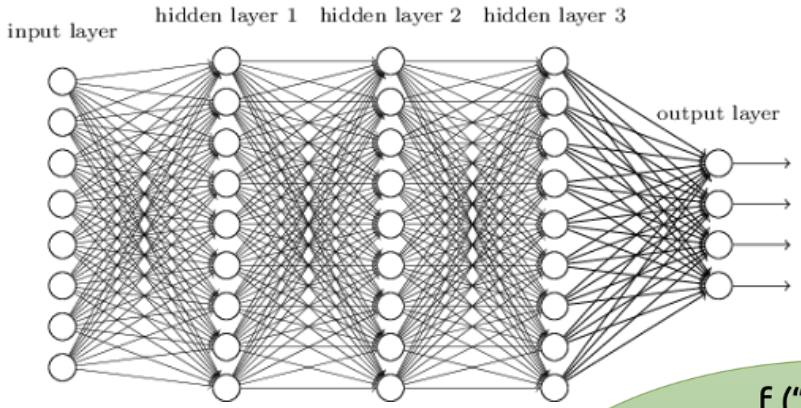
Day 5: Classification and Hyperparameters

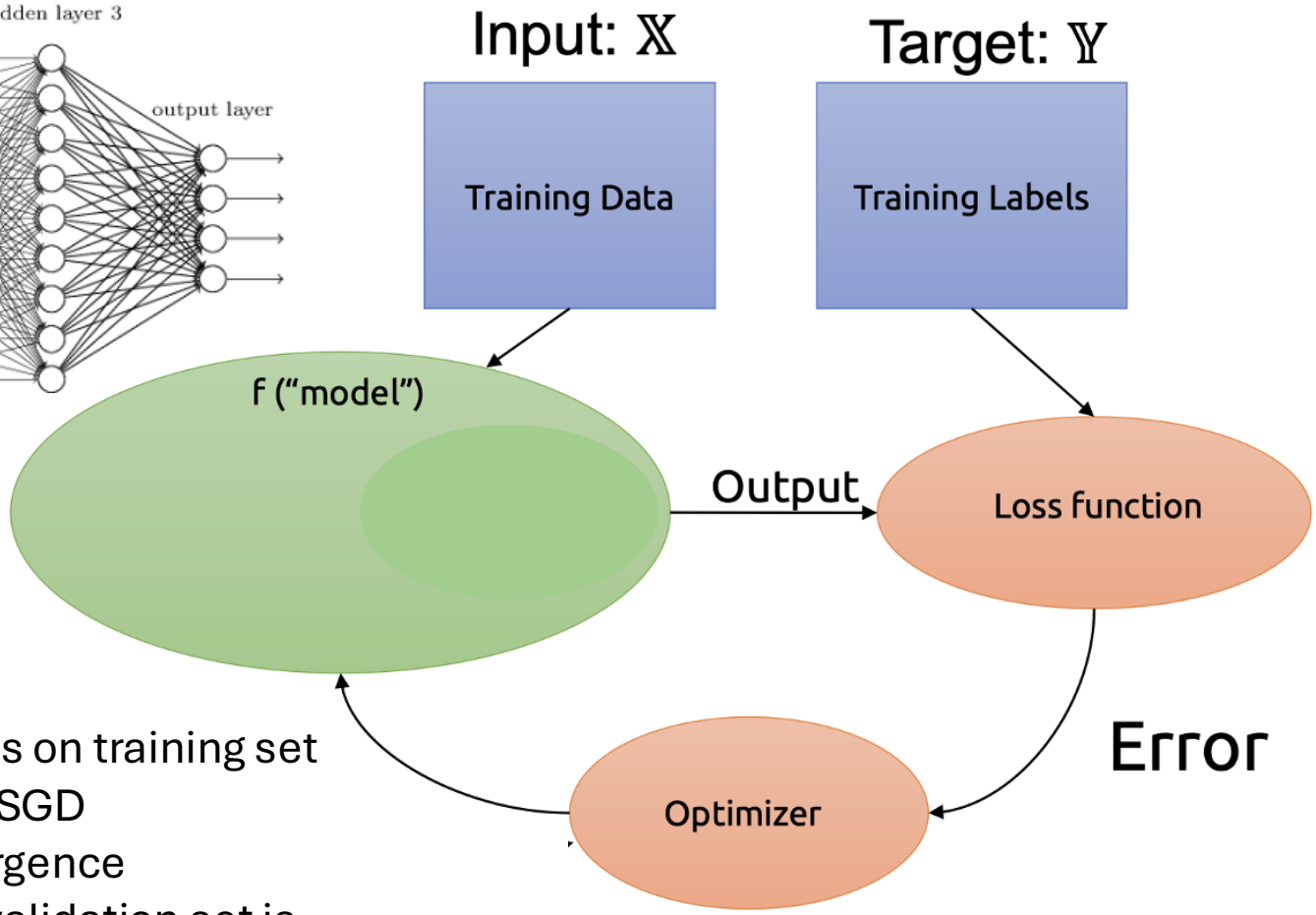
CSCI 1470

Eric Ewing

Thursday, 9/23

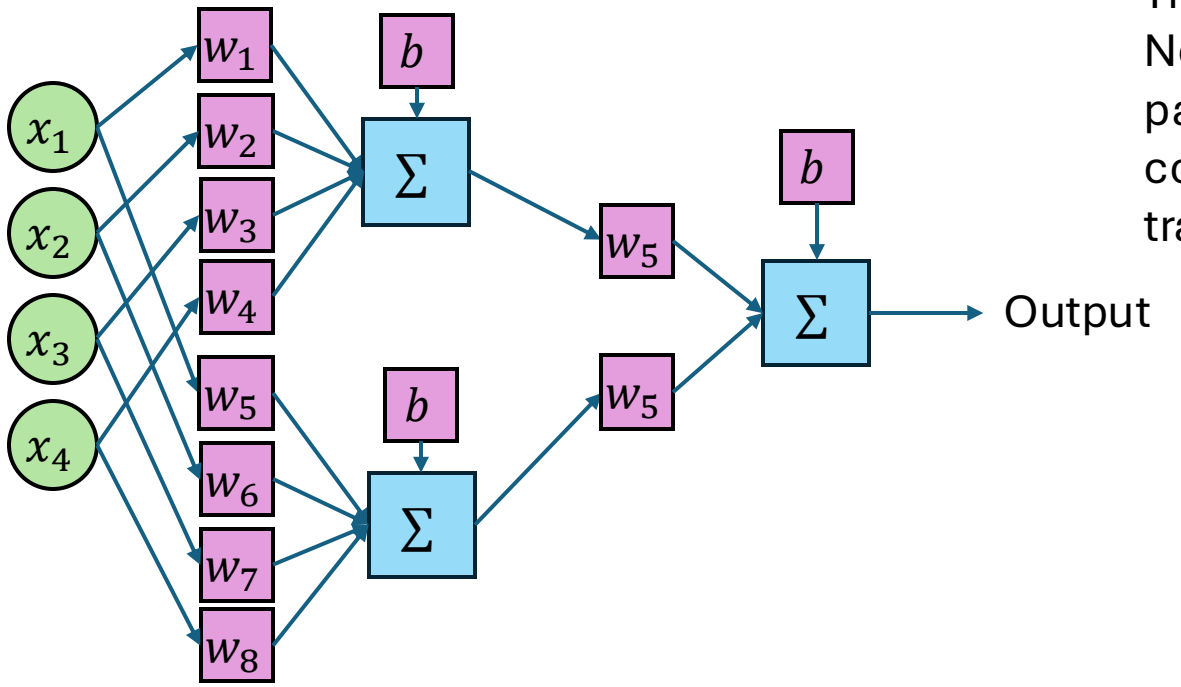
Recap: MLPs





1. Compute Error/Loss on training set
2. Run Backprop and SGD
3. Repeat until convergence
4. If performance on validation set is acceptable, terminate, else try new hyperparameters

Hyperparameters



The **parameters** of a Neural Network are what is trained (e.g., weights and biases).

