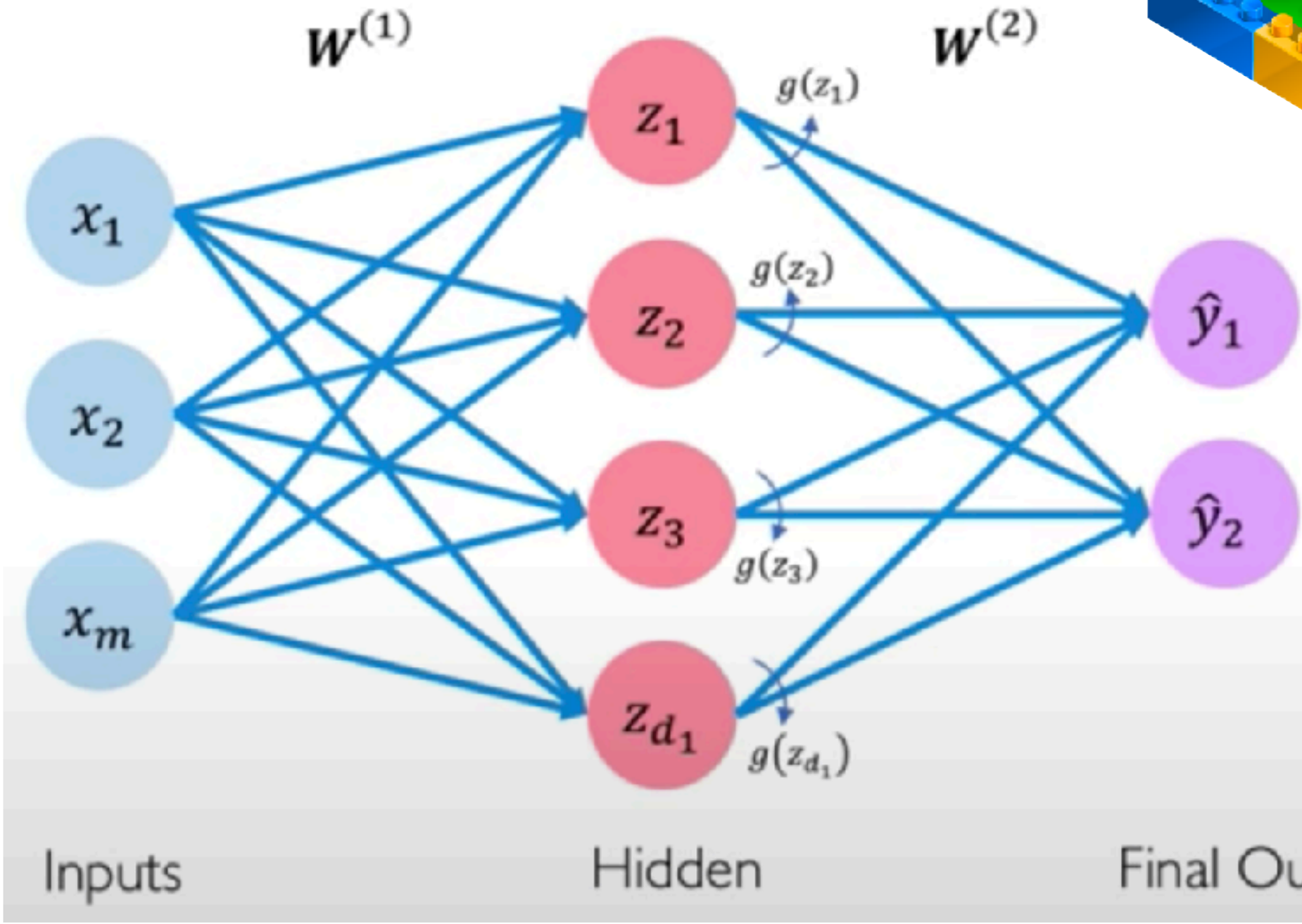
- Simplified notation/drawing



$$\hat{y} = g(w_0 + x^T w)$$

Single Layer Perceptron (NN)





$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)} \right)$$

Summary: Layers & Weights

- Layers extract representation of the DATA fed into them, which are supposed to be more meaningful for decoding the DATA

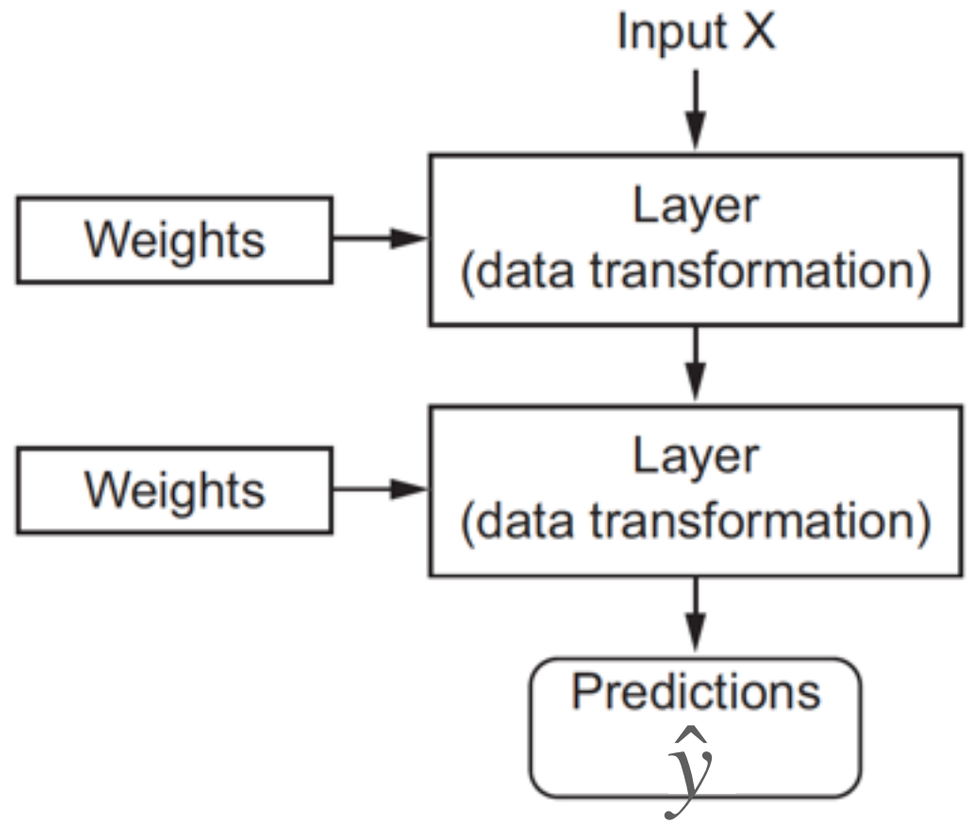
- Some DATA goes in, and
  comes out in a more useful form

- You can 🤔 of layers as “filters”

- Weights control what the layer is doing to it’s input DATA

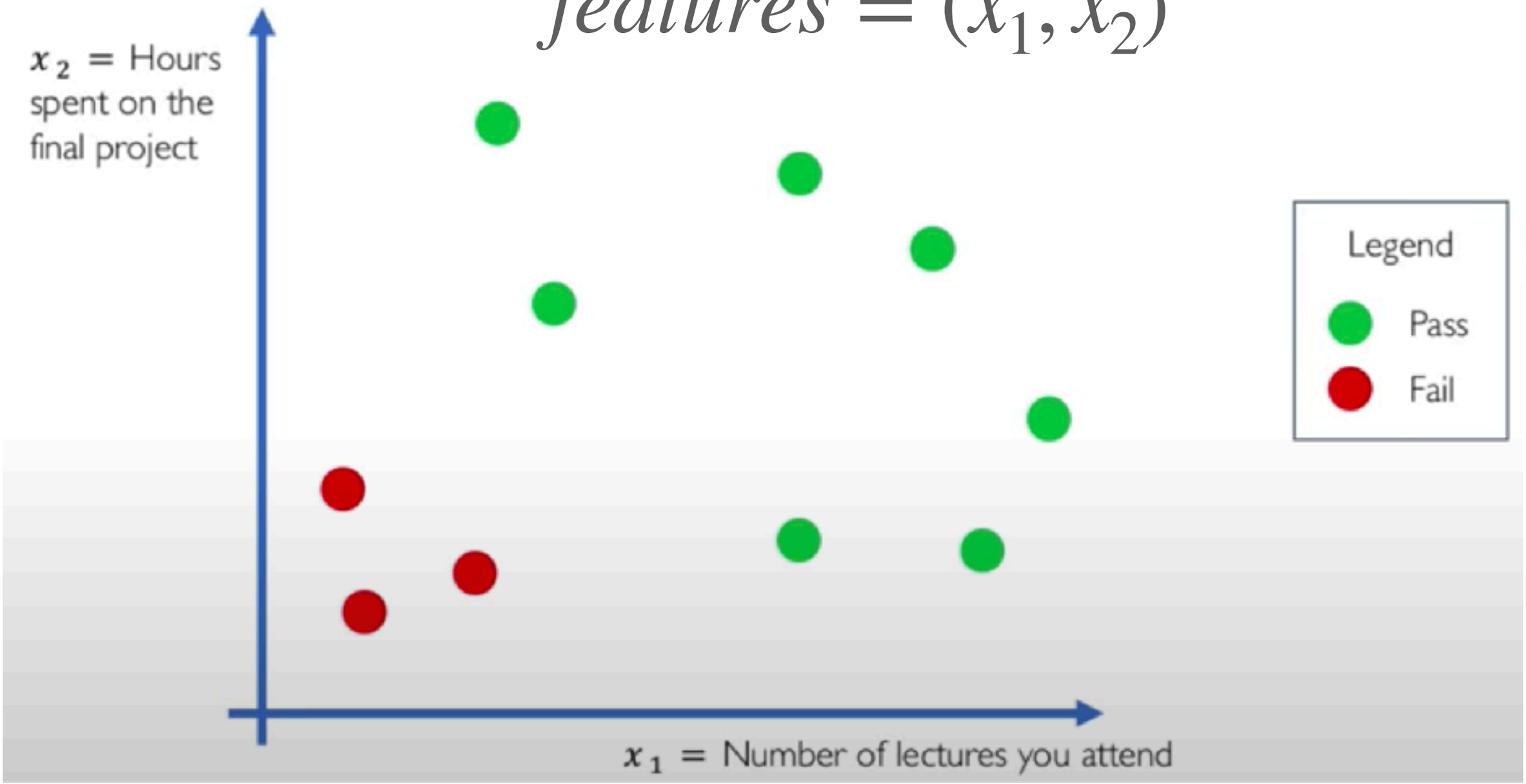
- The depth of the model is the number of layers contribute to the lea process

- GOAL: Find right values for the weights such that the network maps the input to its right target



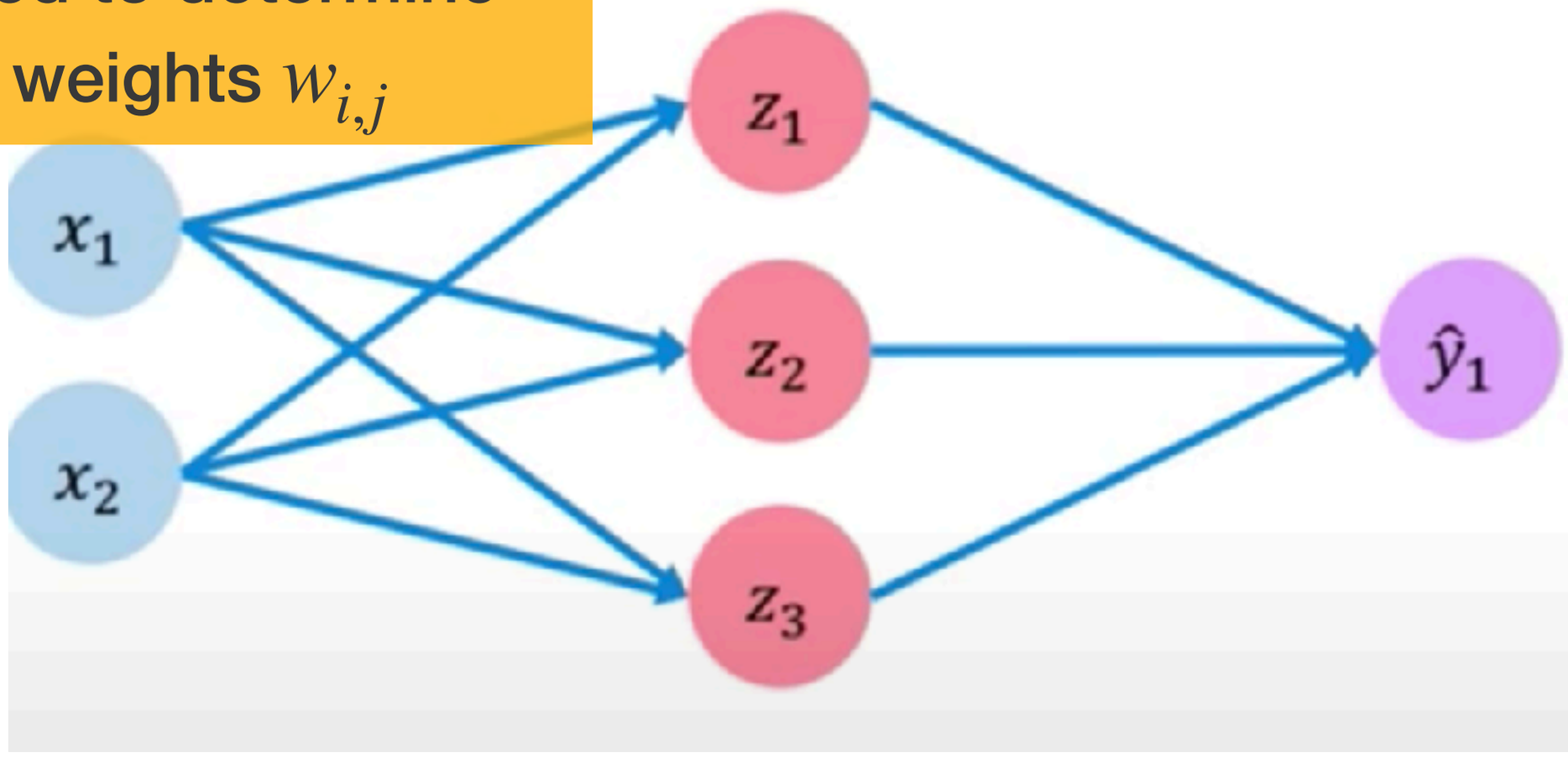
MIT Example Problem: Will I Pass This Class?
The DATA

