Midterm Review

Snehal, Sean, Adhitya, Lucas, Ziga

Agenda

Agenda

- Fundamentals

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers
- CNNs

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

- CNNs
- Modern Vision Networks

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

- CNNs
- Modern Vision Networks
- Generative Models - VAEs, GANs, Diffusion

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

10 MIN BREAK

- CNNs
- Modern Vision Networks
- Generative Models - VAEs, GANs, Diffusion

Agenda

- Fundamentals
- NLP - Word Embeddings, RNNs + LSTMs, Transformers

10 MIN BREAK

- CNNs
- Modern Vision Networks
- Generative Models - VAEs, GANs, Diffusion

We’ll reference prelim questions throughout review

Disclaimer

Disclaimer

Topics covered are NOT indicative of content appearing on the Midterm

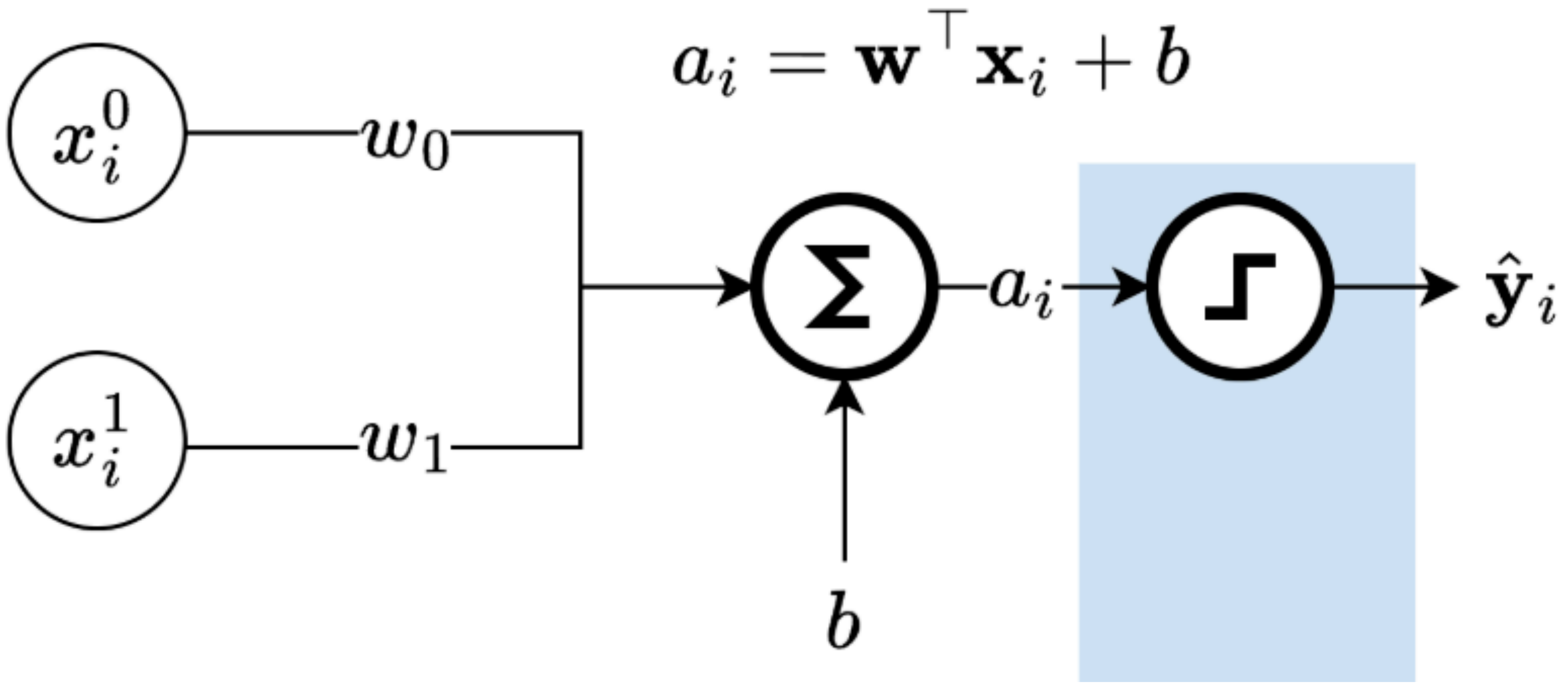
Disclaimer

Topics covered are NOT indicative of content appearing on the Midterm

Material covered today should NOT be interpreted as a suggestion or hint for the Midterm's scope

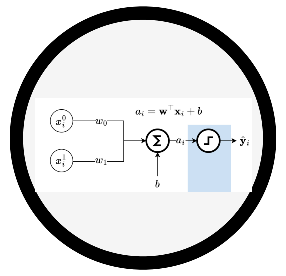
Fundamentals

You already know what a Neuron is!



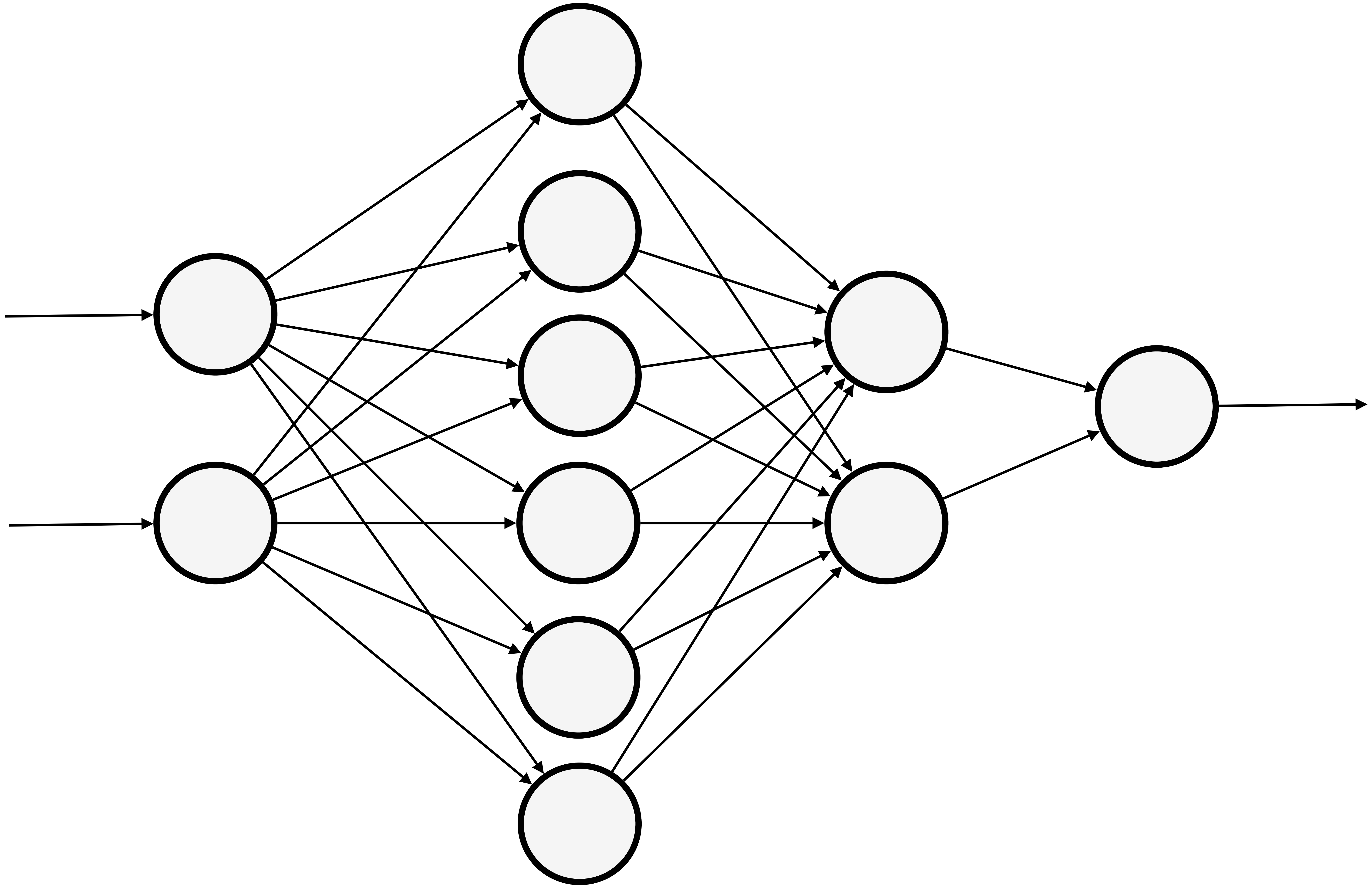
Neural Net

And that multiple neurons form a Neural Net!



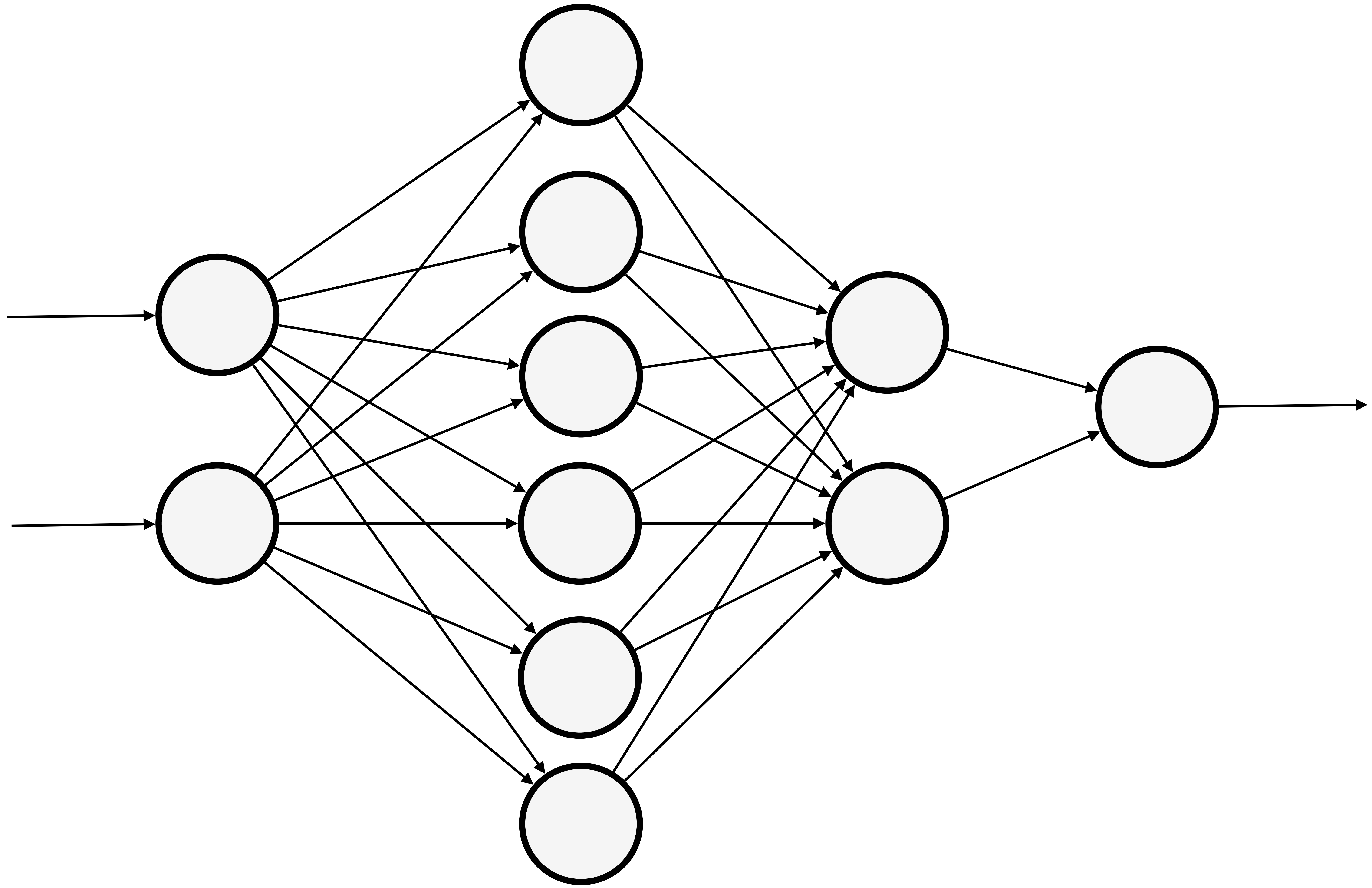
Neural Net

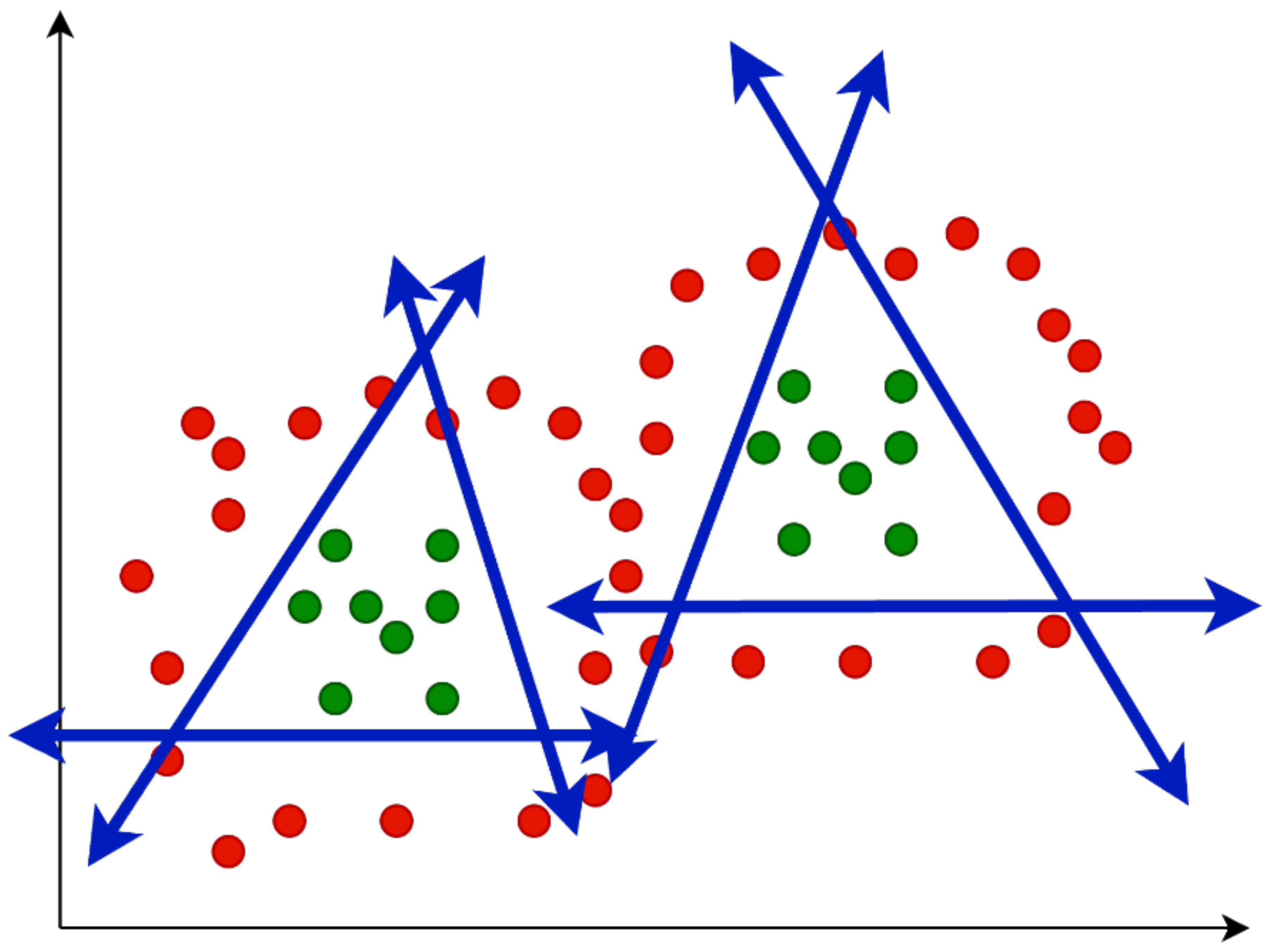
And that multiple neurons form a Neural Net!



Neural Net

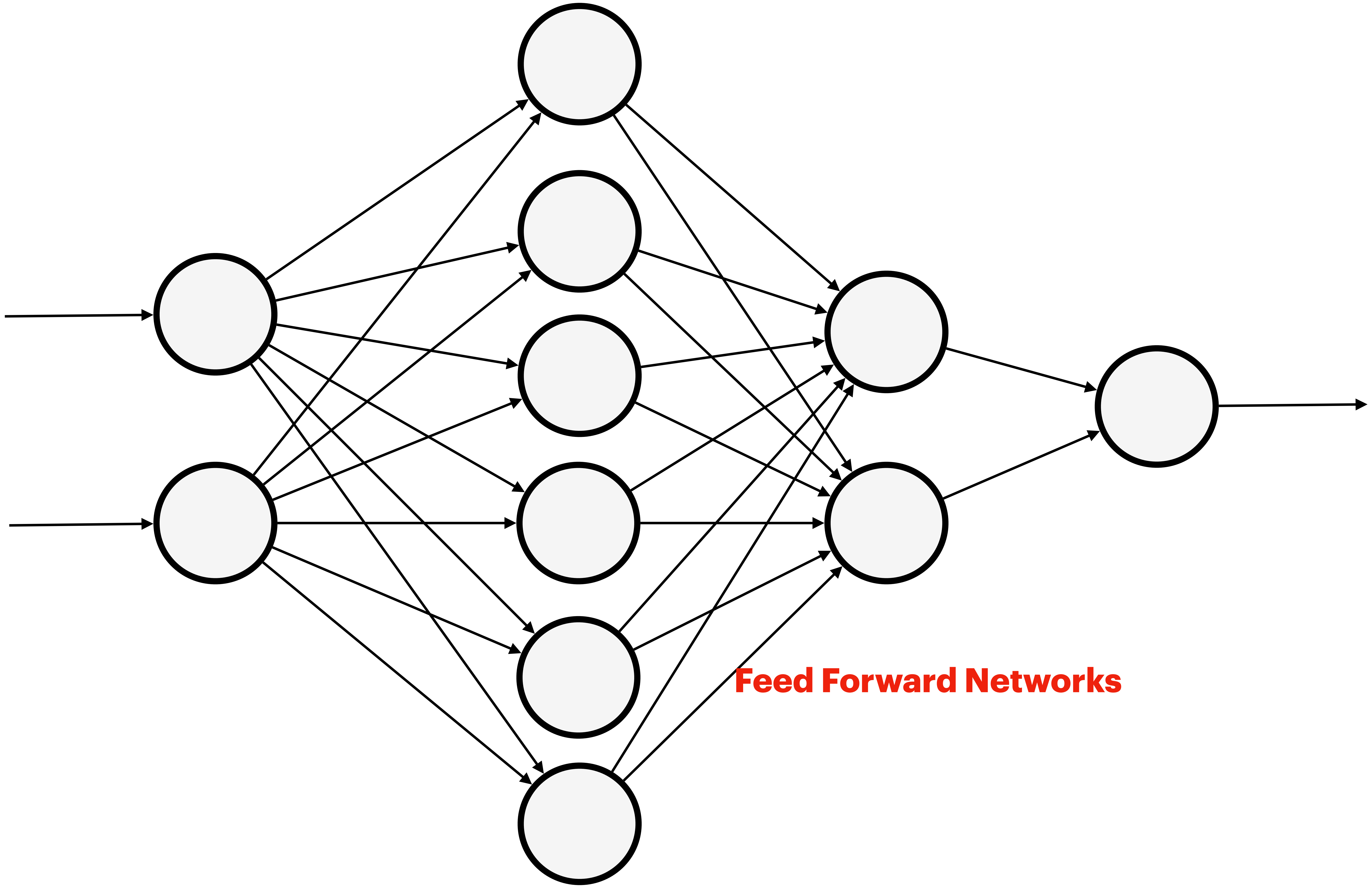
And that multiple neurons form a Neural Net!

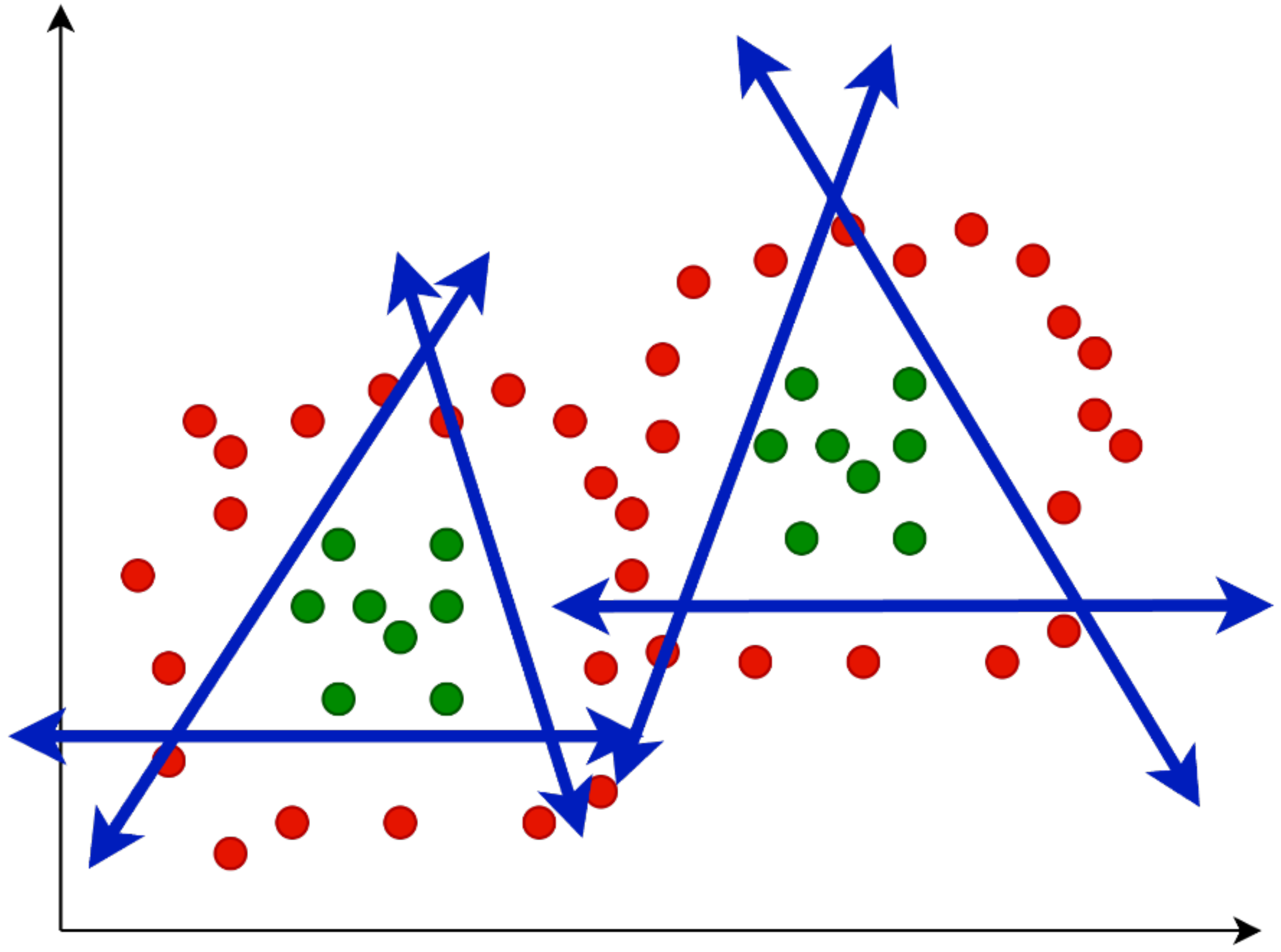




Neural Net

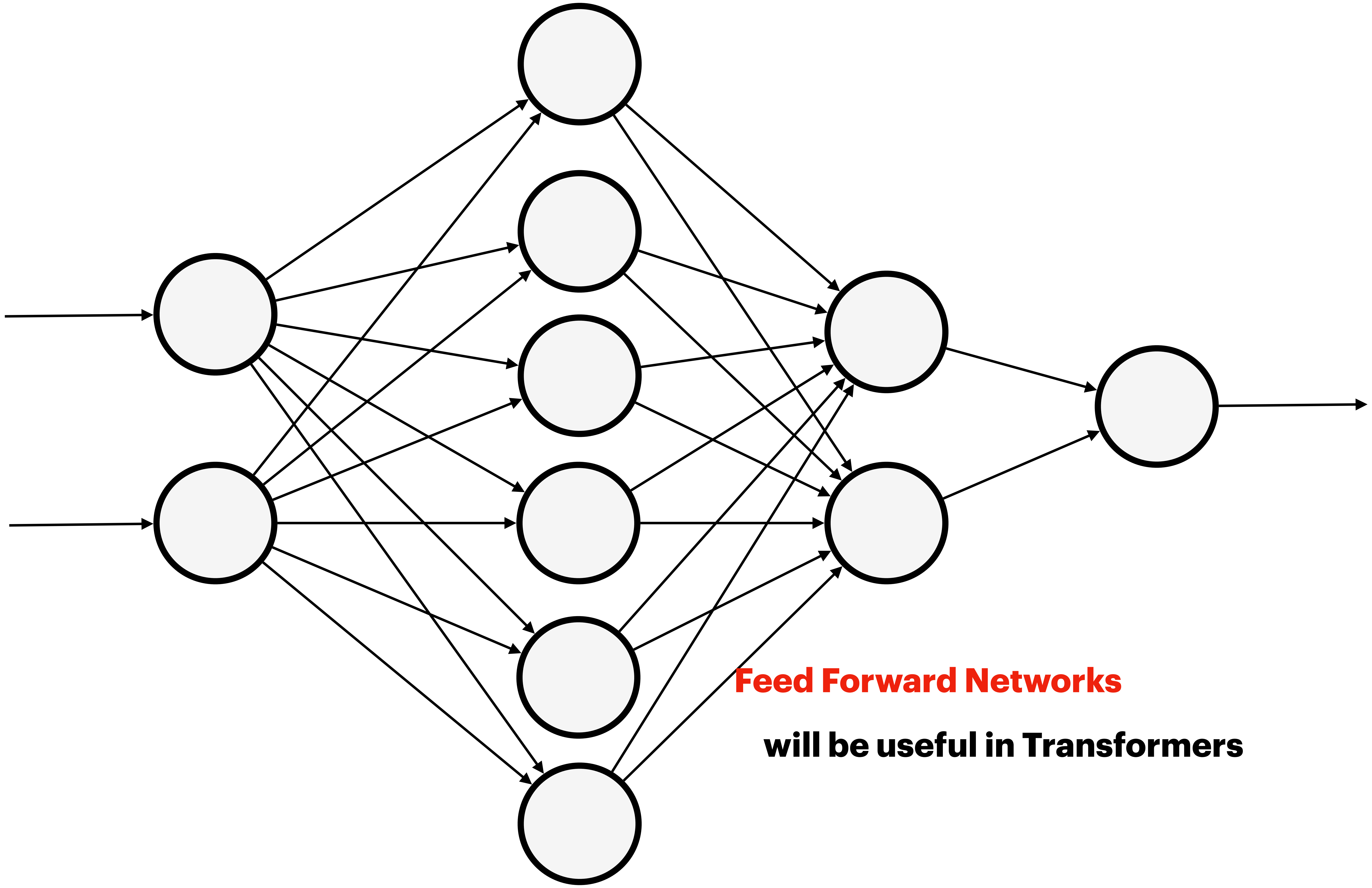
And that multiple neurons form a Neural Net!

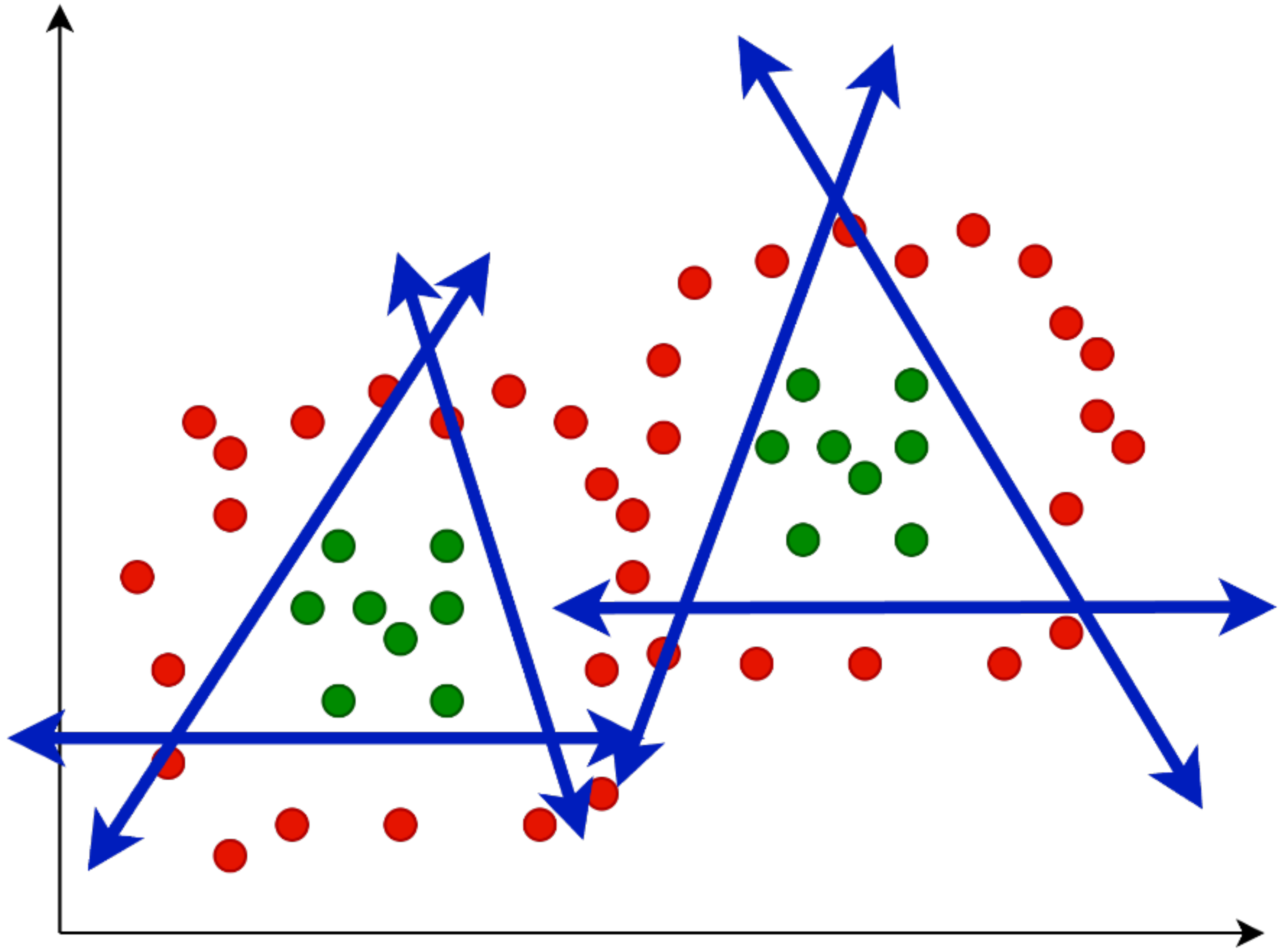




Neural Net

And that multiple neurons form a Neural Net!



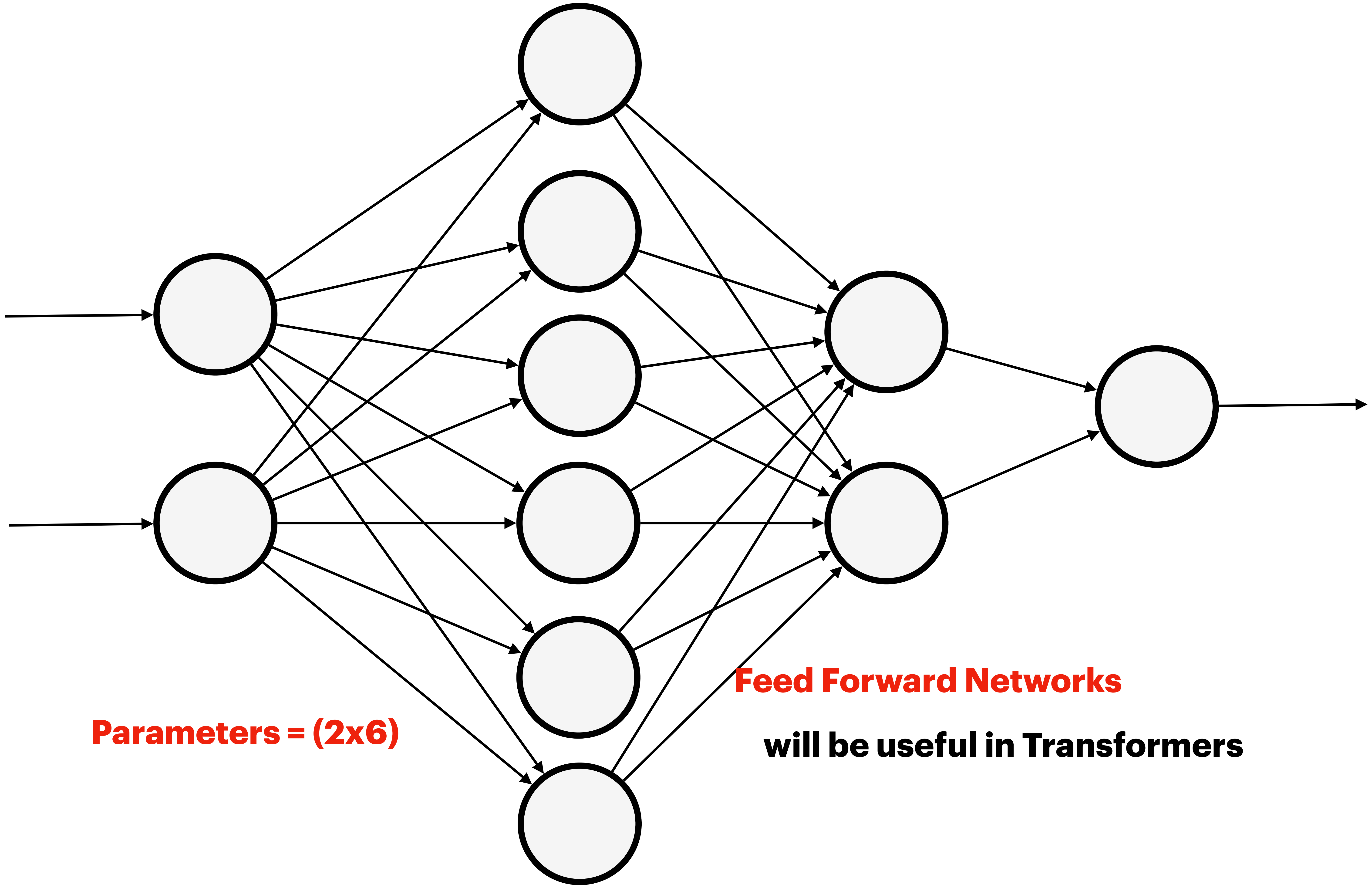


Feed Forward Networks

will be useful in Transformers

Neural Net

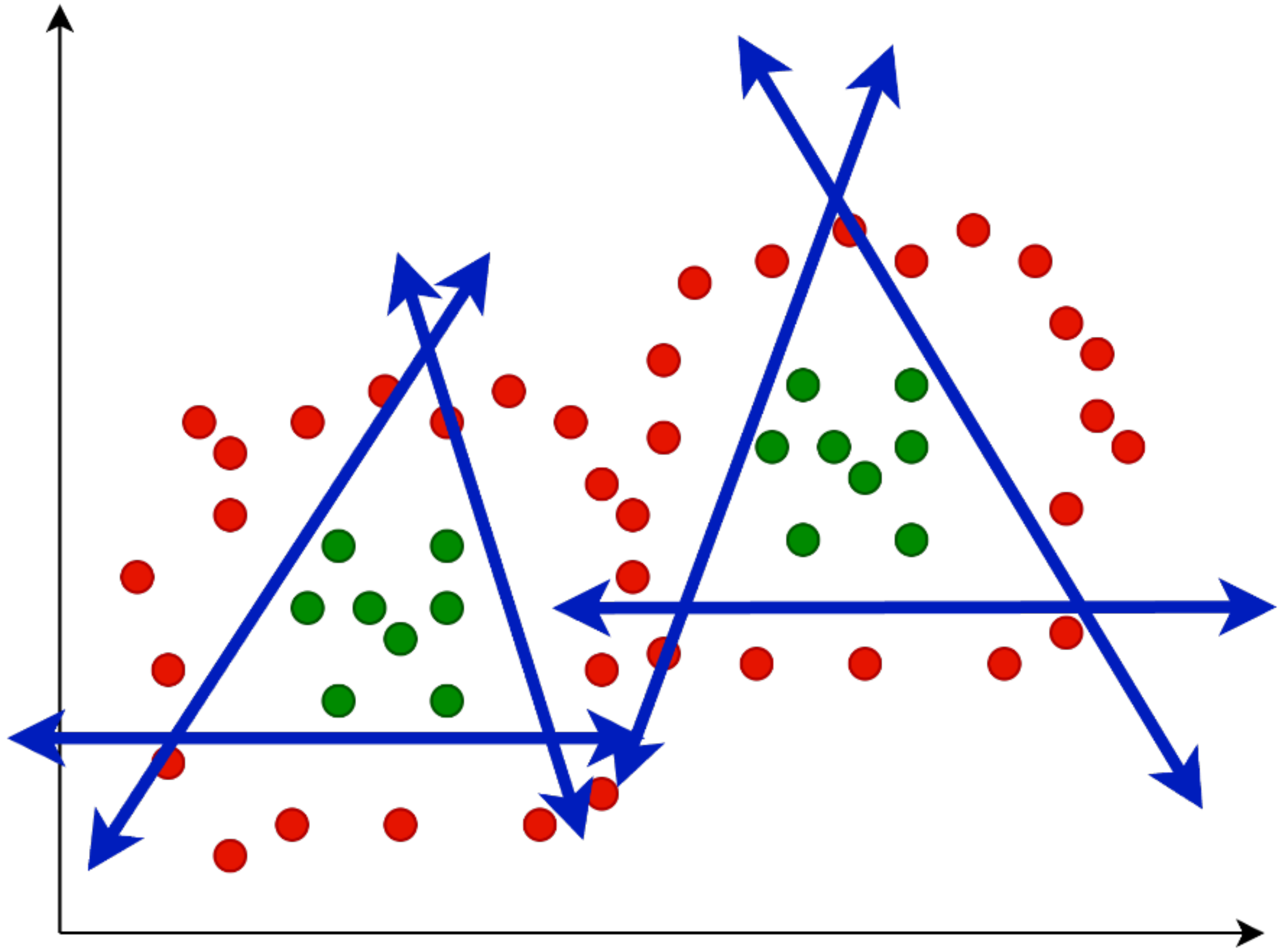
And that multiple neurons form a Neural Net!