The **hyperparameters** of a Neural Network are the parameters that **you** have control of that control that training.

Data Preprocessing

Data Preprocessing

Normalization: Shift each feature to be of a similar scale

Data Preprocessing

Normalization: Shift each feature to be of a similar scale

Apply: $x' = \frac{x - min(x)}{min(x) - max(x)}$ to every feature in dataset, shifting each feature to lie within [0, 1].

Data Preprocessing

Normalization: Shift each feature to be of a similar scale

Apply: $x' = \frac{x - min(x)}{min(x) - max(x)}$ to every feature in dataset, shifting each feature to lie within [0, 1].

Before normalization: different features can be different scales, different units, etc.

Distance A: 10 inches
Distance B: 25 inches
Distance C: 40 inches

Distance A: 1000 inches
Distance B: 2500 inches
Distance C: 4000 inches

After normalization: Each feature has same impact on model

Normalized Features: [0, 0.5, 1]

Normalized Features: [0, 0.5, 1]

Data Preprocessing

Normalization: Shift each feature to be of a similar scale

Apply: $x' = \frac{x - min(x)}{min(x) - max(x)}$ to every feature in dataset, shifting each feature to lie within [0, 1].

Before normalization: different features can be different scales, different units, etc.

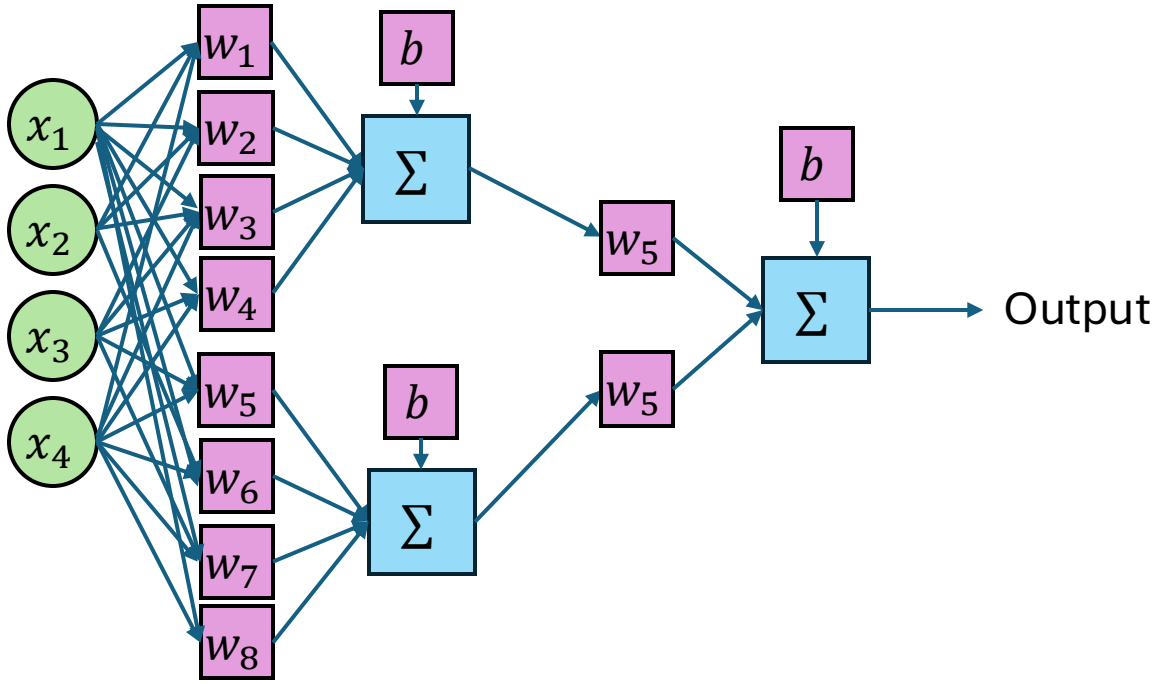
After normalization: Each feature has same impact on model

Gradient Descent converges faster when working with normalized data!