$$features = (x_1, x_2)$$

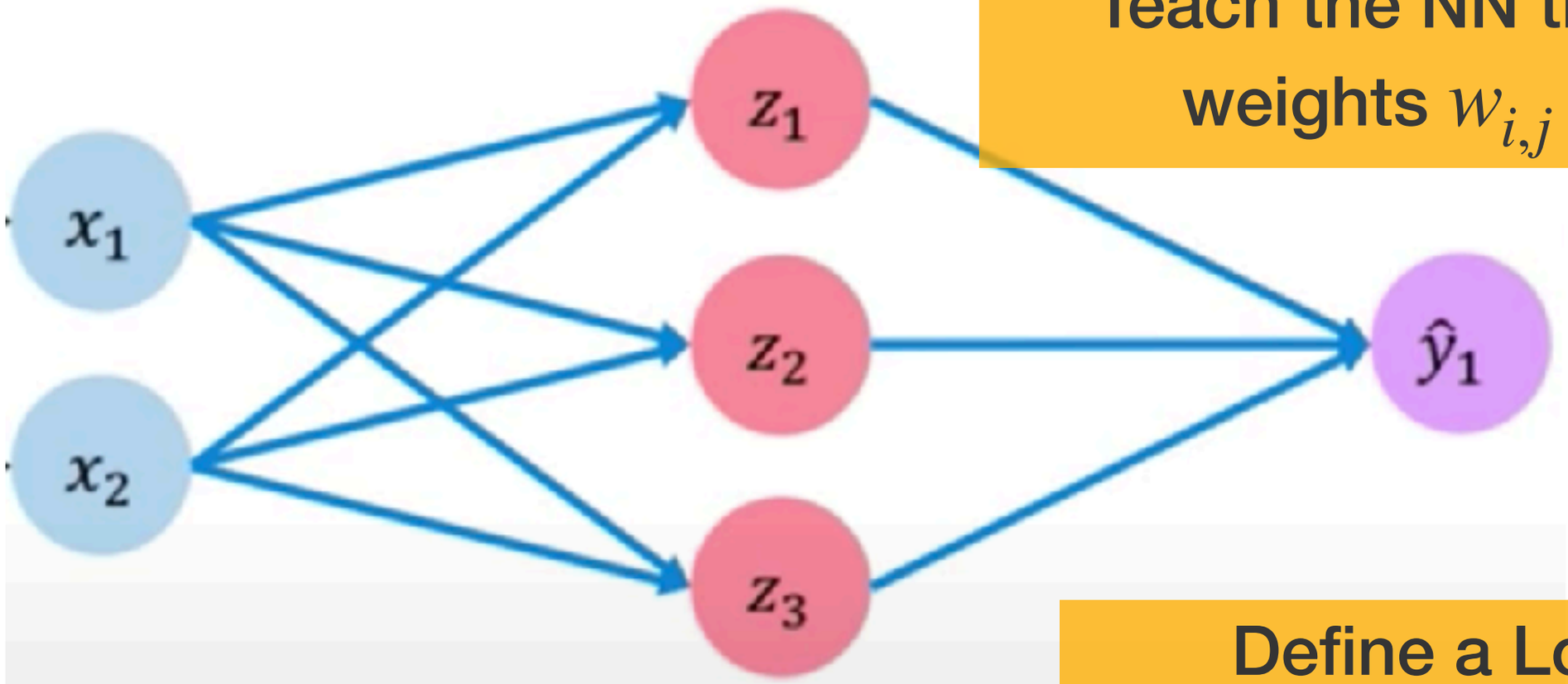


MIT Example Problem: Will I Pass This Class?
The Single Layer NN

Need to determine weights $w_{i,j}$



MIT Example Problem: Will I Pass This Class? The LOSS



Teach the NN the weights $w_{i,j}$

Define a Loss

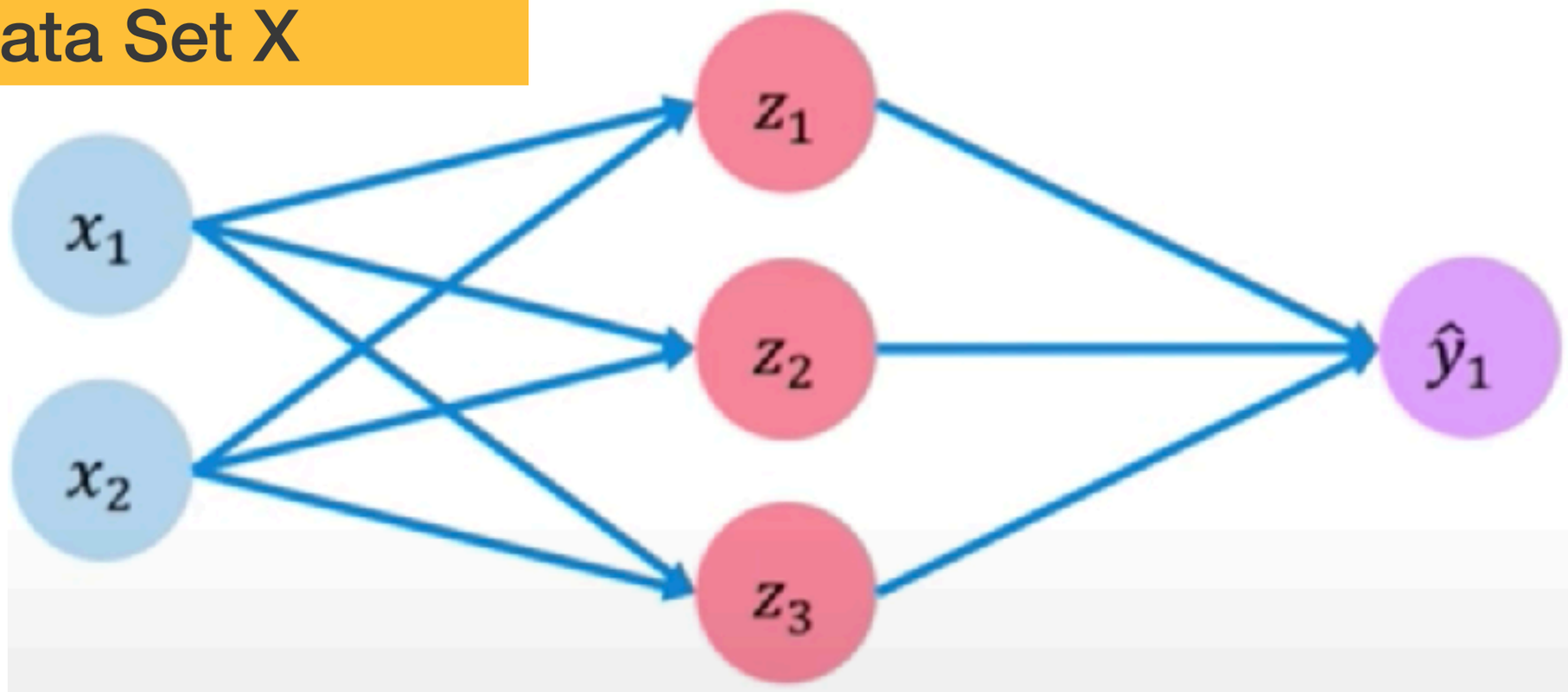
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Prediction $\hat{y}$, target y

MIT Example Problem: Will I Pass This Class? The Empirical LOSS (COST)

- Measure the TOTAL LOSS over the entire DATASET

Data Set X



Total Loss is the Average

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

Prediction $\hat{y}$, target y

MIT Example Problem: Will I Pass This Class?

Binary Cross Entropy Loss

- Here the output target is binary (fail or pass, 0 or 1)

- We want to predict the probability to pass the class

Define $P(y|x) = \begin{cases} \hat{p} & \text{if } y=1 \\ (1-\hat{p}) & \text{if } y=0 \end{cases}$

- A good classifier give high value of $\hat{p}$ when the target is y=1, so what we wants to
  maximize $\hat{p}$ when y=1, maximized (1-$\hat{p}$) when y=o

  - Maximize the Bernoulli Distribution

$$p(y|x) = \hat{p}^{y}(1-\hat{p})^{1-y} \Rightarrow \quad \begin{aligned} y=1 &\rightarrow P(y|x) = \hat{p} \\ y=0 &\rightarrow P(y|x) = 1-\hat{p} \end{aligned}$$

$$\log P(y|x) = y\log\hat{p} + (1-y)\log(1-\hat{p}) \qquad \frac{\partial \log P(y|x)}{\partial \hat{p}} = 0 \Rightarrow \hat{p} = y$$