Parameters = (2x6)

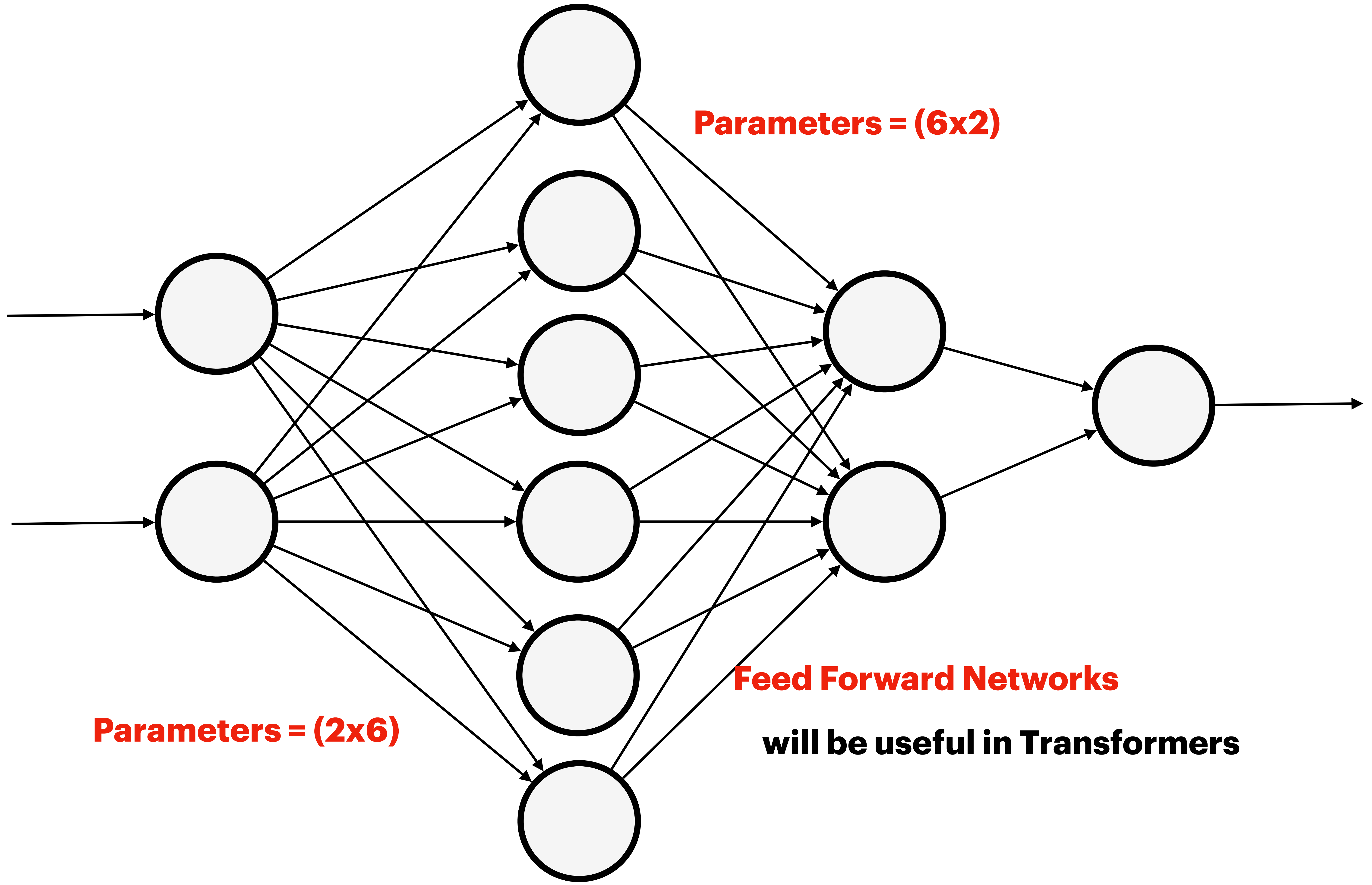
Feed Forward Networks

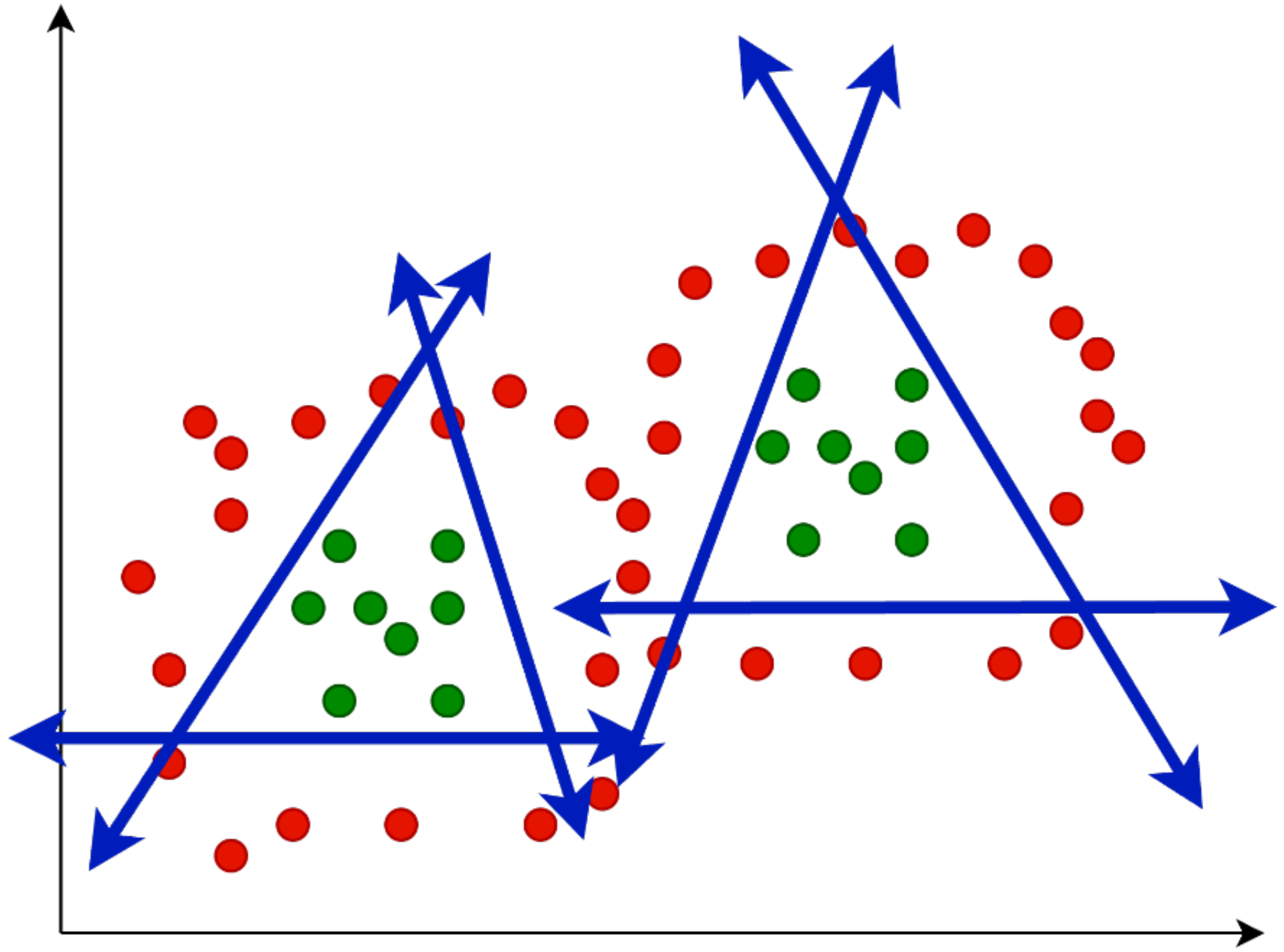
will be useful in Transformers



Neural Net

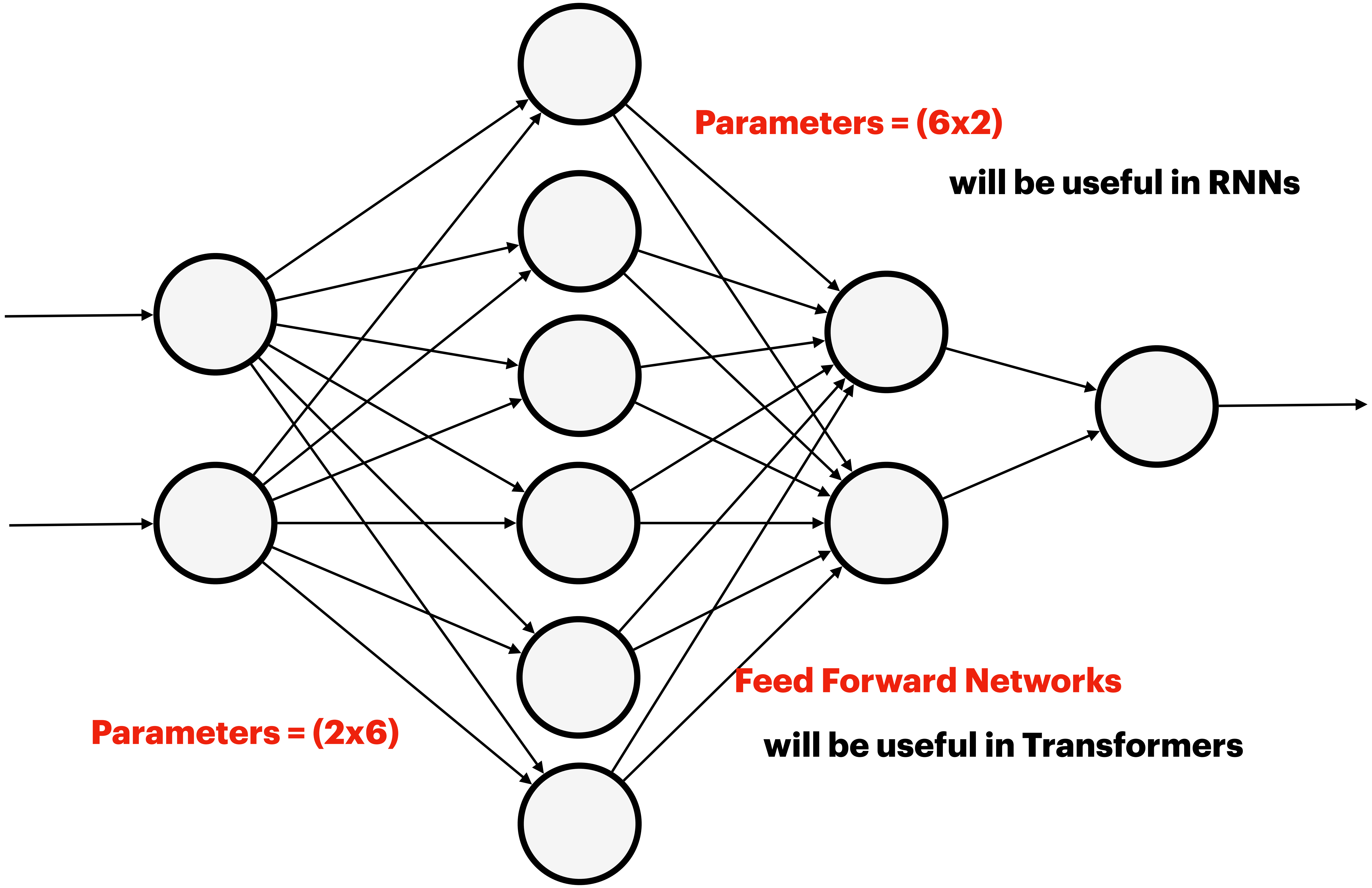
And that multiple neurons form a Neural Net!

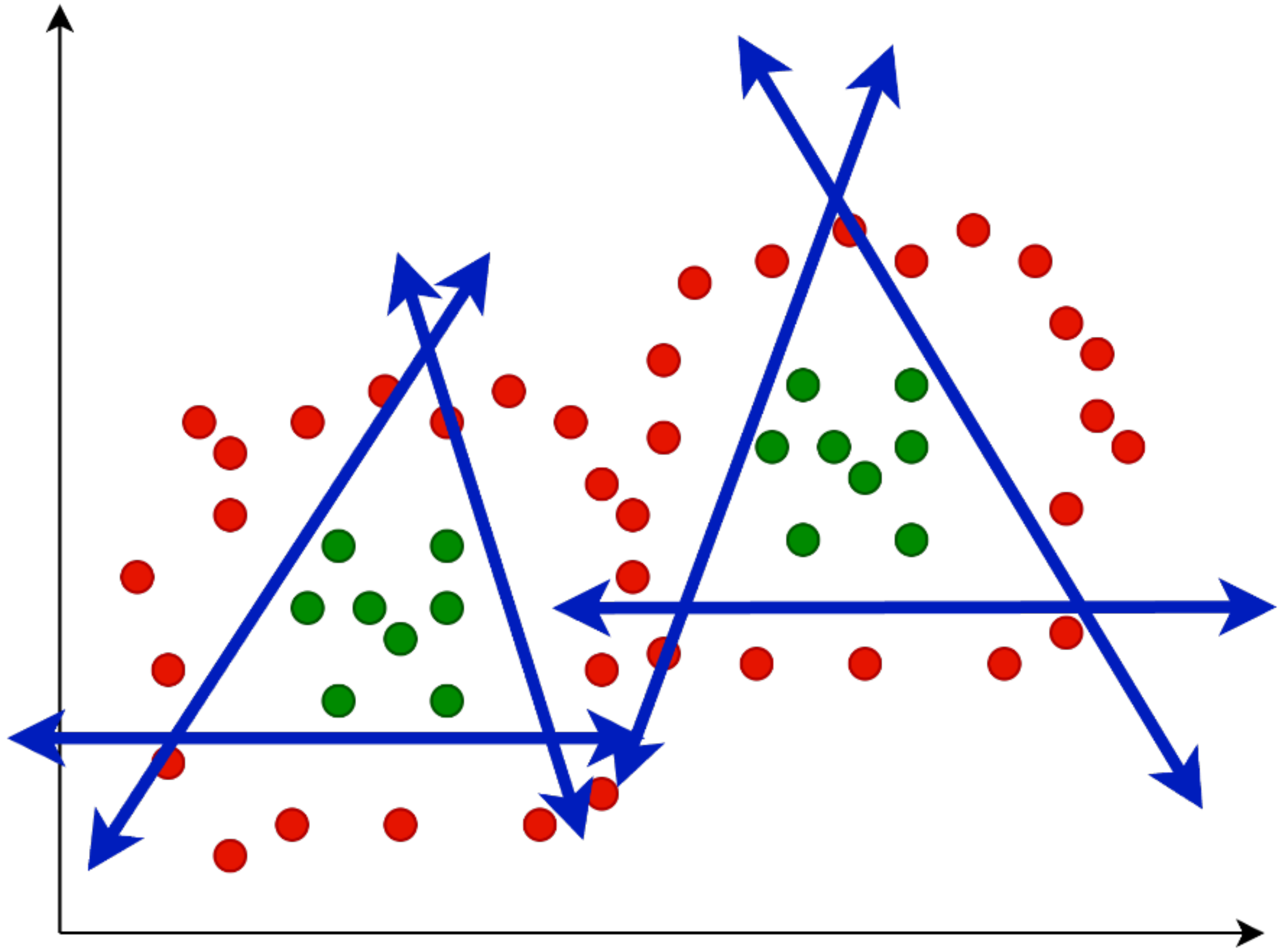




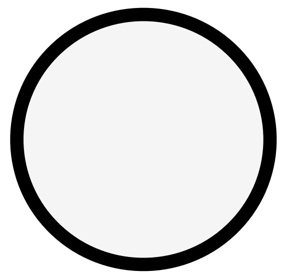
Neural Net

And that multiple neurons form a Neural Net!

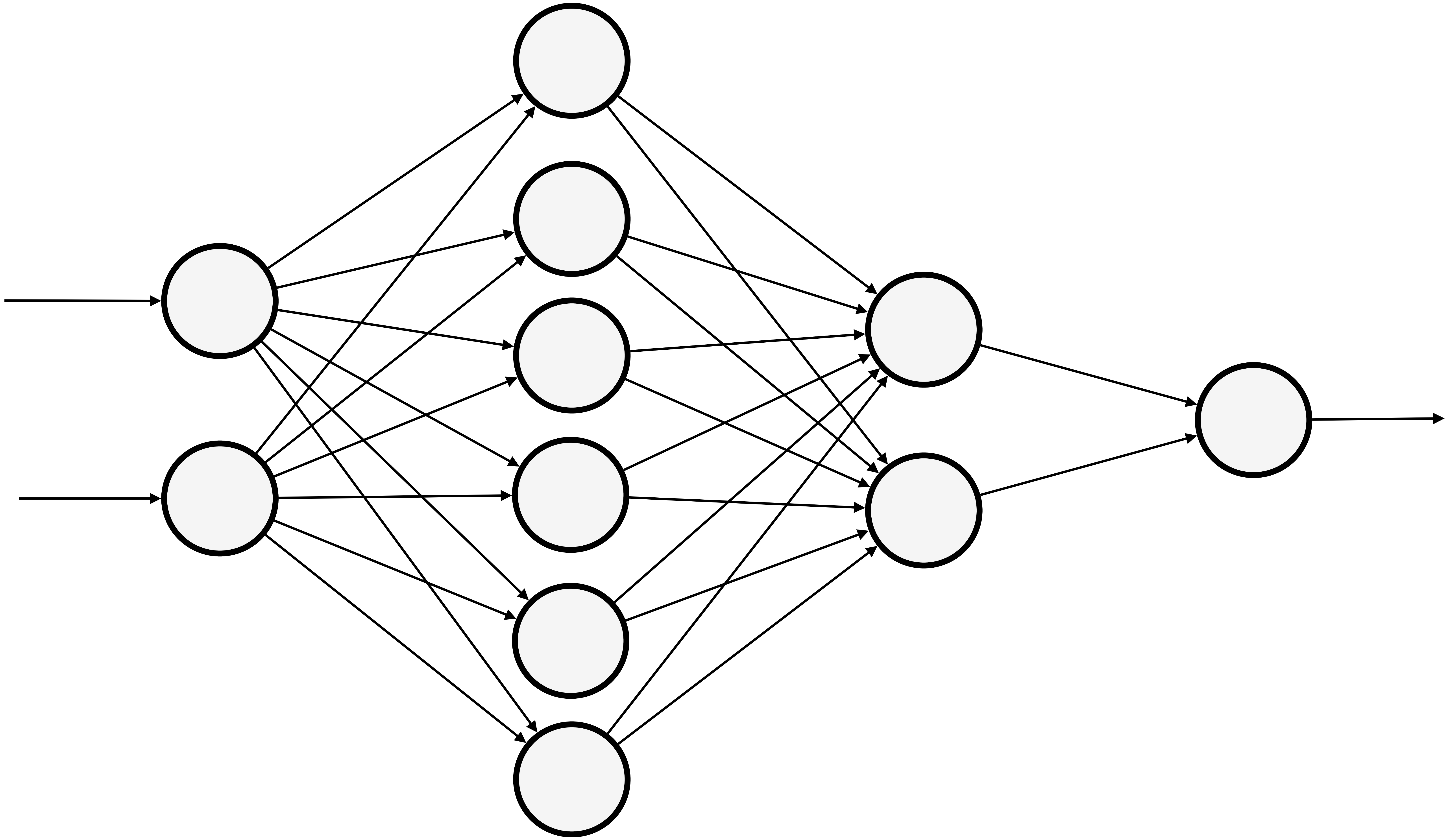




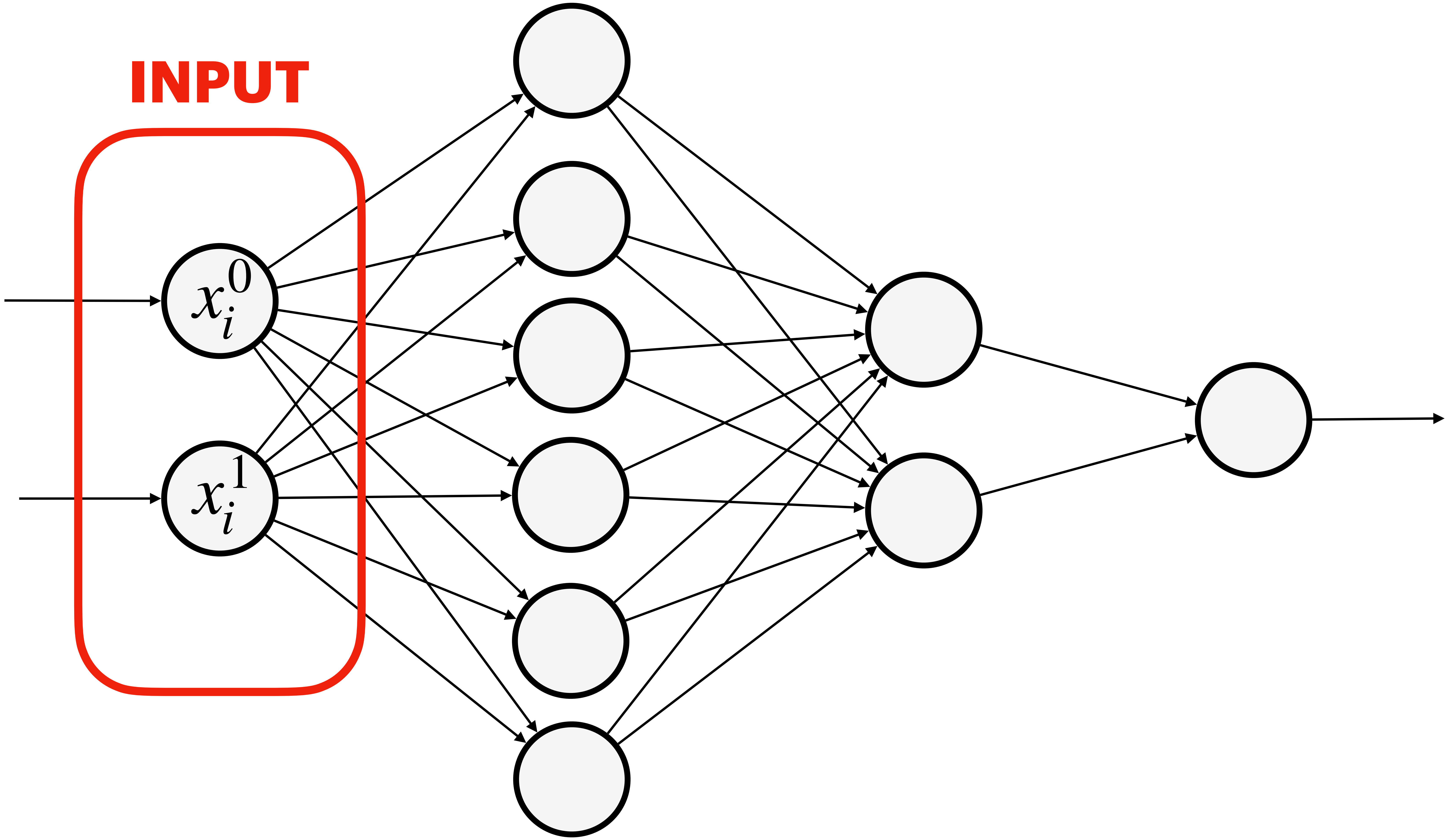
High Level Structure



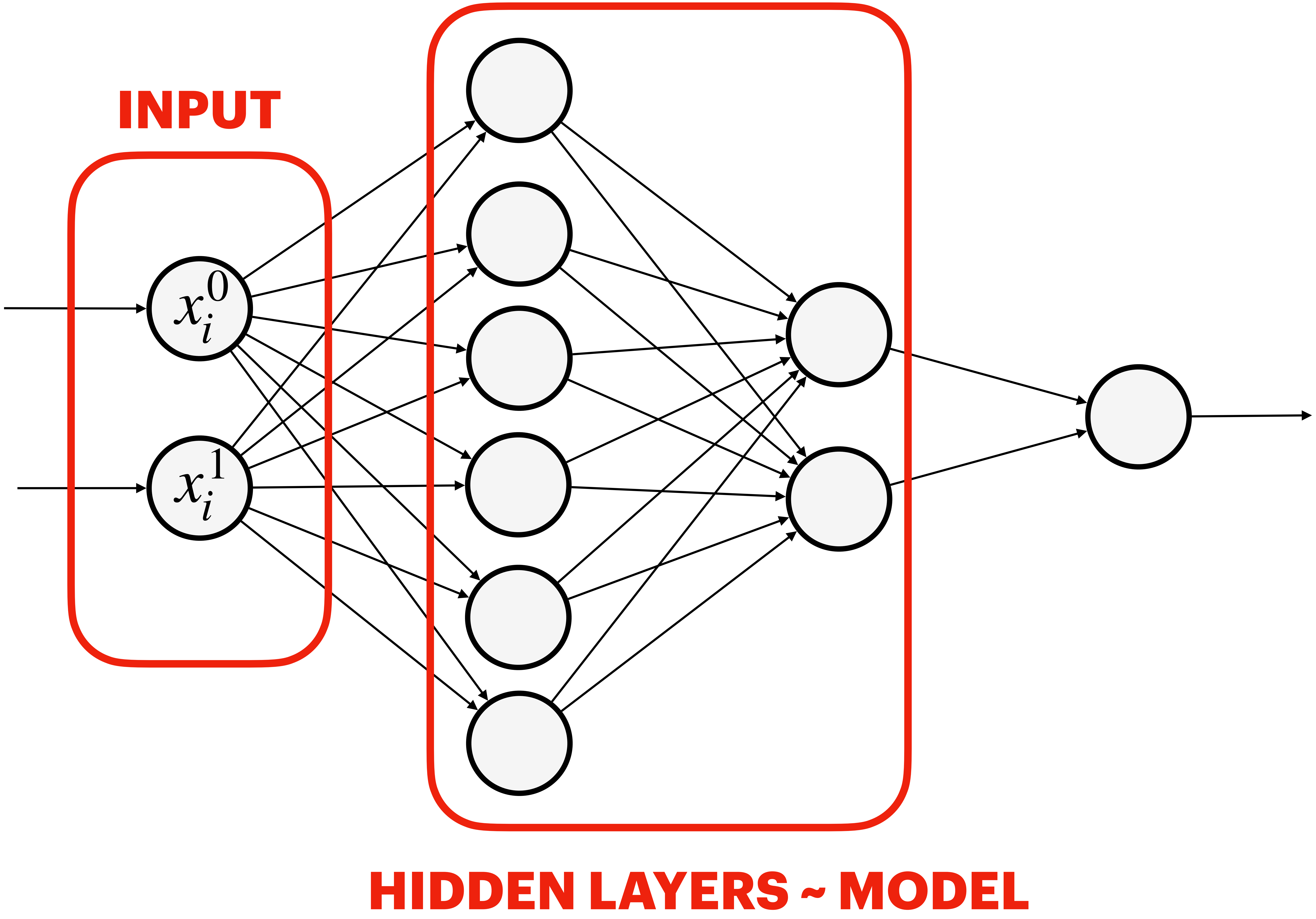
High Level Structure



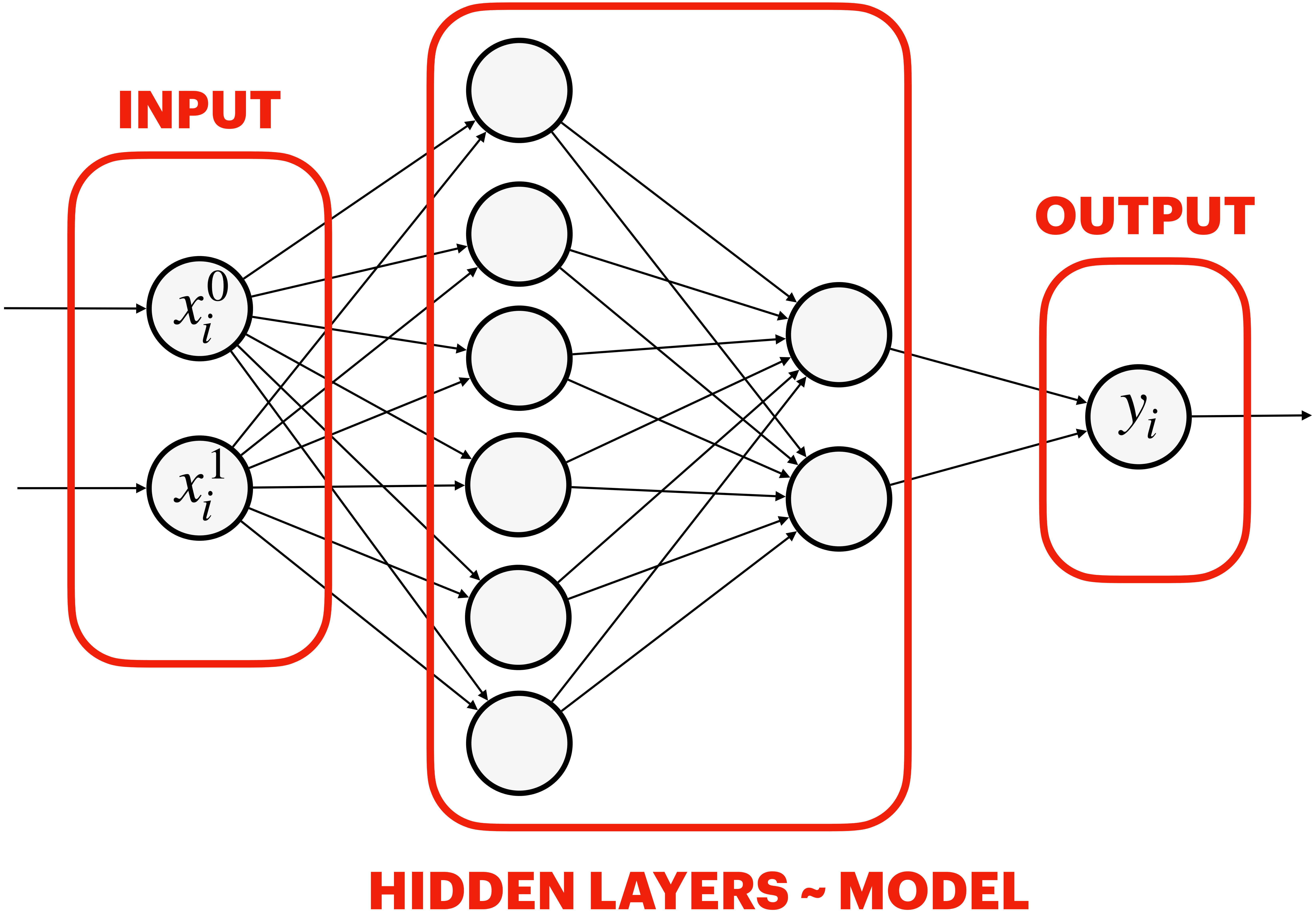
High Level Structure



High Level Structure



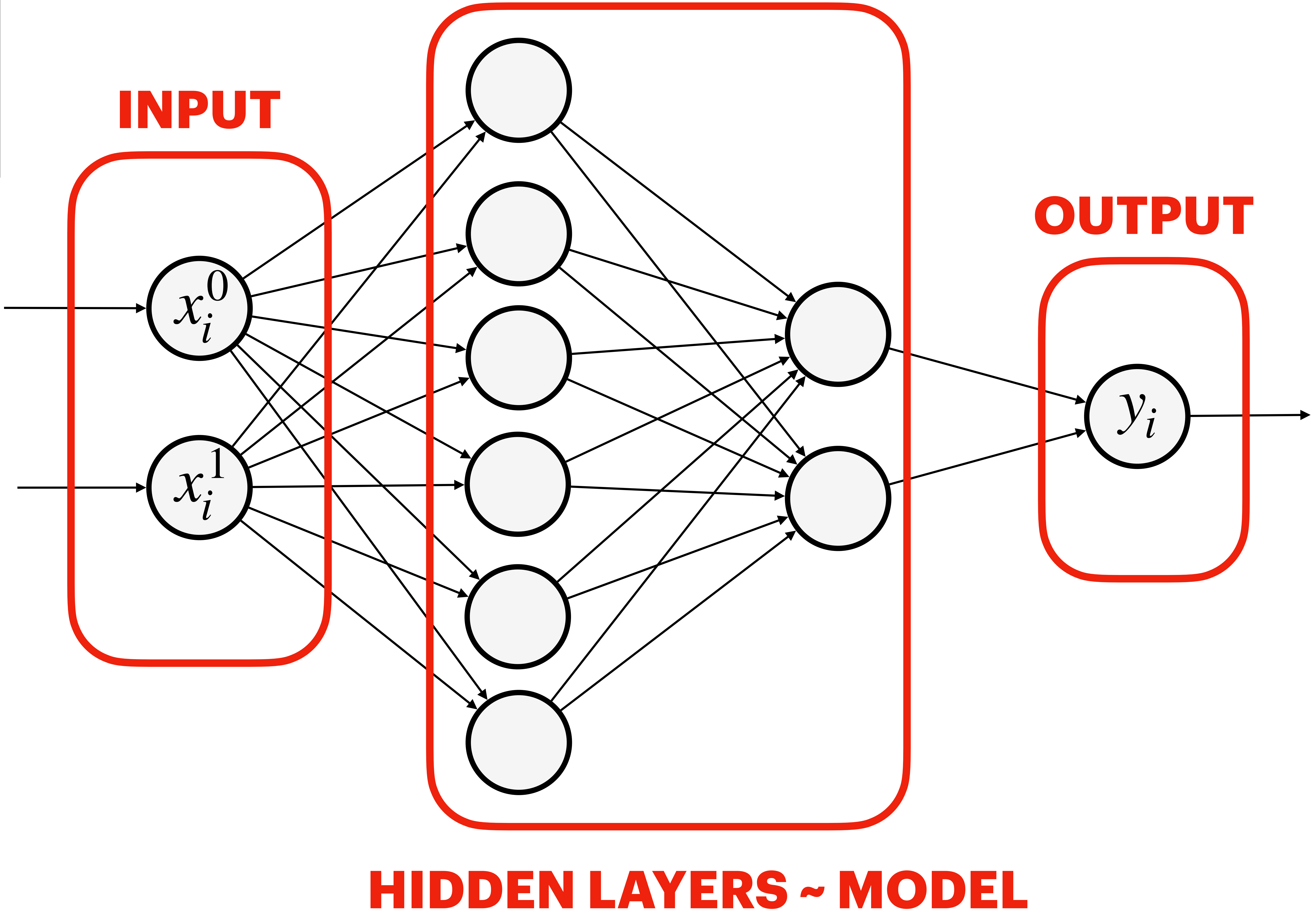
High Level Structure



High Level Structure

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT



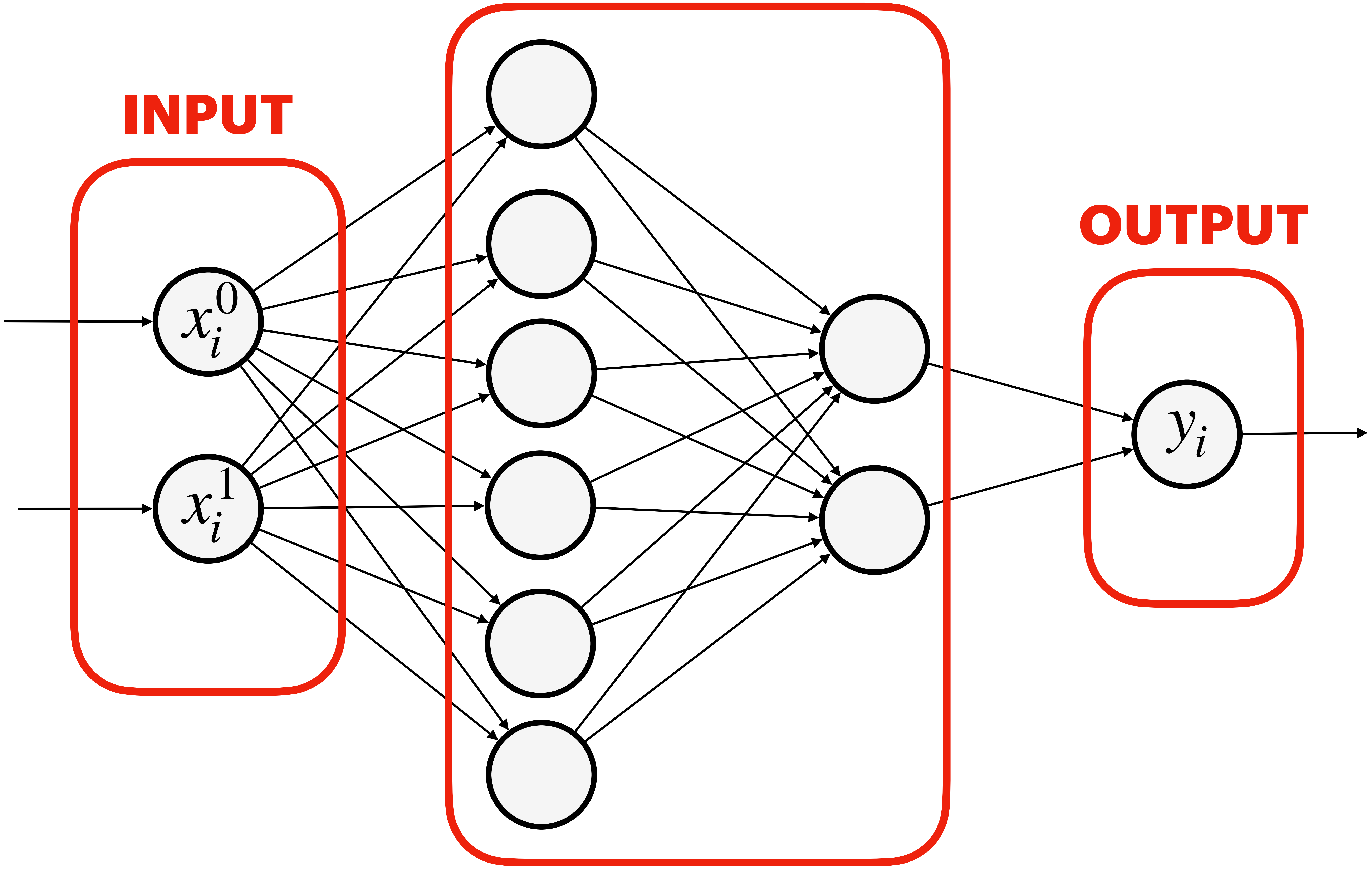
High Level Structure

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT

IMAGE

INPUT



OUTPUT

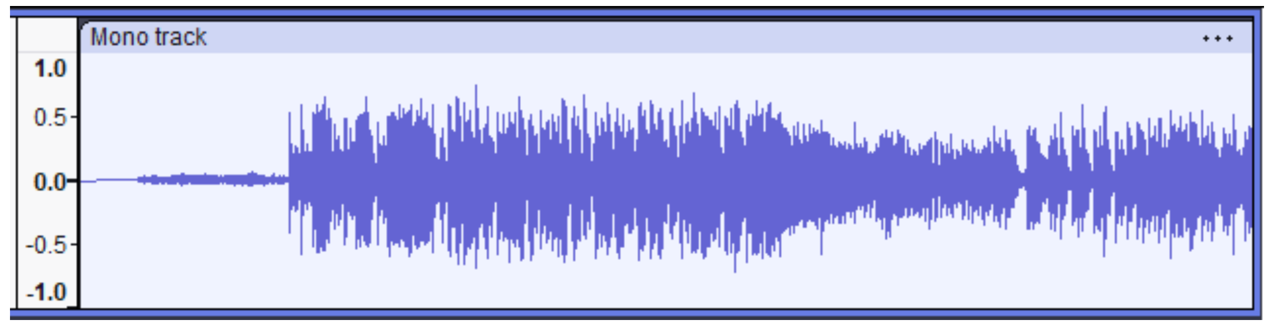
HIDDEN LAYERS ~ MODEL

High Level Structure

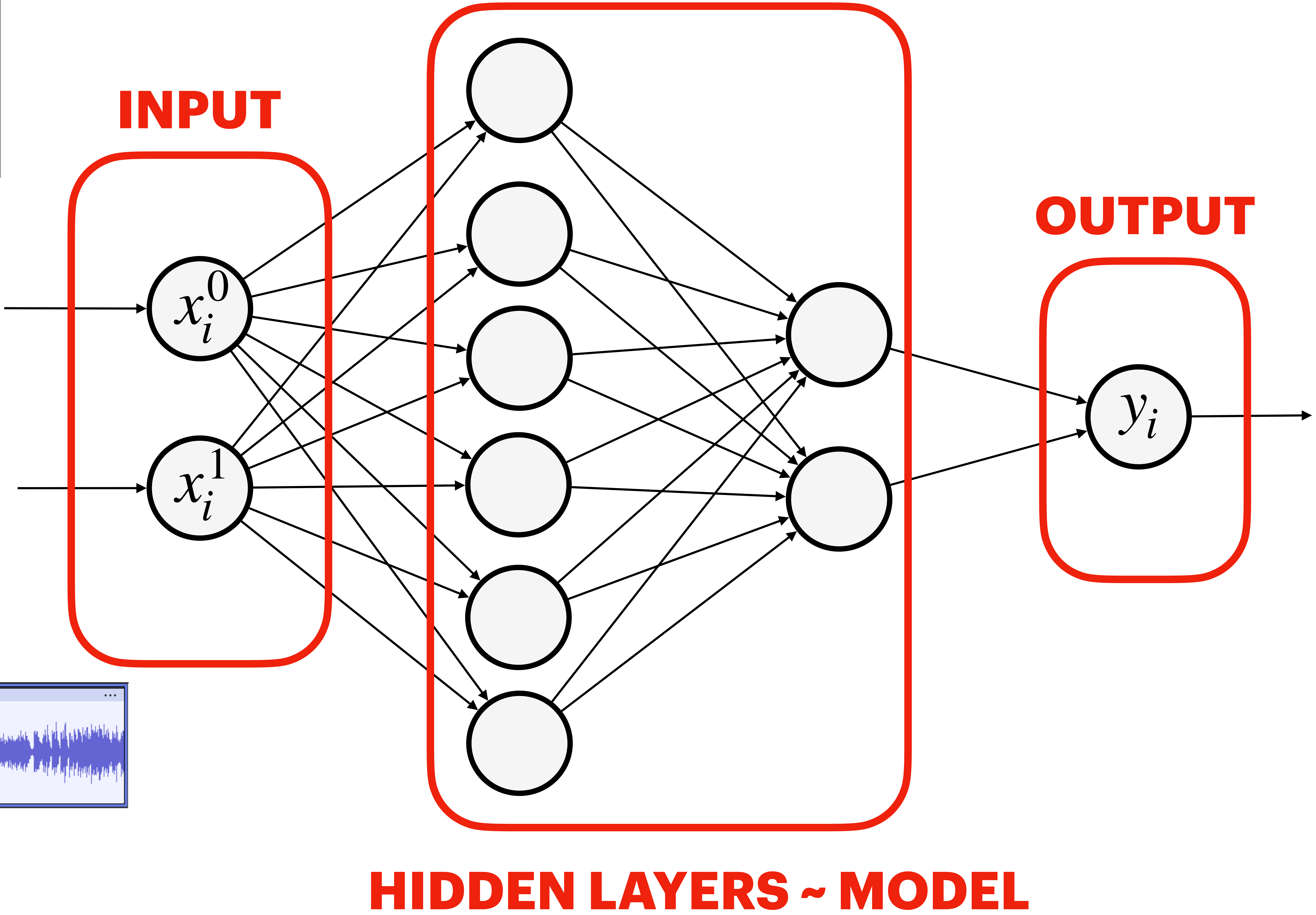
Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT

IMAGE



AUDIO

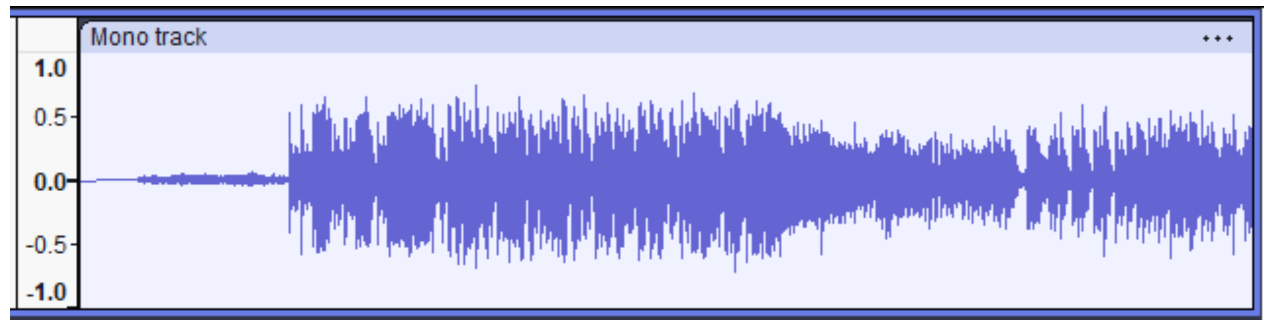


High Level Structure

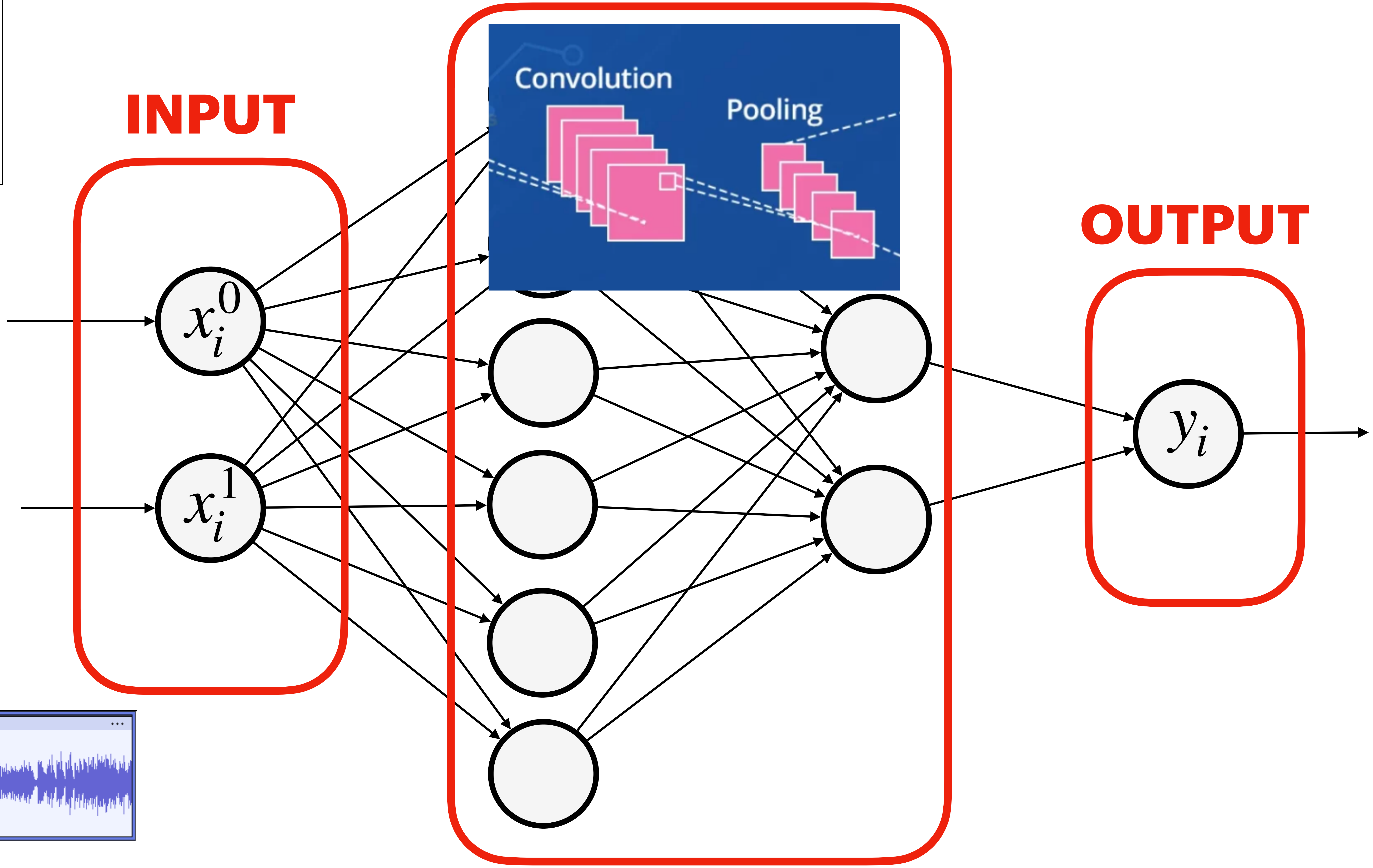
Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT

IMAGE

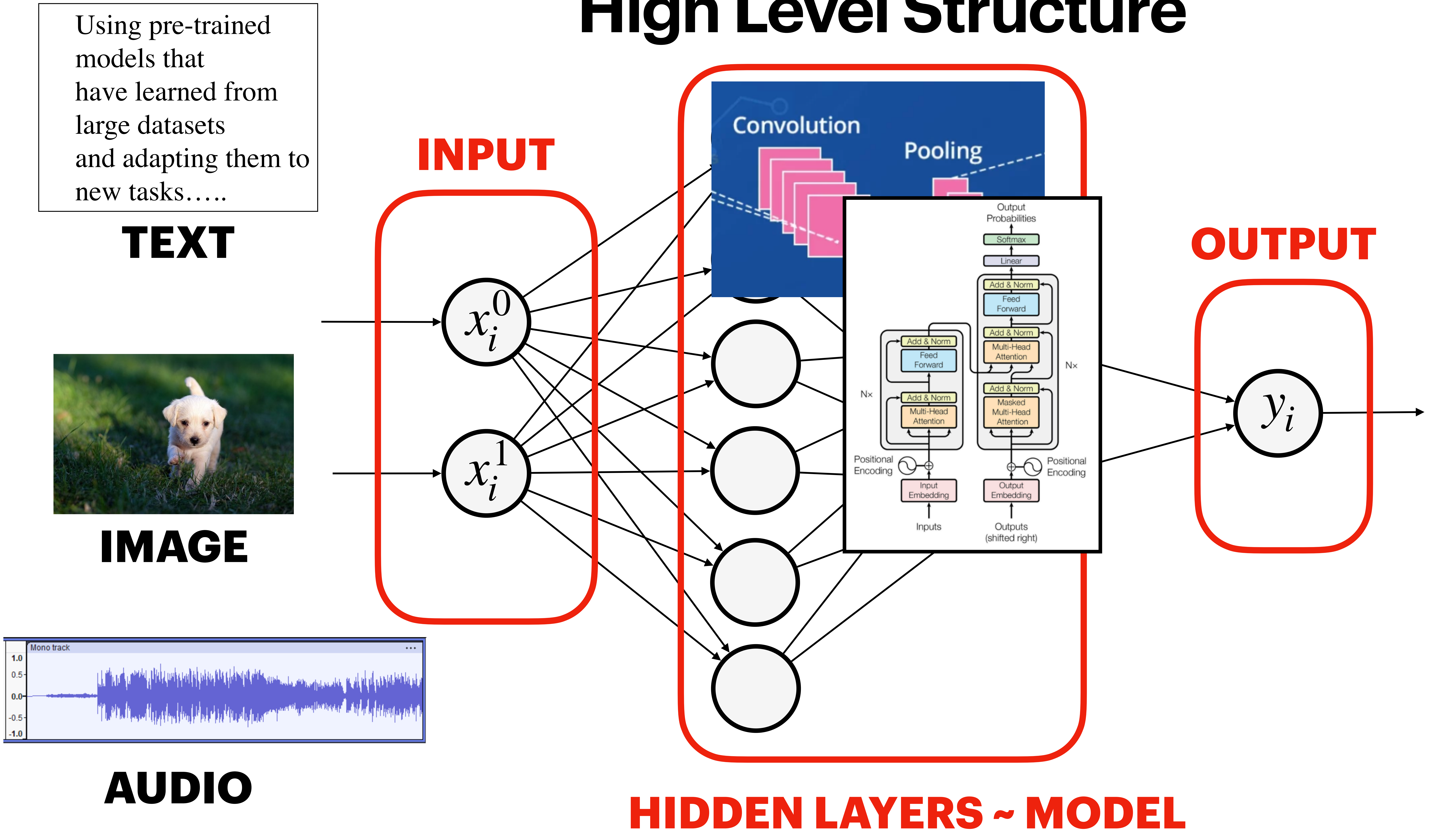


AUDIO



HIDDEN LAYERS ~ MODEL

High Level Structure

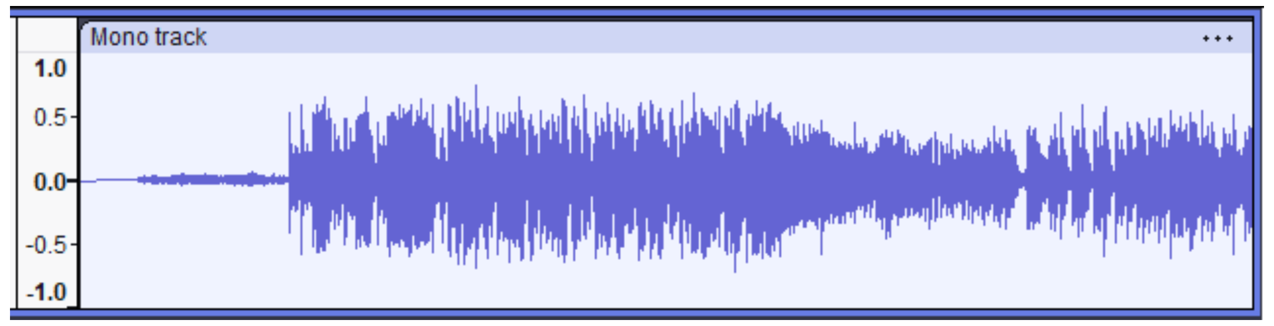


High Level Structure

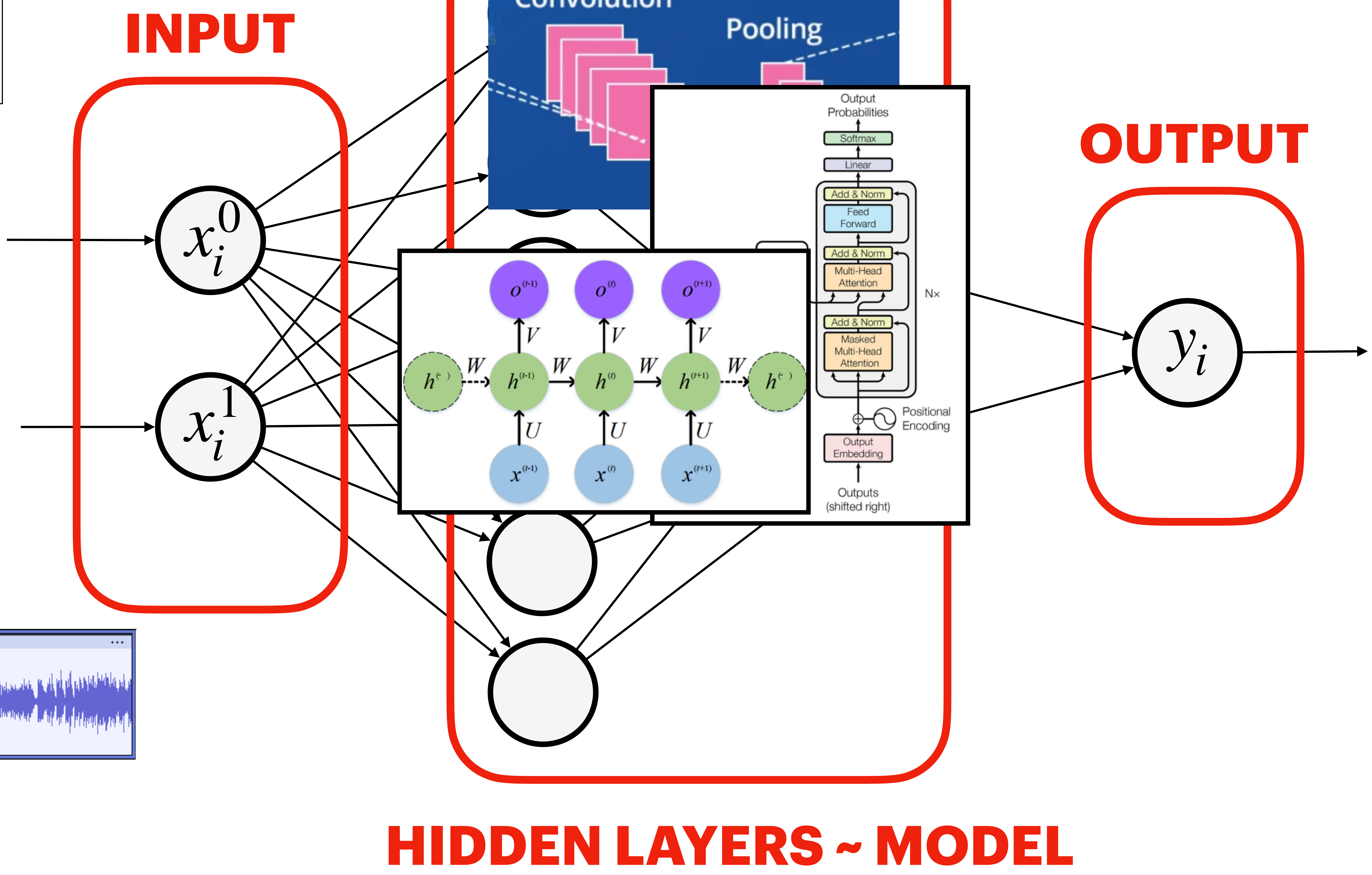
Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT

IMAGE

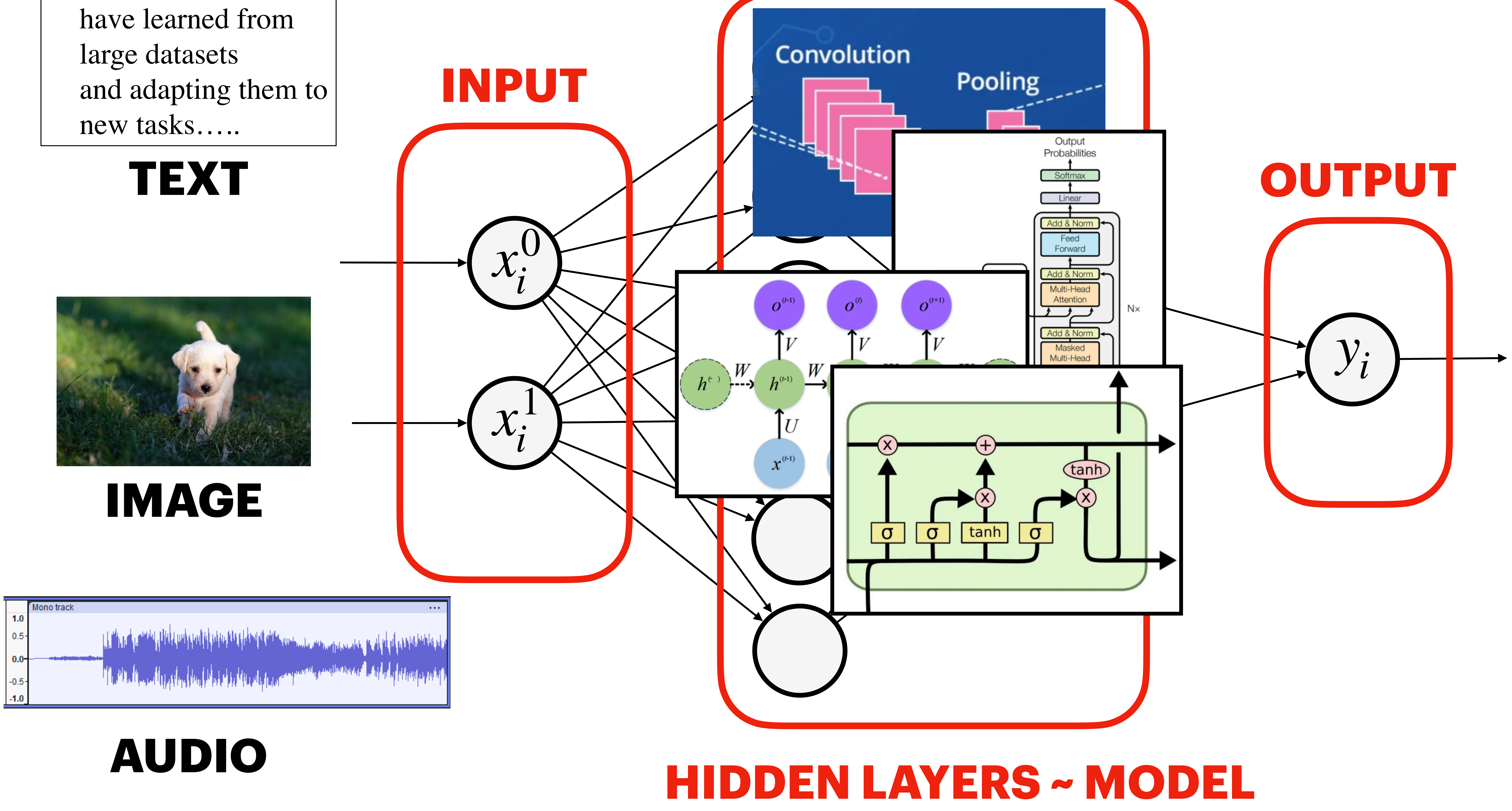


AUDIO



High Level Structure

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

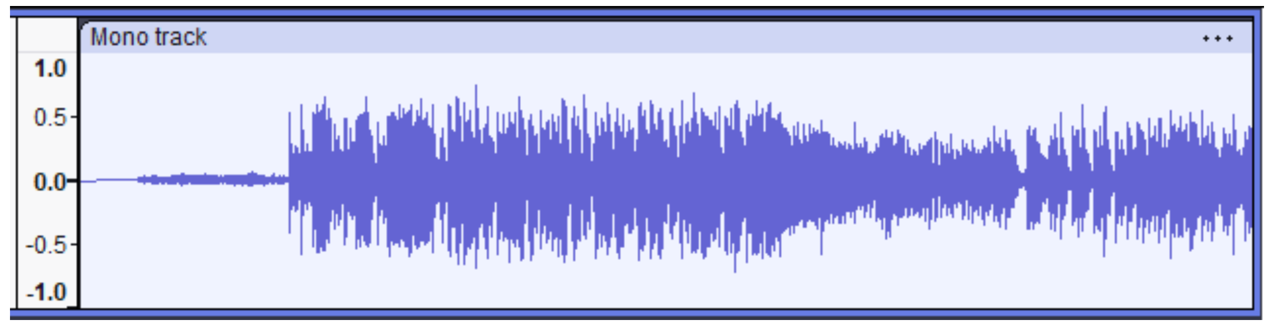


High Level Structure

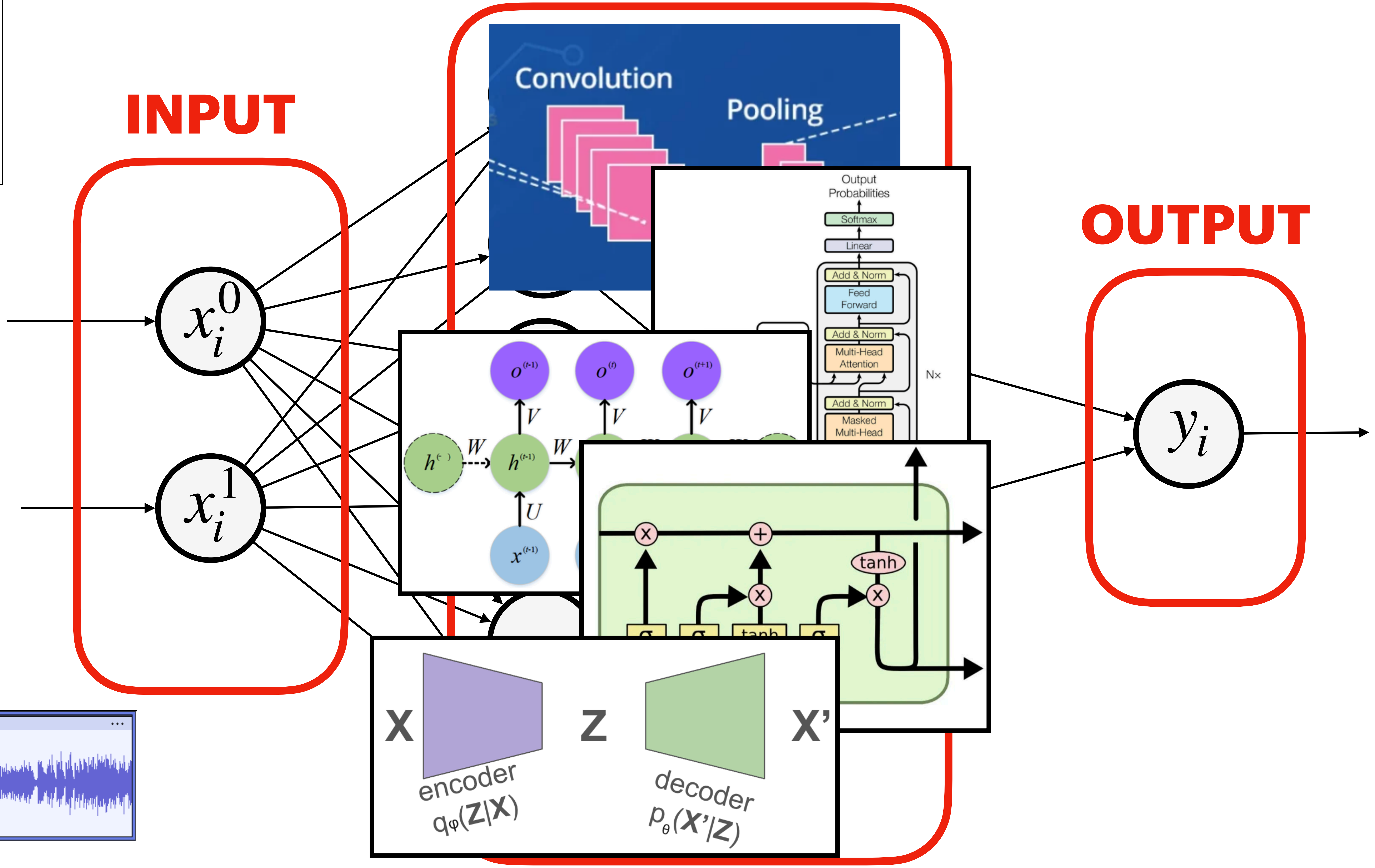
Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

TEXT

IMAGE

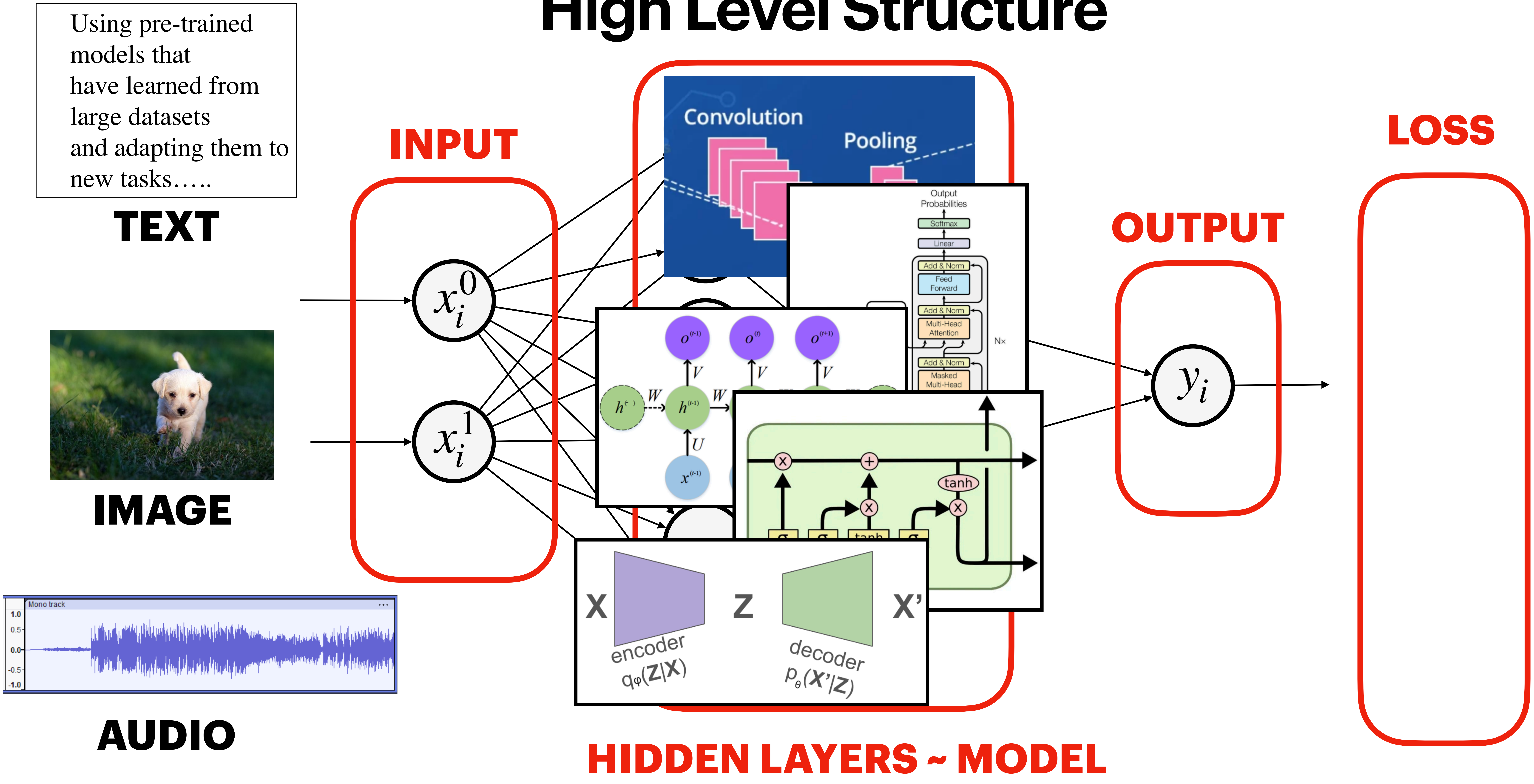


AUDIO

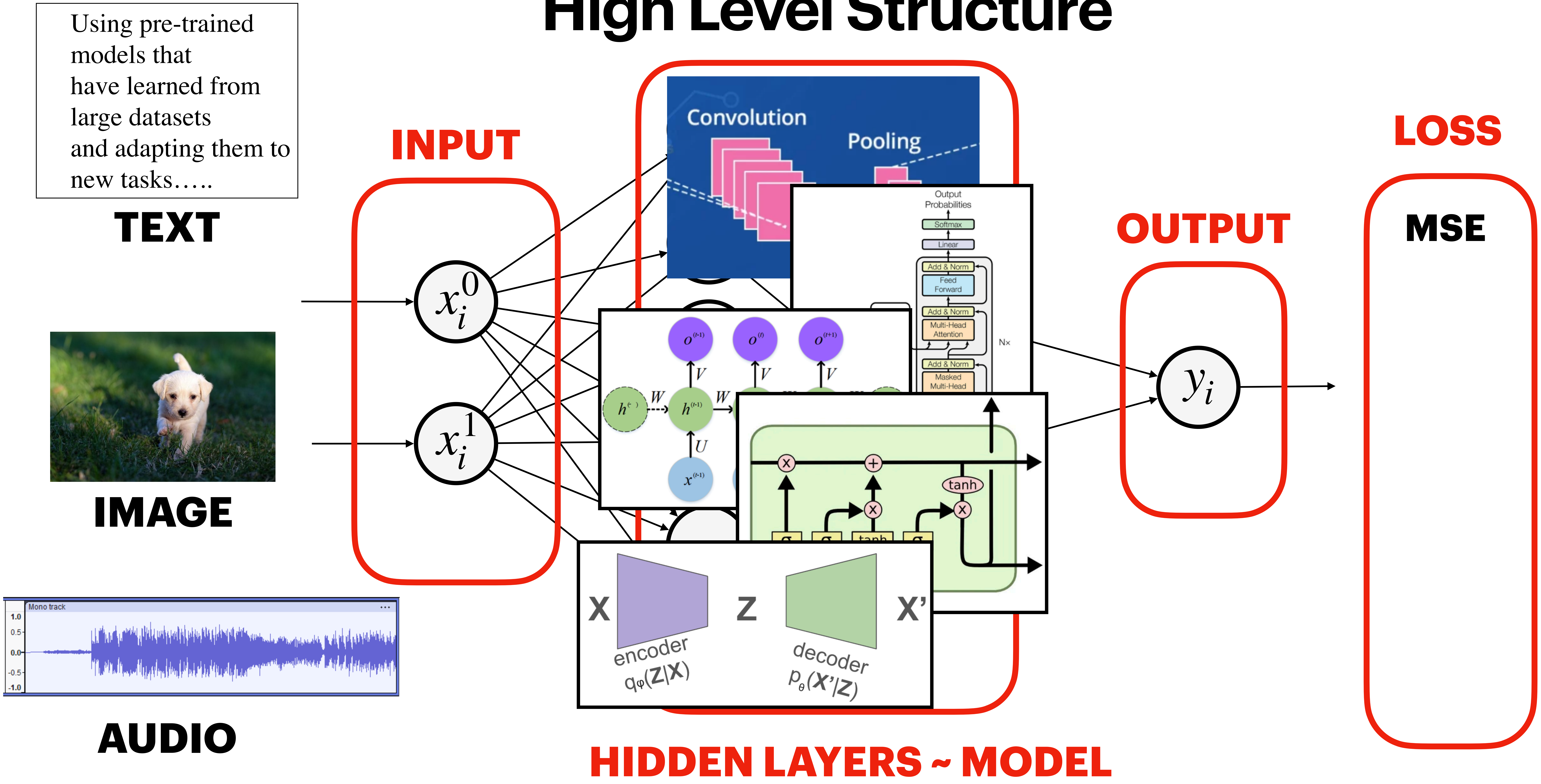


HIDDEN LAYERS ~ MODEL

High Level Structure



High Level Structure



High Level Structure

High Level Structure

High Level Structure

High Level Structure

High Level Structure

High Level Structure

Plan

- Cover each section at a high level
- Deep dive into the models

Fundamentals

LOSS FUNCTIONS

Loss Functions*

Loss Functions*

measure distance from ideal distribution/

measure how unlikely your data us

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

LOWER IS BETTER

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

LOWER IS BETTER

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

- Regression setting - Mean Squared Error

LOWER IS BETTER

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

- Regression setting - Mean Squared Error
- Classification setting - Cross Entropy

LOWER IS BETTER

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

- Regression setting - Mean Squared Error
- Classification setting - Cross Entropy

LOWER IS BETTER

*Other loss functions covered in second half

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

- Regression setting - Mean Squared Error ← **distance**
- Classification setting - Cross Entropy

LOWER IS BETTER

*Other loss functions covered in second half

Loss Functions*

measure distance from ideal distribution/
measure how unlikely your data us

- Regression setting - Mean Squared Error ← **distance**
- Classification setting - Cross Entropy ← **unlikelihood**

LOWER IS BETTER

*Other loss functions covered in second half

Loss Functions*

- Regression setting - Mean Squared Error
- Classification setting - **Cross Entropy** ← **unlikelihood**

Cross Entropy
beyond the formula

- You've been introduced to it as a formula

Add a negative sign to turn it into a loss, i.e. something to minimize:

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like **data likelihood** with a **negative sign**

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like **data likelihood** with a **negative sign**

output
softmax

y_0

y_1

y_2

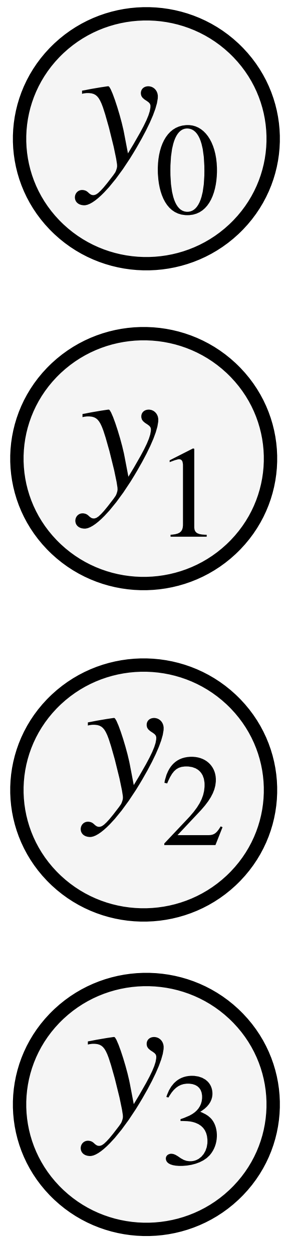
y_3

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like **data likelihood** with a **negative sign**

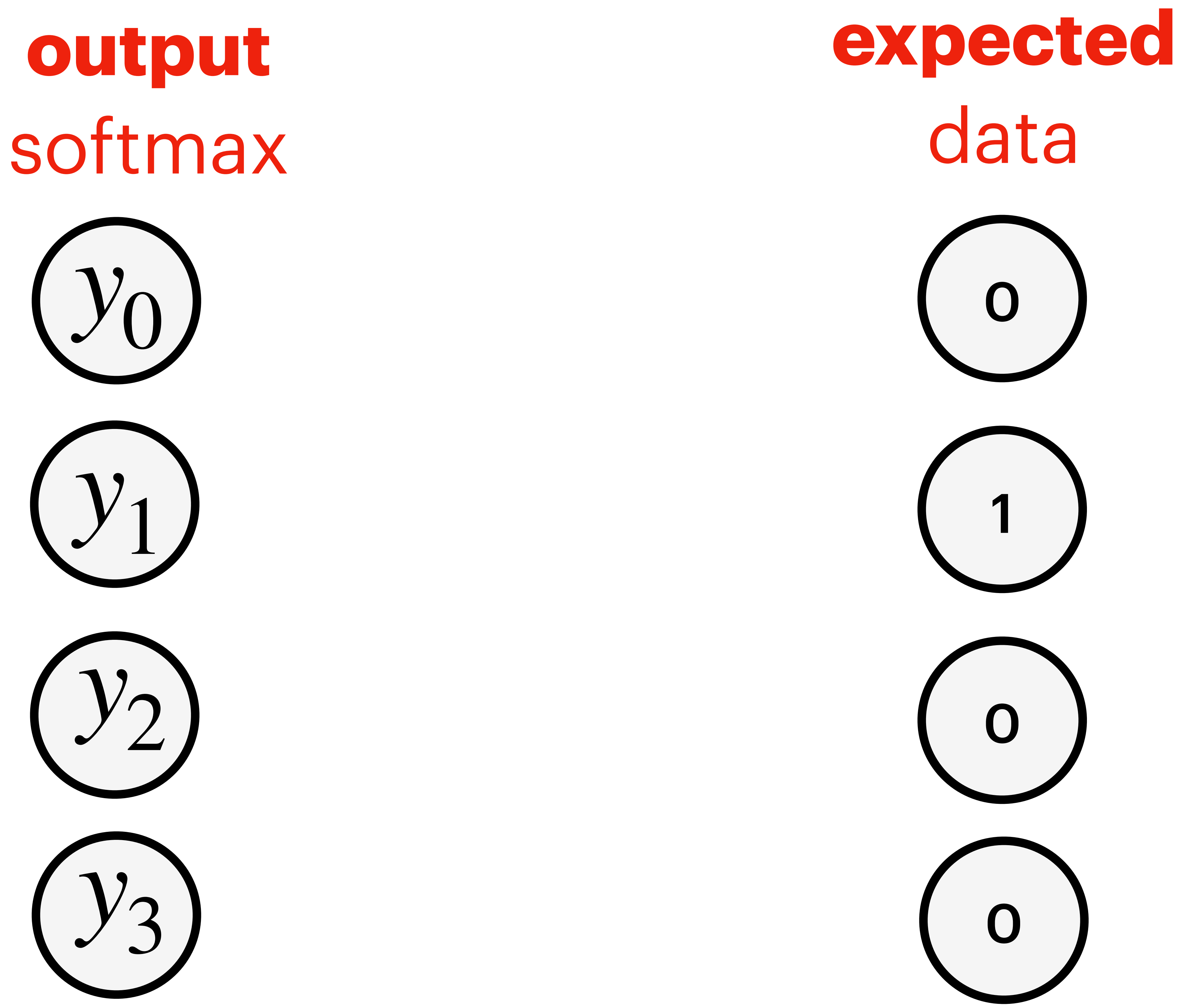
output
softmax



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

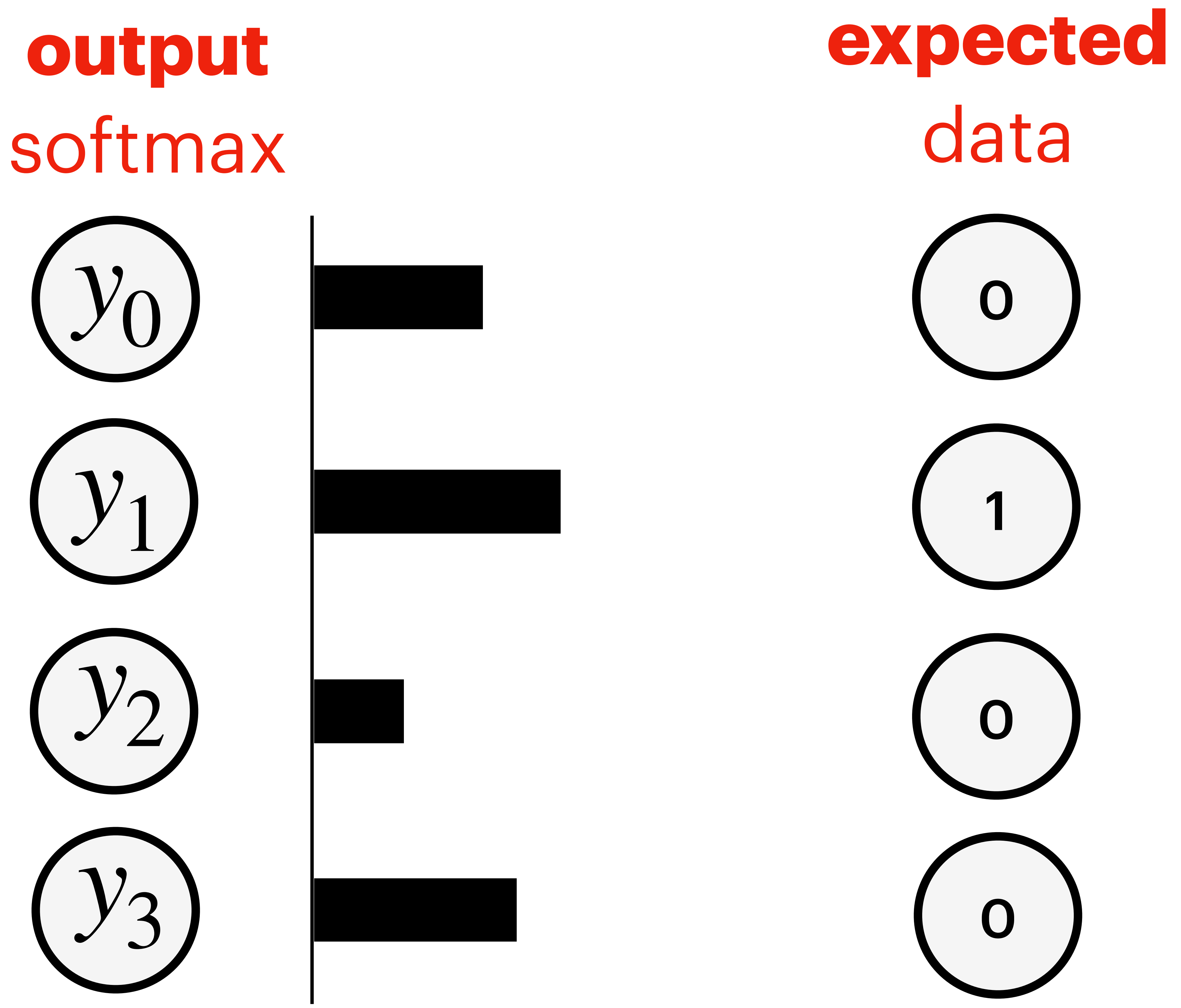
- But think of it like **data likelihood** with a **negative sign**



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

- But think of it like **data likelihood** with a **negative sign**



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

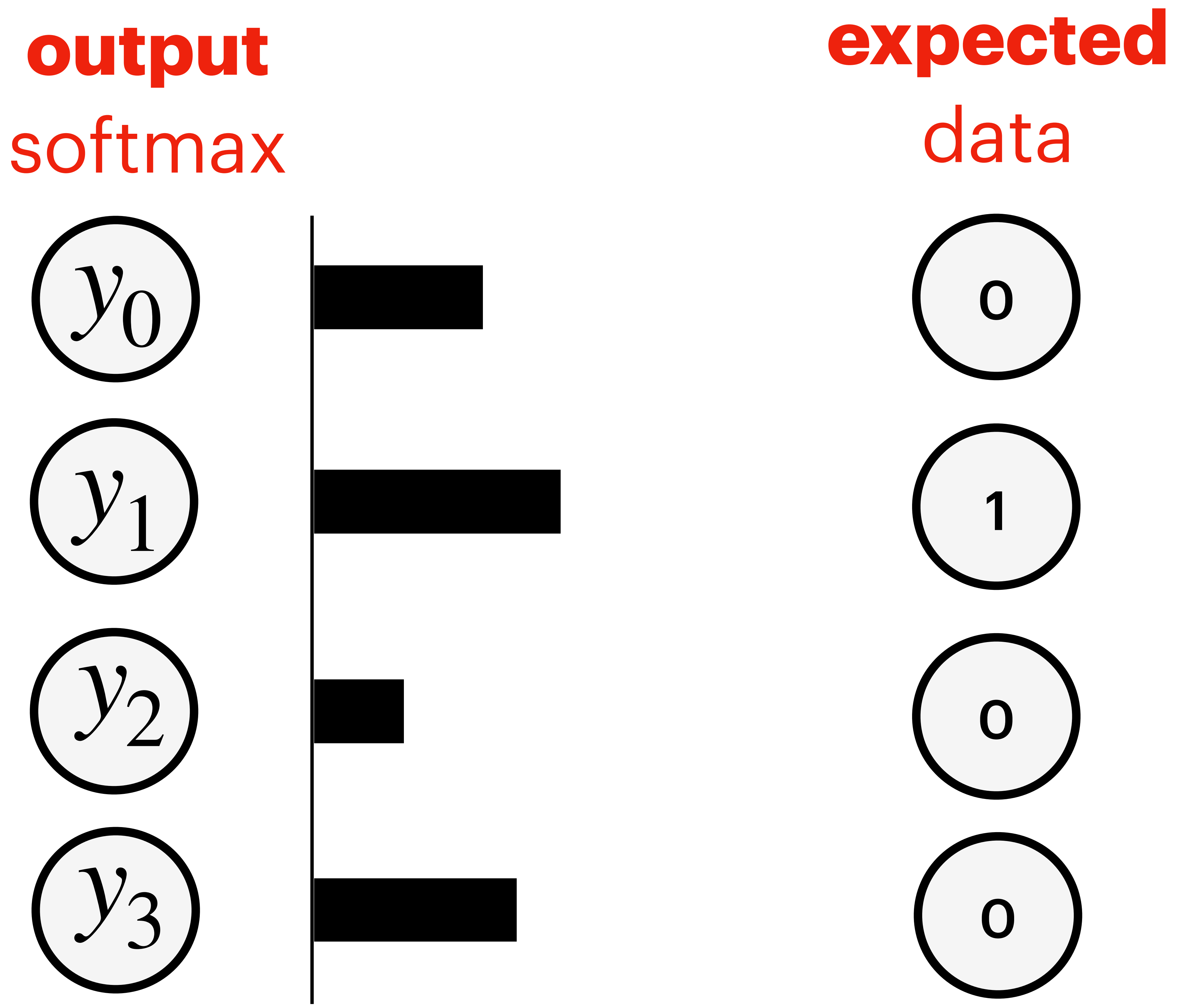
- But think of it like **data likelihood** with a **negative sign**

What is the probability of getting my expected data?