Ioffe, Sergey; Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

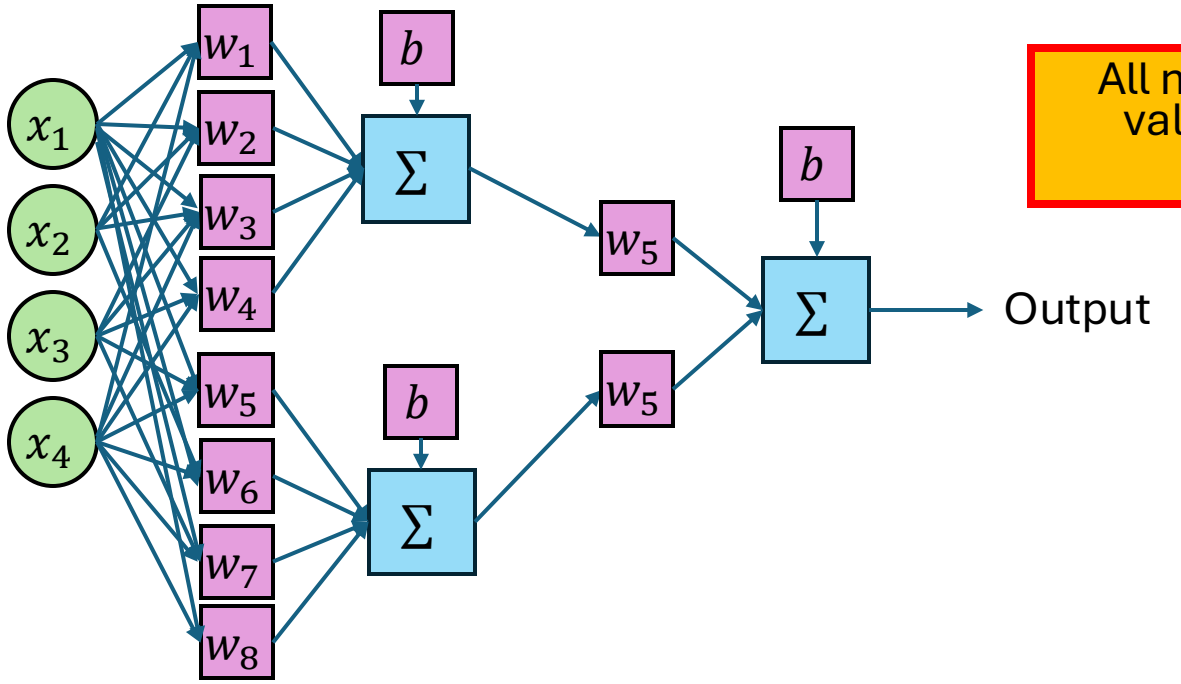
Network Initialization

What if we begin with all parameters set to 0?