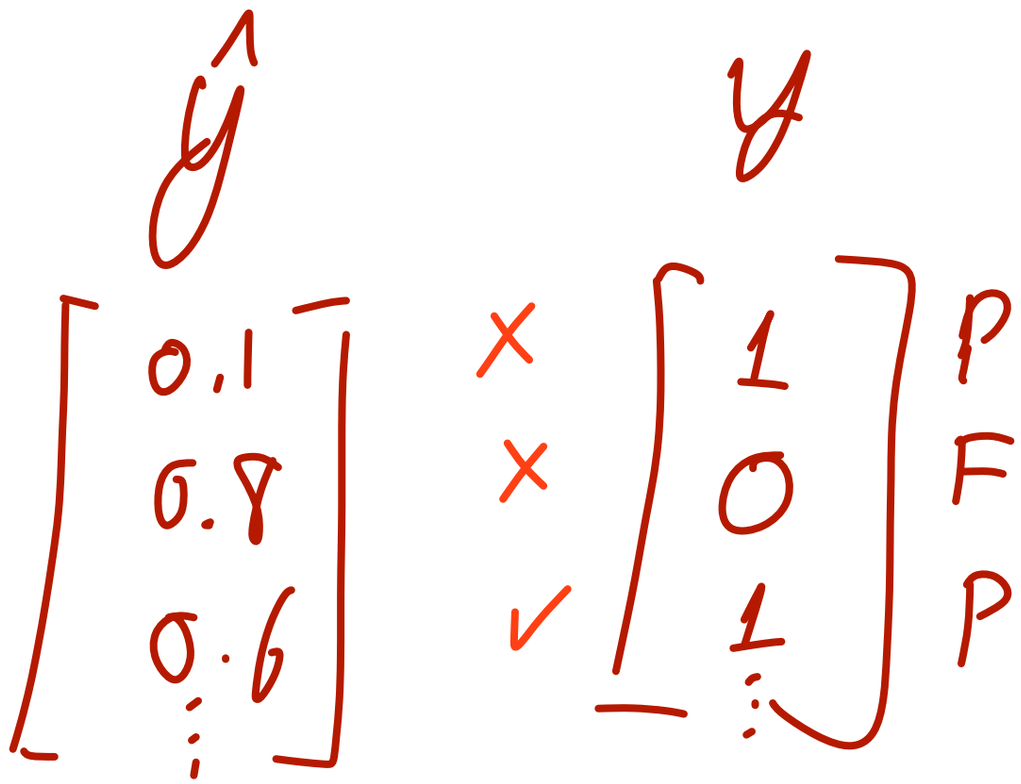
Binary Cross Entropy Loss

$$p(y\,|\,x) = \hat{p}^{y}(1-\hat{p})^{1-y}$$

$$logP(y\,|\,x) = ylog\hat{p} + (1-y)log(1-\hat{p})$$

$$BCE(y,\hat{p}) = -logP(y\,|\,x) = -ylog\hat{p} - 1 - y)log(1-\hat{p})$$

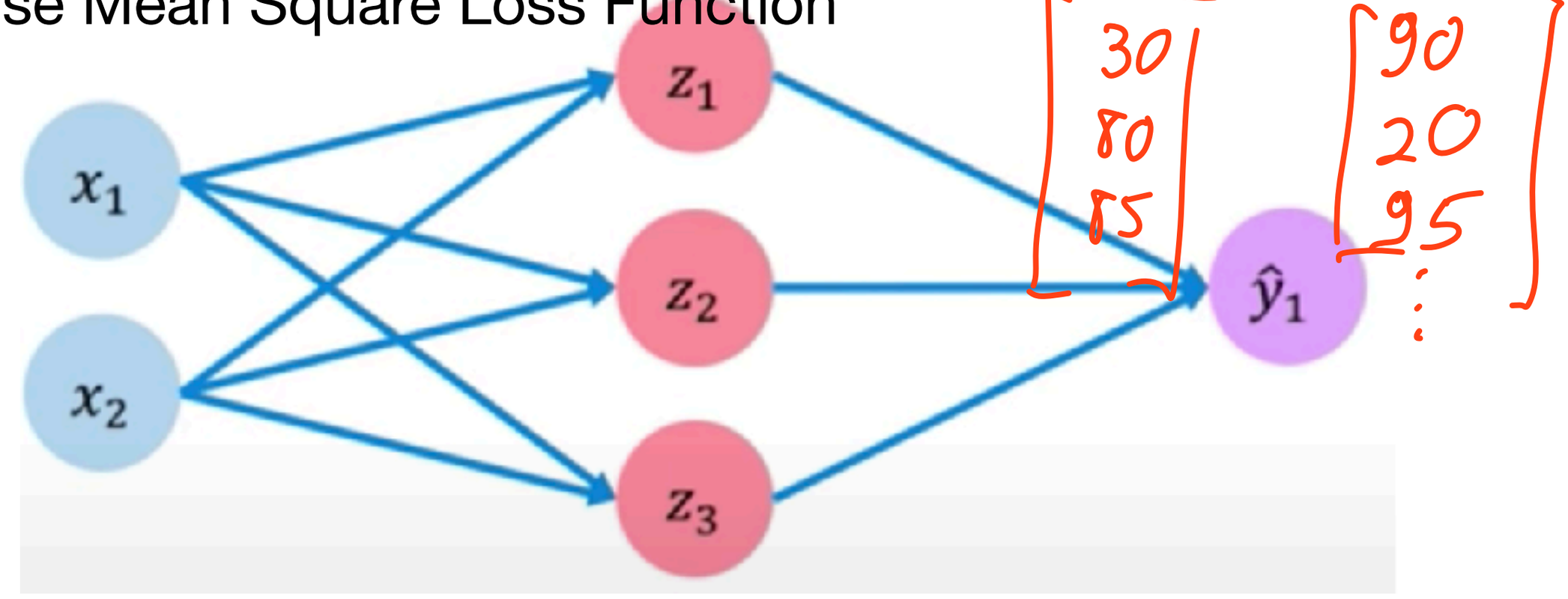
MIT Example Problem: Will I Pass This Class?
Binary Cross Entropy Loss



$$\mathscr{L}(w) = \frac{1}{n}\left(\sum_i y^i \log \hat{y} + \left(1 + y^i\right) \log\left(1 - \hat{y}^i\right)\right)$$

MIT Example Problem: Will I Pass This Class? Regression (try to predict the grade)

- Try to predict continuous real numbers
- Use Mean Square Loss Function



$$\mathscr{L}(w) = \frac{1}{n} \sum_{i} \left(y^i - \hat{y}^i \right)^2$$

Loss Optimization

- Find the NN weights that give the minimus loss

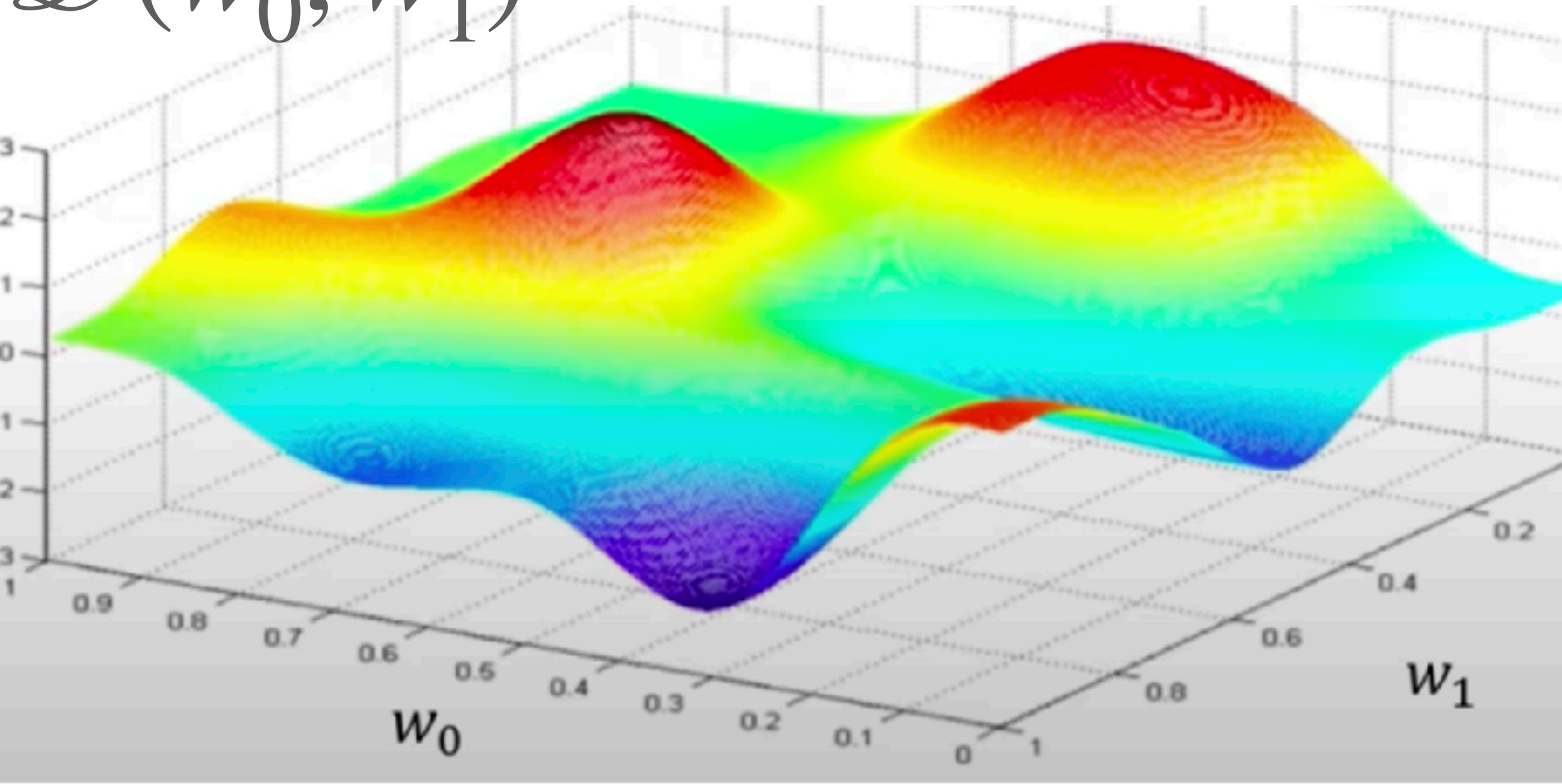
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$W^* = argmin_W \mathcal{L}(W)$$

$$W = \{ W^{(0)}, W^{(1)}, \ldots \}$$

LEARNING

$\mathscr{L}(w_0, w_1)$

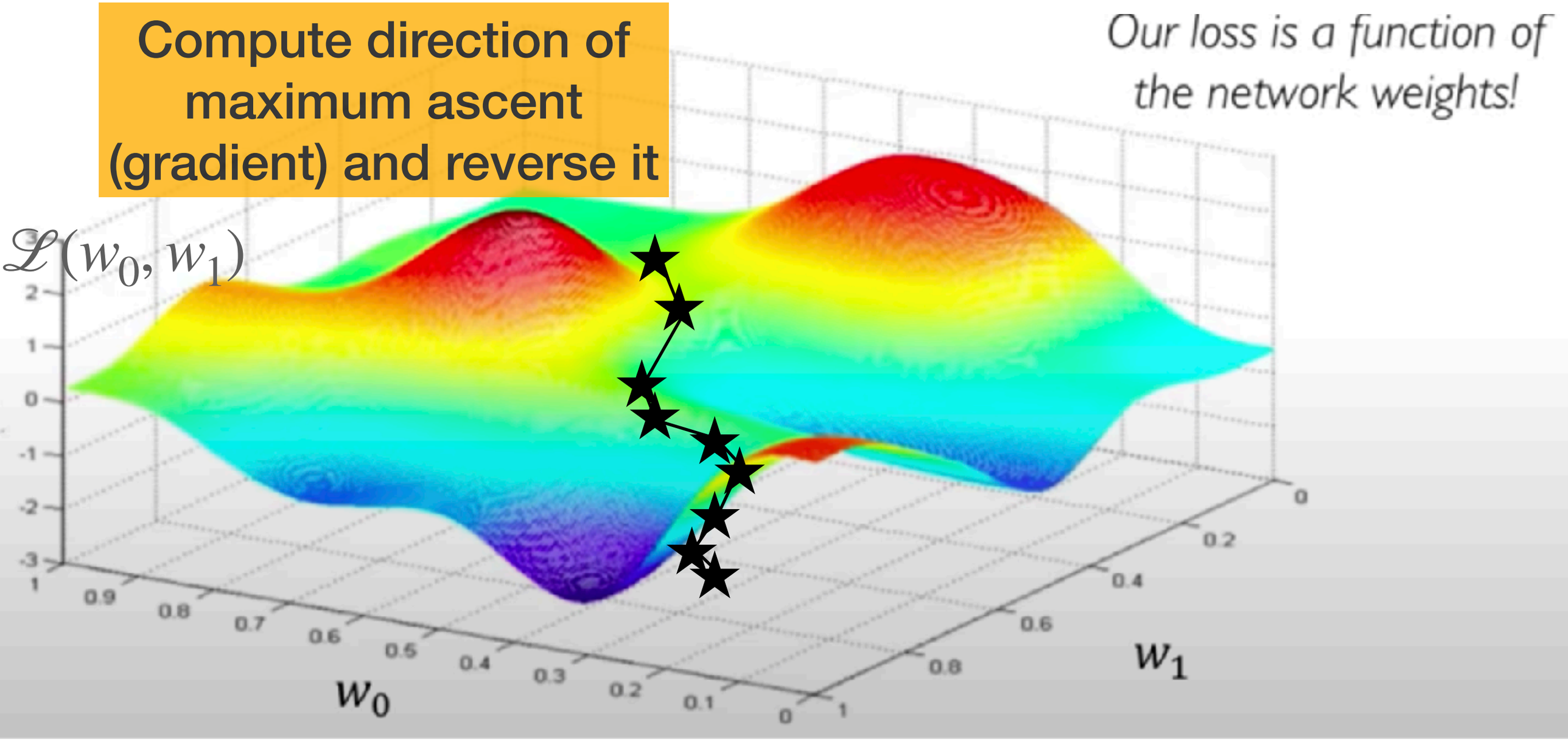


Loss Optimization: Gradient Descent

- The Loss is a function of the NN weights

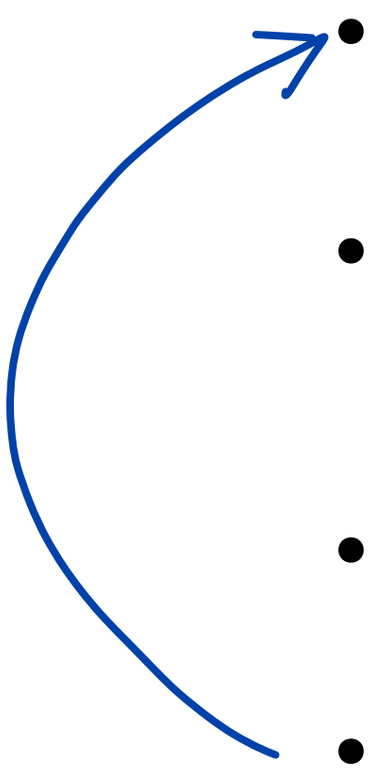
$$W^* = argmin_W \mathcal{L}(W)$$

Compute direction of maximum ascent (gradient) and reverse it



Gradient Descent

- Algorithm
  - Initialize weights randomly
  - Loop until convergence:
    - Compute Gradient $\frac{\partial \mathcal{L}(w)}{\partial w}$
    - Take a step η (the learning rate - how fast you want to achieve the goal)
    - Update weights in the opposite direction $w \rightarrow w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$
    - Return weights