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i)]$$

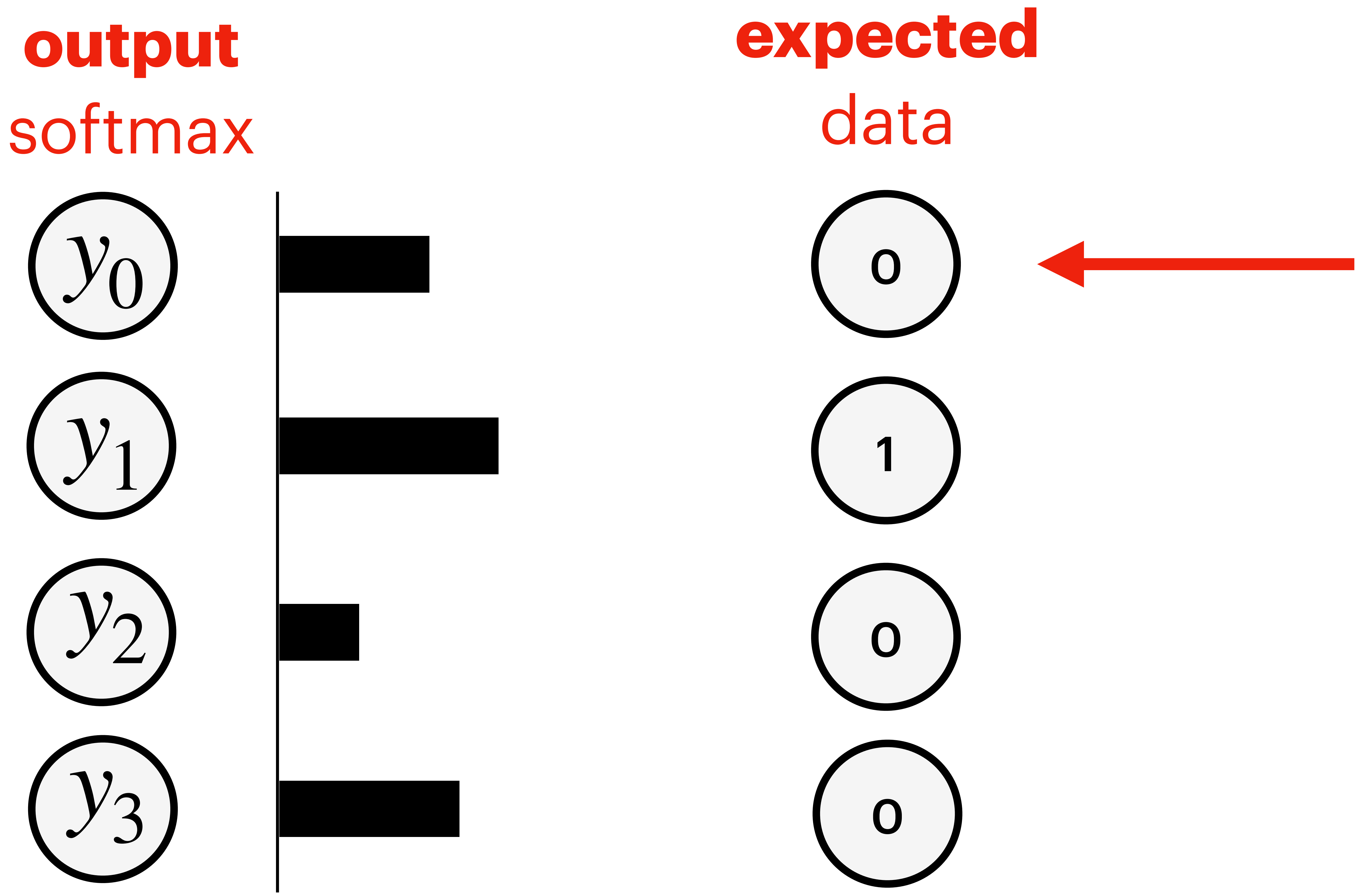
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like data likelihood with a negative sign



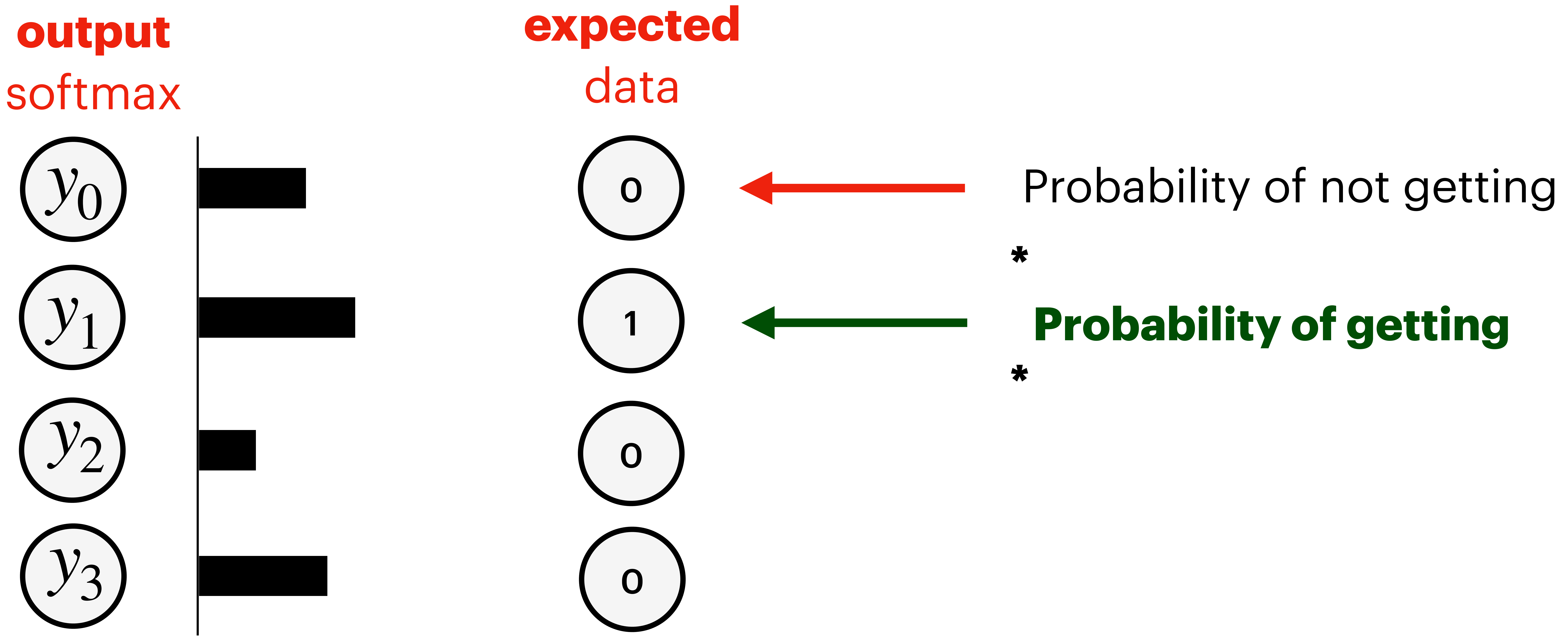
Probability of not getting

*

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

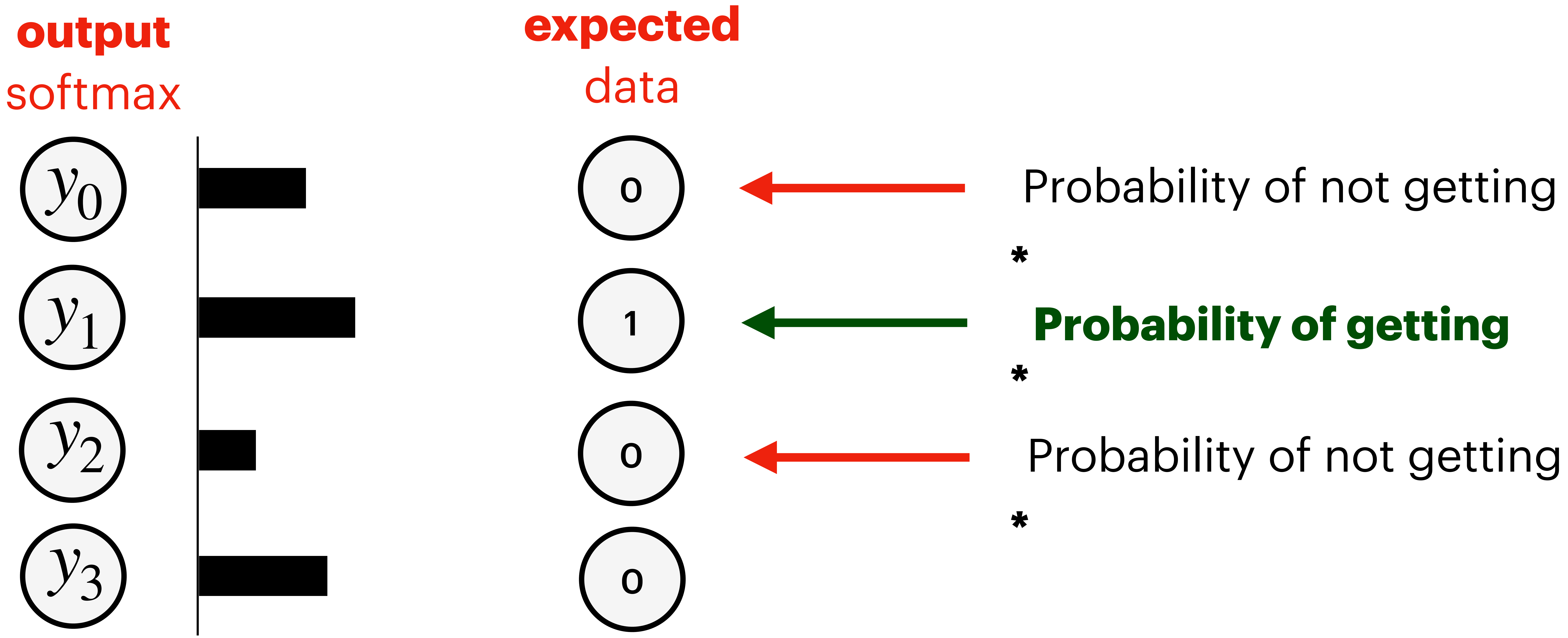
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

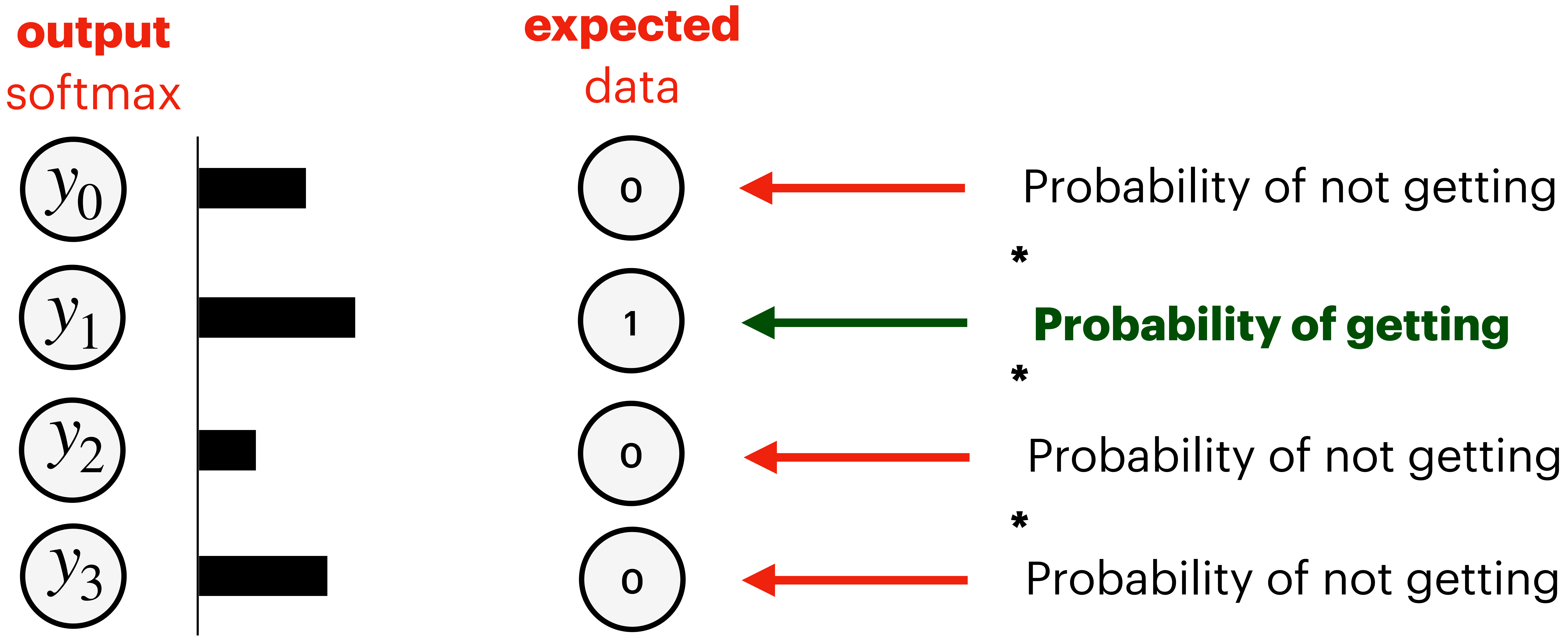
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

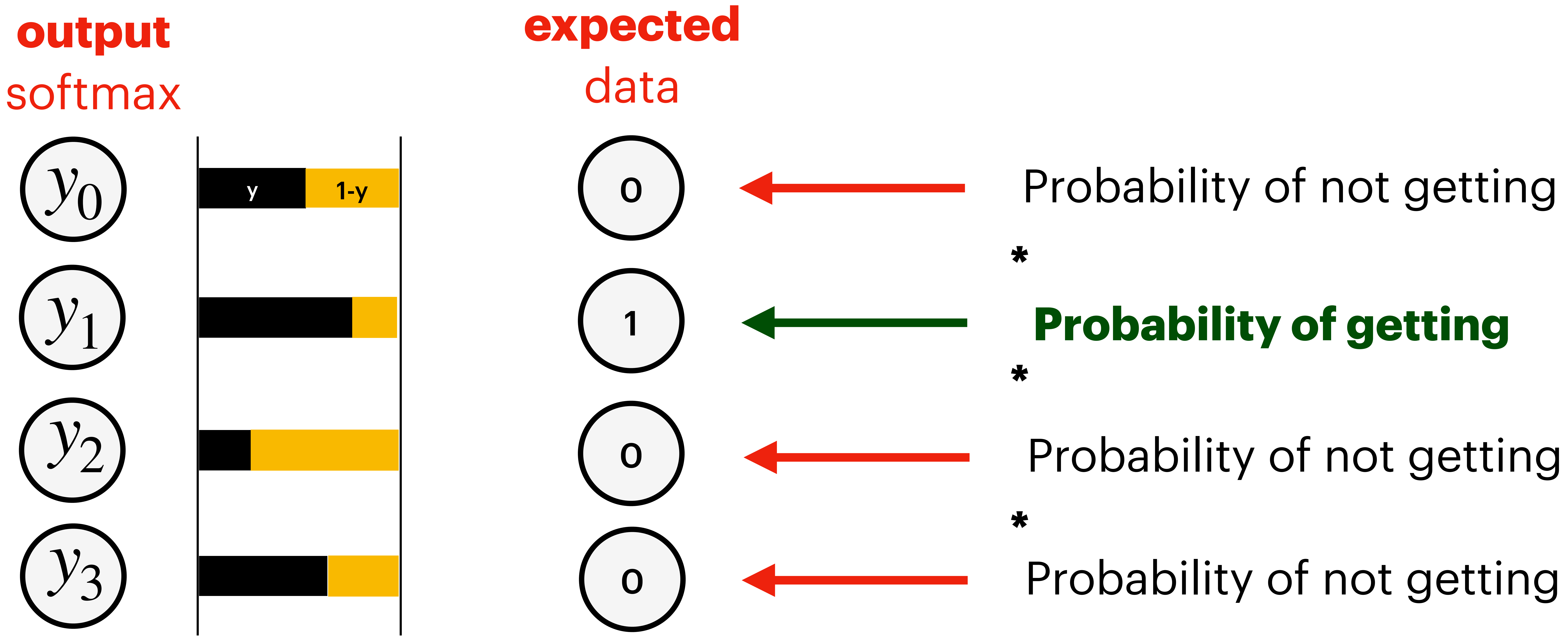
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

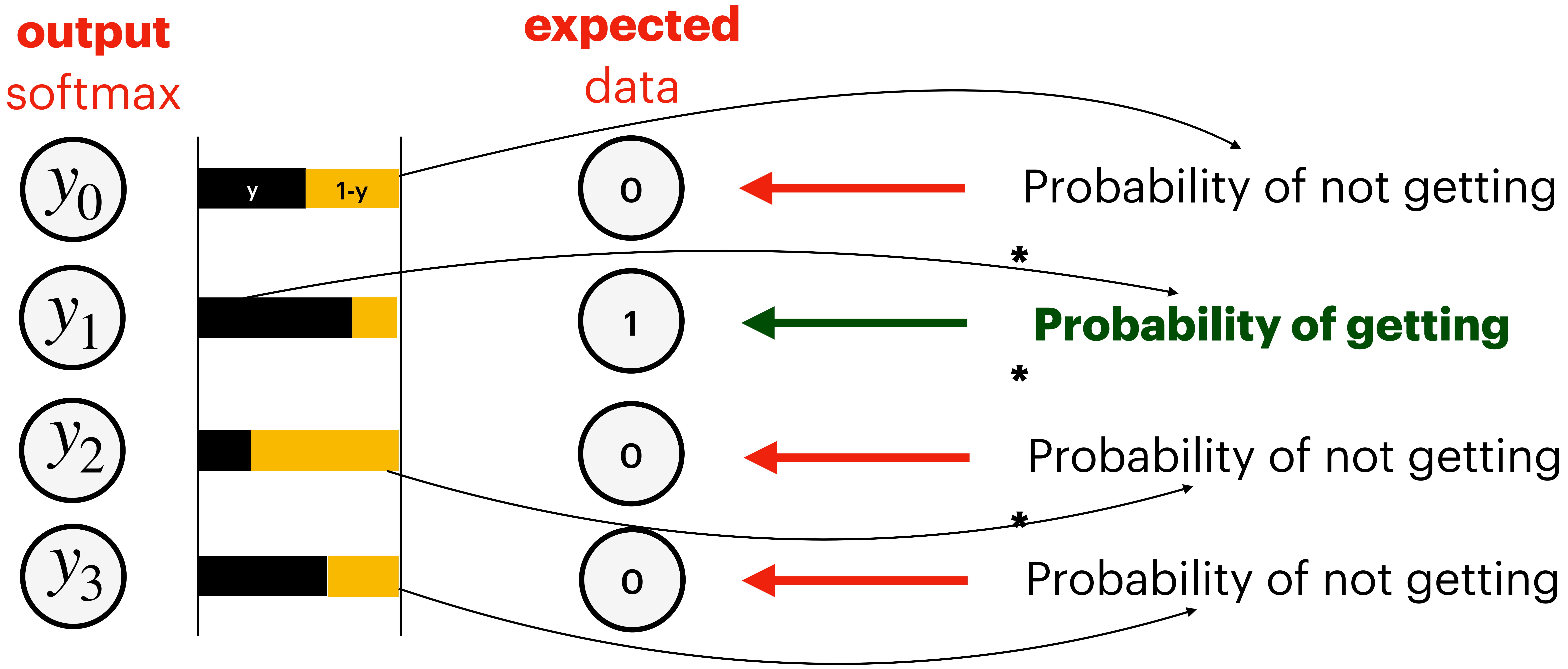
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

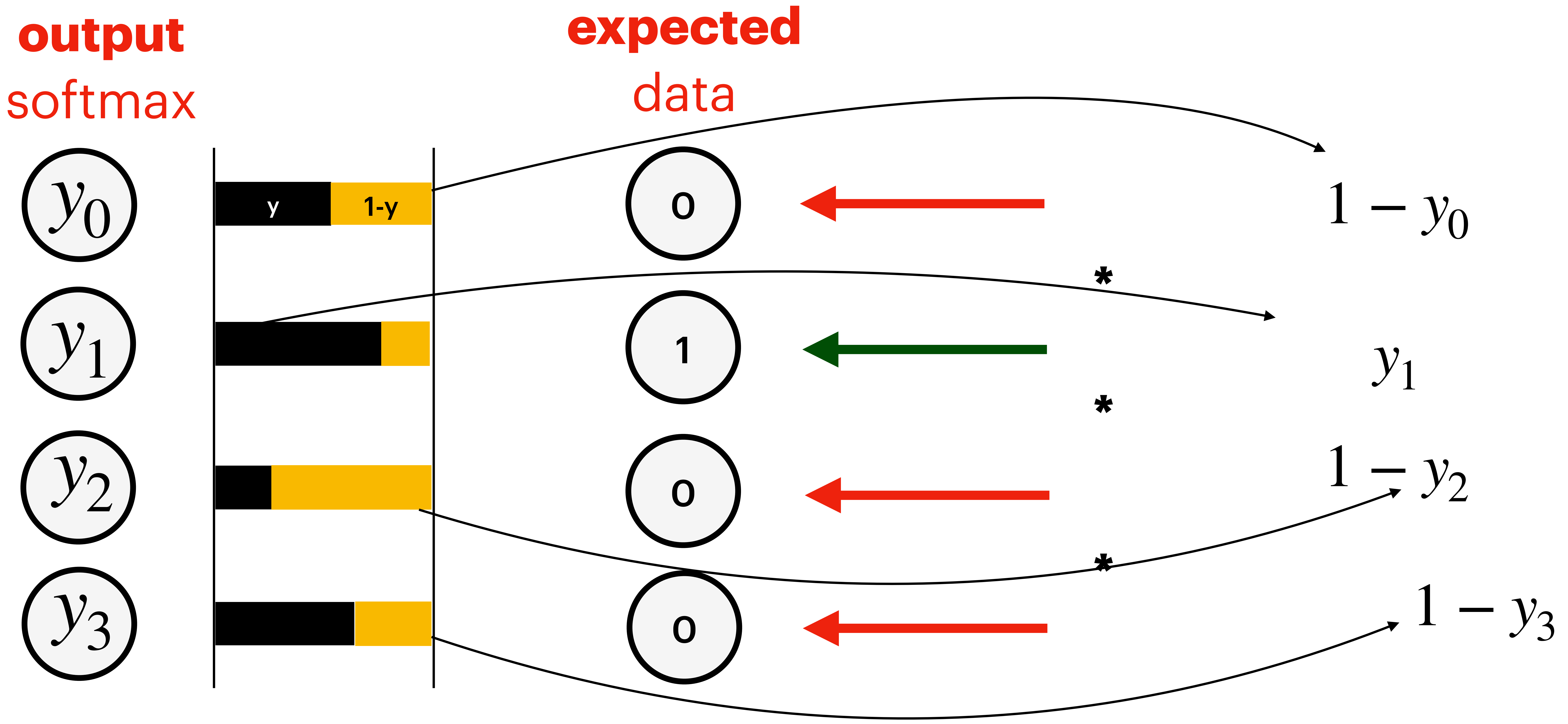
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like data likelihood with a negative sign

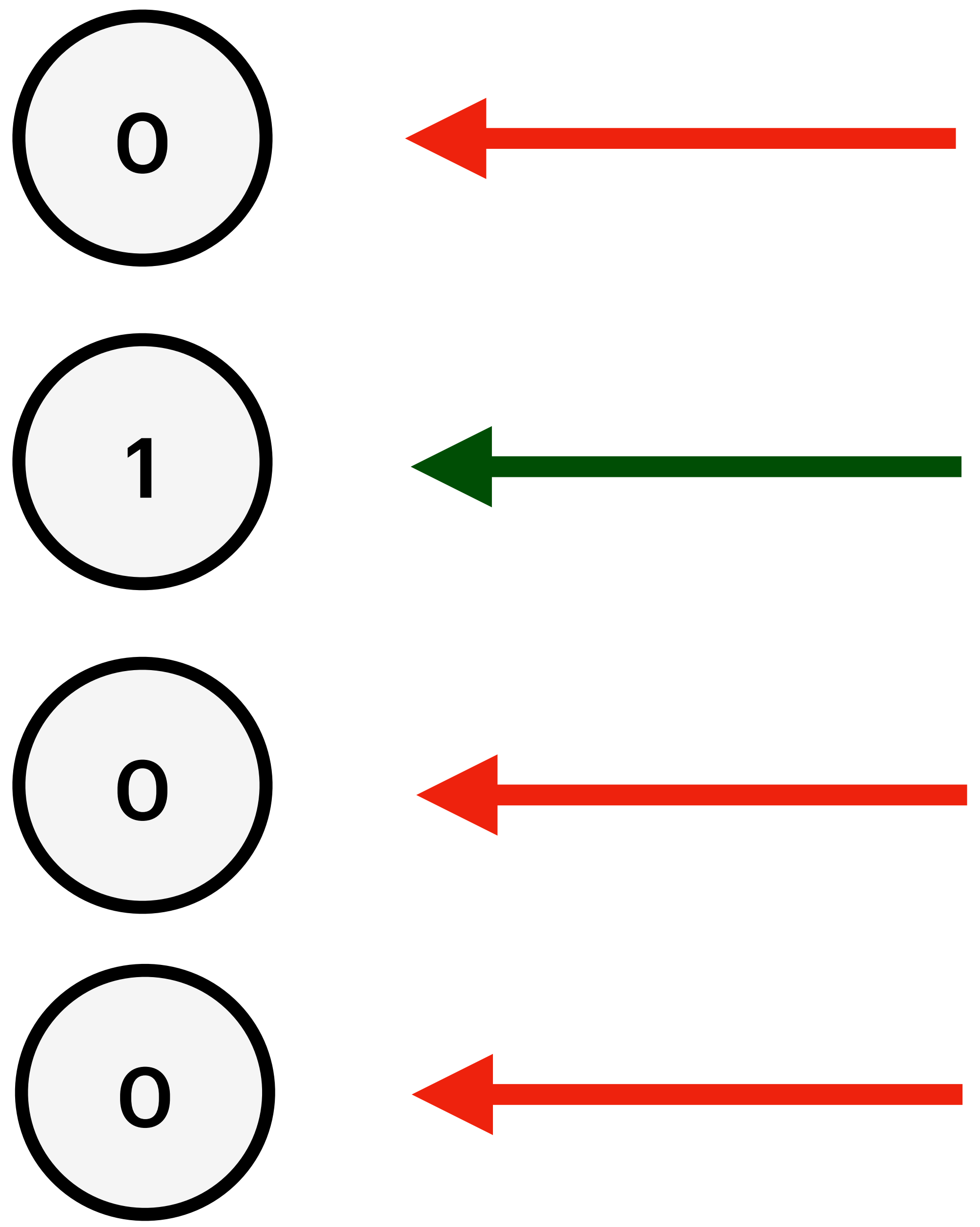


Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like data likelihood with a negative sign

expected
data



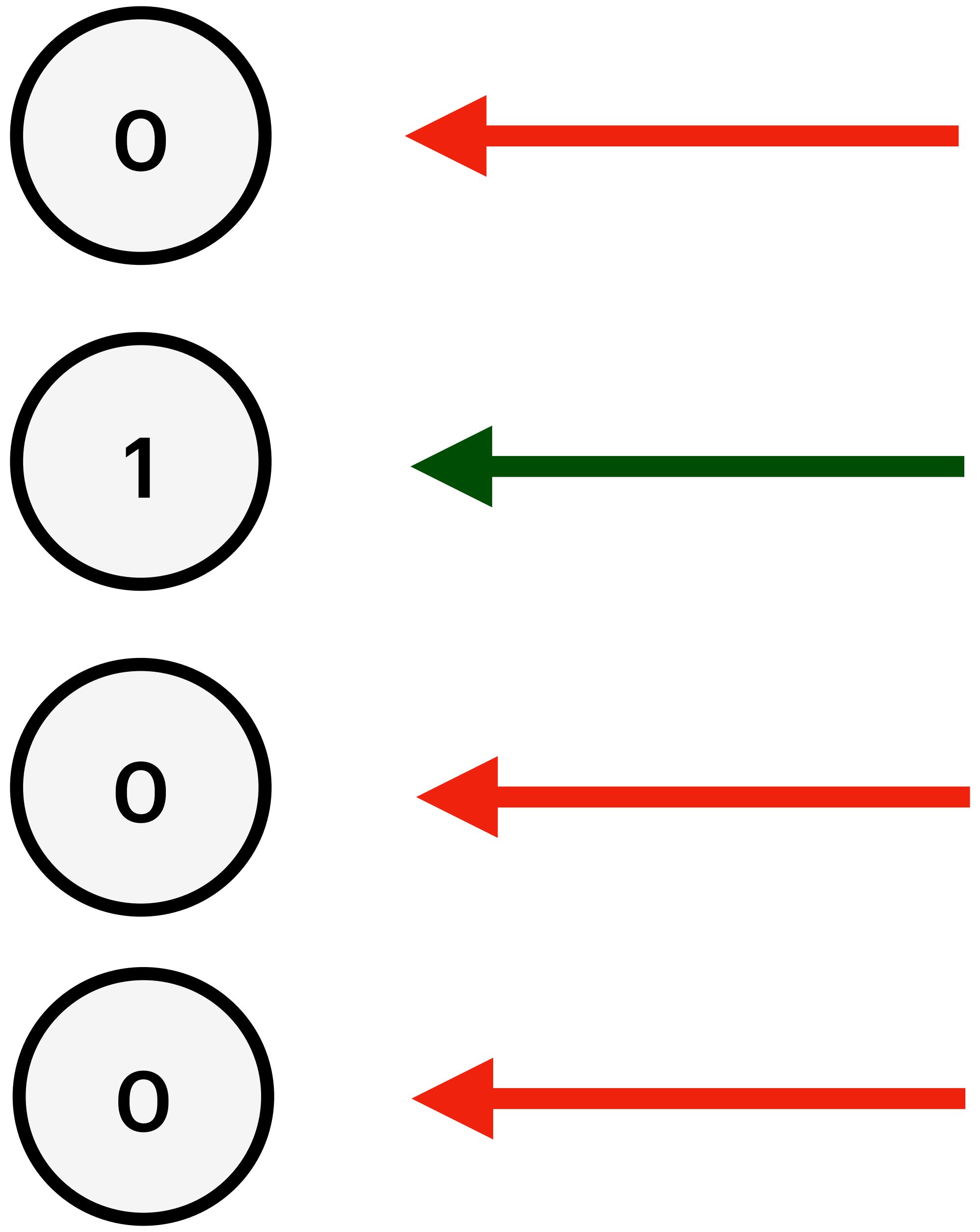
$$1 - y_0 \\ * \\ y_1 \\ * \\ 1 - y_2 \\ * \\ 1 - y_3$$

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

- But think of it like data likelihood with a negative sign

expected
data

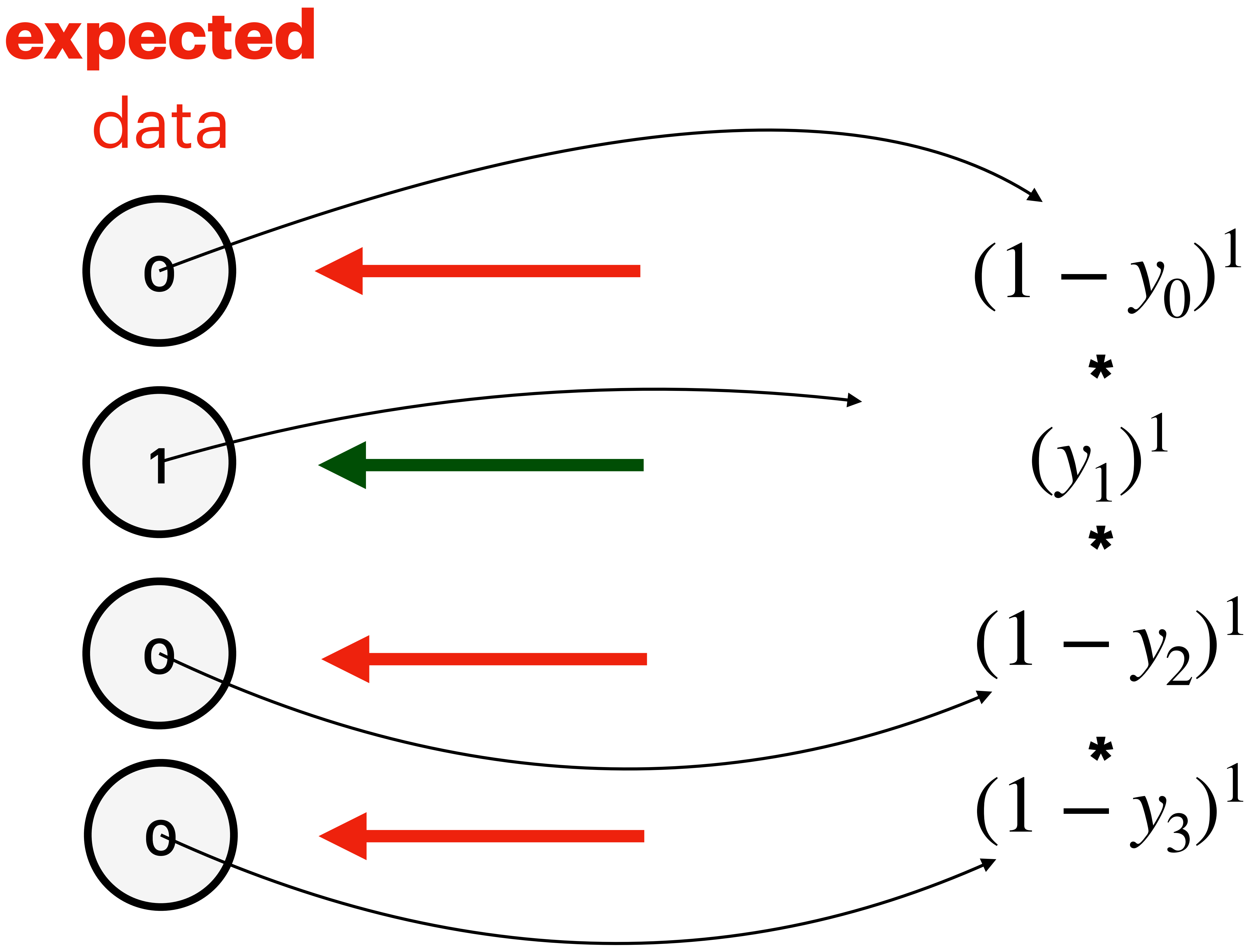


$$(1 - y_0)^1$$
$$*$$
$$(y_1)^1$$
$$*$$
$$(1 - y_2)^1$$
$$*$$
$$(1 - y_3)^1$$

Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

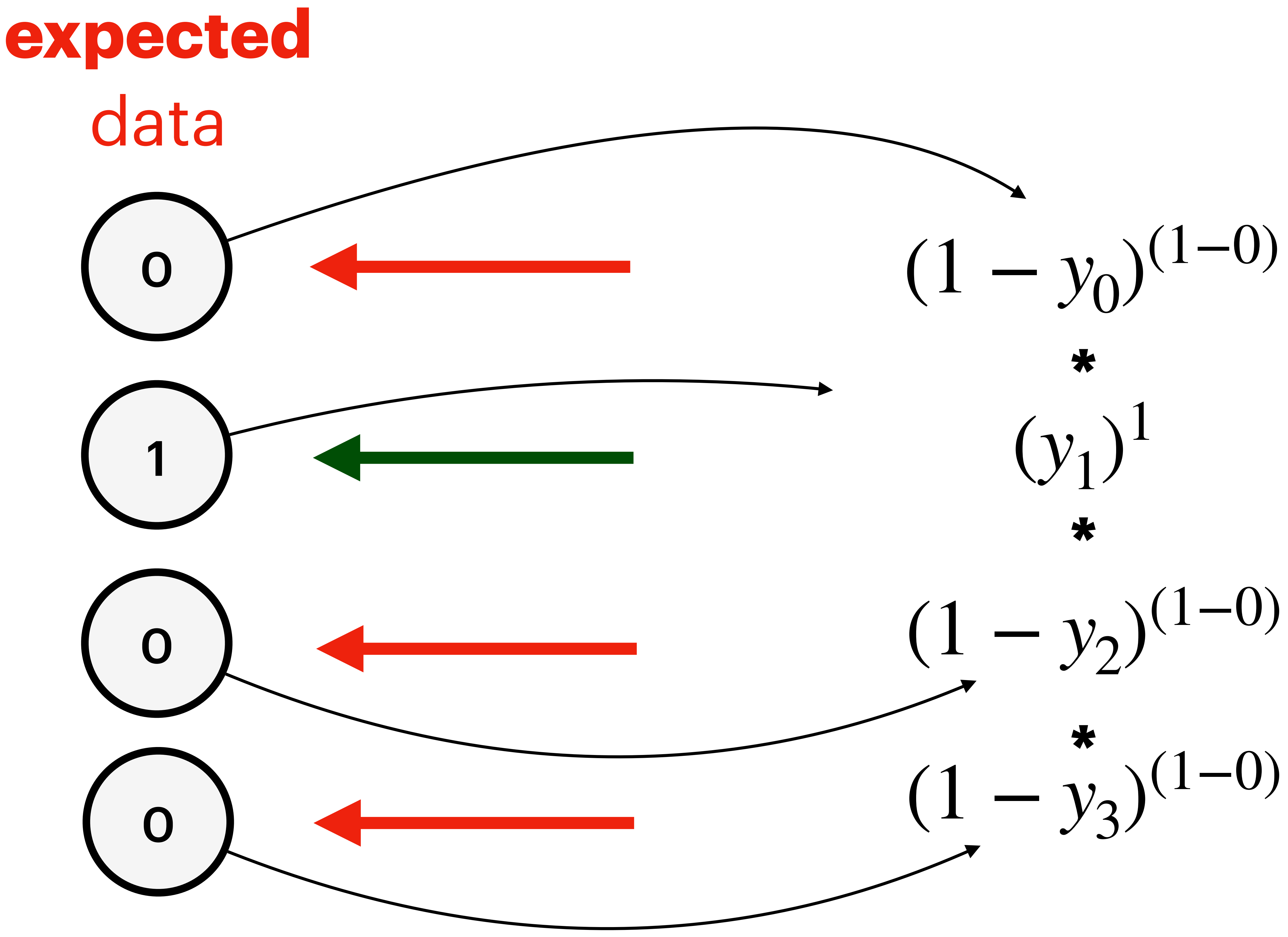
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i))]$$

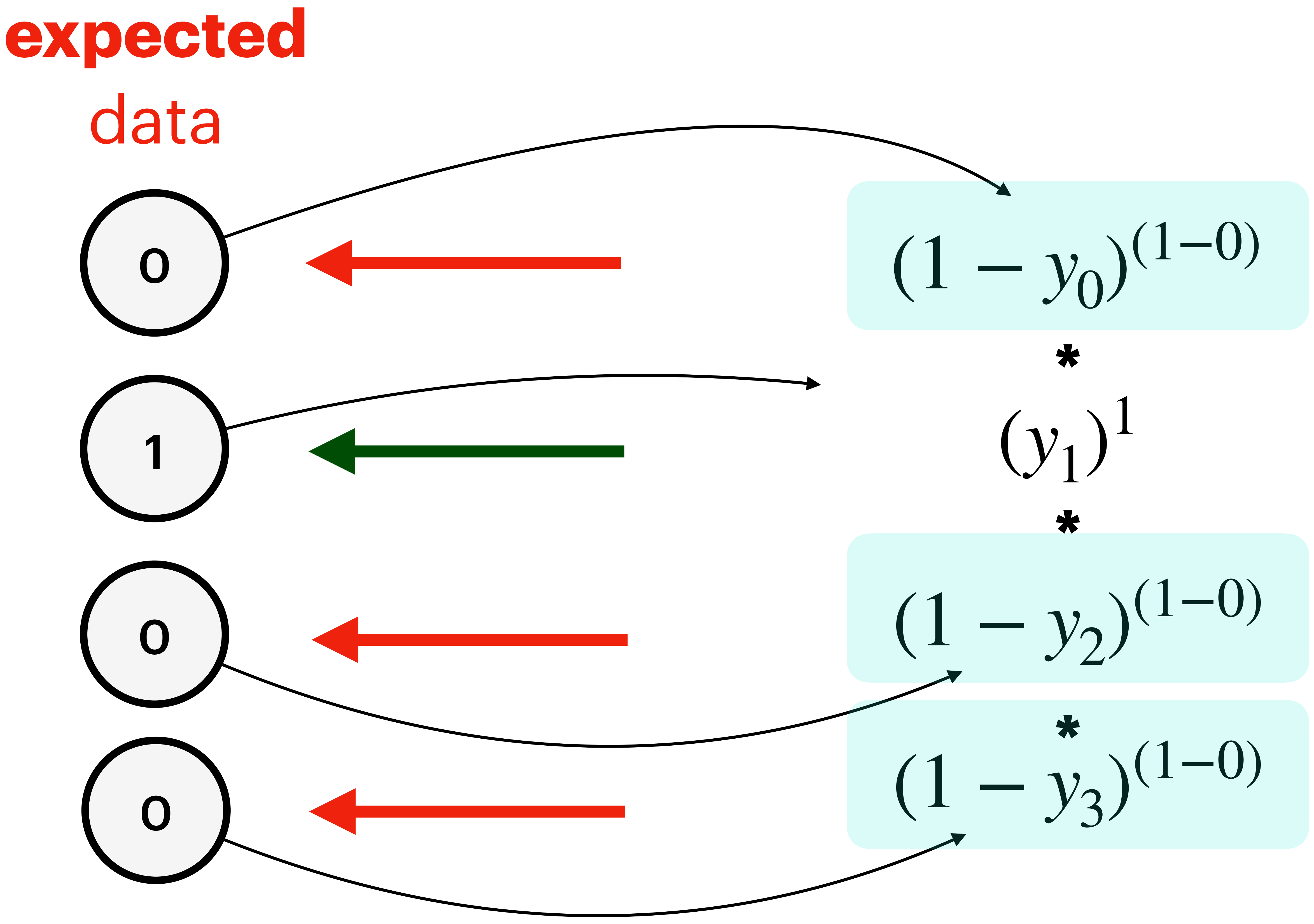
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1-\mathbf{y}_i)\log(1-\hat{\mathbf{y}}_i))]$$

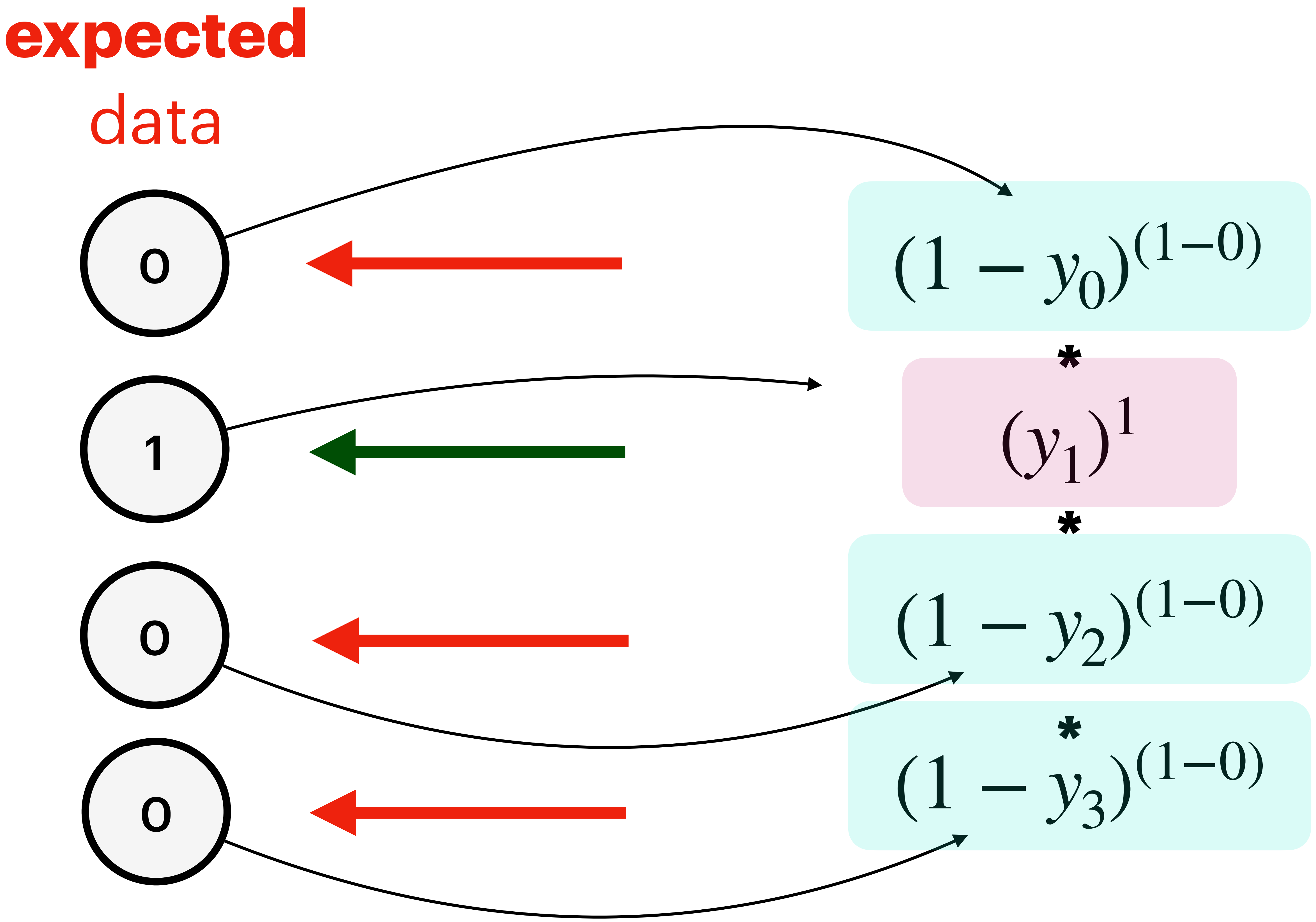
- But think of it like data likelihood with a negative sign



Cross Entropy

$$\ell(\hat{\mathbf{y}}_i, \mathbf{y}_i) = -\log p(\mathbf{y}_i|\mathbf{x}_i) = -[\mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i)\log(1 - \hat{\mathbf{y}}_i))]$$

- But think of it like data likelihood with a negative sign



High Level Structure

High Level Structure

on the other end of the pipeline

Fundamentals

INPUT PROCESSING

Input Processing

needs to be encoded

Input Processing

needs to be encoded

TEXT

Using pre-trained
models that
have learned from
large datasets
and adapting them to
new tasks…..