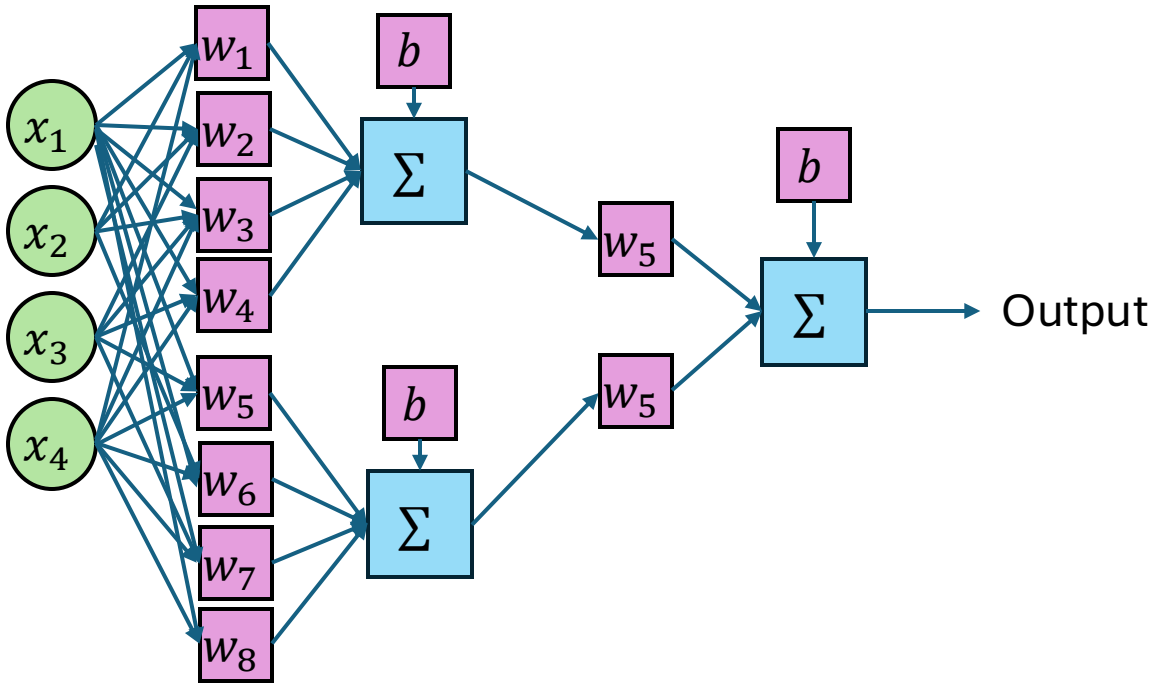
Network Initialization

What if we begin with all parameters set to 0?