BACKPROPAGATION

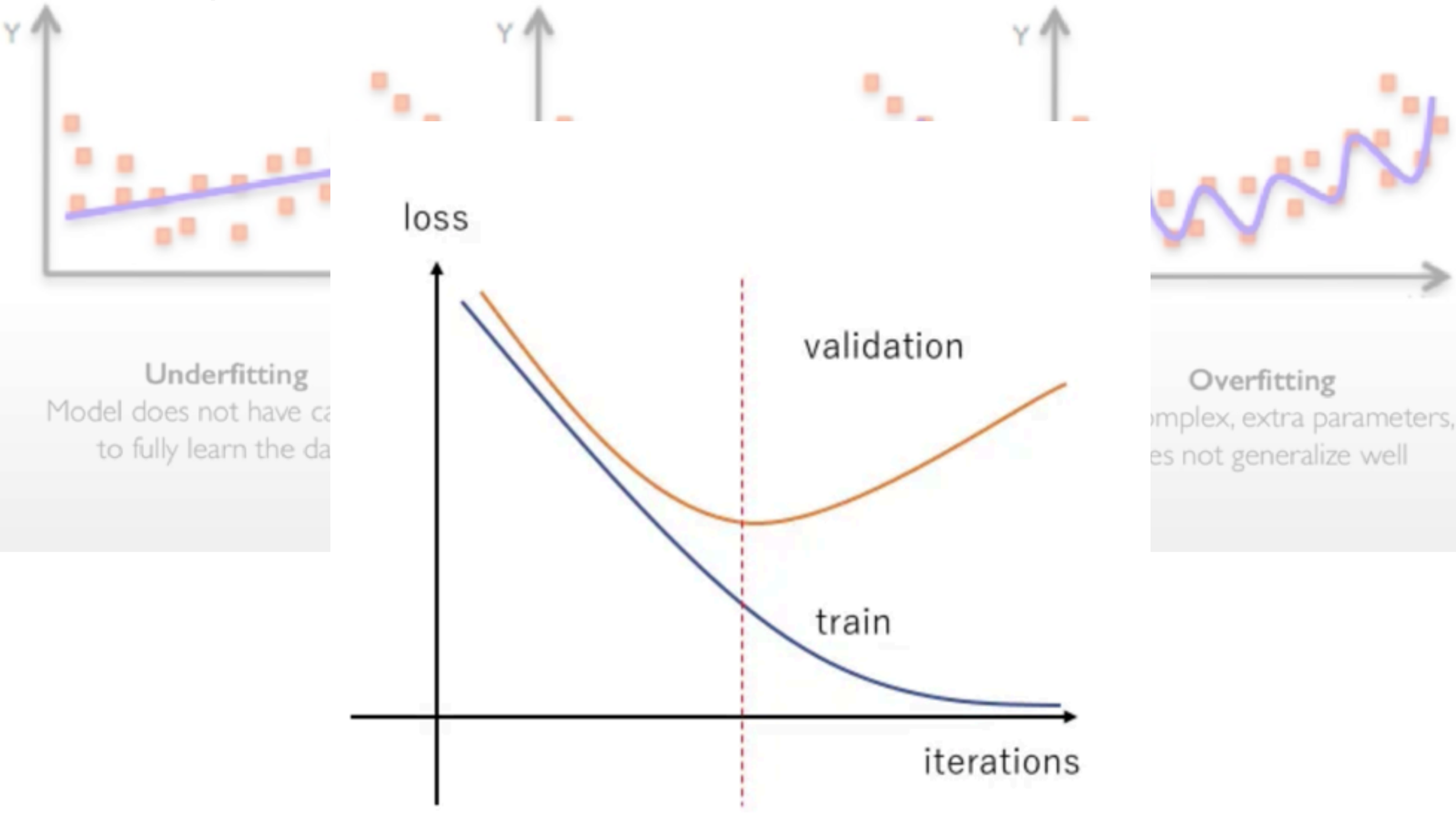
- **Backpropagation** is the algorithm that computes the gradient of a loss function with respect to the weights of the NN in an efficient way

- It is essential to do so in an efficient way in order to cope with Multi-Layer Networks.

- Backpropagation is calculating the gradient iterating backward from the last layer to the network input

Overfitting

- Stop Loss



Summary: Training

- We divide our labeled DATA into~80% training and 20% validation (sometimes also keep some DATA for test)

- In each epoch we run on a batch (1000s of samples) and adjust the weights

- Our success is measured by the accuracy of our predictions

- This gap between training accuracy and test accuracy is overfitting: Overfitting is a central issue

- We want to eliminate the model from learning to memorize the training DATA, which will improve the model's generalization to unseen DATA.

GNN & HEP

Graph NN & High Energy Physics

The Philosophy

- GNNs are a class of deep learning architectures that implement strong relational inductive biases for learning functions that operate on graphs.

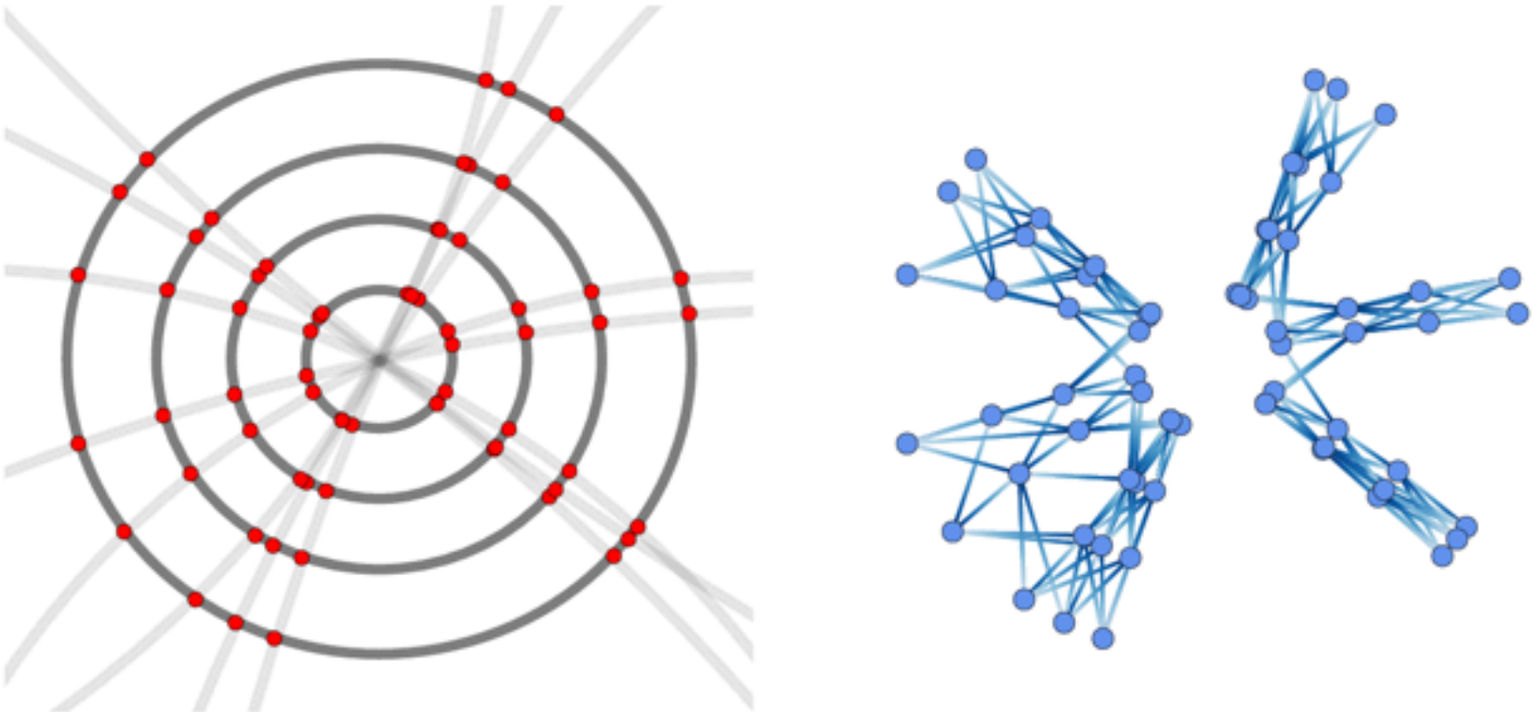
- Inductive Bias means that using GNN we can inject our physics understanding (e.g. Lagrangians) into the problem

- GNN implements a form of parameterized message-passing whereby information is propagated across the graph, allowing sophisticated edge-, node-, and graph-level outputs to be computed

- Within a GNN there are one or more standard neural network building blocks, typically fully connected layers, which implement the message computations and propagation functions.