Input Processing

needs to be encoded

TEXT

Using pre-trained
models that
have learned from
large datasets
and adapting them to
new tasks…..

ONE HOT ENCODING

Input Processing

needs to be encoded

TEXT

Using pre-trained
models that
have learned from
large datasets
and adapting them to
new tasks…..

BAG OF WORDS

ONE HOT ENCODING

Input Processing

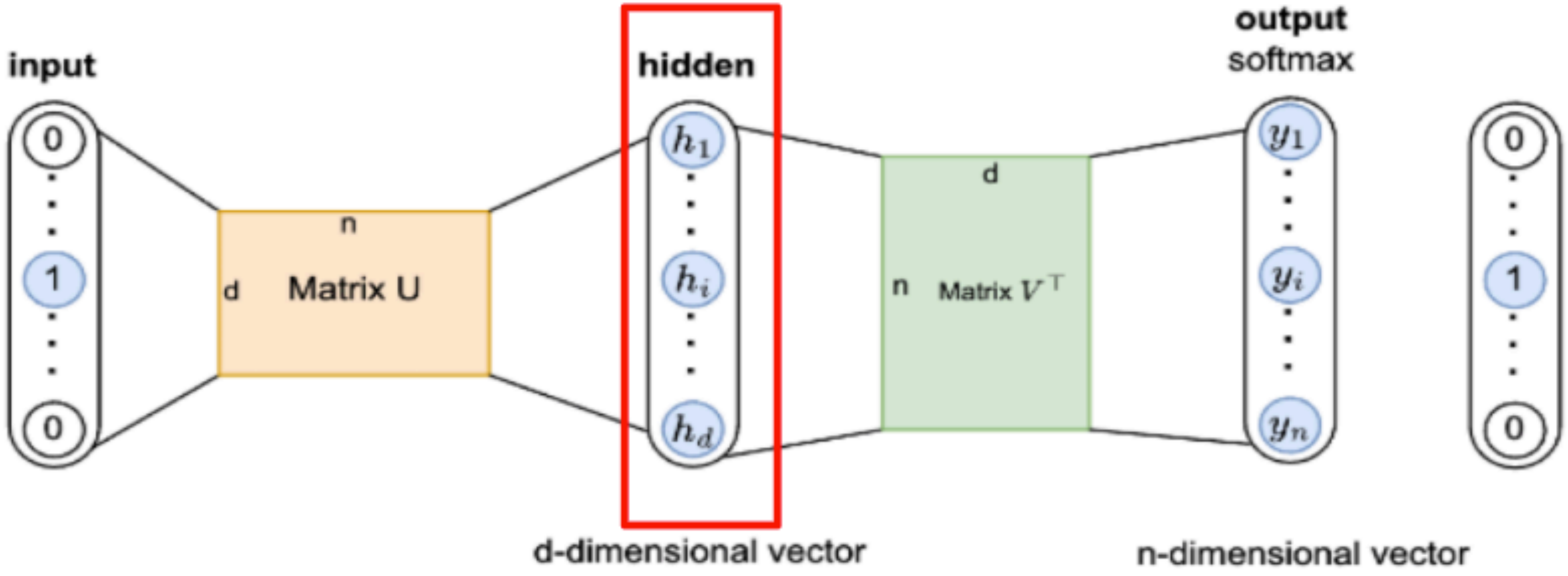
needs to be encoded

BAG OF WORDS

TEXT

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

ONE HOT ENCODING



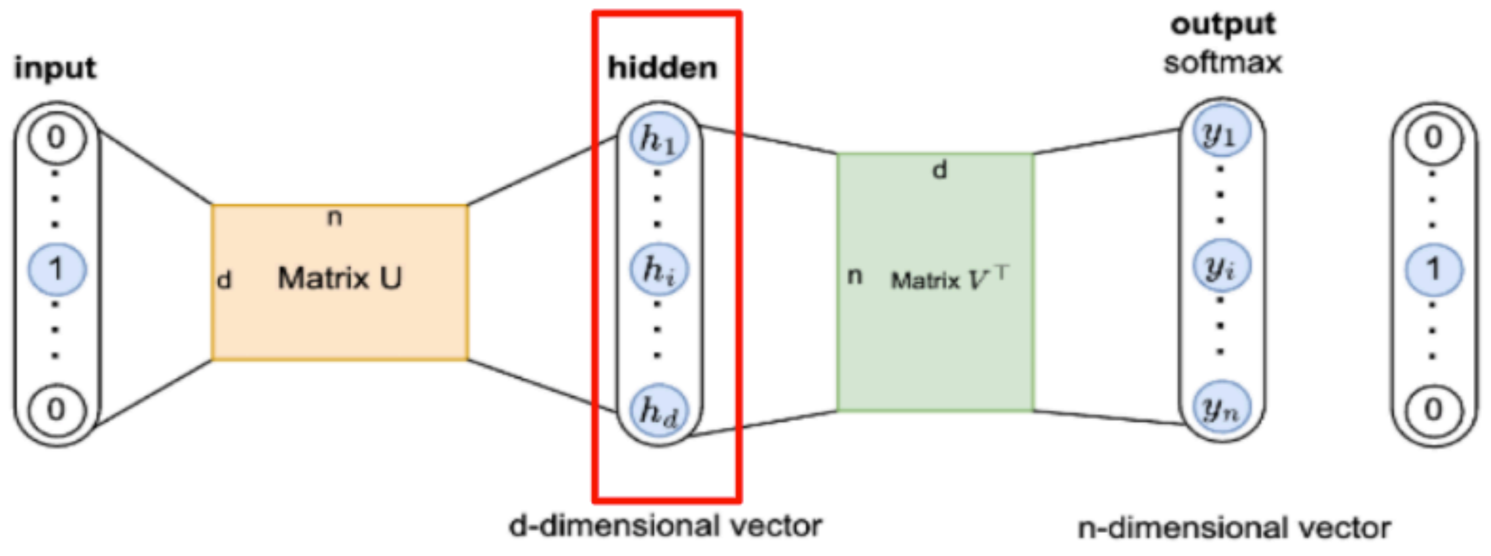
EMBEDDINGS

Input Processing

needs to be encoded

TEXT

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

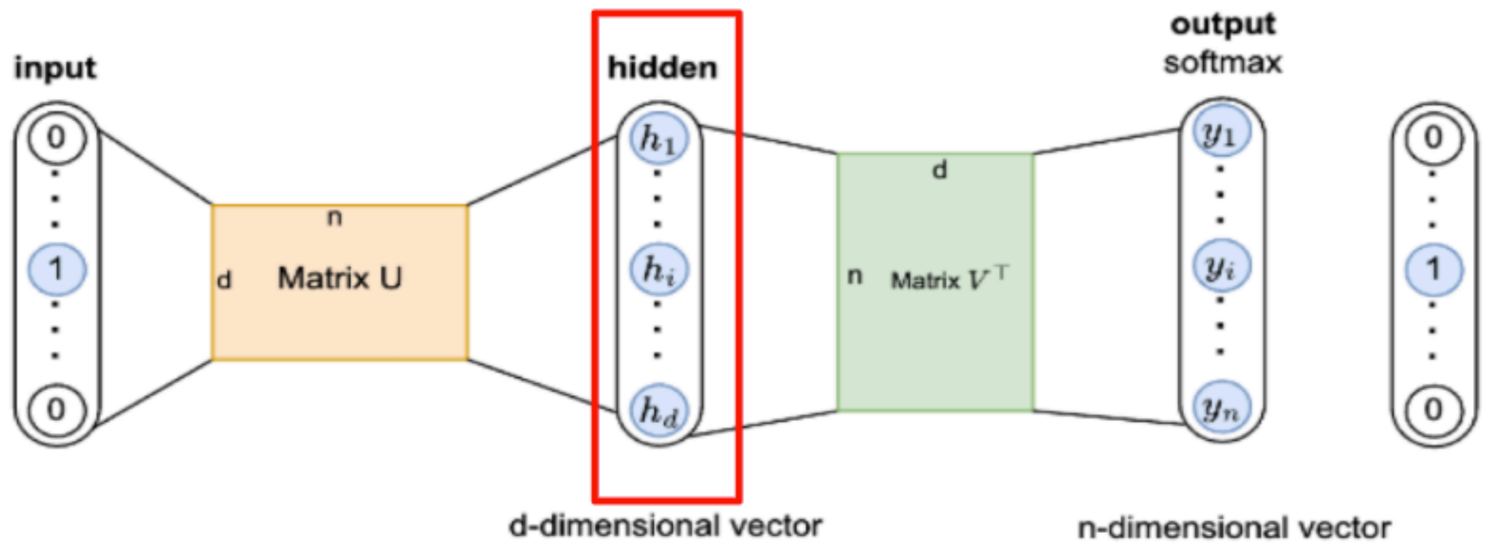


Input Processing

needs to be encoded

TEXT

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..



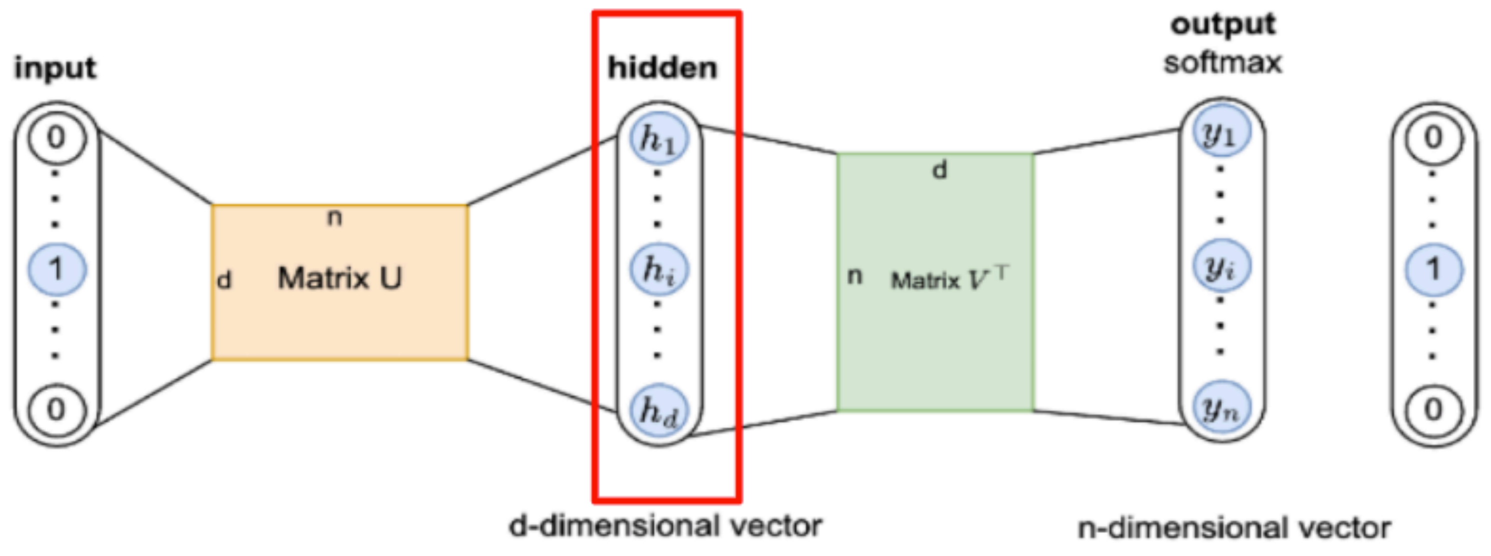
IMAGE

Input Processing

needs to be encoded

TEXT

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..

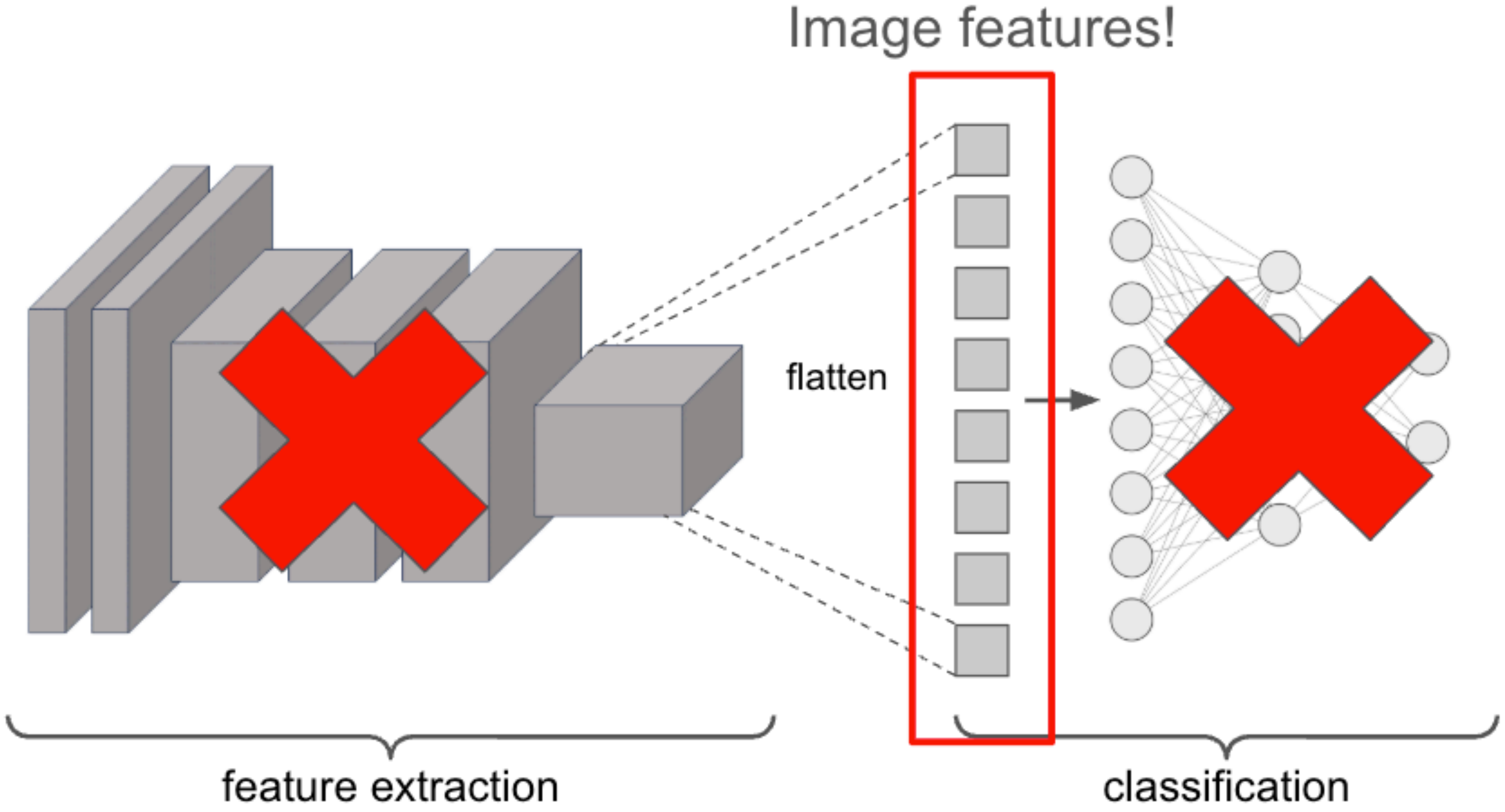


IMAGE

DIRECT PIXEL VALUES

Input Processing

needs to be encoded



EMBEDDINGS

IMAGE

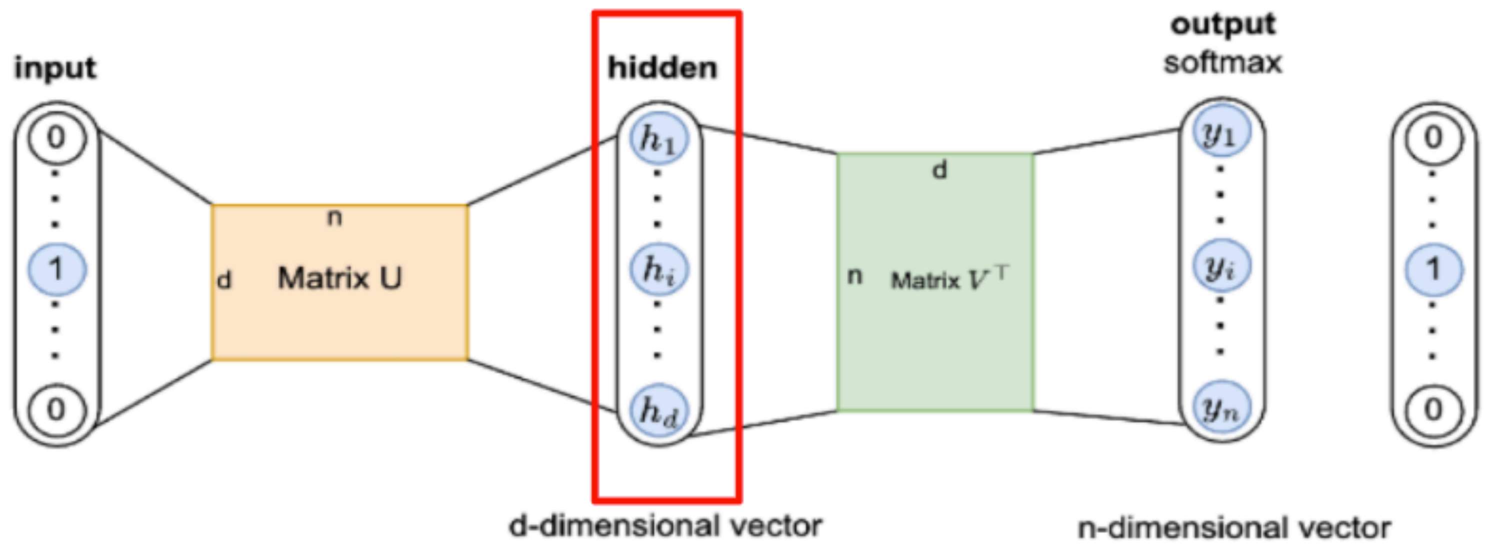
DIRECT PIXEL VALUES

Input Processing

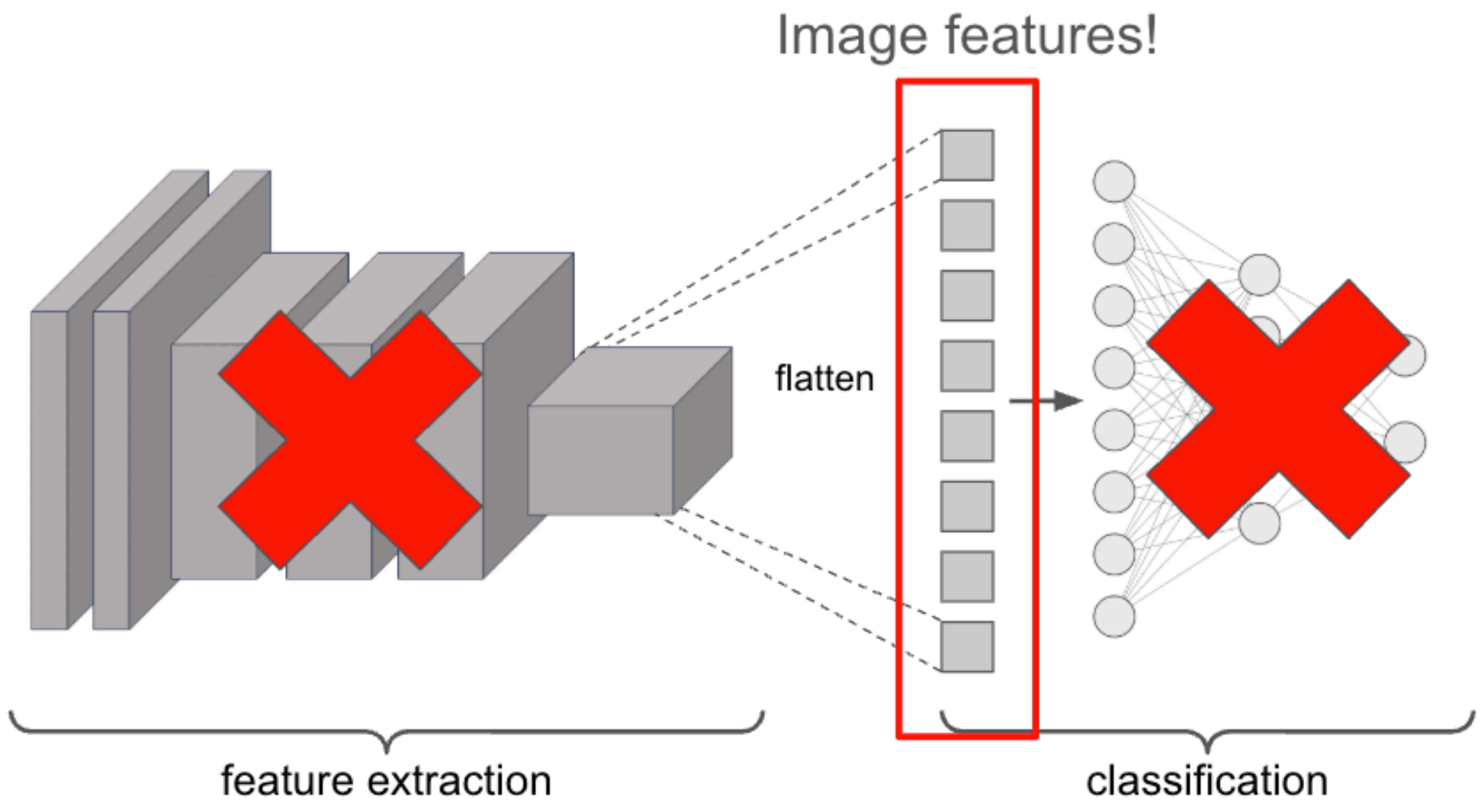
needs to be encoded

TEXT

Using pre-trained models that have learned from large datasets and adapting them to new tasks…..



IMAGE



High Level Structure

Fundamentals

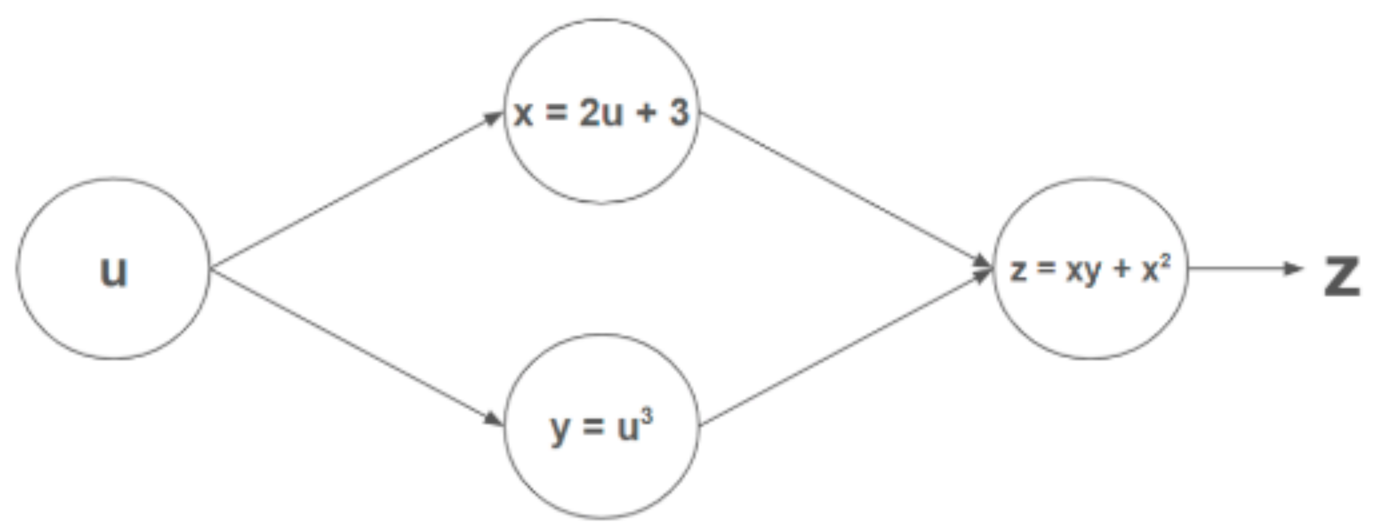
BACKPROPAGATION

Backpropagation

calculate derivatives of loss and update parameters

Problem 4: Mini Backpropagation [15 pts]

Suppose you are given the following network with input u and output z.



Using backpropagation, calculate the derivative of the output z with respect to the input u. You may leave your answer in terms of u, x, and y. Please show your work, including the calculations of any intermediate derivatives you use to derive your final answer.

Fundamentals

OPTIMIZATION

Optimization

backpropagate optimally

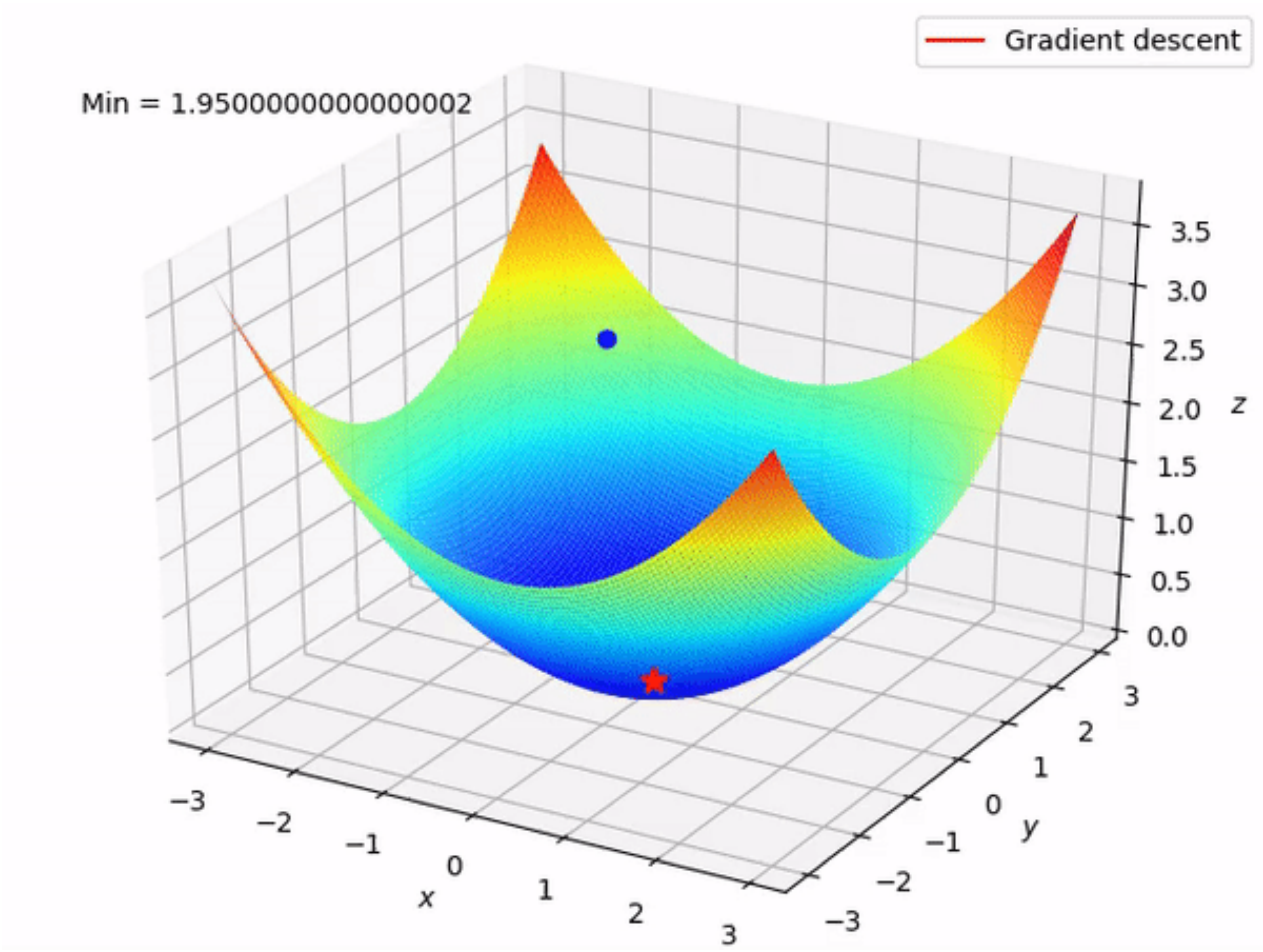
Some Terminology

Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

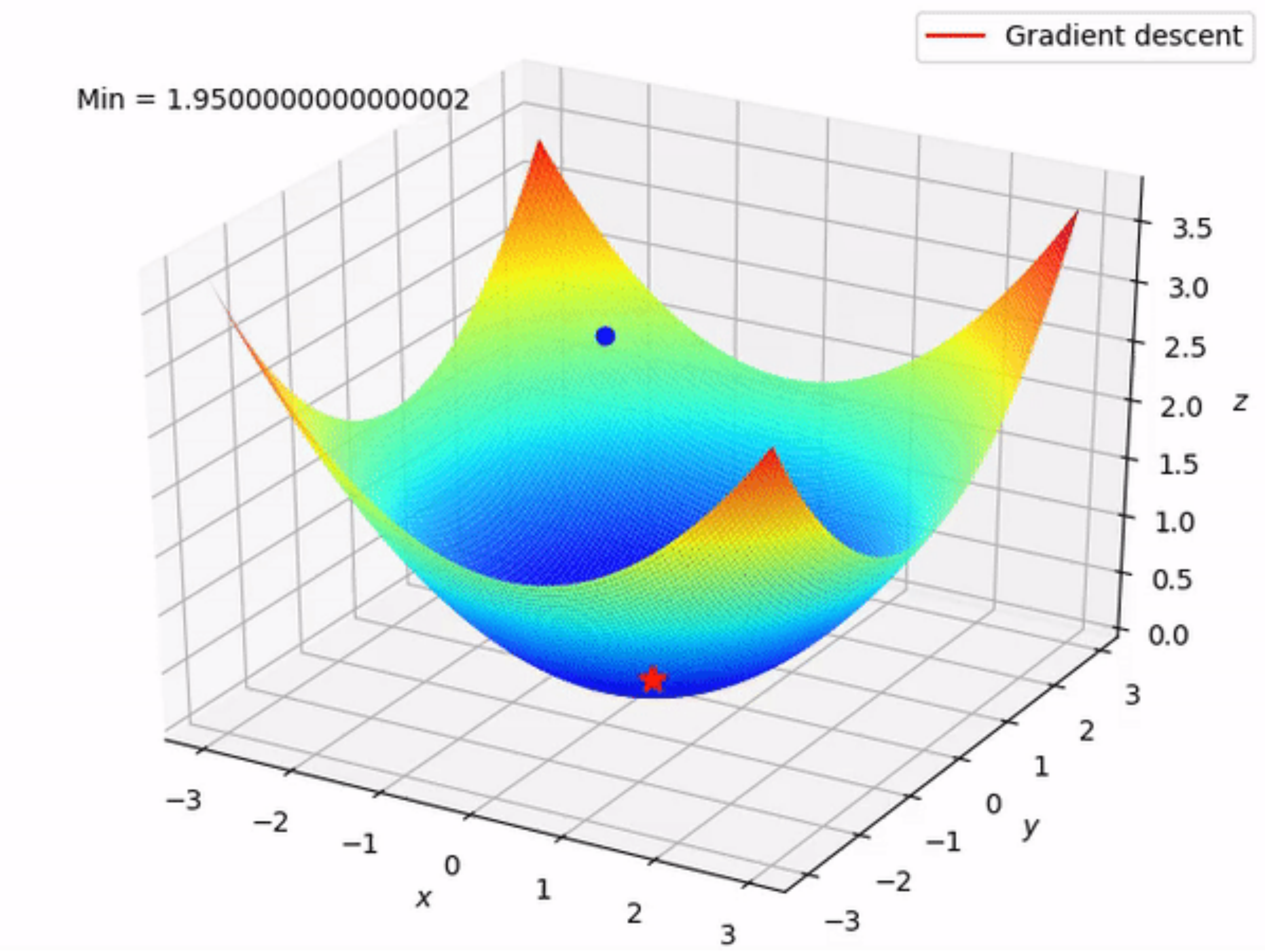


Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$



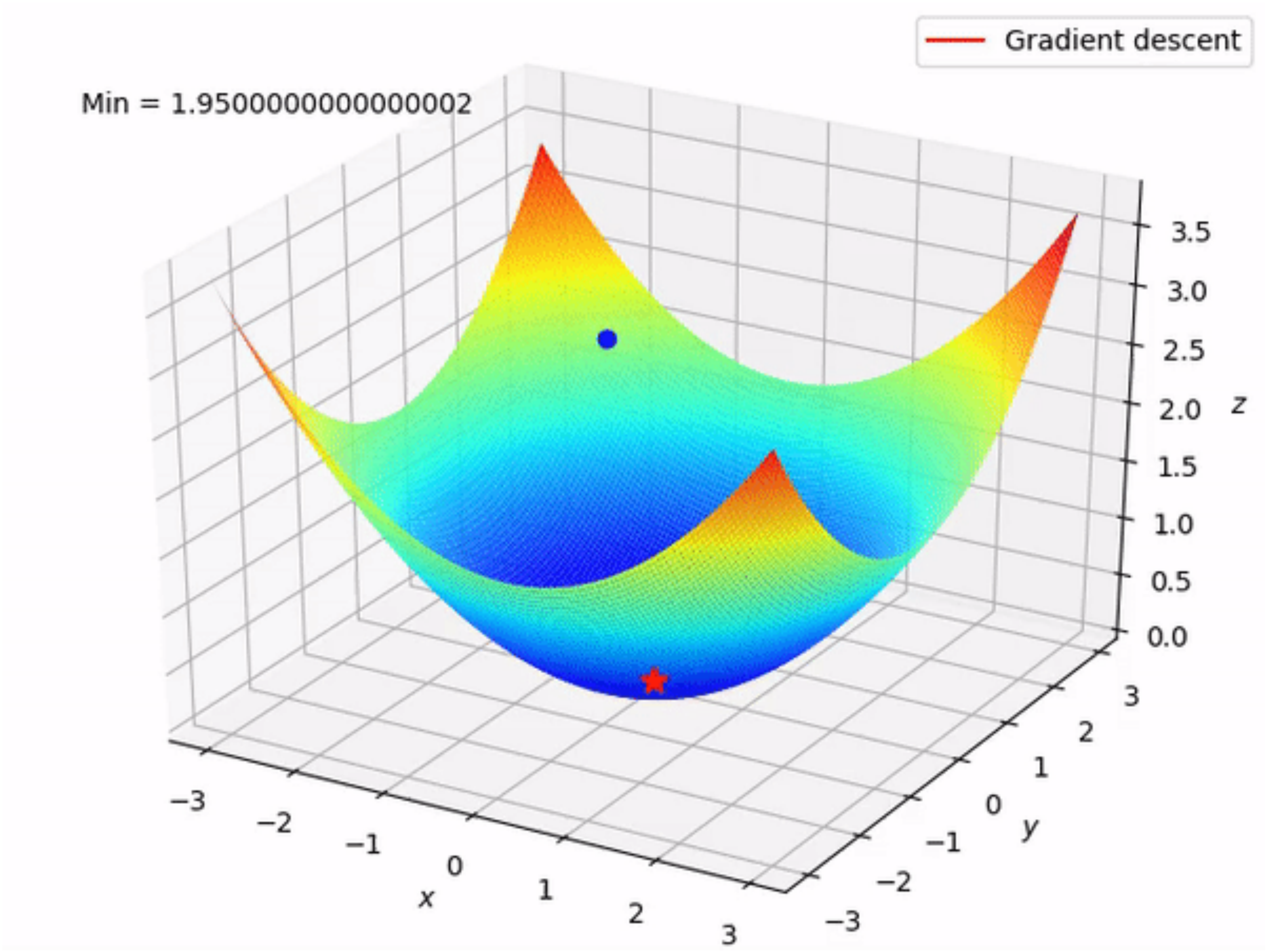
Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

Gradient Calculation



Optimization

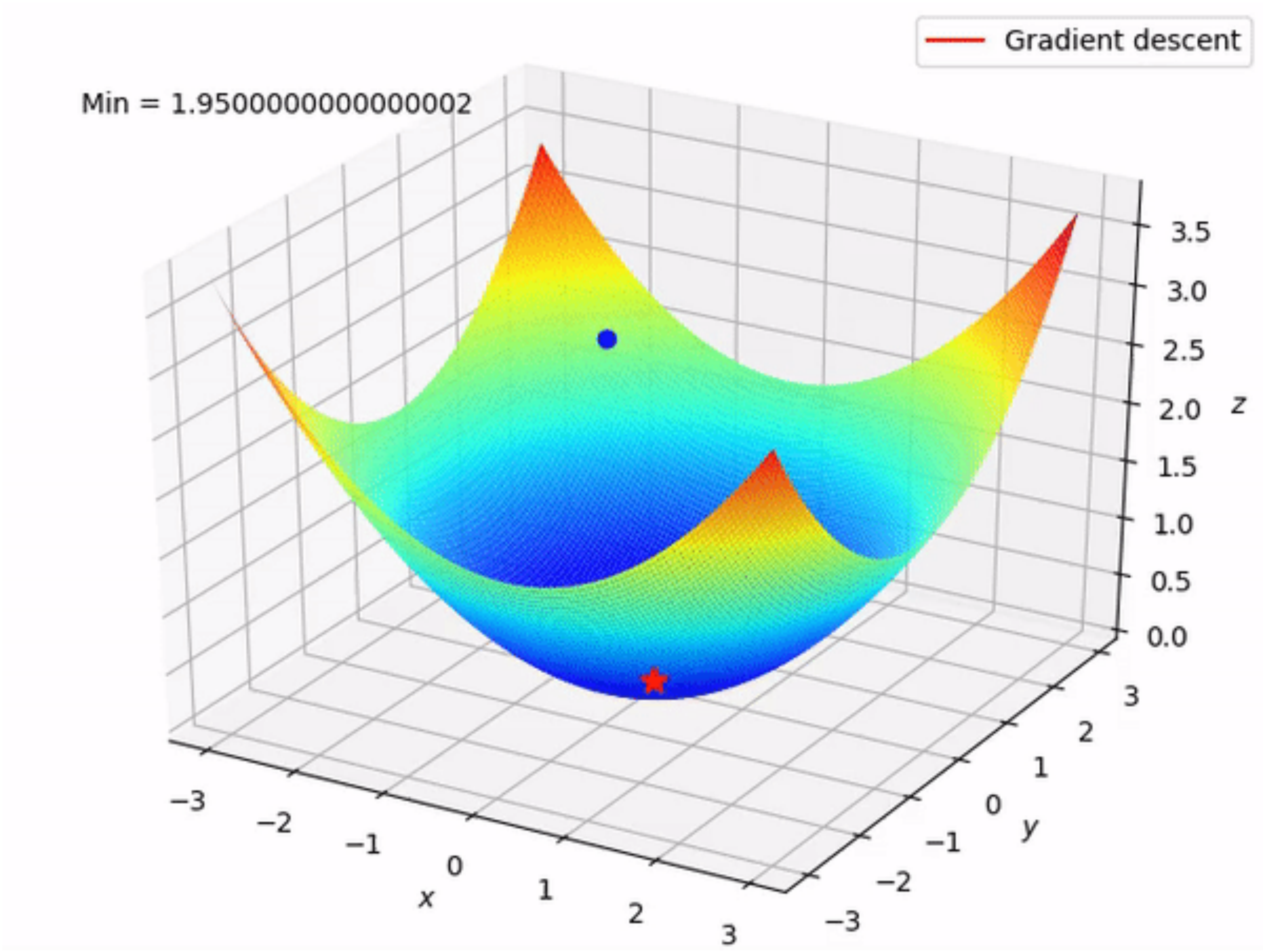
backpropagate optimally

Gradient Descent

Weight Update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

Gradient Calculation



Optimization

backpropagate optimally

Gradient Descent

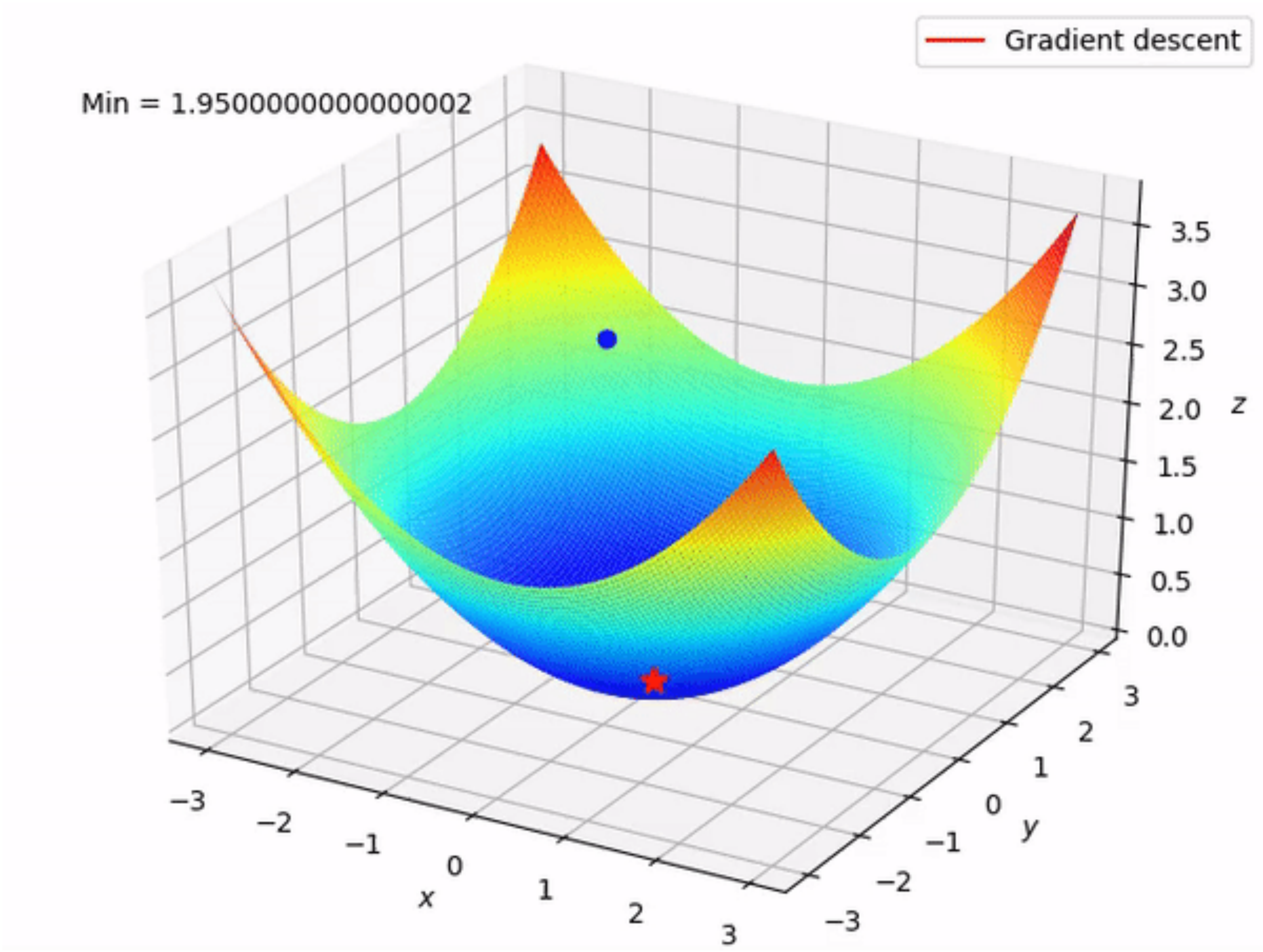
Weight Update

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

Gradient Calculation

compute expensive

$$\mathcal{L}(\mathbf{w}_t) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}_t, \mathbf{x}_i)$$



Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

sample 1

Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

sample 1

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Optimization

backpropagate optimally

Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

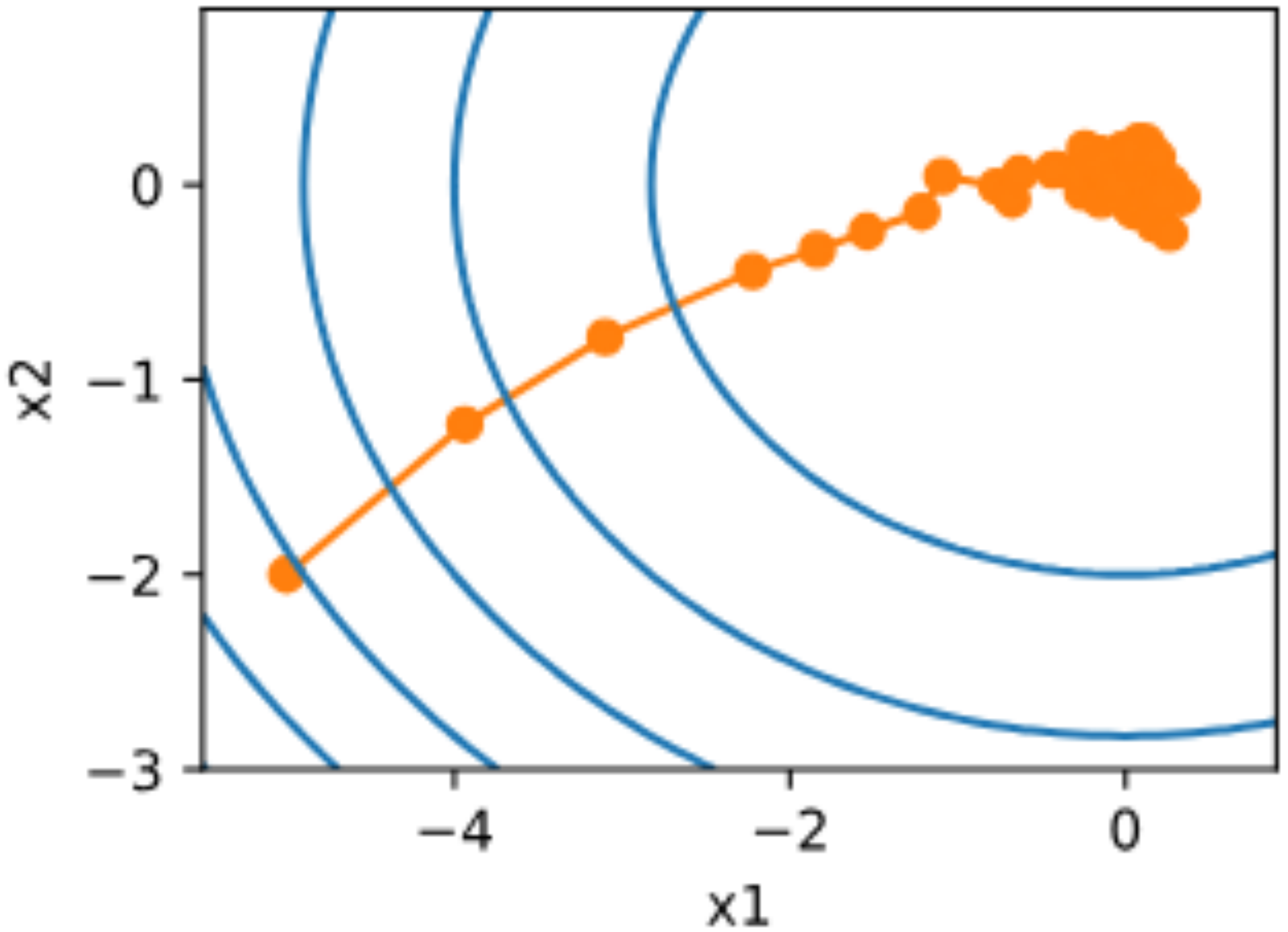
compute expensive

Stochastic Gradient Descent

sample 1

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

fast but noise ball convergence



Optimization

backpropagate optimally

Gradient Descent

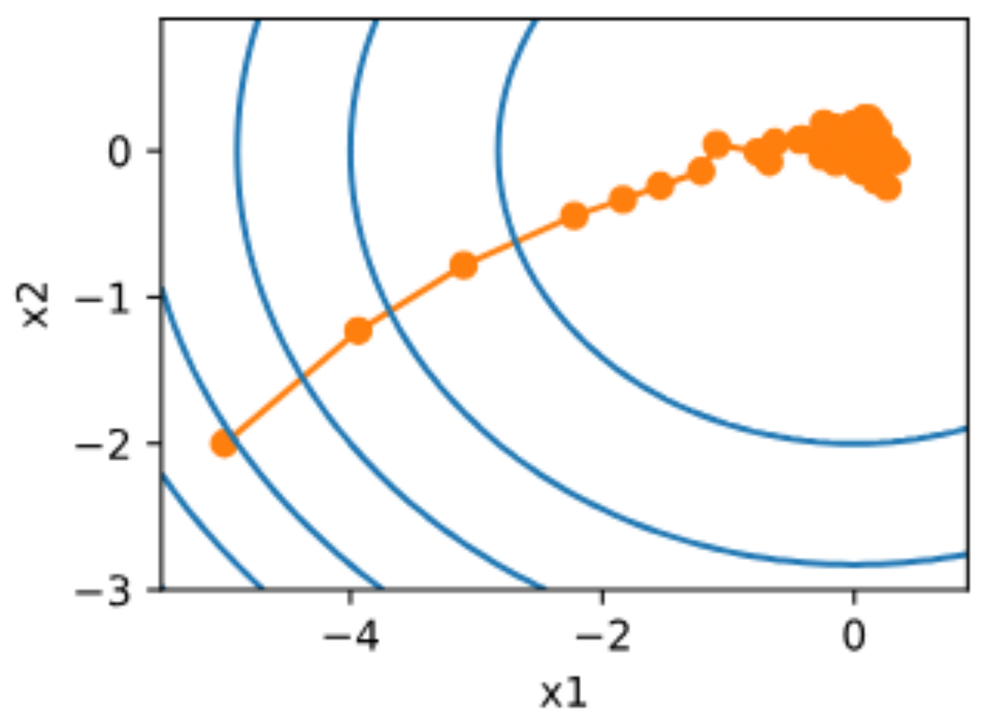
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

fast but noise ball convergence



Optimization

backpropagate optimally

Gradient Descent

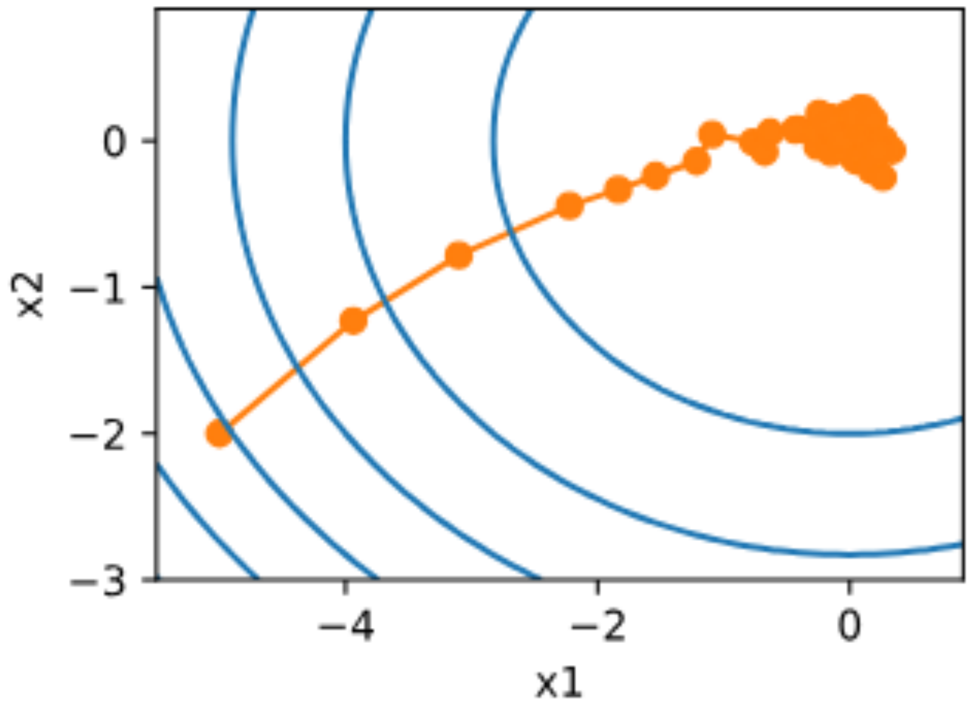
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

fast but noise ball convergence

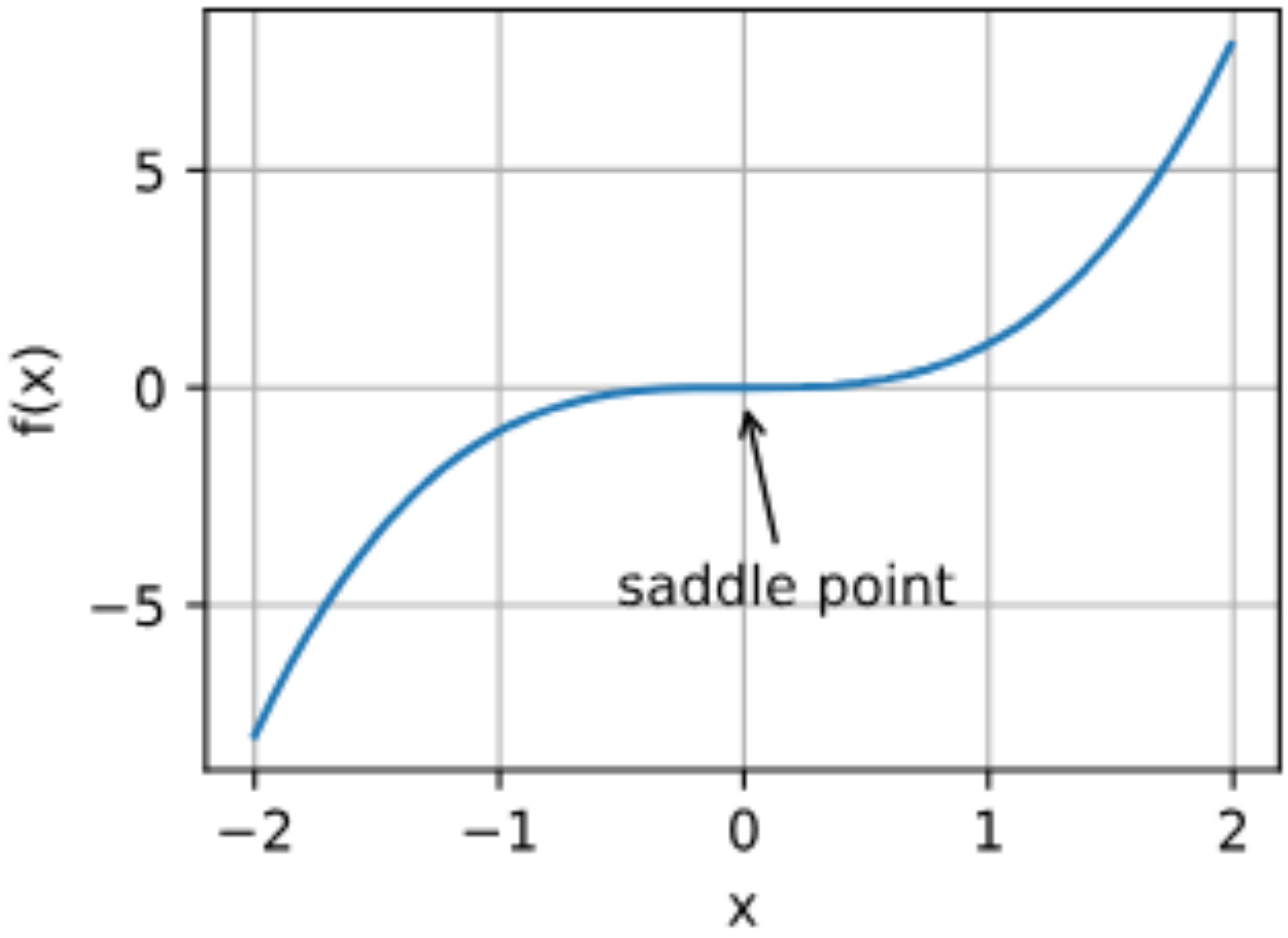


Mini-Batch Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

Optimization

backpropagate optimally



Mini-Batch
Stochastic
Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

without momentum can get stuck in a saddle point

Optimization

backpropagate optimally

Gradient Descent

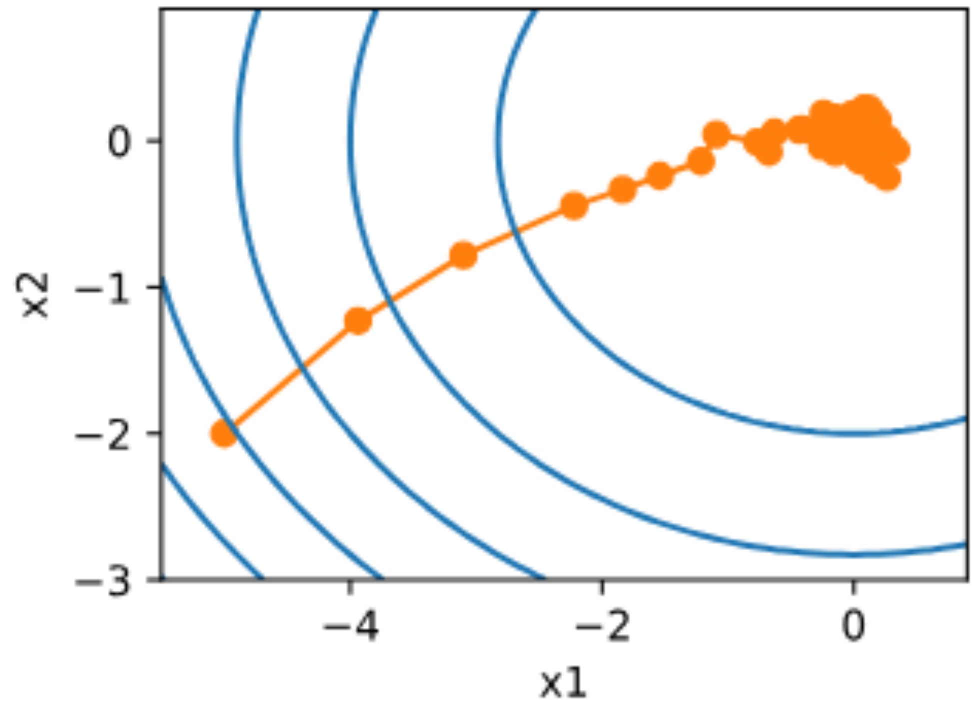
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

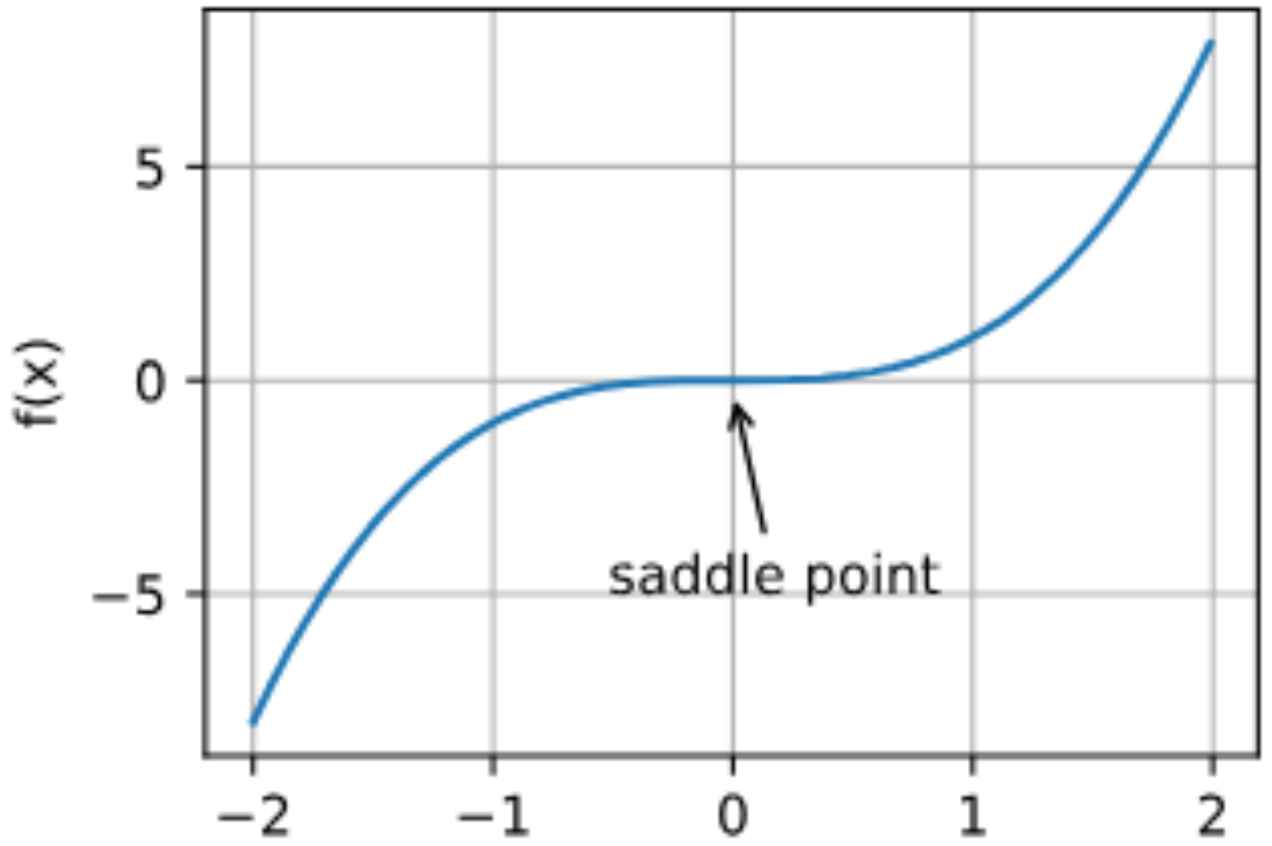
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

fast but noise ball convergence



Mini-Batch Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$



Optimization

backpropagate optimally

Gradient Descent

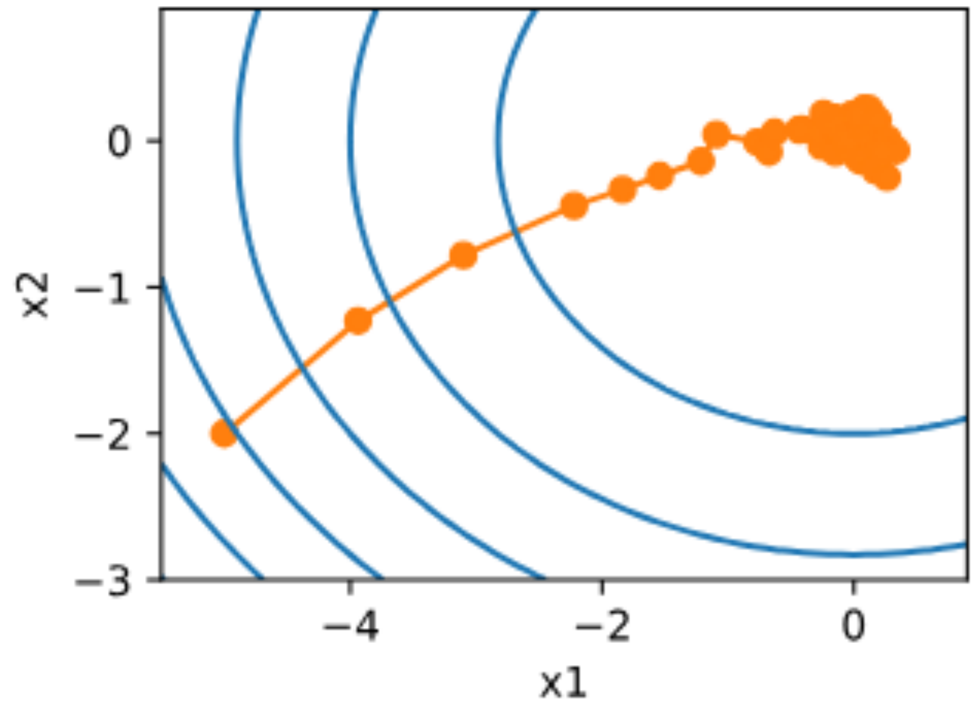
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \mathcal{L}(\mathbf{w}_t)$$

compute expensive

Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

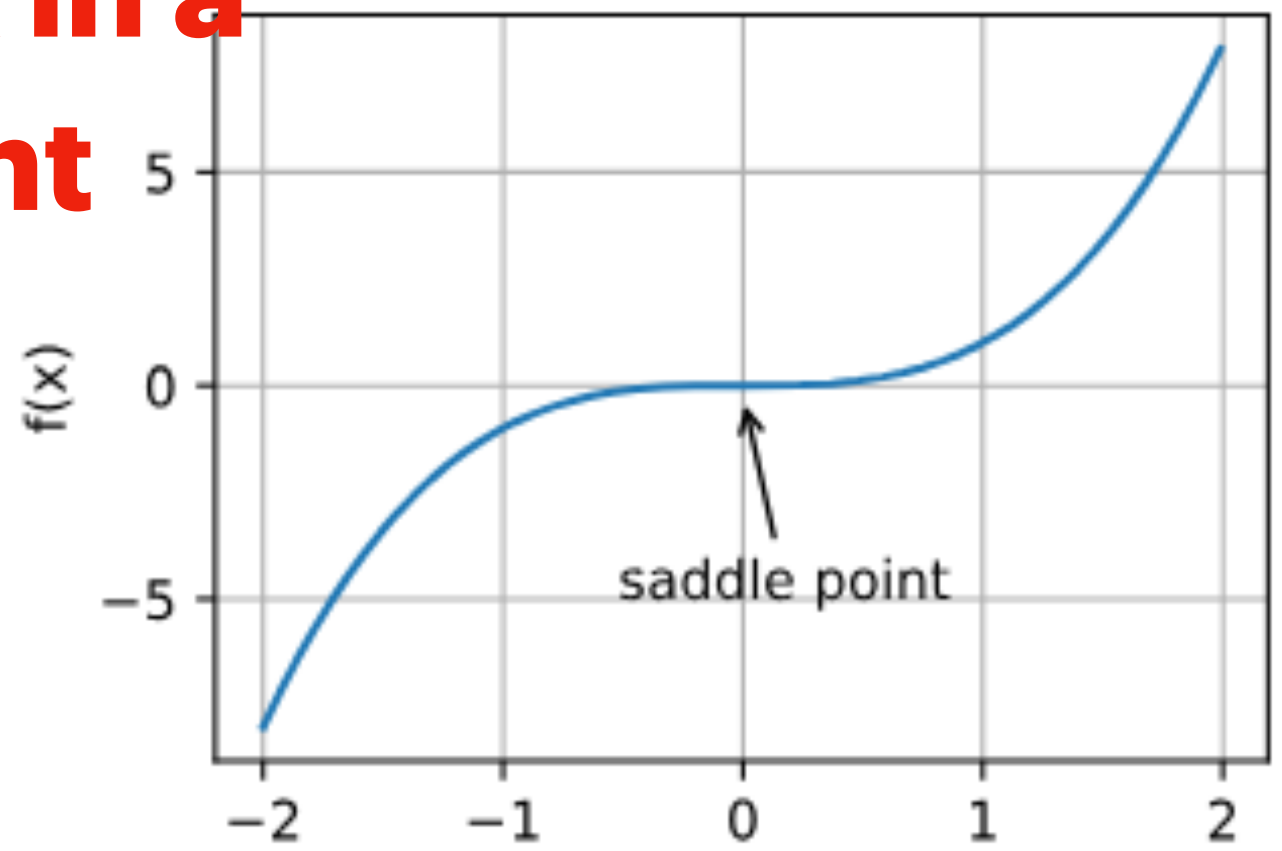
fast but noise ball convergence



Mini-Batch Stochastic Gradient Descent

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{1}{b} \sum_{i \in \mathcal{B}_t} \nabla \ell(\mathbf{w}_t, \mathbf{x}_i)$$

without momentum can get stuck in a saddle point



Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

Optimization

backpropagate optimally

SGD with Momentum

Accumulate Gradients
moving average

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

Optimization

backpropagate optimally

SGD with Momentum

Accumulate Gradients
moving average

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

But can end up stopping early if initial gradients high

Element wise control - so w_x and w_y update at different rates

Optimization

backpropagate optimally

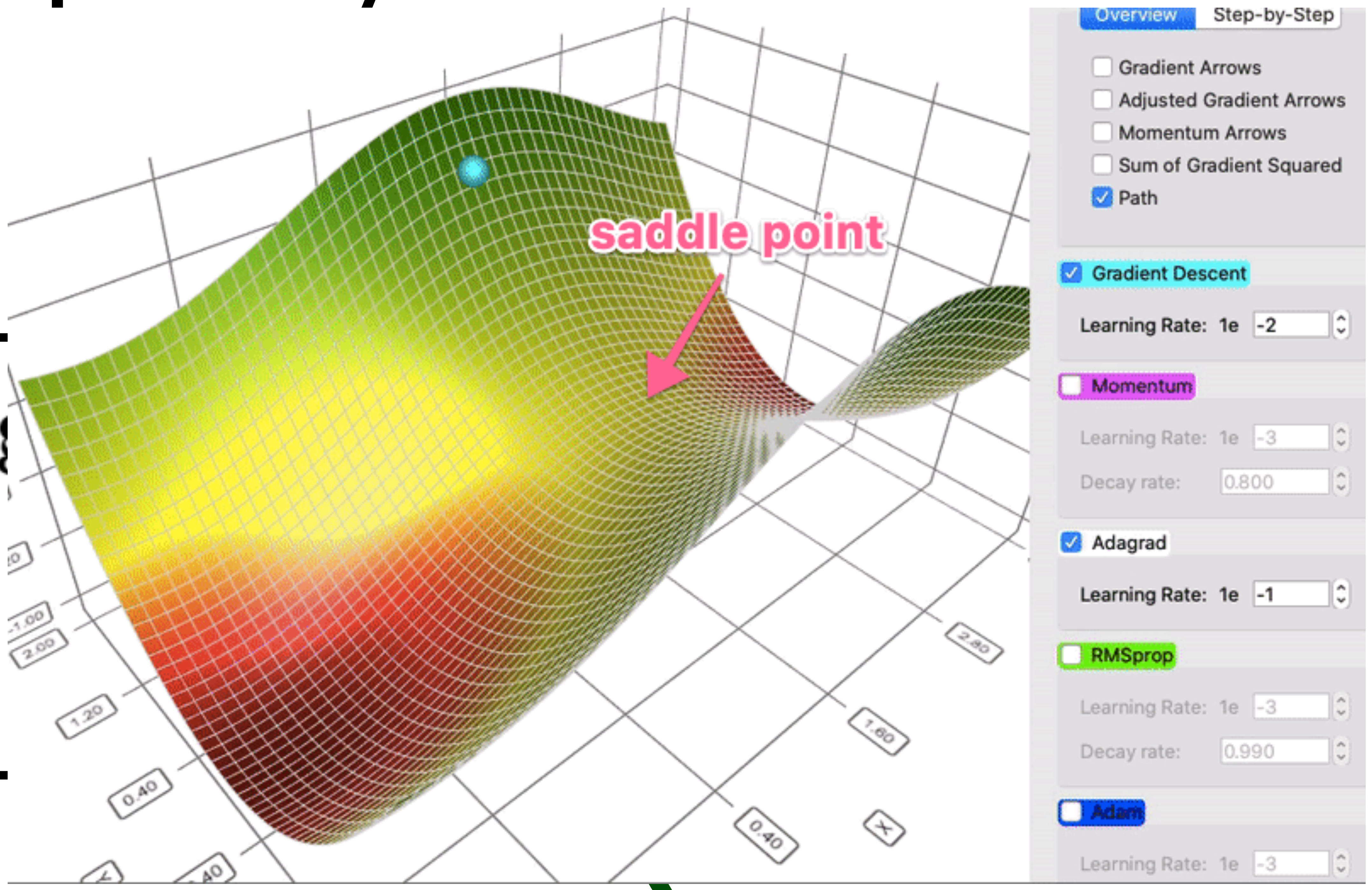
SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \ldots$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \ldots$$



But can end up stopping early if initial gradients high

Element wise control - so w_x and w_y update at different rates

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

RMSProp

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

RMSProp

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

RMSProp

exponential decay moving average
less weight to older gradients

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

note that first moment estimate is not tracked

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

RMSProp

exponential decay moving average

less weight to older gradients

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

note that first moment estimate is not tracked

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

again stuck in a saddle if sparse gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise control - so w_x and w_y update at different rates

But can end up stopping early if initial gradients high

RMSProp

exponential decay moving average less weight to older gradients

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

note that first moment estimate is not tracked

Optimization

backpropagate optimally

SGD with Momentum

$$\mathbf{m}_{t+1} = \mu \mathbf{m}_t - \alpha \nabla l(\mathbf{w}_t; \mathbf{x}_i)$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{m}_{t+1}$$

gradient

Adagrad

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

Element wise

w_y

update at different rates

But can end up stopping early if initial gradients high

RMSProp

exponential decay moving average

less weight to older gradients

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1 - \beta)\mathbf{g}_t^2$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\mathbf{v}_{t+1} + \epsilon}} \odot \mathbf{g}_t$$

note that first moment estimate is not tracked

CAN IT BE MADE EVEN BETTER

Optimization

backpropagate optimally

ADAM

combine first momentum tracking from "SGD with momentum" with "exponential decay of RMSProp"

$\mathbb{E}[\widehat{\mathbf{m}}_{t+1}]$

$\mathbb{E}[\widehat{\mathbf{v}}_{t+1}]$

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g}_t$$

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1-\beta_2)\mathbf{g}_t^2$$

$$\widehat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}}$$

$$\widehat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1}+\epsilon}} \odot \widehat{\mathbf{m}}_{t+1}$$

Optimization

backpropagate optimally

ADAM

combine first momentum tracking from "SGD with momentum" with "exponential decay of RMSProp"

$\mathbb{E}[\widehat{\mathbf{m}}_{t+1}]$

$\mathbb{E}[\widehat{\mathbf{v}}_{t+1}]$

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1 - \beta_1)\mathbf{g}_t$$

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1 - \beta_2)\mathbf{g}_t^2$$

$$\widehat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1 - \beta_1^{t+1}}$$

$$\widehat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1 - \beta_2^{t+1}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1} + \epsilon}} \odot \widehat{\mathbf{m}}_{t+1}$$

Optimization

backpropagate optimally

ADAM

$\mathbb{E}[\widehat{\mathbf{m}}_{t+1}]$ $\mathbb{E}[\widehat{\mathbf{v}}_{t+1}]$ →

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g}_t$$
$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1-\beta_2)\mathbf{g}_t^2$$
$$\widehat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}}$$
$$\widehat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}}$$
$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1}} + \epsilon} \odot \widehat{\mathbf{m}}_{t+1}$$

combine first momentum tracking from "SGD with momentum" with "exponential decay of RMSProp"

Optimization

backpropagate optimally

ADAM

$\mathbb{E}[\widehat{\mathbf{m}}_{t+1}]$ $\mathbb{E}[\widehat{\mathbf{v}}_{t+1}]$

$$\mathbf{m}_{t+1} = \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g}_t$$

$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1-\beta_2)\mathbf{g}_t^2$$

$$\widehat{\mathbf{m}}_{t+1} = \frac{\mathbf{m}_{t+1}}{1-\beta_1^{t+1}}$$

$$\widehat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\alpha}{\sqrt{\widehat{\mathbf{v}}_{t+1}+\epsilon}} \odot \widehat{\mathbf{m}}_{t+1}$$

combine first momentum tracking from "SGD with momentum" with "exponential decay of RMSProp"

ADAMW

Optimization

backpropagate optimally

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

ADAMW

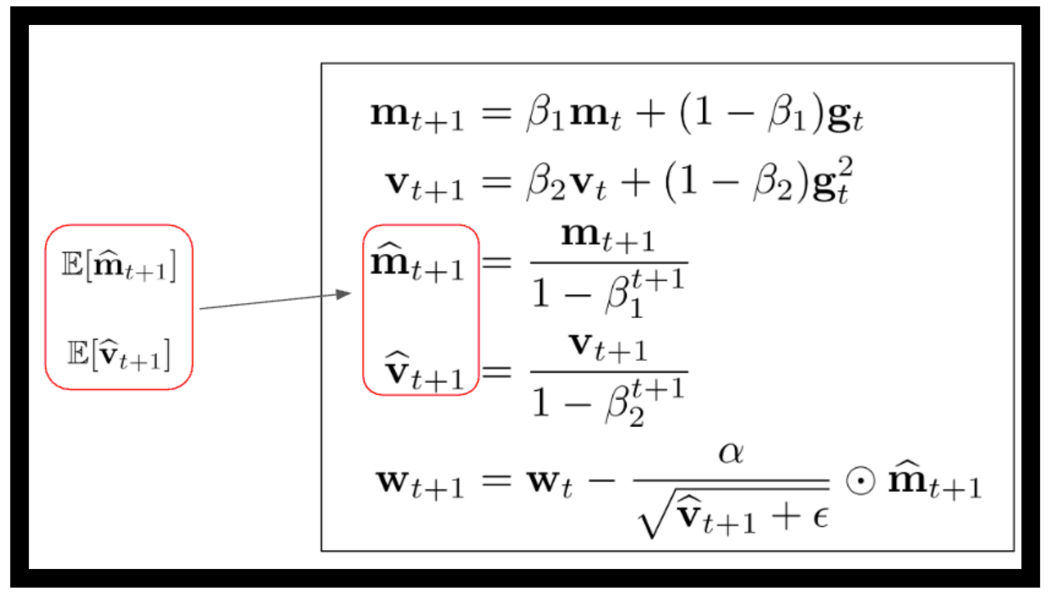
given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Add L2 regularization

Optimization

backpropagate optimally

ADAM



combine first momentum tracking from "SGD with momentum" with "exponential decay of RMSProp"

ADAMW

Add L2 regularization

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

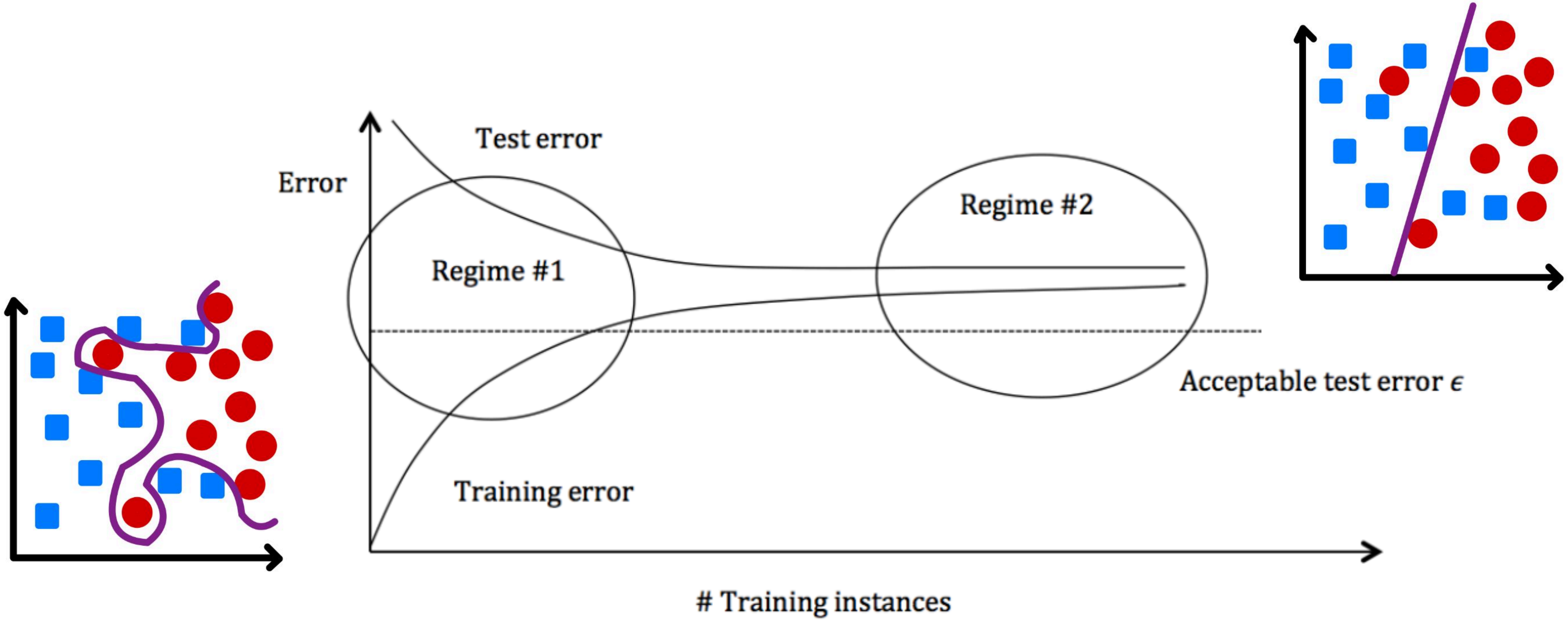
given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1-\beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1-\beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1}\right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Fundamentals