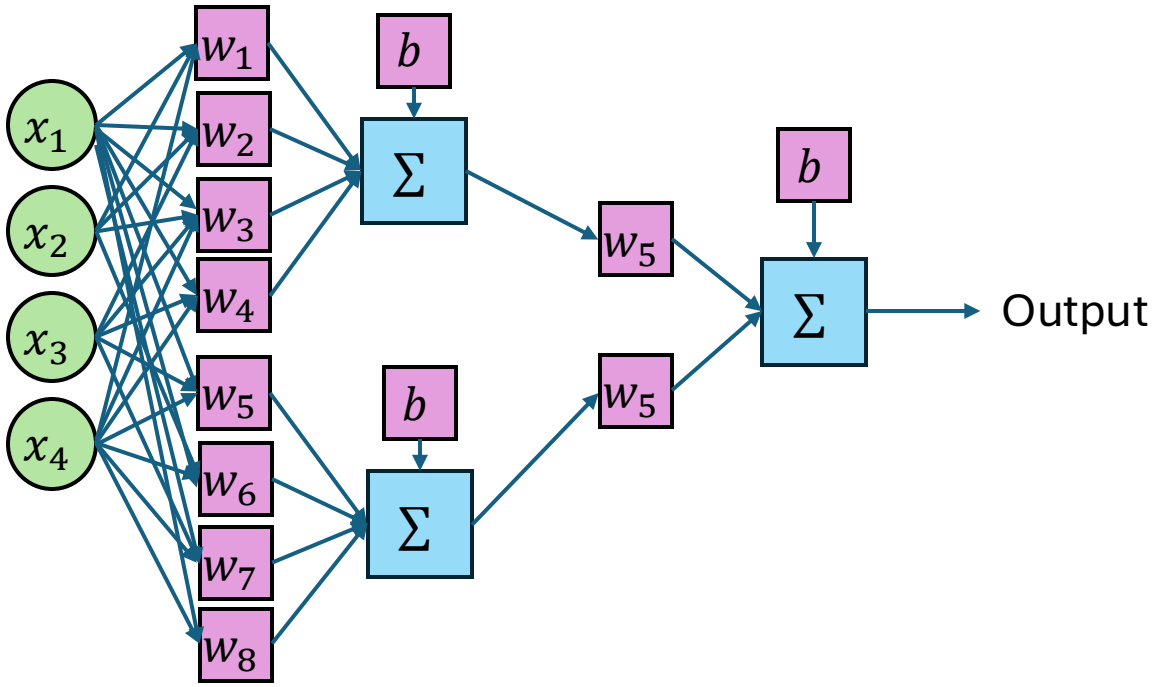


All neurons would have the same value, **gradients would be the same.**

Network Initialization

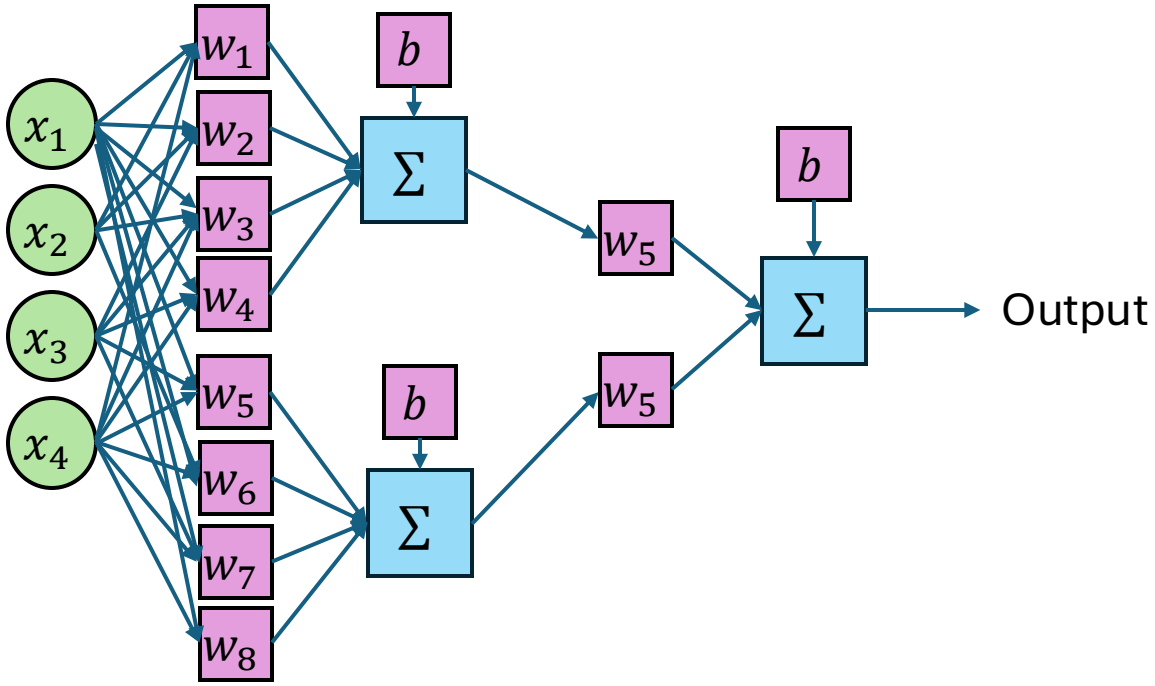


Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