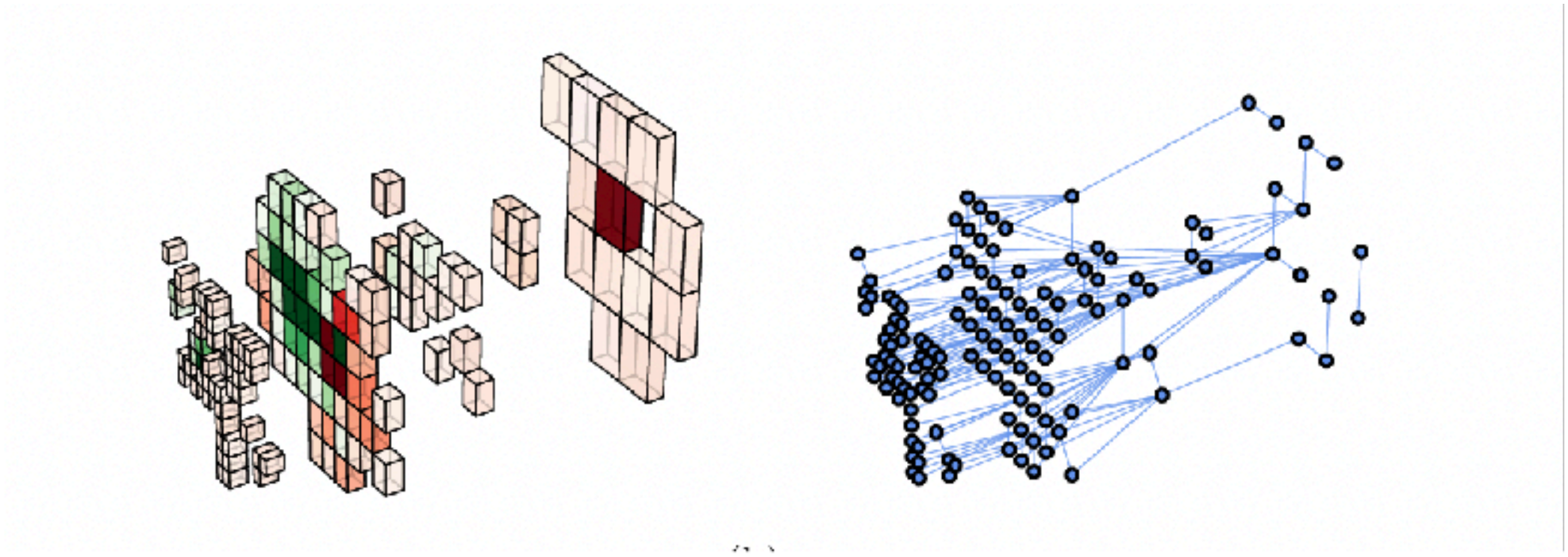
GNN Example 1

clustering tracking detector hits into tracks,



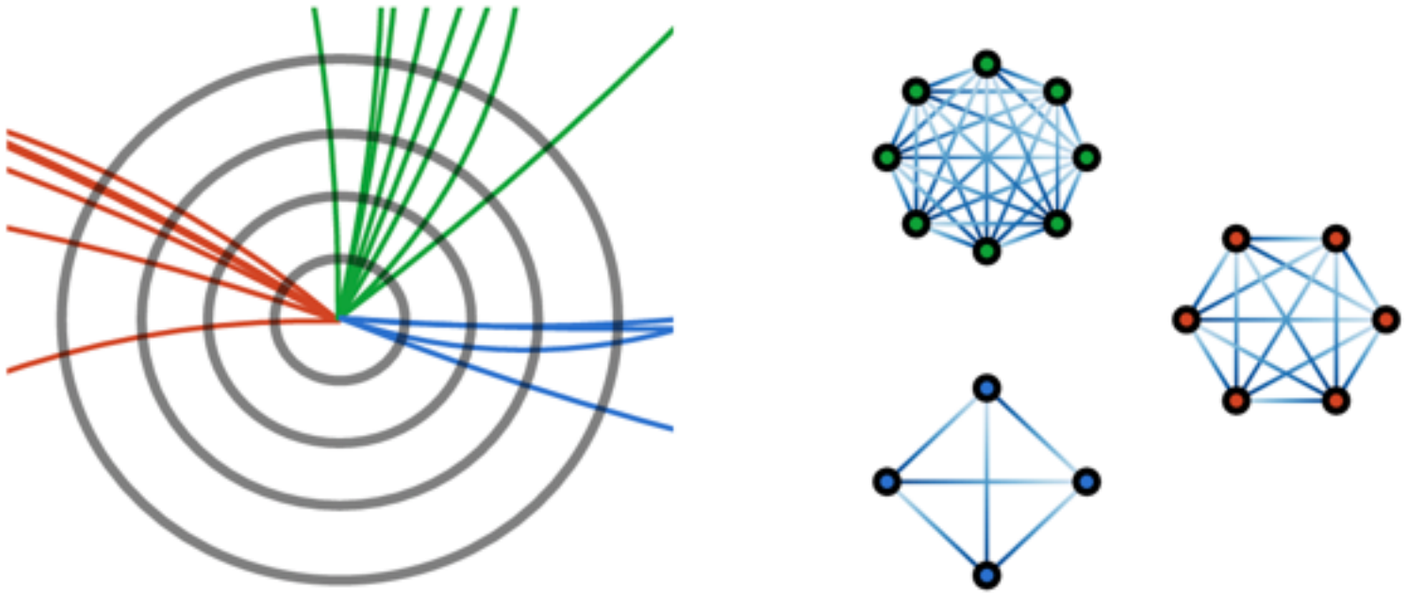
GNN Example 2

Segmenting Calorimeter Cells



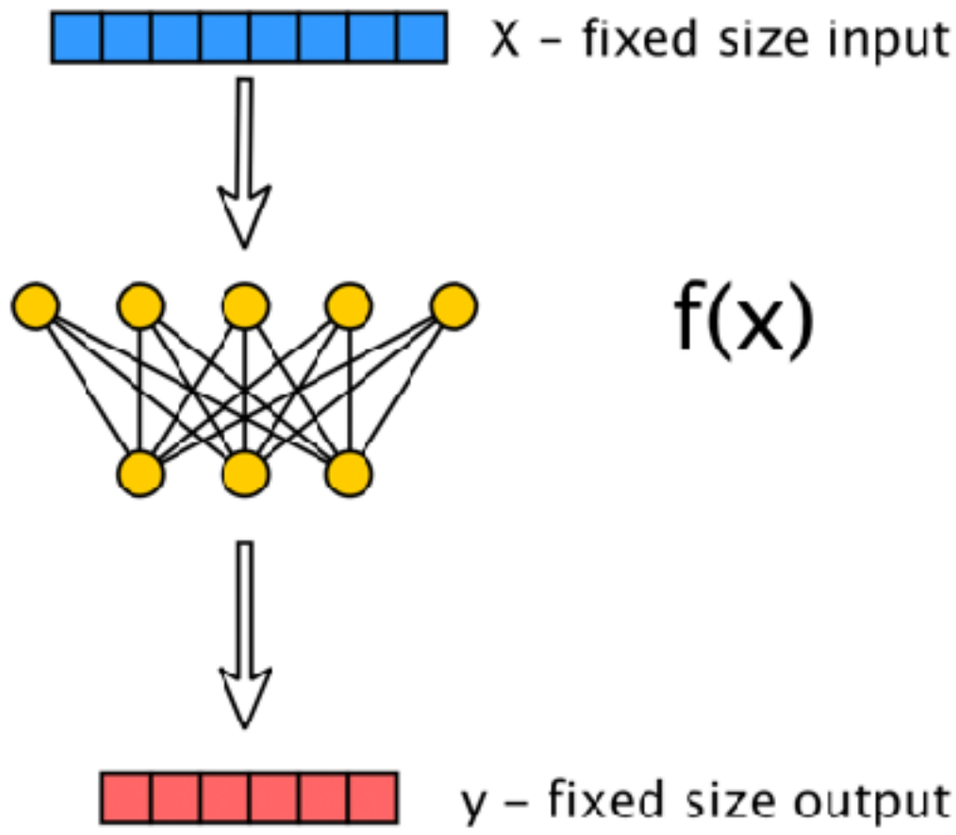
GNN Example 3

Jet Classification



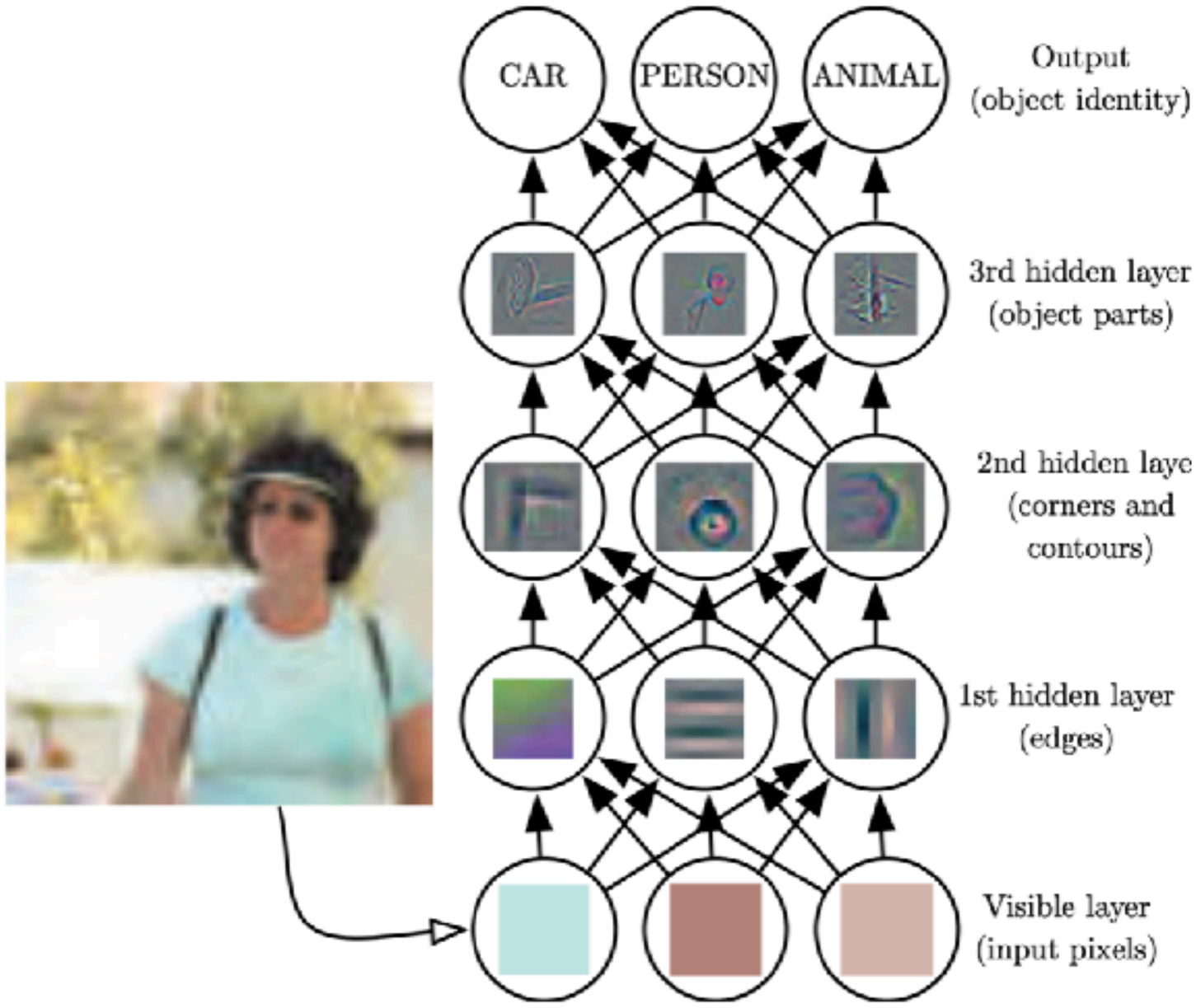
MLP

- MLP (NN) is the basic building block of GNN
- It has a fixed-size of input and output
- MLP is a sequence of matrix multiplications with non-linear element-wise functions between them, which transform the input into the output.
- Weights learnt. by using stochastic gradient descent.
- This structure can, in theory, learn to approximate any function



Hidden Representations

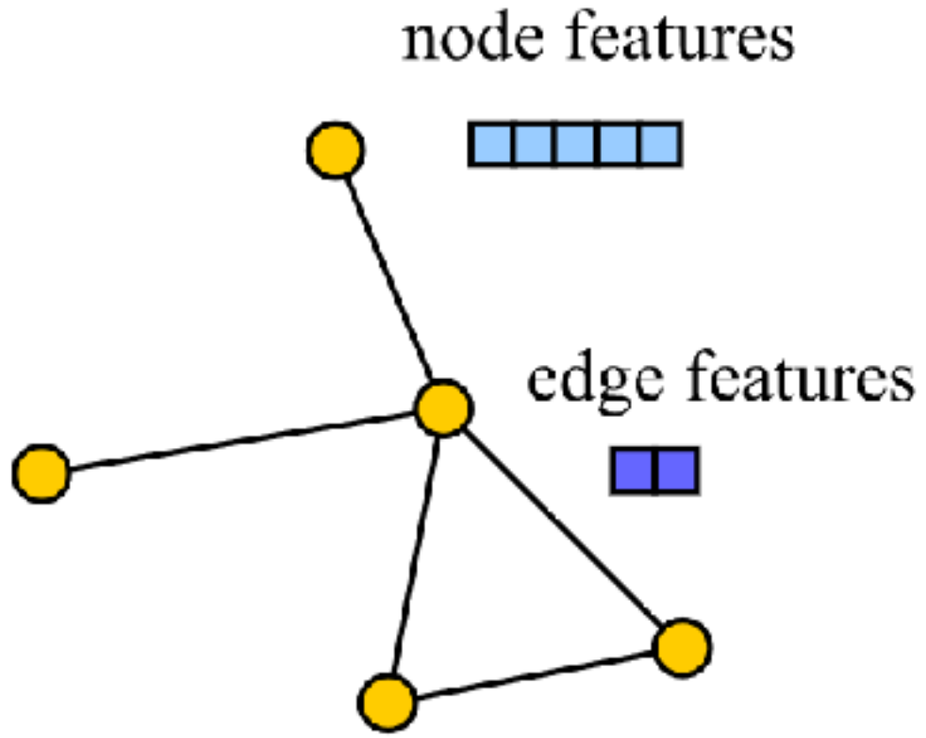
- The network creates its own language via representations



Nodes Represerntation

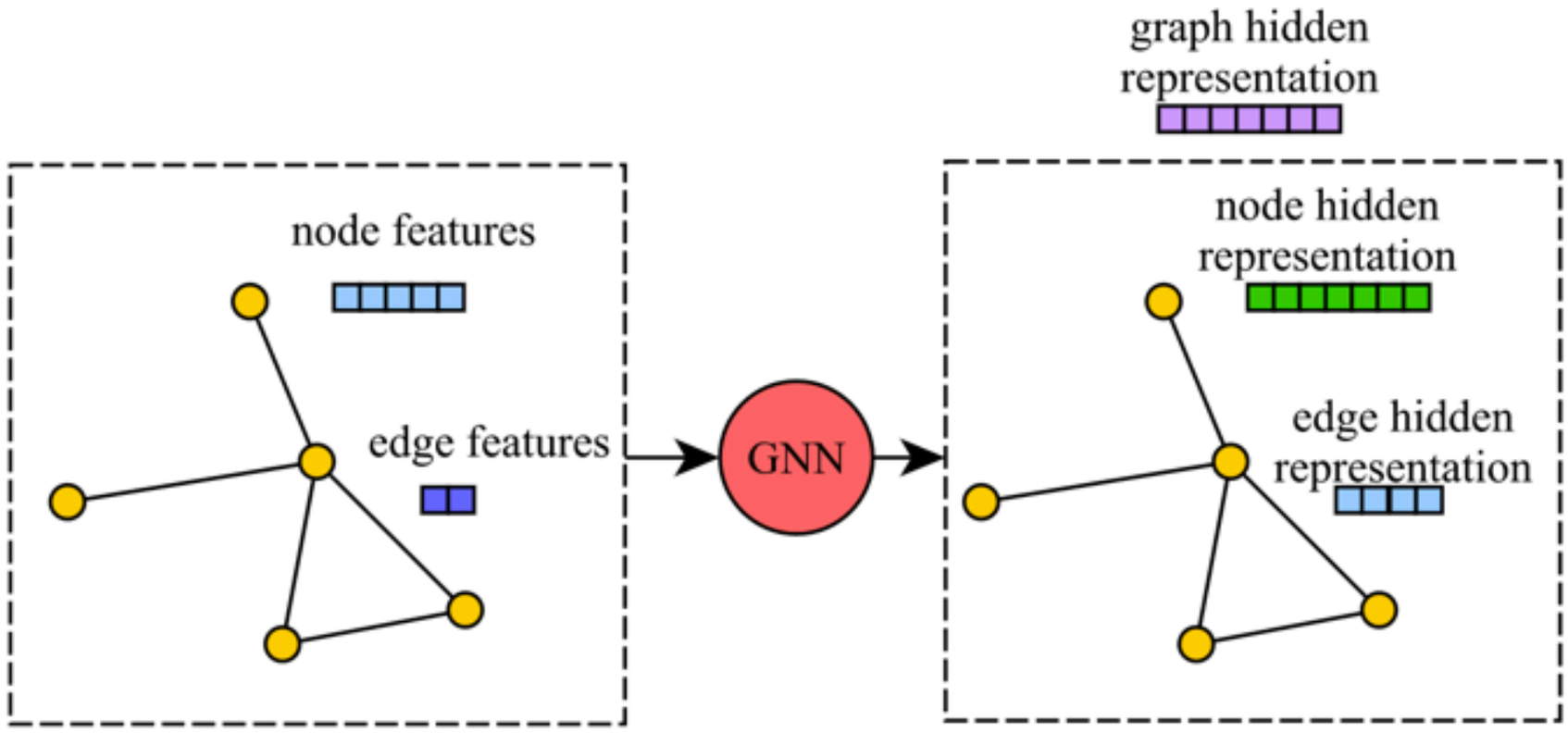
- A network ”creates a representation” for the nodes in the GNN

- By a sequence of data-transformation steps, the neural network is learning to describe the data in its own format such that it will be able to perform the function approximation task it was asked to do.