Regularization

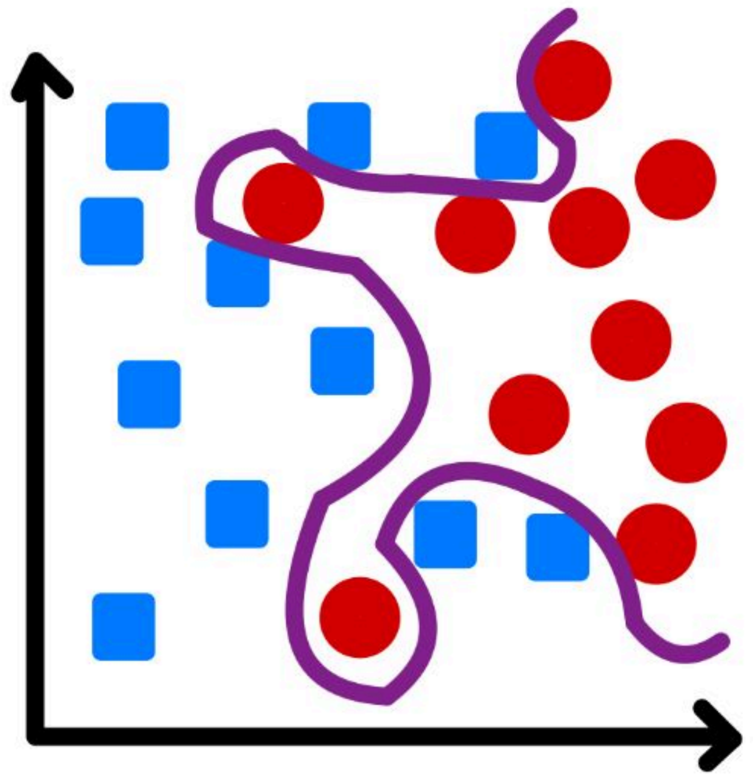
Snehal Bhagat

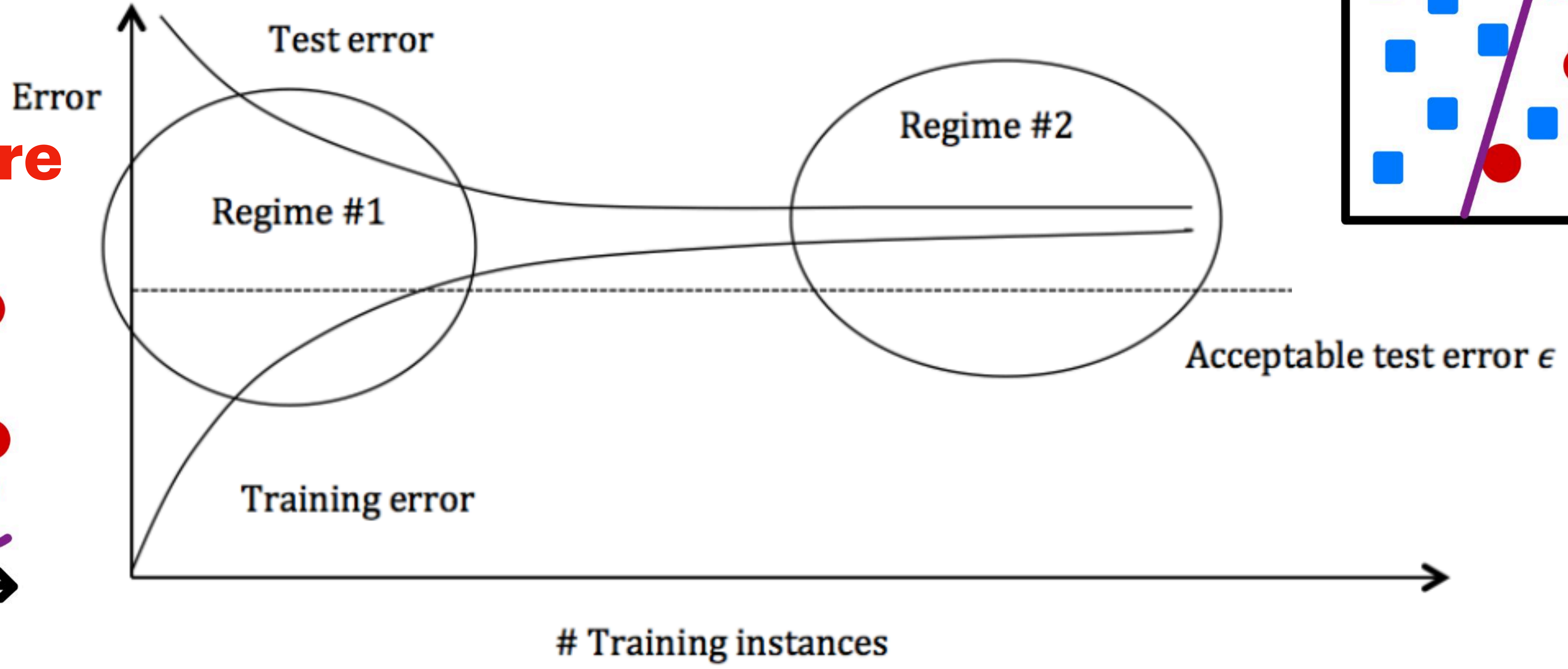
Regularization

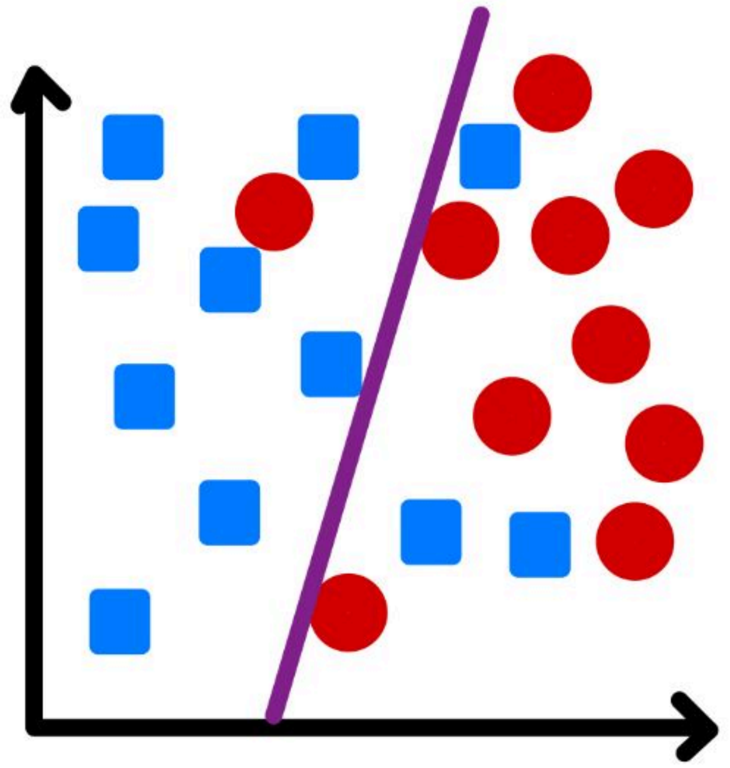


Regularization

OVERFIT
Needs to generalize more

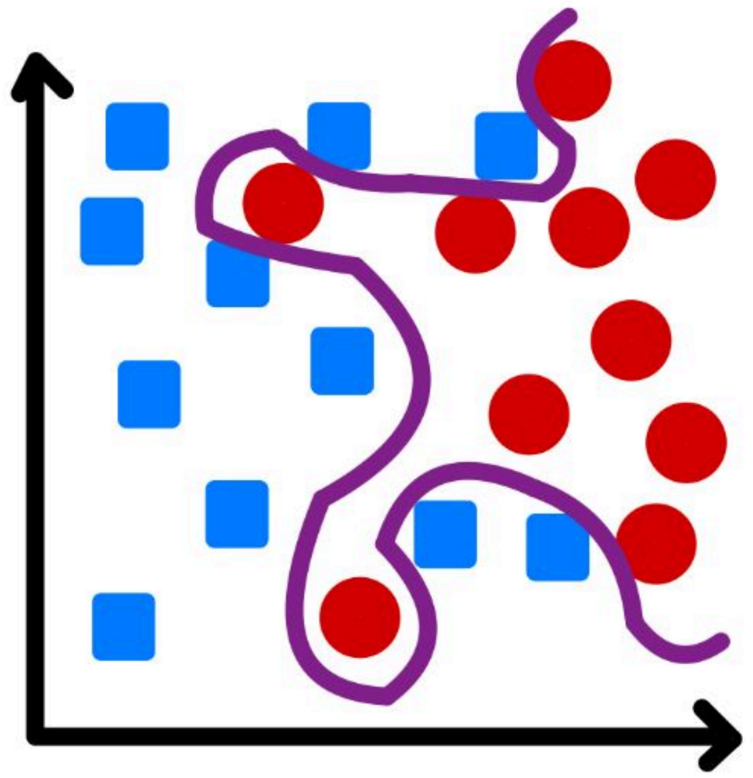


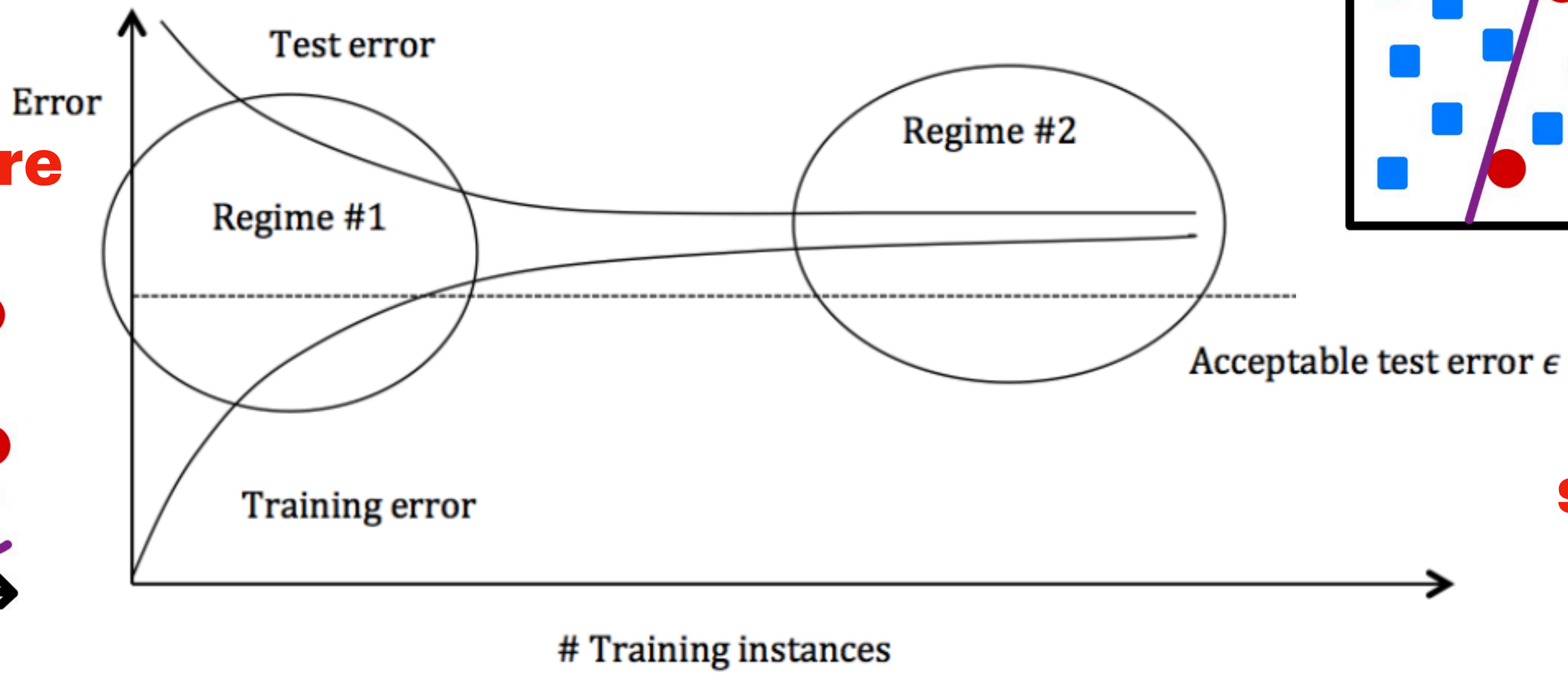


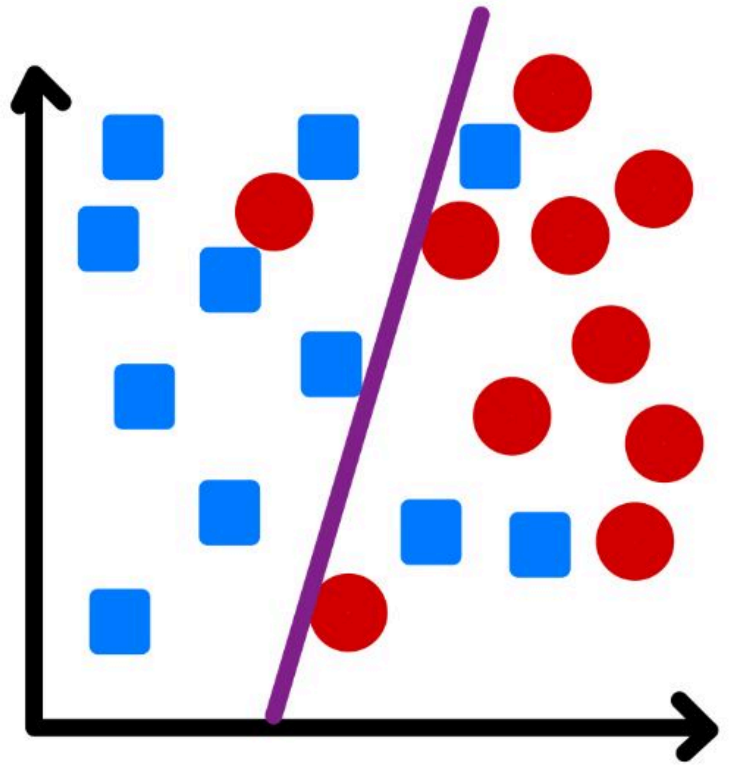


Regularization

OVERFIT
Needs to generalize more





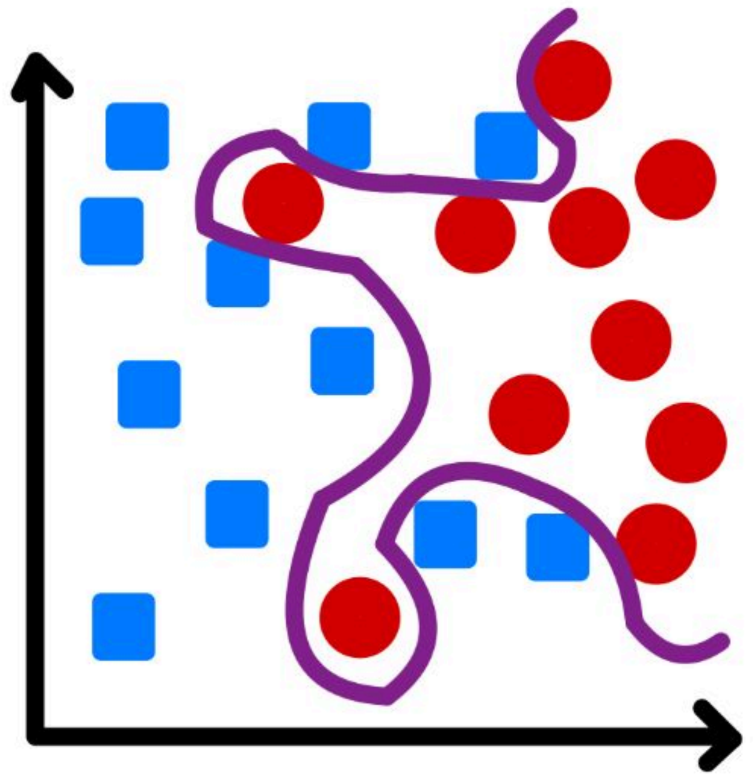


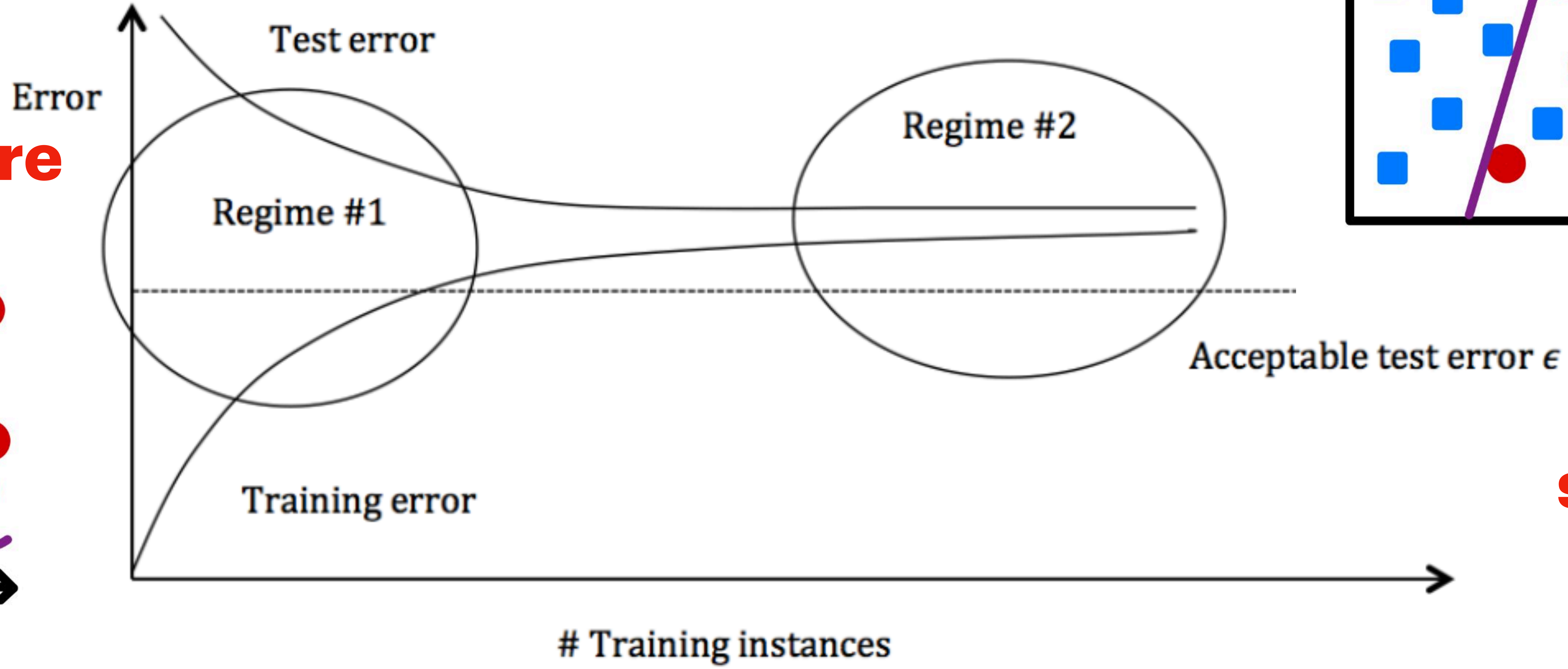
UNDERFIT
Needs to specialize more

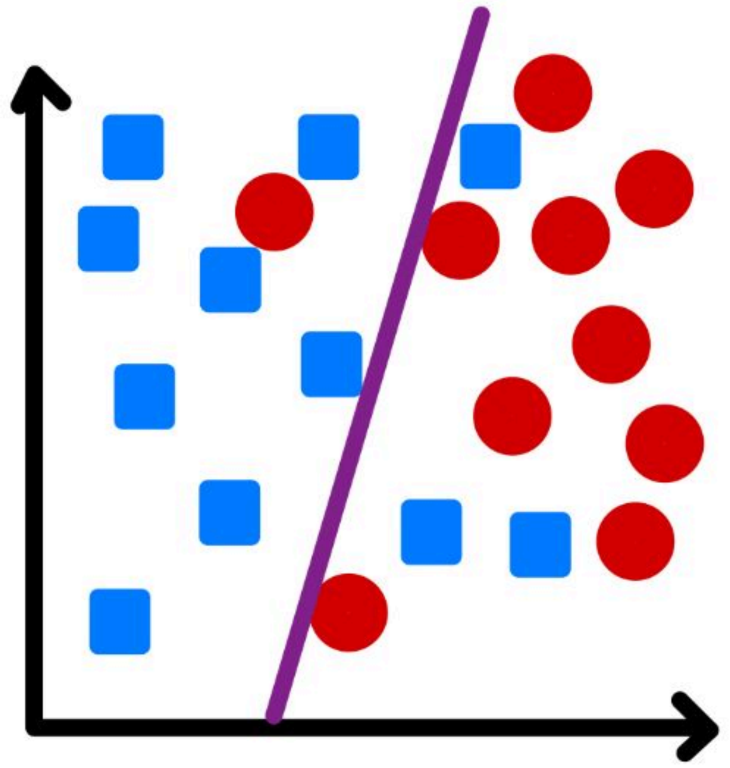
Regularization

ensures model doesn't overfit

OVERFIT
Needs to generalize more





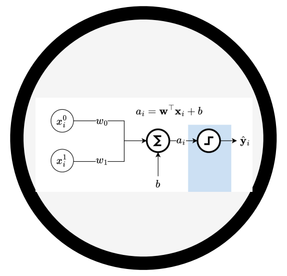


UNDERFIT
Needs to specialize more

Regularization

ensures model doesn’t overfit

Regularization



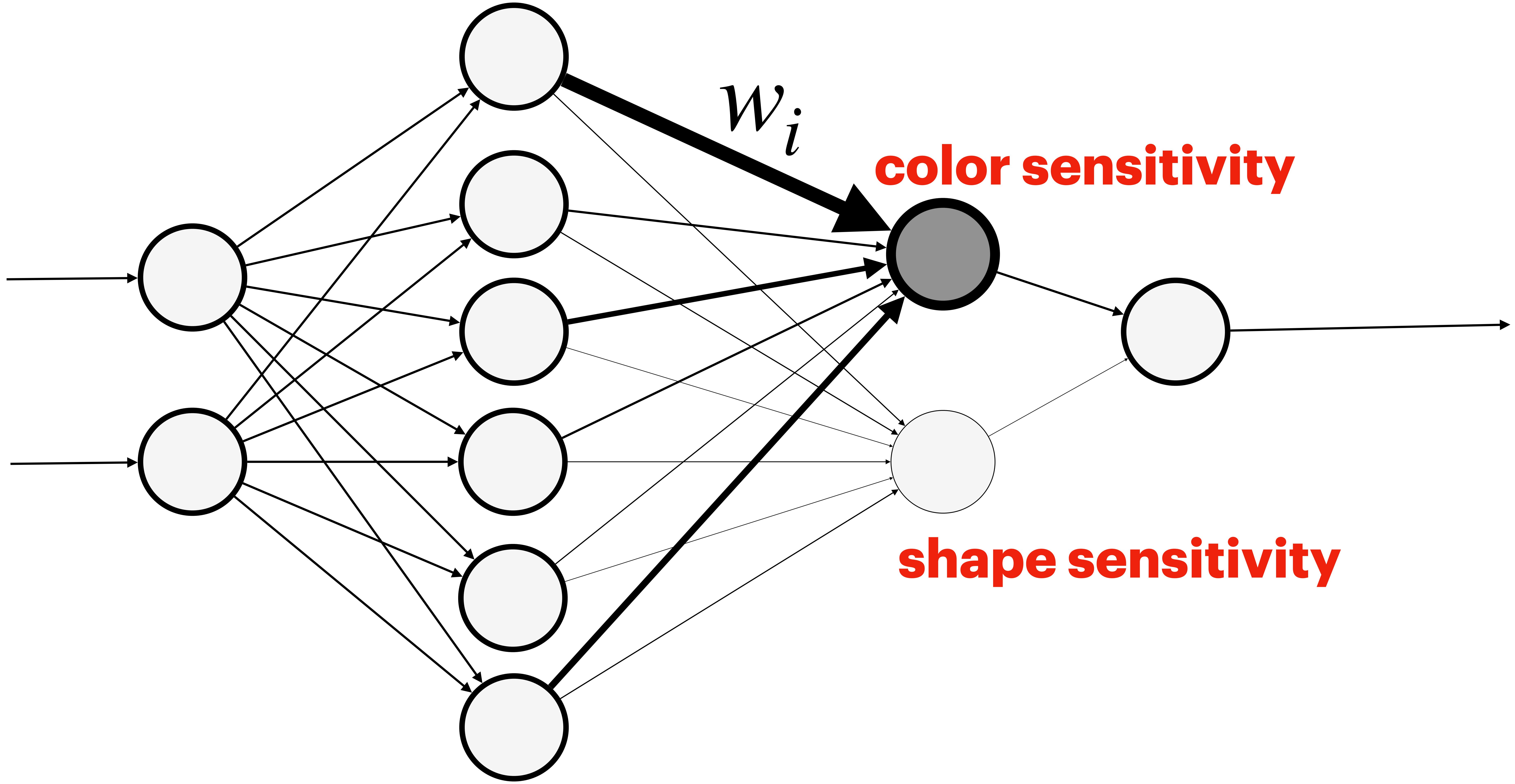
w_i

color sensitivity

shape sensitivity

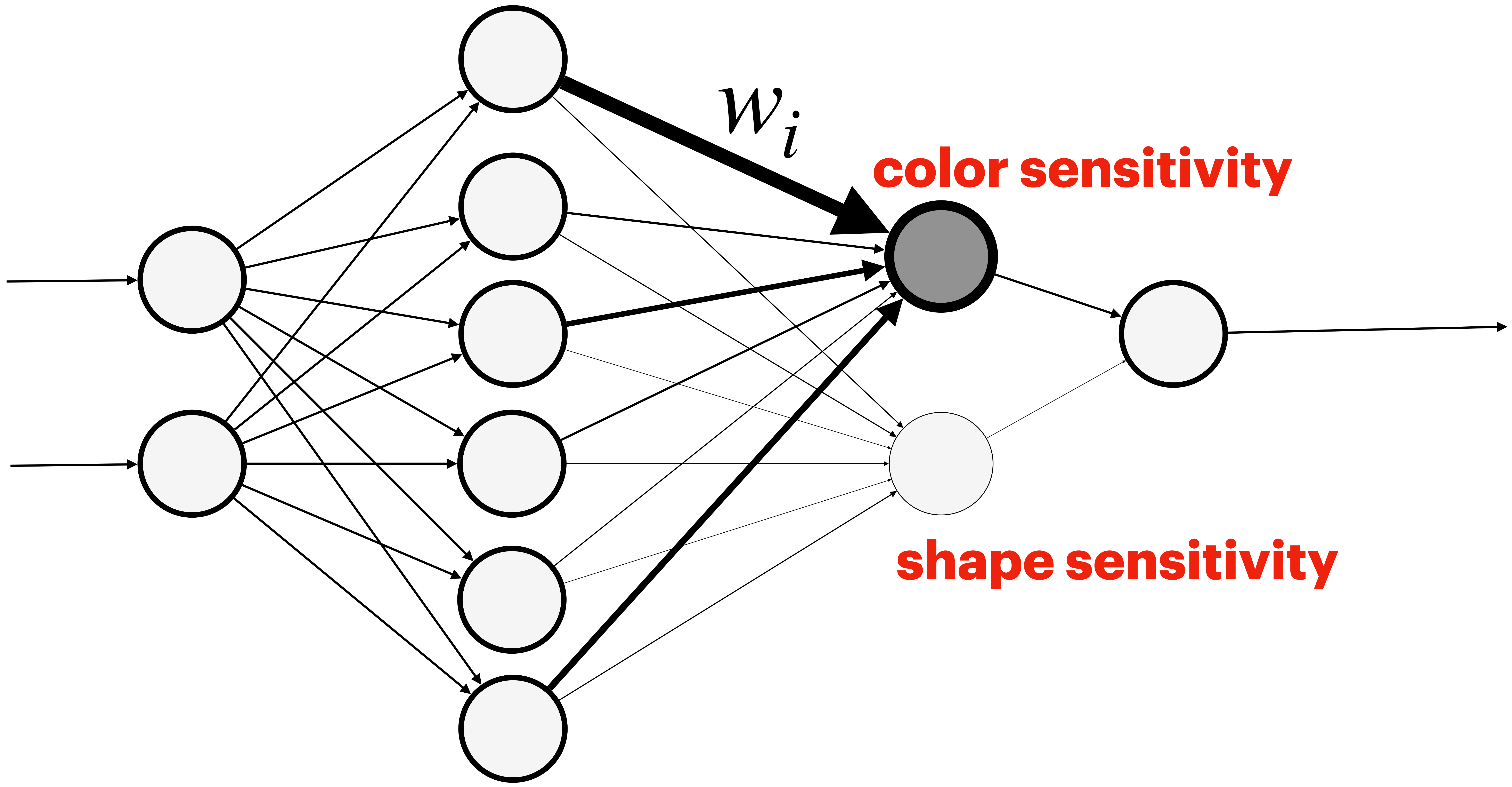
Regularization - ensures model doesn't overfit

Why does overfitting happen (naive)



Regularization - ensures model doesn't overfit

Why does overfitting happen (naive)



so ideally we would like more evenly distributed weights

Regularization - ensures model doesn't overfit

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

Regularization - ensures model doesn't overfit

So what are some techniques to tackle this

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven

Regularization - ensures model doesn't overfit

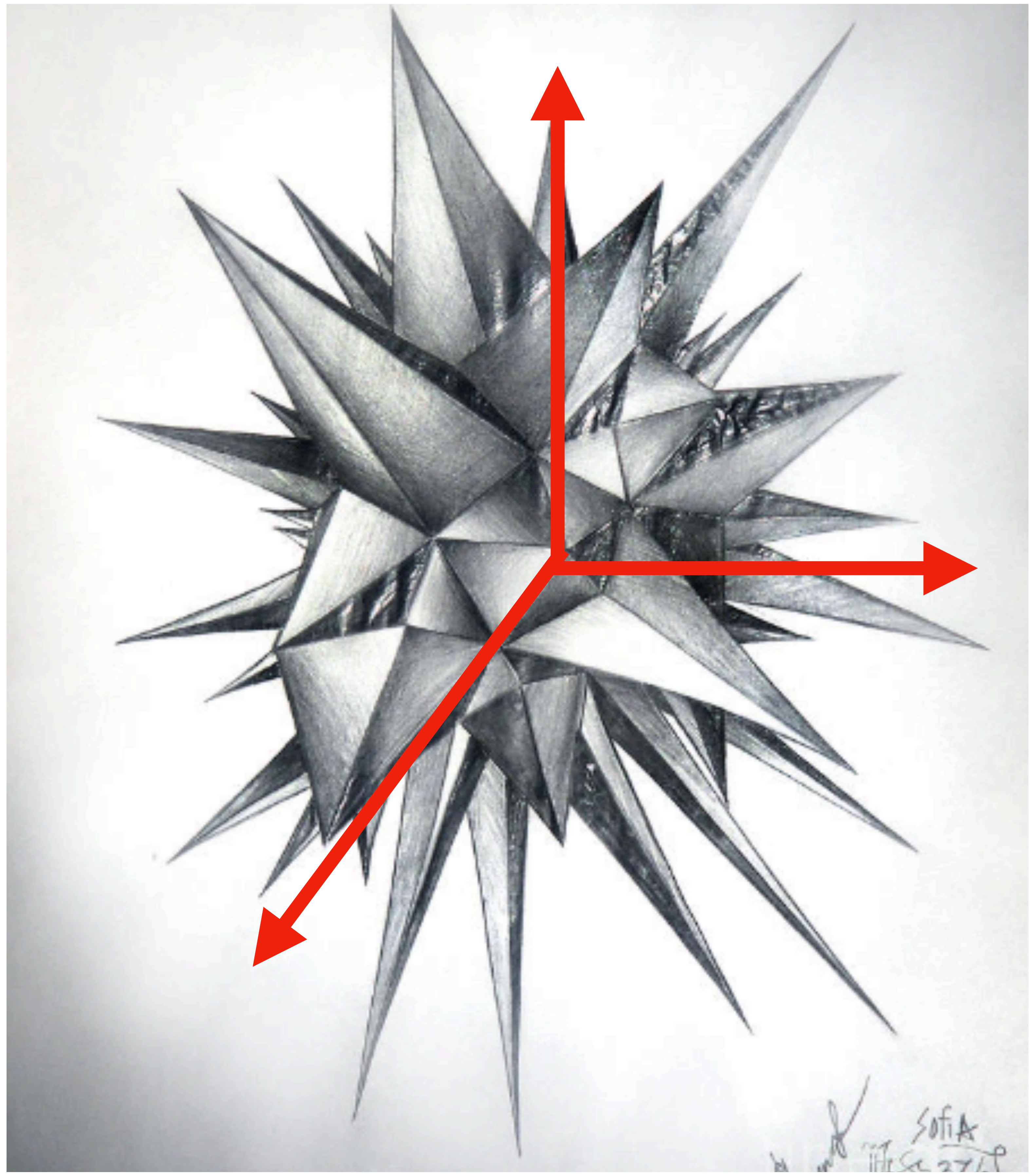
So what are some techniques to tackle this

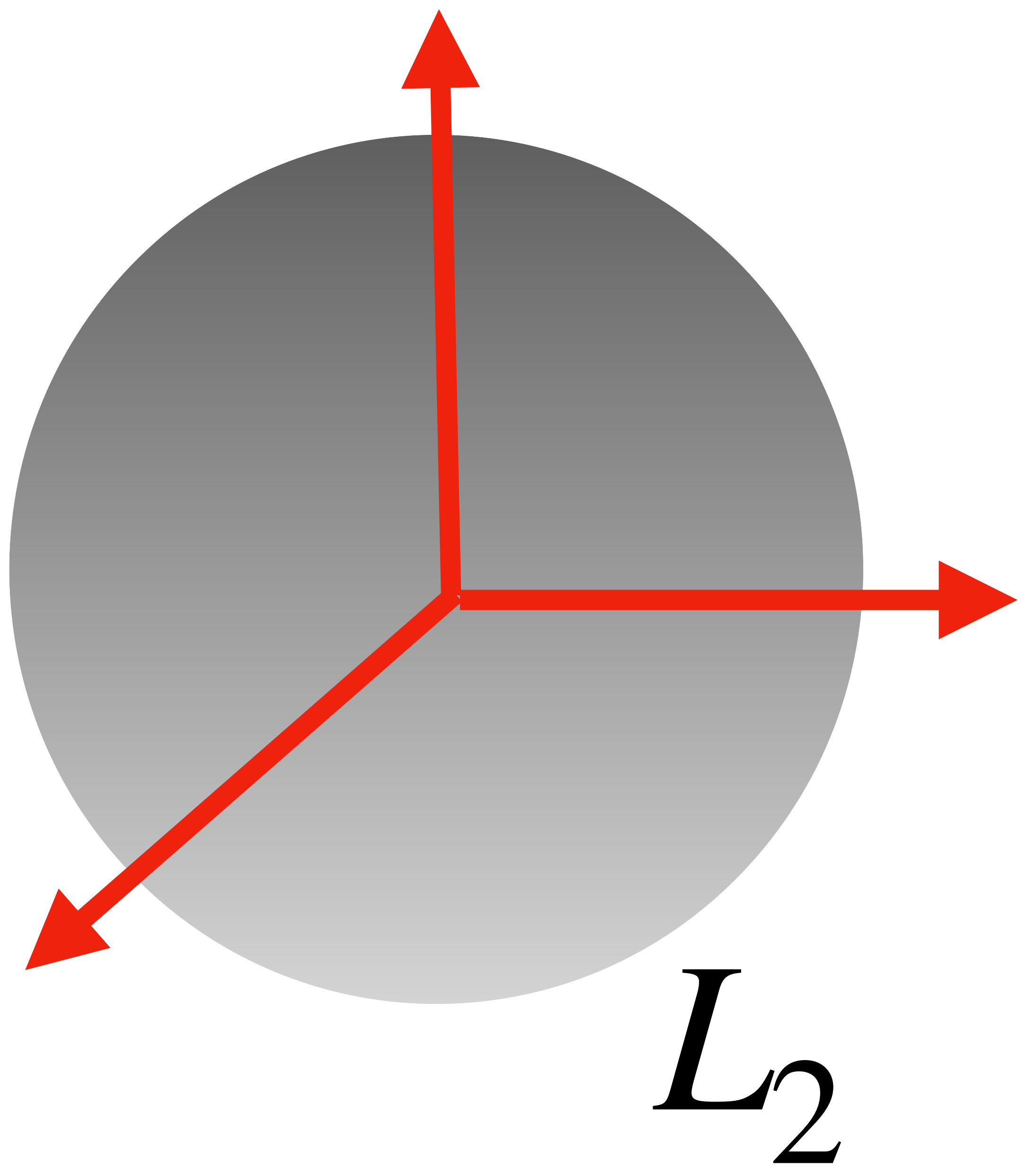
- Early Stopping - before it has the time to become uneven
- L1/L2 regularization

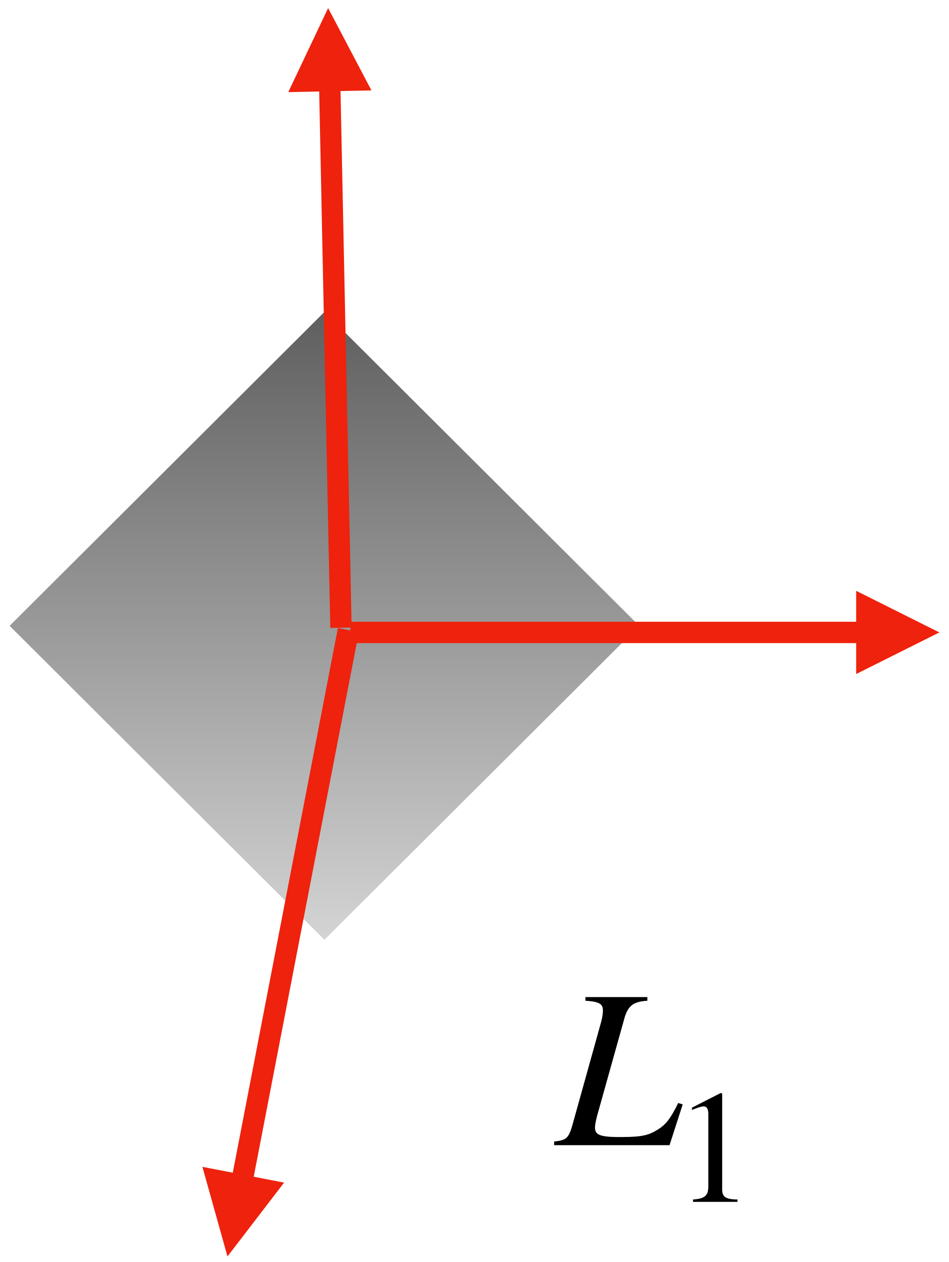
Regularization

- L1/L2 regularization

Naive Visualization of Weights







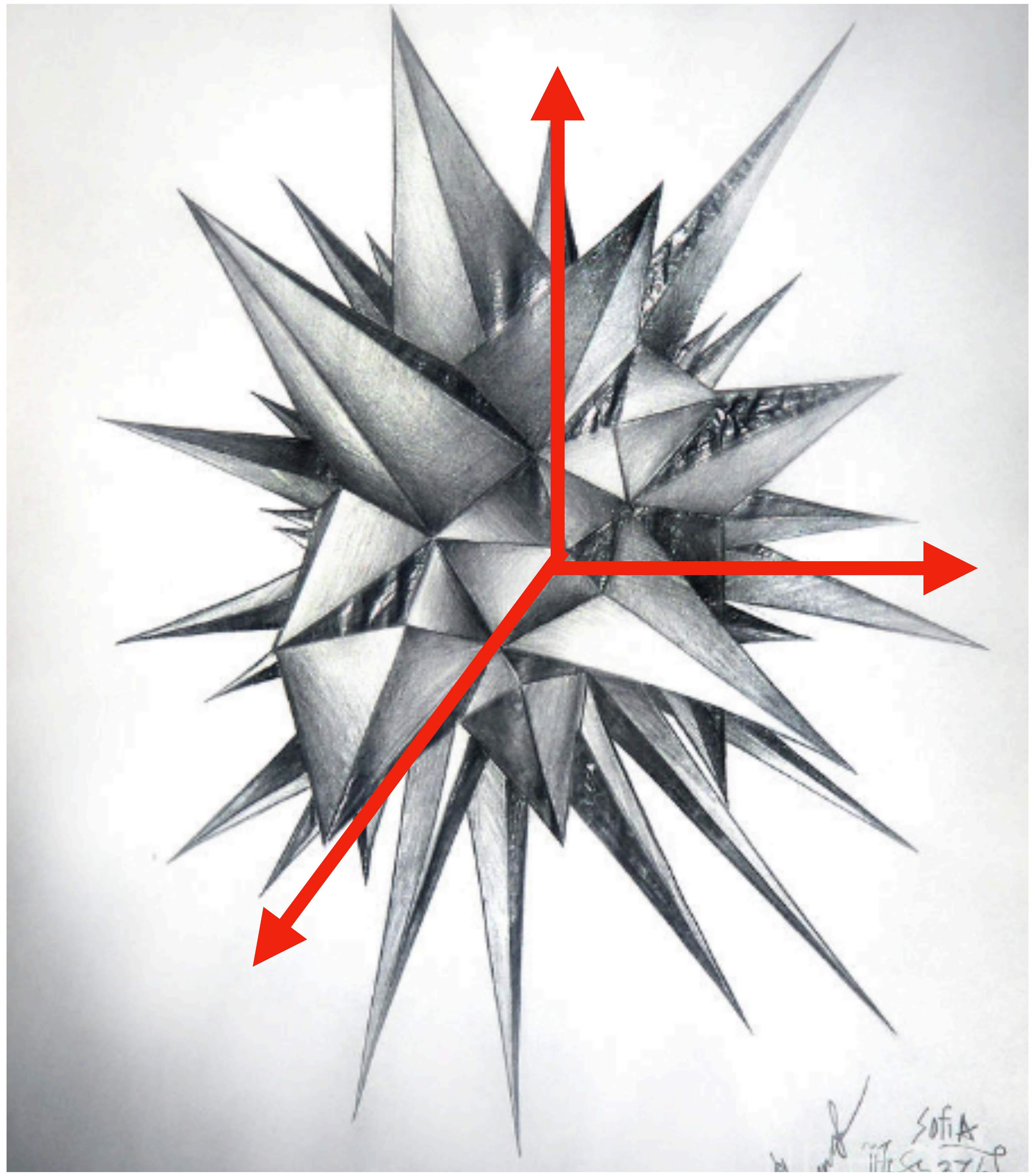
$$\underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}(\mathbf{w}, D) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

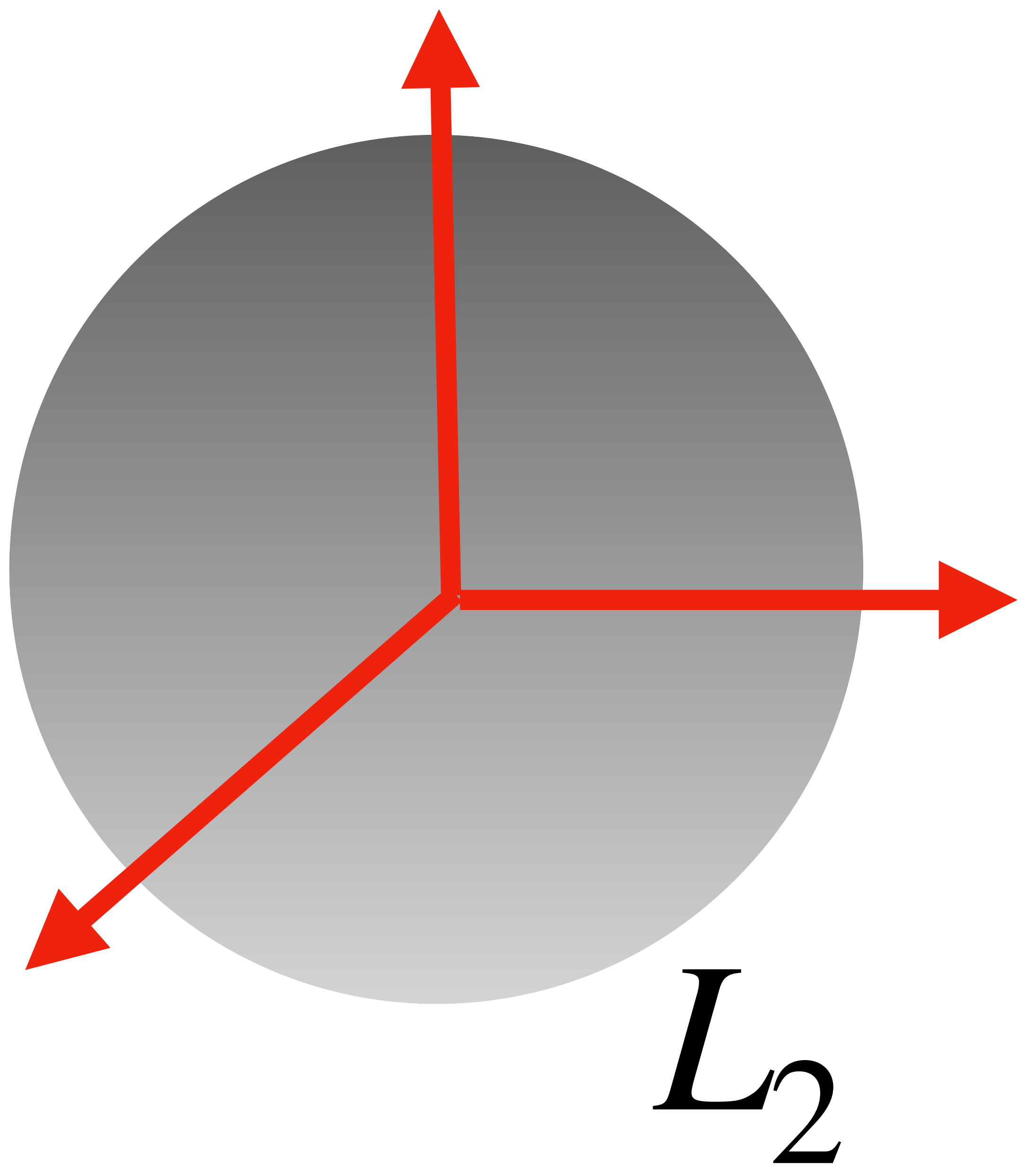
$$\underset{\mathbf{w}}{\operatorname{argmin}} \mathcal{L}(\mathbf{w}, D) + \lambda|\mathbf{w}|$$