GNN

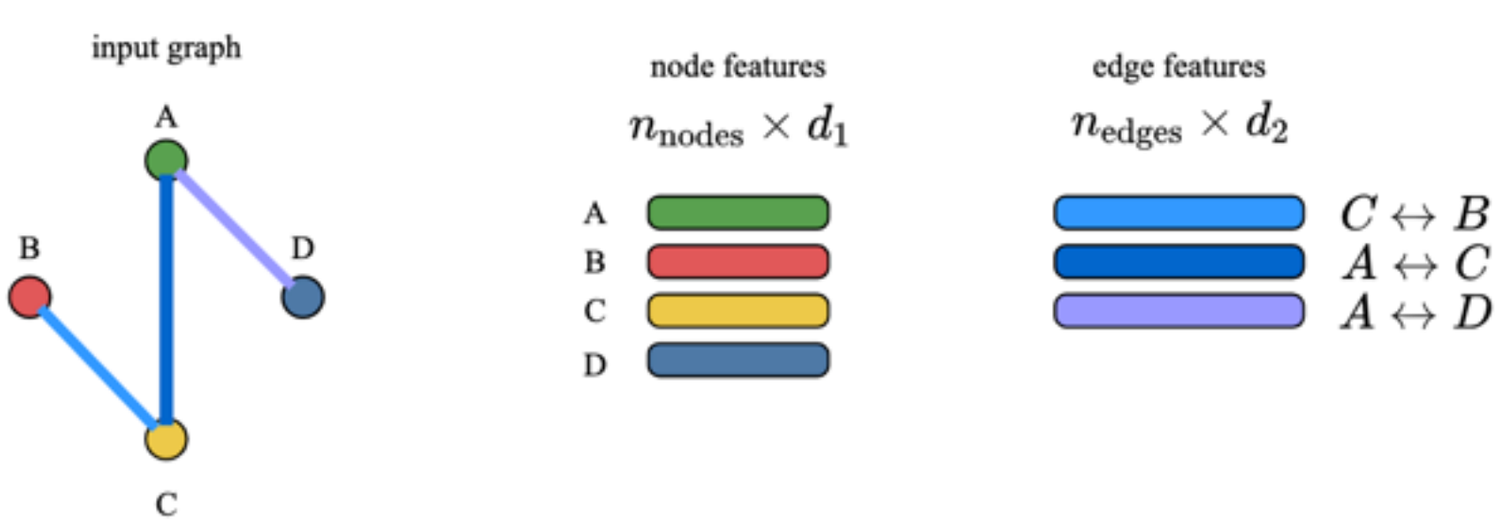
- The hidden representation term is a key concept in GNNs. Looking at the GNN as a black box, it allows for creating neural networks that operate on **variable size sets and graphs** and create a hidden representation for the components - the nodes, edges and graph



GNN Inputs and Equivariance

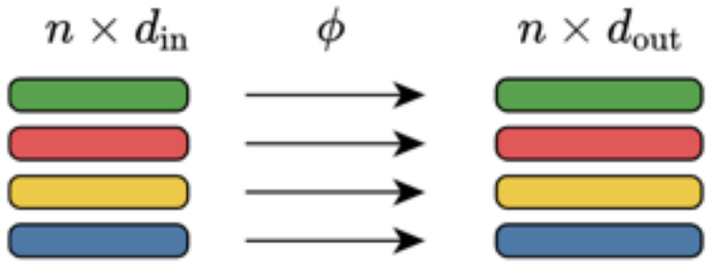
- Each Node , Edge has Features

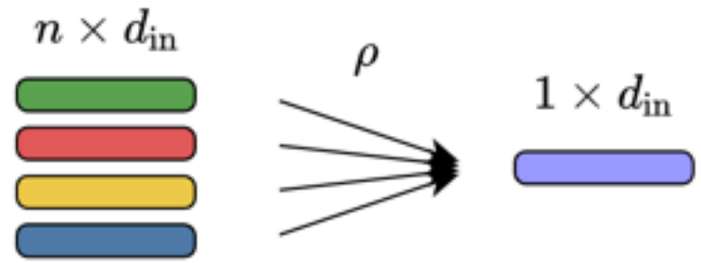
- GNN is Equivariant:
  The output of the GNN must be equivariant to the arbitrary labeling of the nodes.
  Permutation of the input permutes the output



Equivariant Building Blocks

- Basic operations: Parallel Update and Permutation invariant aggregation





Parallel Update (ϕ)

Permutation invariant aggregation (ρ)

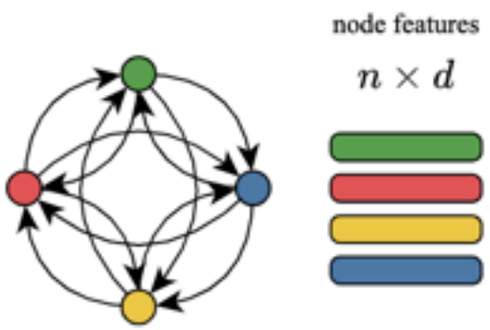
GN Blocks

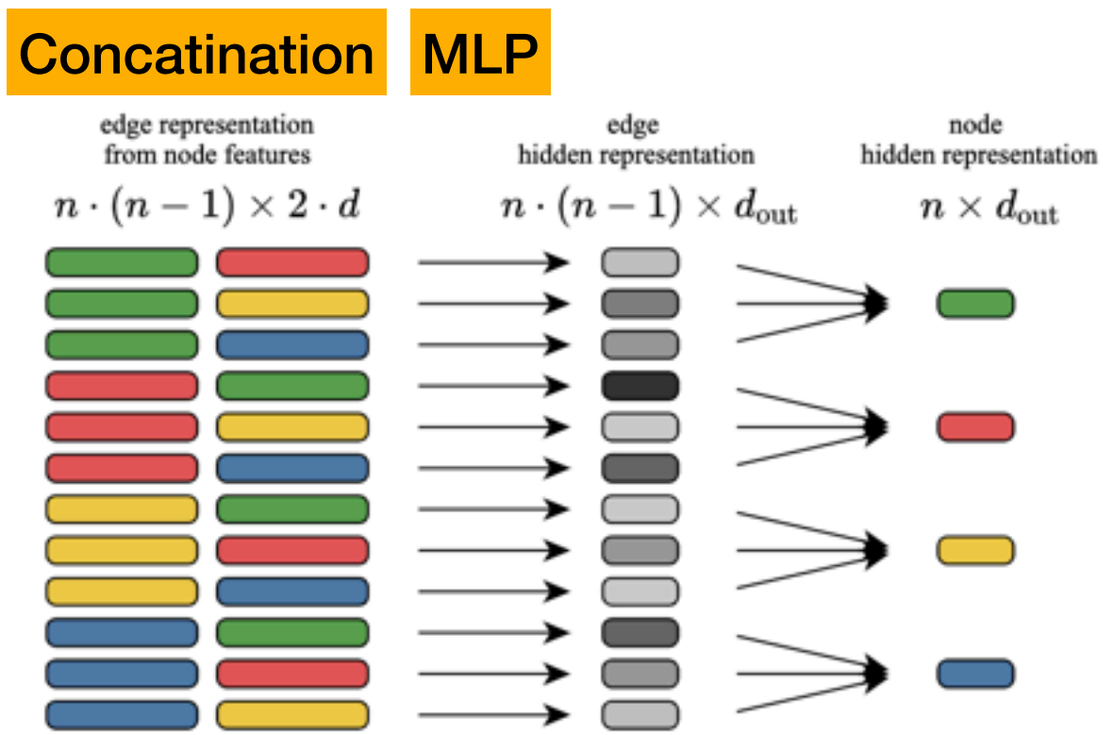
- Architecture Example: Graph with only input node features and directional edges

- First, the node features are used to construct an edge representation simply by concatenating the node features

- The same MLP is used to create edge (hidden) representations

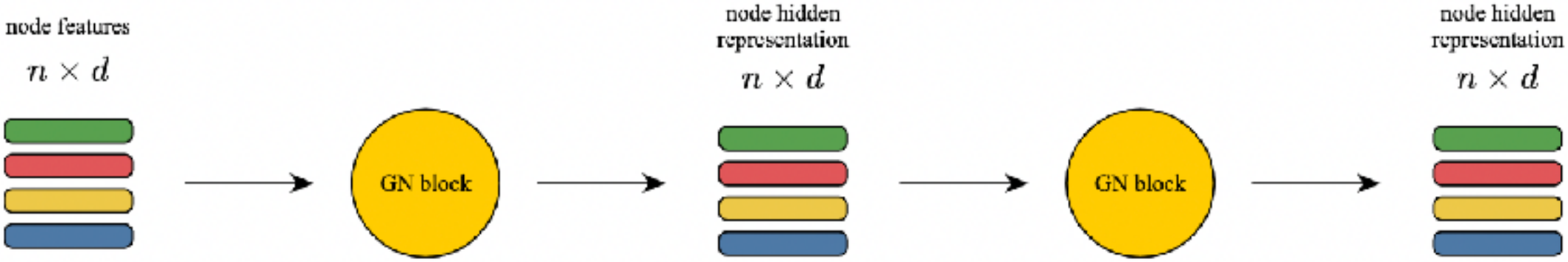
- Edges aggregated to create (hidden) node representations





Stacking GN Blocks

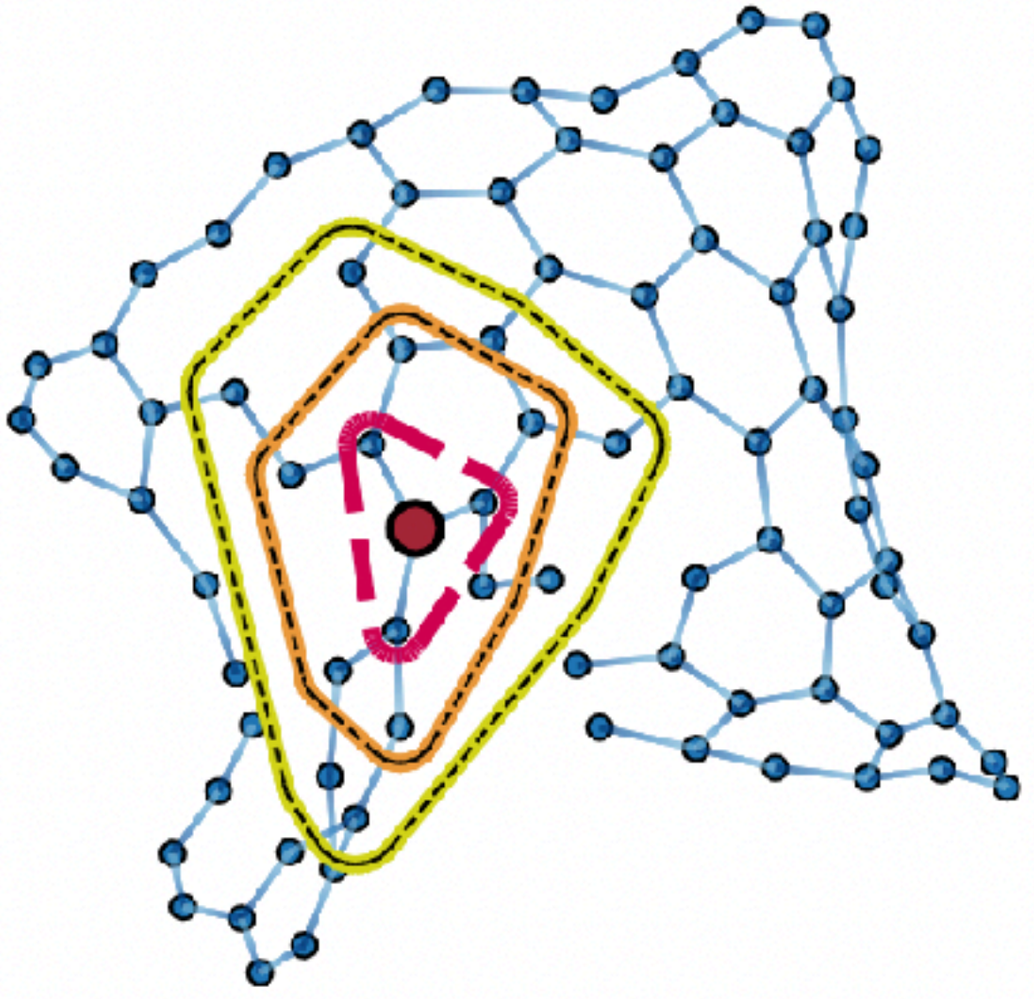
- Stacking GN Blocks increases the depth of the network, which allows for approximating more complex functions.



Increasing the Receptive Field

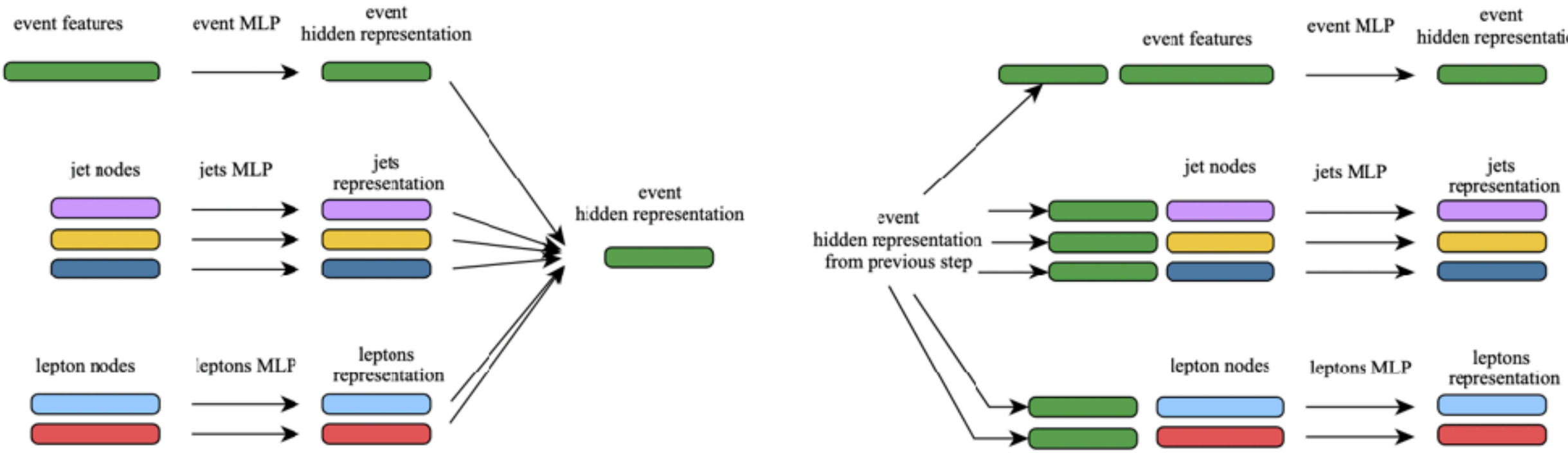
- Stacking GN Blocks increases the receptive field of a node

- Each iteration communicates with a remoter circle of neighbors



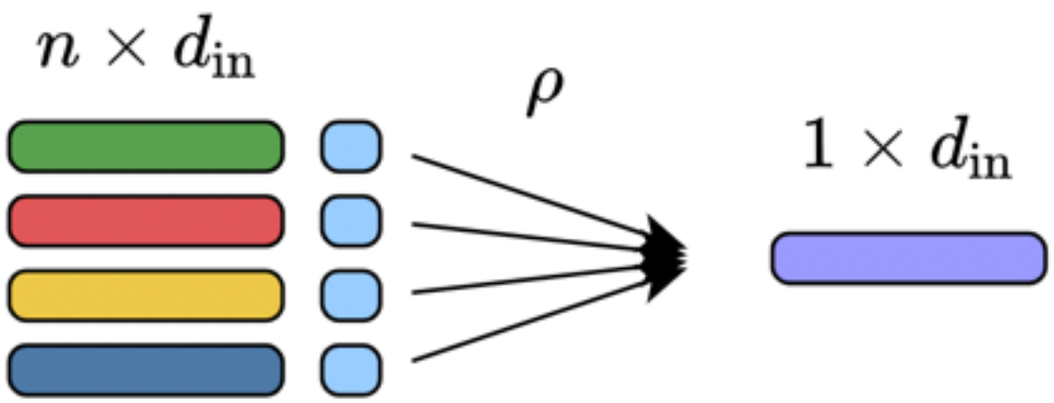
A Typical Design Example

- Creating GNN from Jets and Lepton features



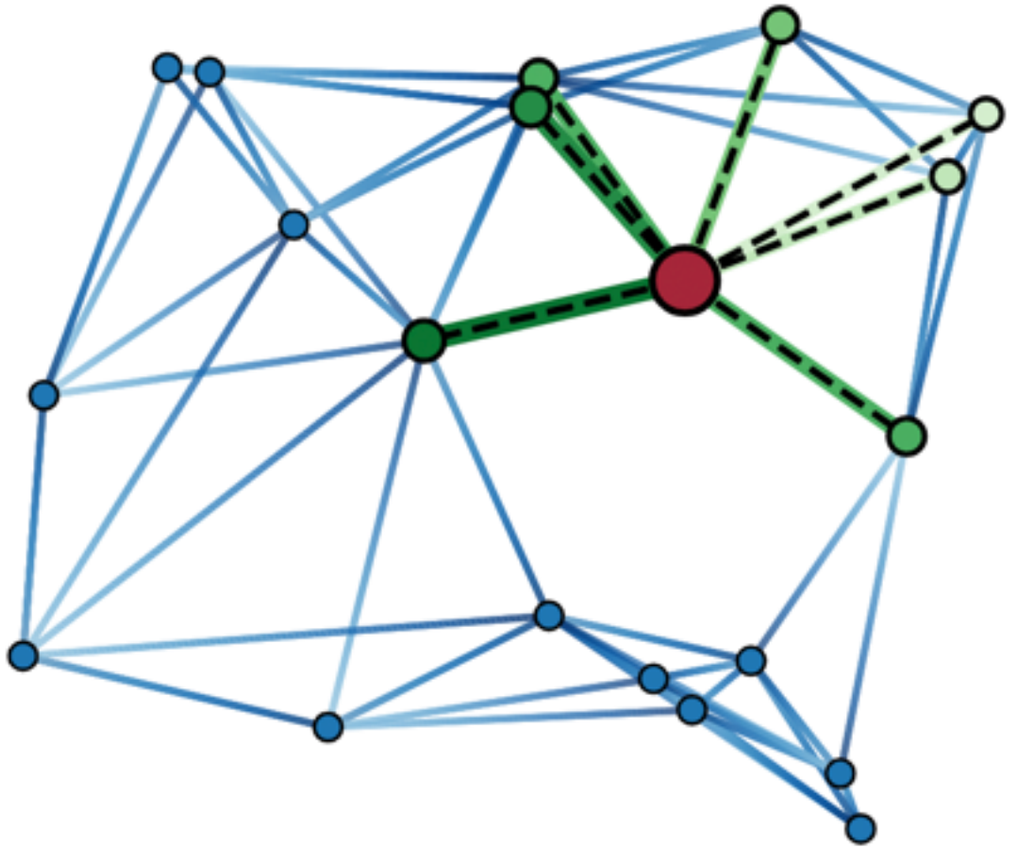
Attention

- Use weighted sums in the various aggregation functions of the GN



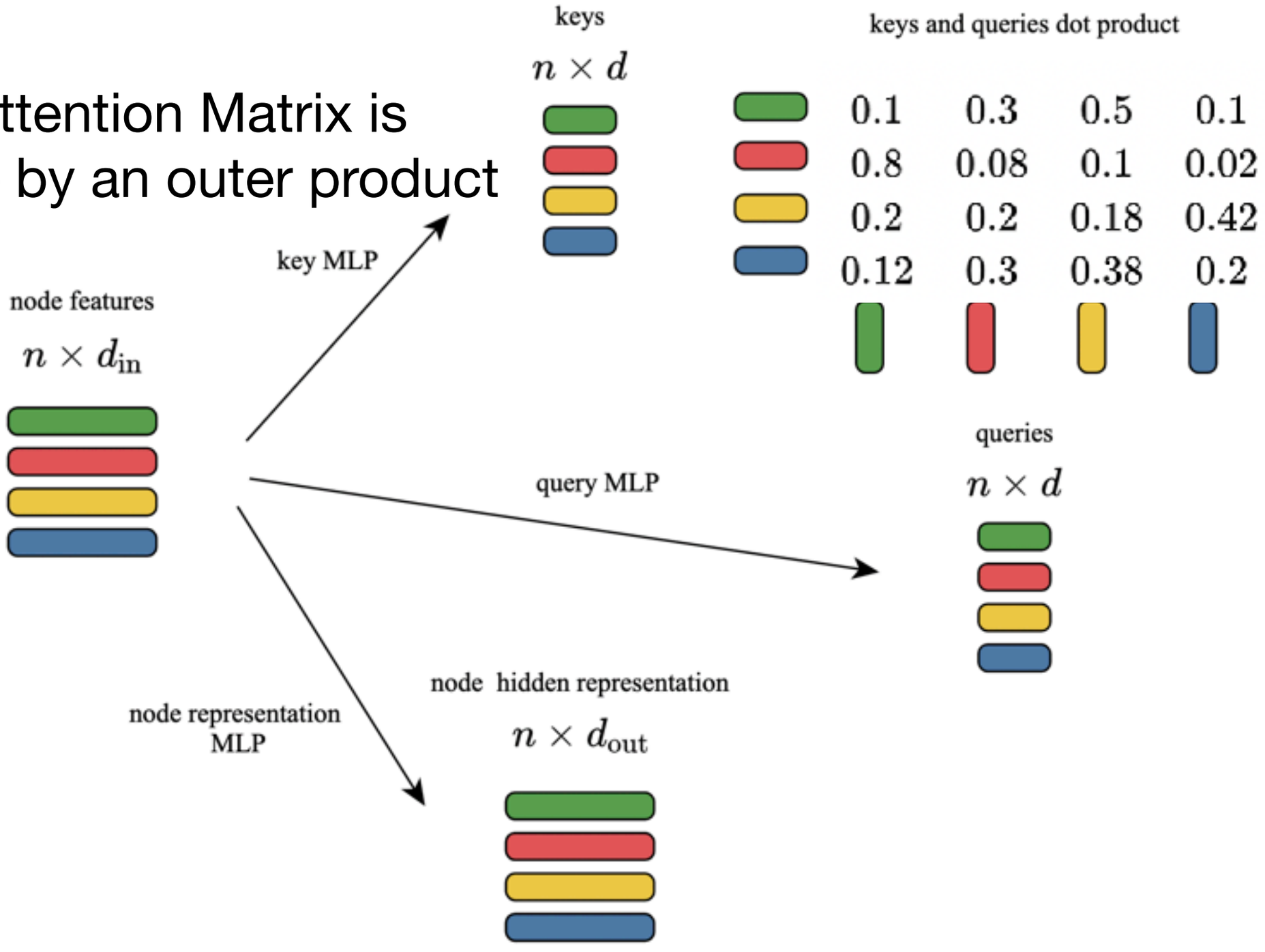
- Give the network flexibility in determining the relative importanc of elements in a set to a particular task

- The red node is collecting information from all its incoming edges, which have different weights assigned to them, shown as different shades of green.