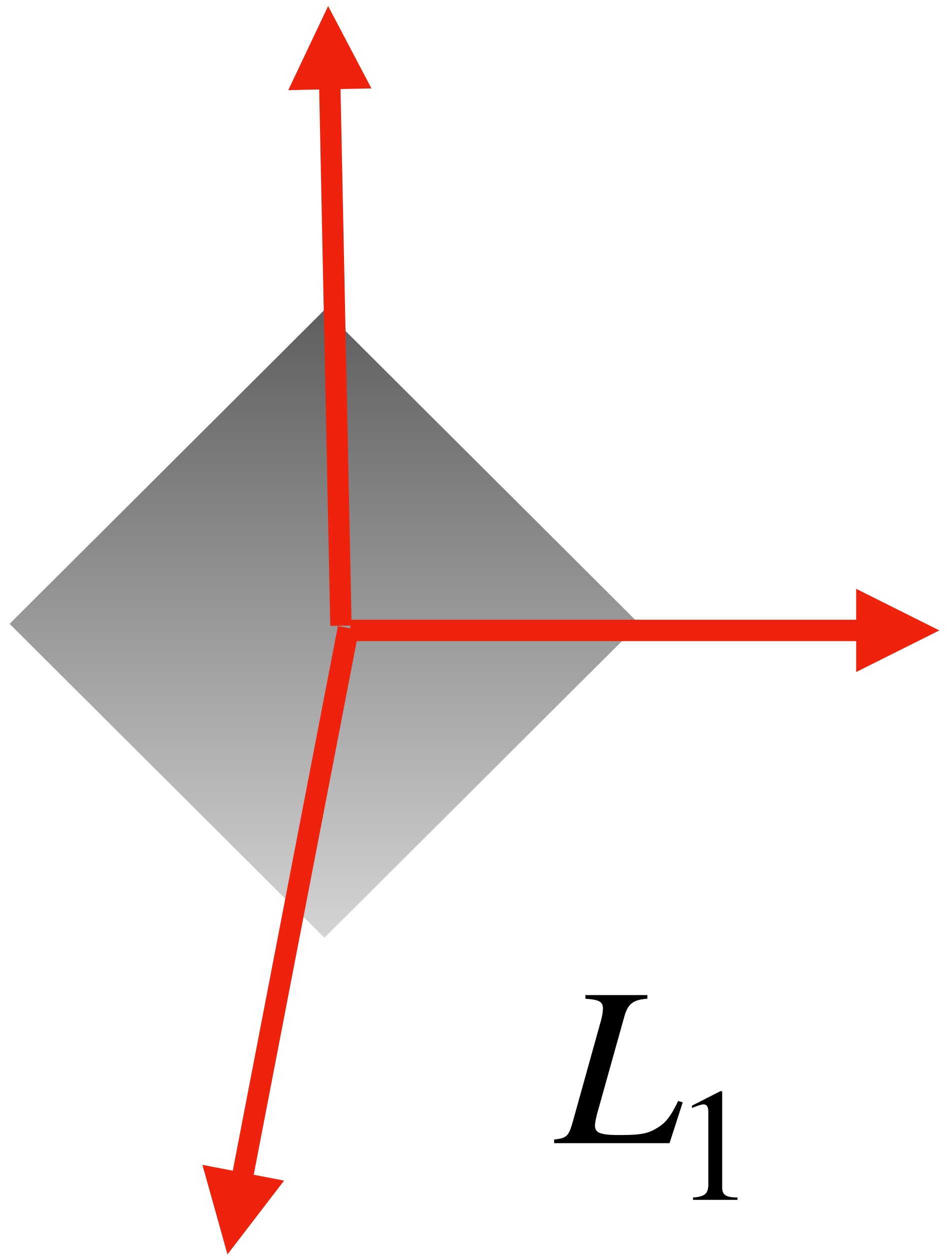
Regularization - ensures model doesn't overfit

- L1/L2 regularization

Naive Visualization of Weights







$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w}, D) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w}, D) + \lambda|\mathbf{w}|$$

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

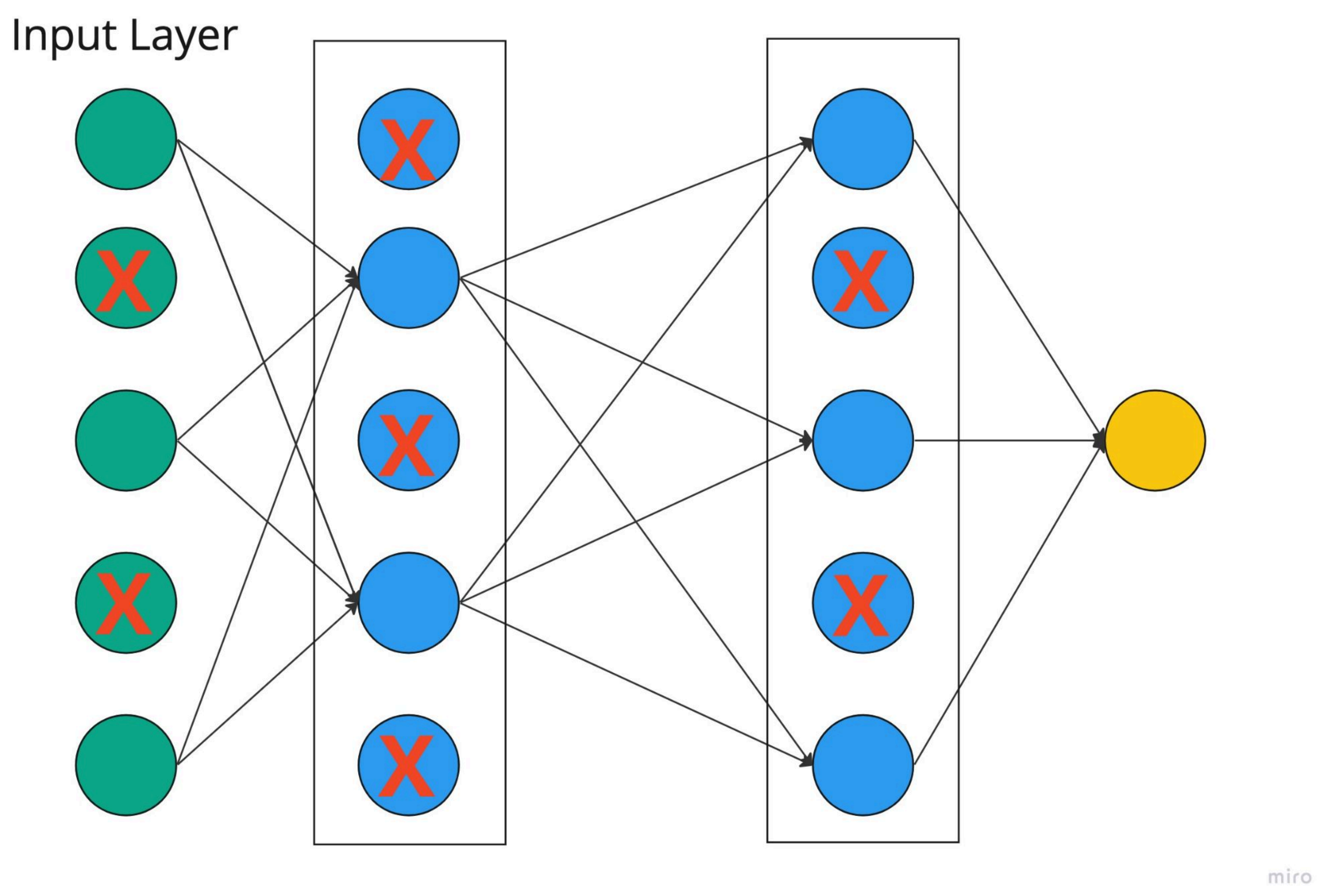
- Early Stopping - before it has the time to become uneven
- L1/L2 regularization

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

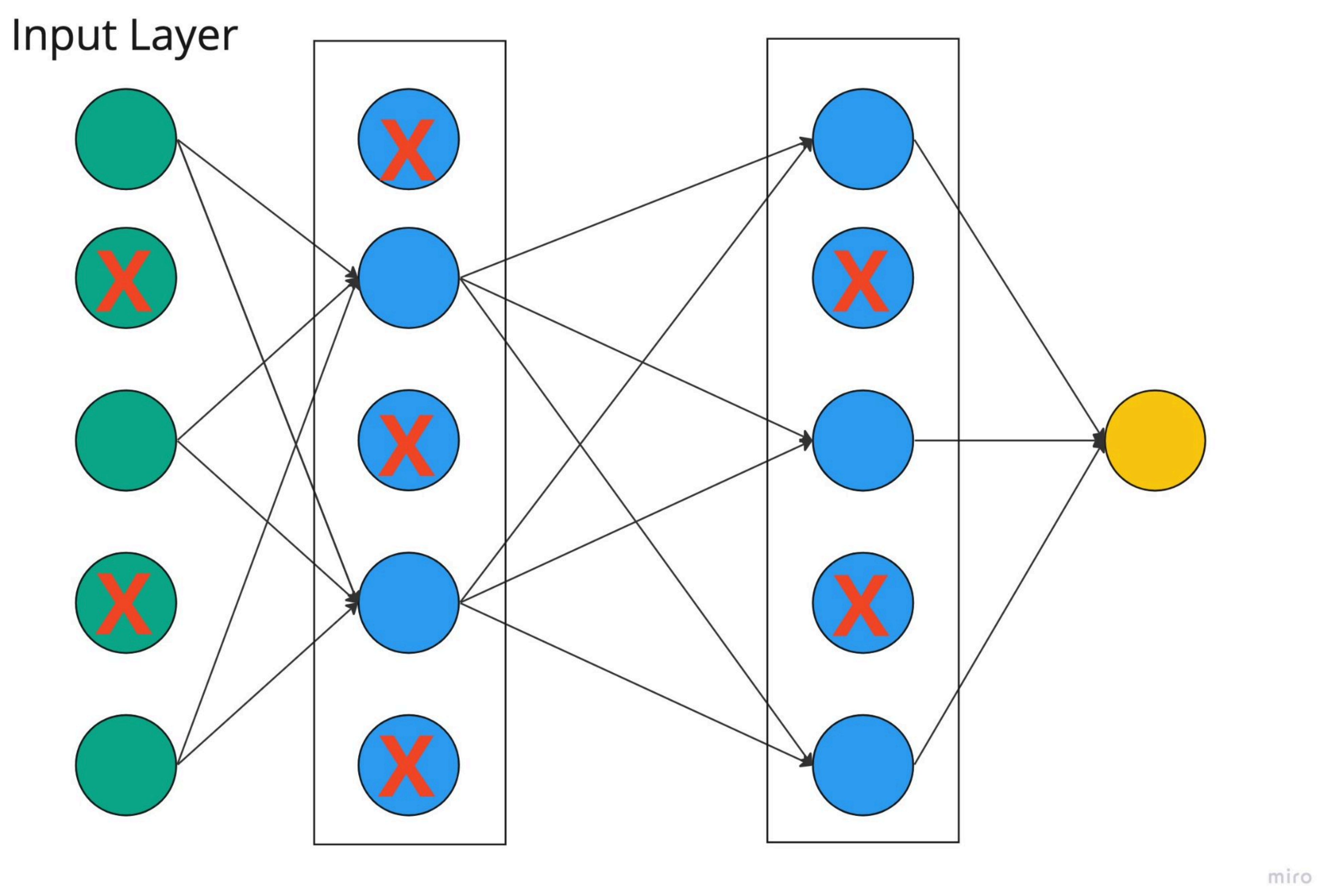
- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout

Regularization - ensures model doesn’t overfit



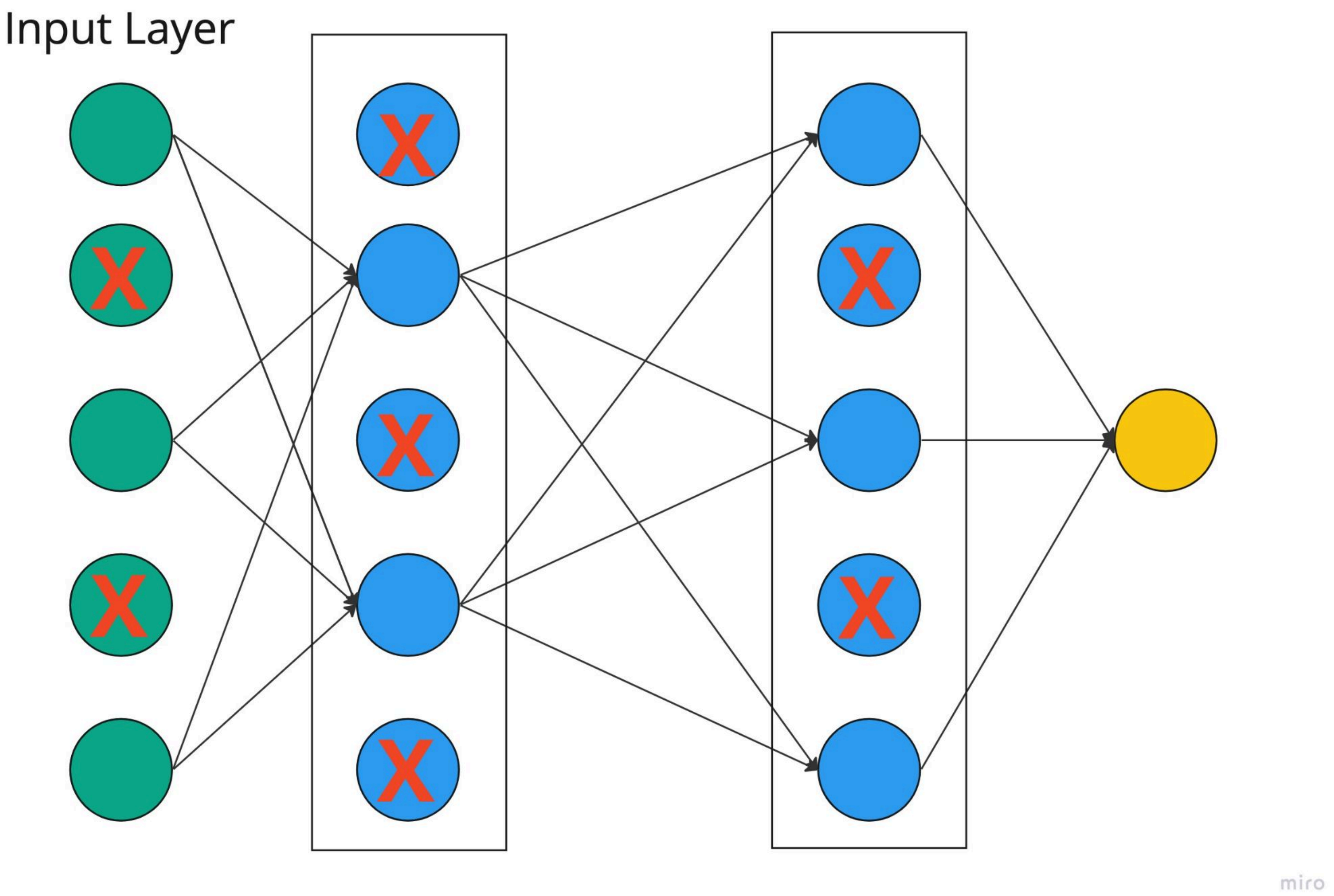
Regularization - ensures model doesn't overfit

- Dropout



Regularization - ensures model doesn’t overfit

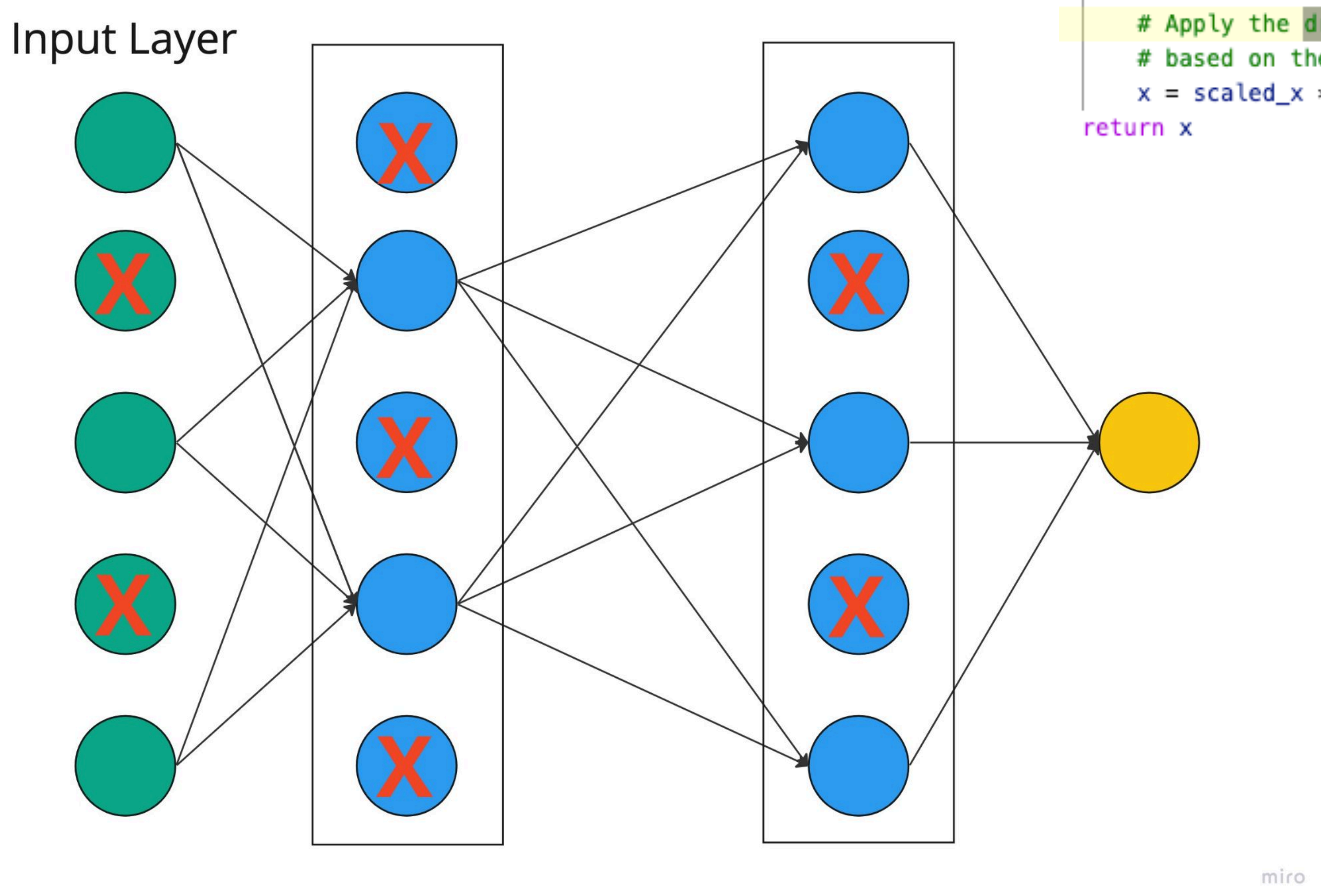
- Dropout



Ensures no over-dependence on specific neurons

Regularization - ensures model doesn't overfit

- Dropout




```
if self.training:
    #########
    # Generate a binary mask where each element has a value of True
    # if it is retained (not dropped out) and False if it is dropped out.
    mask = torch.rand(x.size()) > self.p

    # Scale the input tensor 'x' to compensate for the dropped out elements.
    # This scaling ensures that the expected value remains the same.
    scaled_x = x / (1 - self.p)

    # Apply the dropout mask to the input tensor by zeroing out the elements
    # based on the mask.
    x = scaled_x * mask.float()
return x
```

Ensures no over-dependence on specific neurons

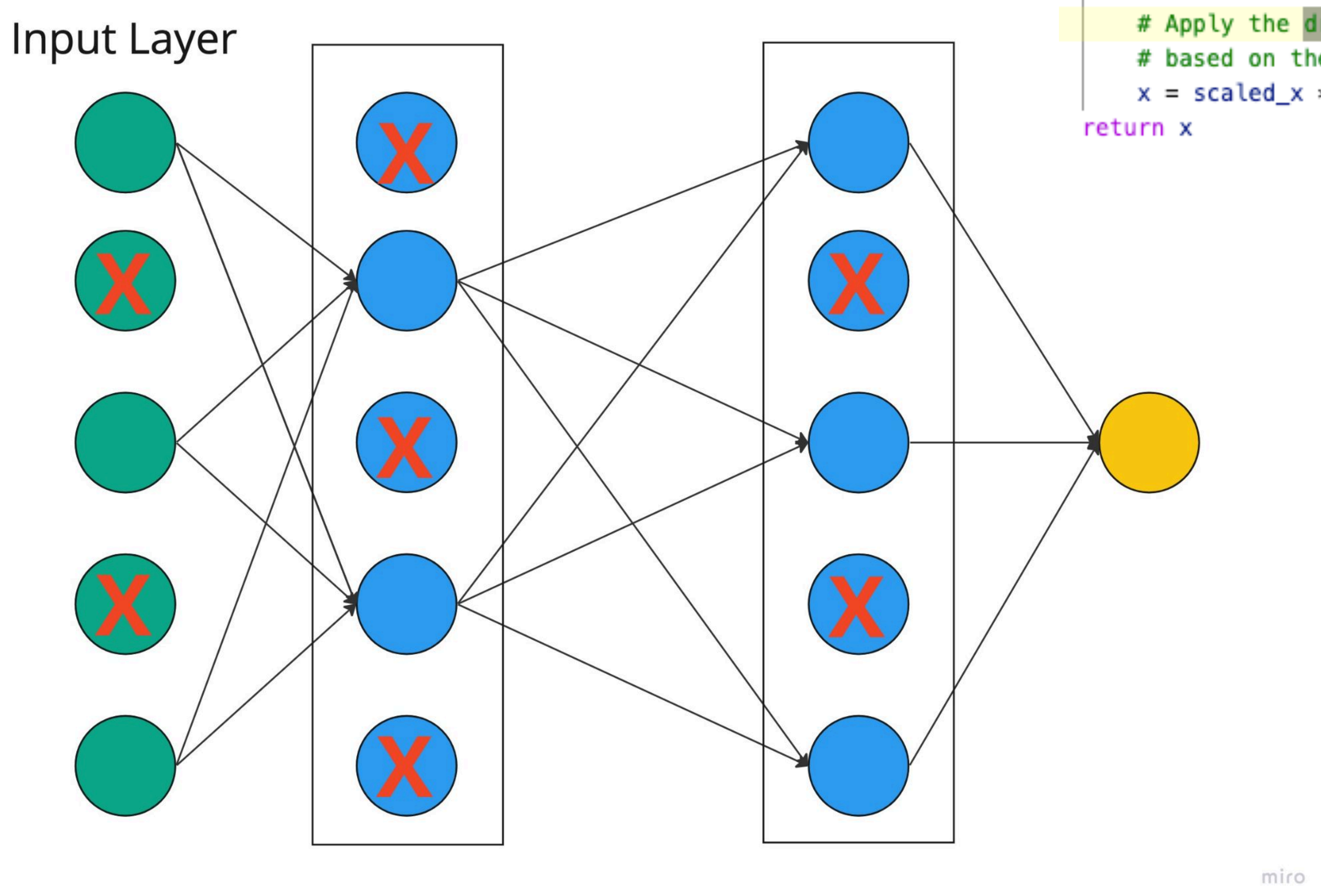
Regularization - ensures model doesn’t overfit

- Dropout

```
if self.training:
    #########
    # Generate a binary mask where each element has a value of True
    # if it is retained (not dropped out) and False if it is dropped out.
    mask = torch.rand(x.size()) > self.p

    # Scale the input tensor 'x' to compensate for the dropped out elements.
    # This scaling ensures that the expected value remains the same.
    scaled_x = x / (1 - self.p)

    # Apply the dropout mask to the input tensor by zeroing out the elements
    # based on the mask.
    x = scaled_x * mask.float()
return x
```



either
divide by 1-p in training time
OR
you can multiply by 1-p in test time

Ensures no over-dependence on specific neurons

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout

Regularization - ensures model doesn't overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout
- Increased Learning Rate

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout
- Increased Learning Rate
- Skip Connections

Regularization - ensures model doesn't overfit

Regularization - ensures model doesn’t overfit

- Increased Learning Rate

Regularization - ensures model doesn’t overfit

- Increased Learning Rate

Noisier updates

Regularization - ensures model doesn't overfit

- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Regularization - ensures model doesn't overfit

- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Improves generalization

Regularization - ensures model doesn't overfit

- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Improves generalization

- Skip Connections

Regularization - ensures model doesn't overfit

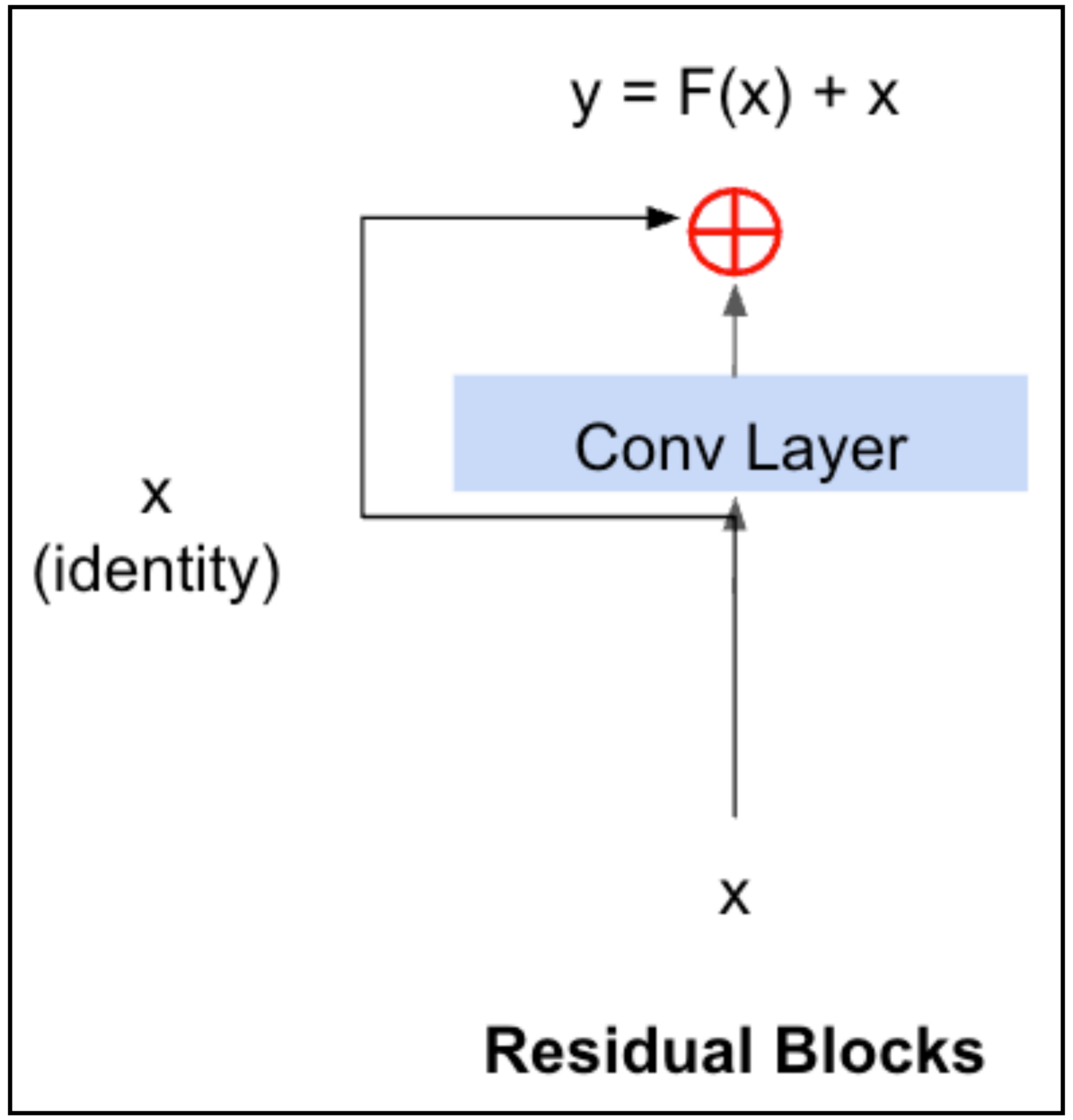
- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Improves generalization

- Skip Connections



Regularization - ensures model doesn’t overfit

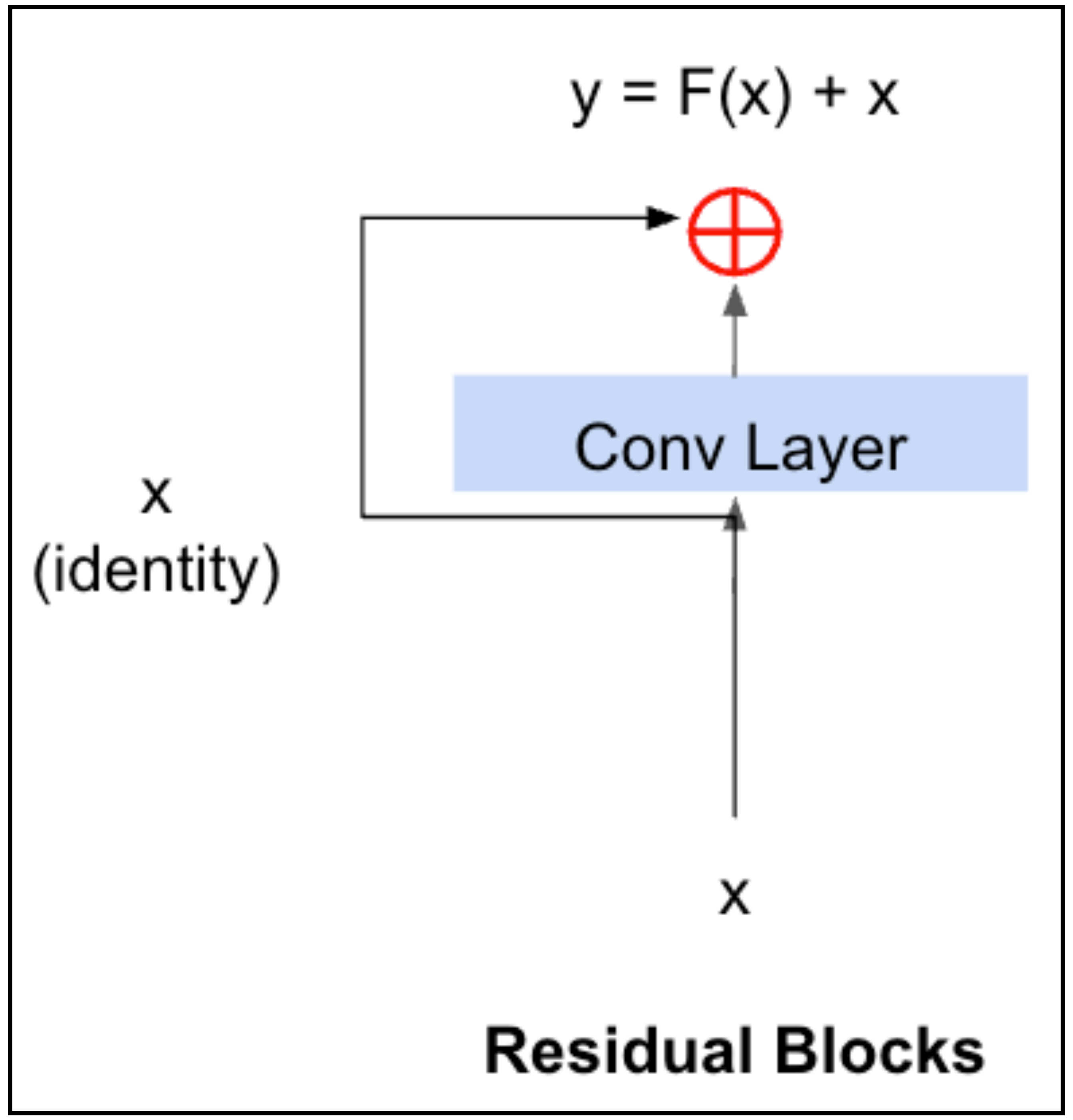
- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Improves generalization

- Skip Connections



$$\frac{\delta L}{\delta x} = \frac{\delta L}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta L}{\delta y}\boxed{(F'(x))}$$

Plain

$$\frac{\delta L}{\delta x} = \frac{\delta L}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta L}{\delta y}\boxed{(1 + F'(x))}$$

ResNet

Regularization - ensures model doesn’t overfit

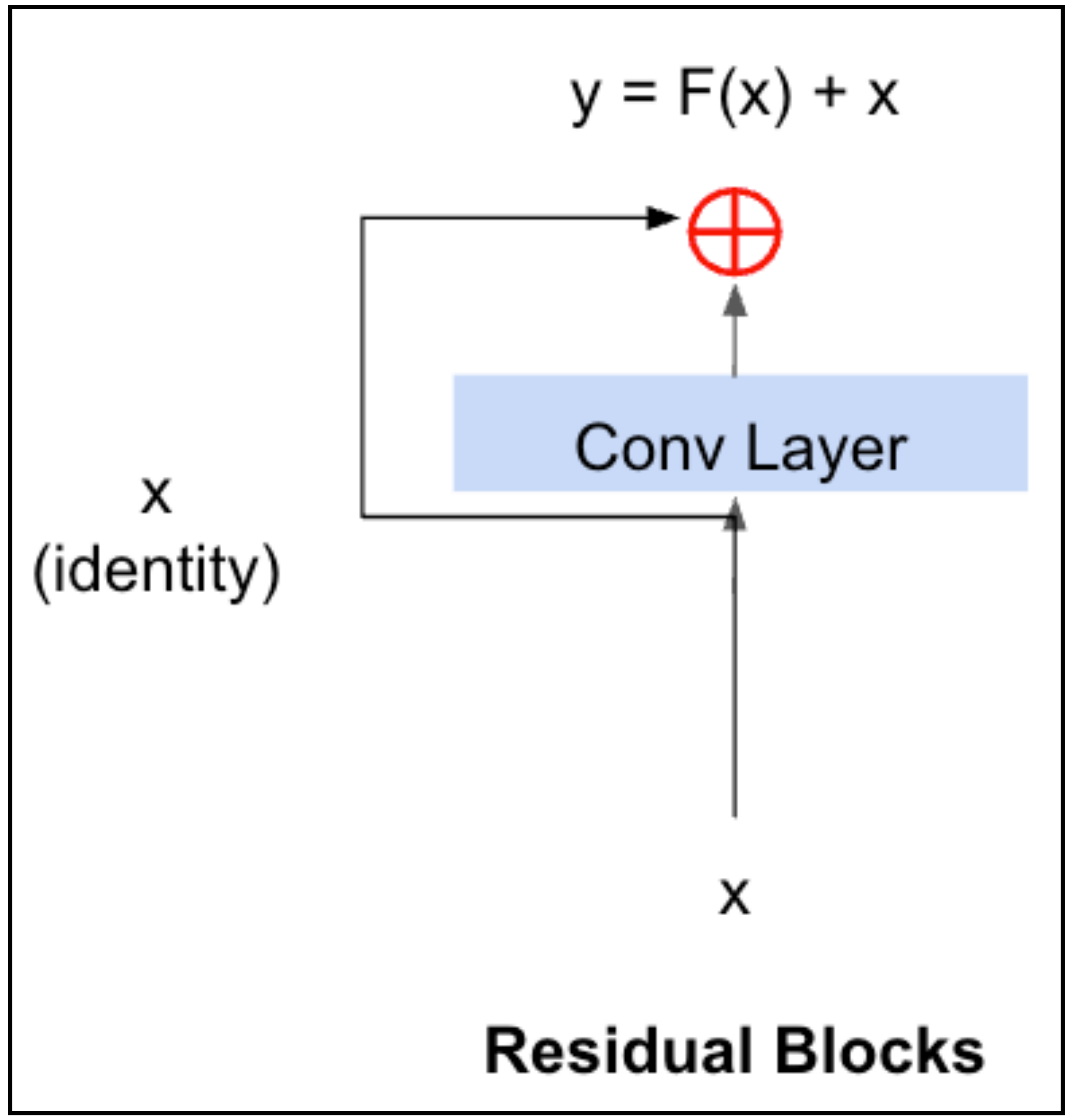
- Increased Learning Rate

Noisier updates

Stochastically, more of the loss landscape accounted for in gradient descent

Improves generalization

- Skip Connections



$$\frac{\delta L}{\delta x} = \frac{\delta L}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta L}{\delta y}(F'(x))$$

Plain

$$\frac{\delta L}{\delta x} = \frac{\delta L}{\delta y} * \frac{\delta y}{\delta x} = \frac{\delta L}{\delta y}(1 + F'(x))$$

ResNet

Gradient that would’ve otherwise vanished

Regularization - ensures model doesn’t overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout
- Learning Rate

Regularization - ensures model doesn't overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout
- Learning Rate
- Batch Norm

Regularization - ensures model doesn't overfit

So what are some techniques to tackle this

- Early Stopping - before it has the time to become uneven
- L1/L2 regularization
- Dropout
- Learning Rate
- Batch Norm
- Layer Norm

Regularization - ensures model doesn't overfit

- Batch Norm

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Regularization - ensures model doesn't overfit

- Batch Norm

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Can't do at test - time

Regularization - ensures model doesn't overfit

- Batch Norm

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Can't do at test - time

Maintain running average

Regularization - ensures model doesn't overfit

- Batch Norm

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Can't do at test - time

Maintain running average

Batch Normalization for **fully-connected** networks

x: N × D

Normalize ↓

μ,σ: 1 × D

ɣ,β: 1 × D

y = ɣ(x-μ)/σ+β

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

x: N×C×H×W

Normalize ↓ ↓ ↓

μ,σ: 1×C×1×1

ɣ,β: 1×C×1×1

y = ɣ(x-μ)/σ+β

Regularization - ensures model doesn't overfit

Layer Normalization

Batch Normalization for
fully-connected networks

$\mathbf{x}: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$\mathbf{x}: N \times D$

Normalize

$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Regularization - ensures model doesn't overfit

Layer Normalization

Batch Normalization for fully-connected networks

$x: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x - \mu)/\sigma + \beta$

Layer Normalization for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$x: N \times D$

Normalize

$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$y = \gamma(x - \mu)/\sigma + \beta$

will be useful in Transformers

Regularization - ensures model doesn't overfit

Layer Normalization

Batch Normalization for
fully-connected networks

$$x: N \times D$$

Normalize

$$\mu, \sigma: 1 \times D$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

Layer Normalization for
fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

$$x: N \times D$$

Normalize

$$\mu, \sigma: N \times 1$$

$$\gamma, \beta: 1 \times D$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

will be useful in Transformers

Regularization - ensures model doesn't overfit

Instance Normalization

Batch Normalization for convolutional networks

$x: N \times C \times H \times W$

Normalize

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x - \mu)/\sigma + \beta$

Instance Normalization for convolutional networks
Same behavior at train / test!

$x: N \times C \times H \times W$

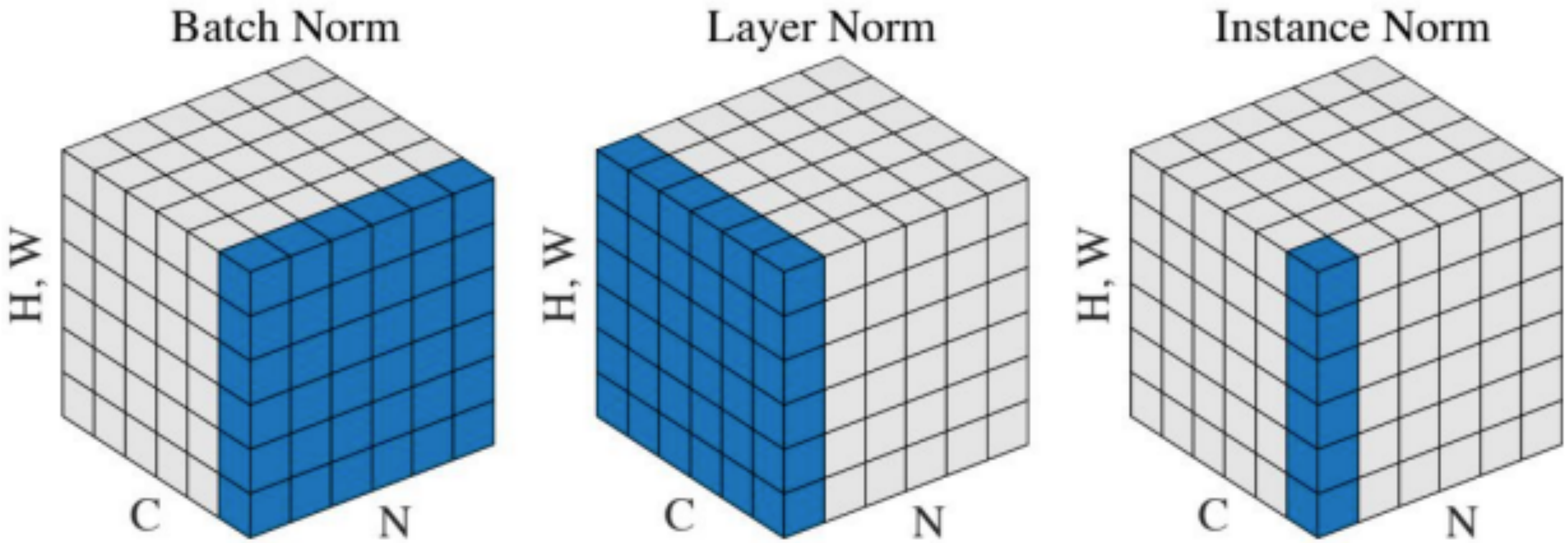
Normalize

$\mu, \sigma: N \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x - \mu)/\sigma + \beta$

Regularization - ensures model doesn't overfit



Few-Shot Principles

Few-Shot Principles

A. Inputs need encoding

Few-Shot Principles

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function

Few-Shot Principles

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function
C. Loss ~ distance from ideal distribution

Few-Shot Principles

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function
C. Loss ~ distance from ideal distribution
D. Purpose of training is to maximize the likelihood of training data

Few-Shot Principles

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function
C. Loss ~ distance from ideal distribution
D. Purpose of training is to maximize the likelihood of training data
E. Optimization algorithms help backprop reach low loss optimally

Few-Shot Principles

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function
C. Loss ~ distance from ideal distribution
D. Purpose of training is to maximize the likelihood of training data
E. Optimization algorithms help backprop reach low loss optimally
F. Regularization ensures you don't overfit

Few-Shot Principles

they prime you to understand deep-learning

A. Inputs need encoding
B. Training/Learning → weight/parameter updates → Backpropagation → Loss function
C. Loss ~ distance from ideal distribution
D. Purpose of training is to maximize the likelihood of training data
E. Optimization algorithms help backprop reach low loss optimally
F. Regularization ensures you don't overfit

HIDDEN LAYERS ~ MODEL