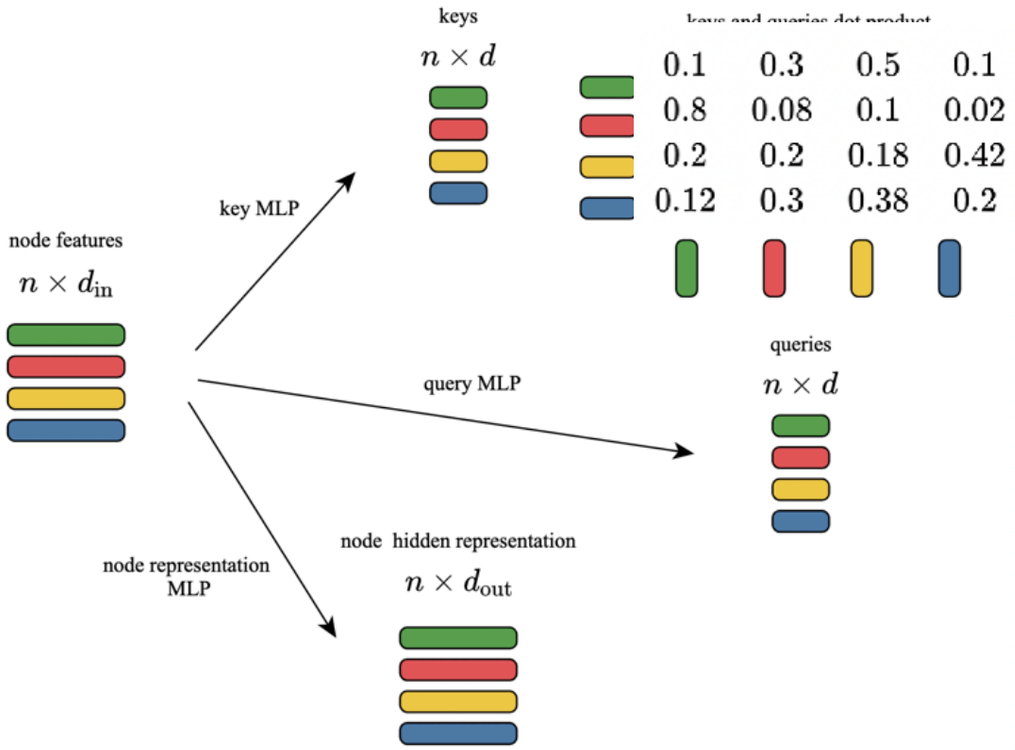
Key Query Attention

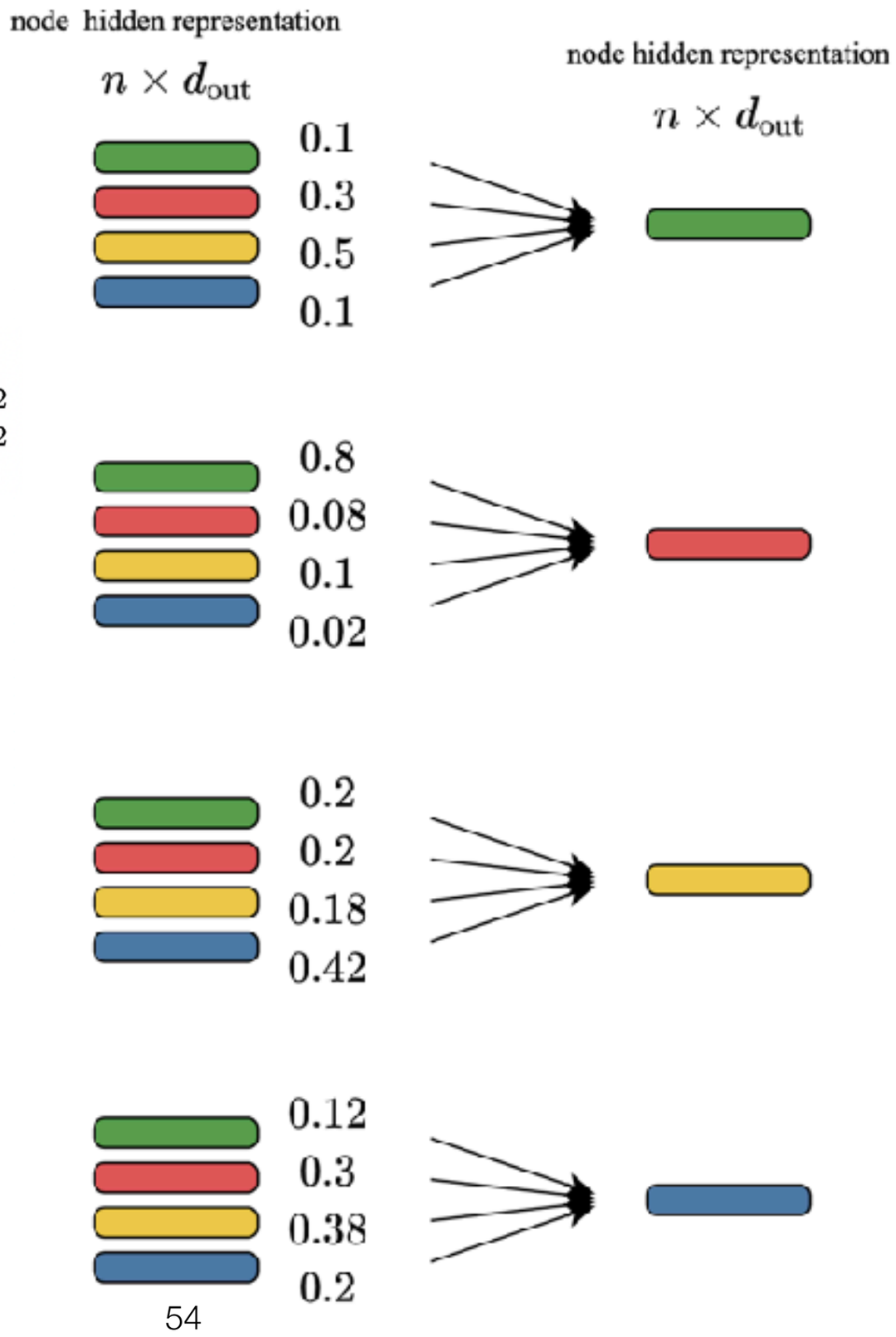
- 2 MLPs create the Keys and the Queries
- The attention Matrix is made by an outer product



Key Query Attention

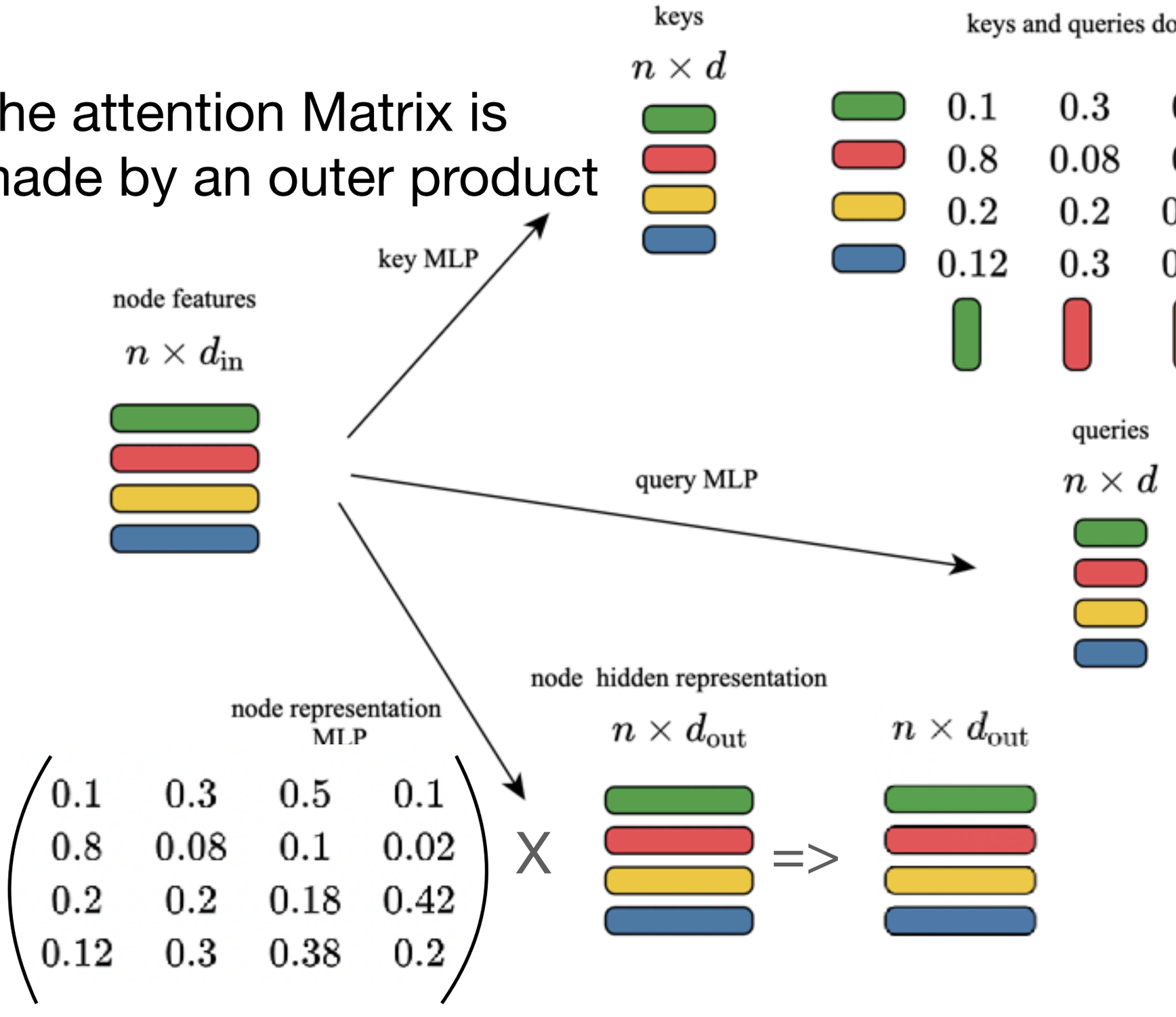
- 2 MLPs create the Keys and the Queries
- The attention Matrix is made by an outer product





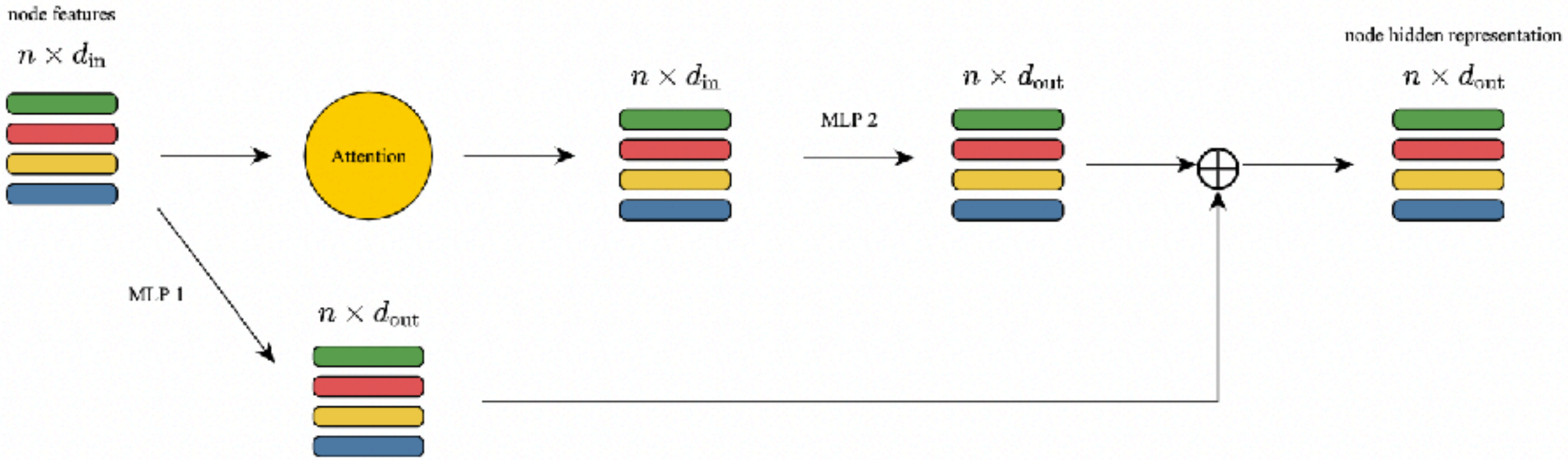
Key Query Attention

- 2 MLPs create the Keys and the Queries
- The attention Matrix is made by an outer product



Deep set GN block

- Here the nodes are not connected by edges, yet you allow the nodes to communicate with each other via an attention mechanism



Summary

- We are in the midst of a revolution that is taking the HEP world by a storm

- Deep Learning is changing the world of HEP

- DL allows faster, more efficient, and more reliable (with reduced systematics) HEP Data analyses

- DL will also improve our Data taking via improved triggers and real time analyses

- DL is the new calculus…
  In 10 years there will be no HEP Data taking and analysis without Deep Learning… So better take the ride (or stay behind)