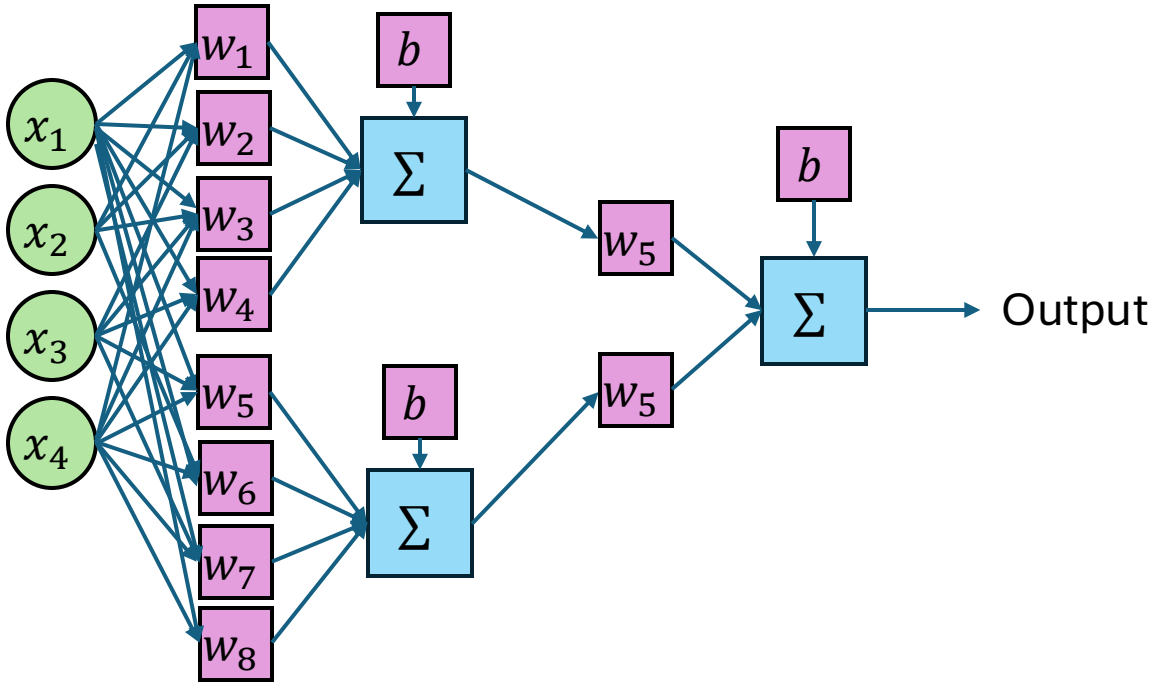
Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow, variance of output increases

Network Initialization

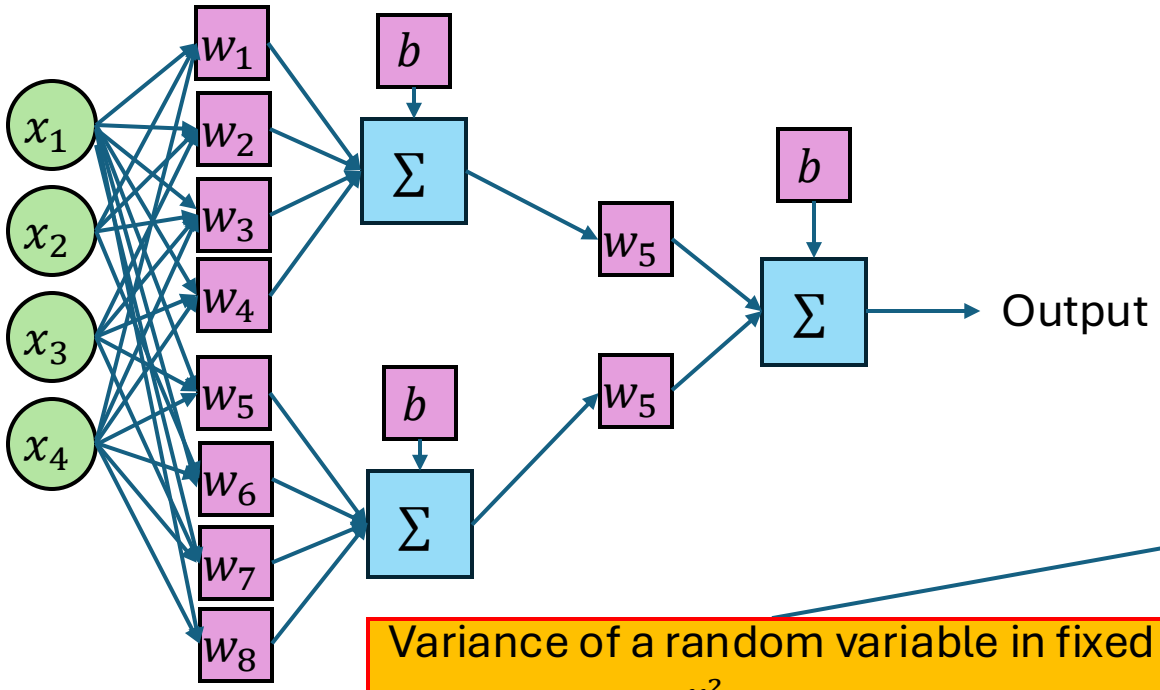


Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

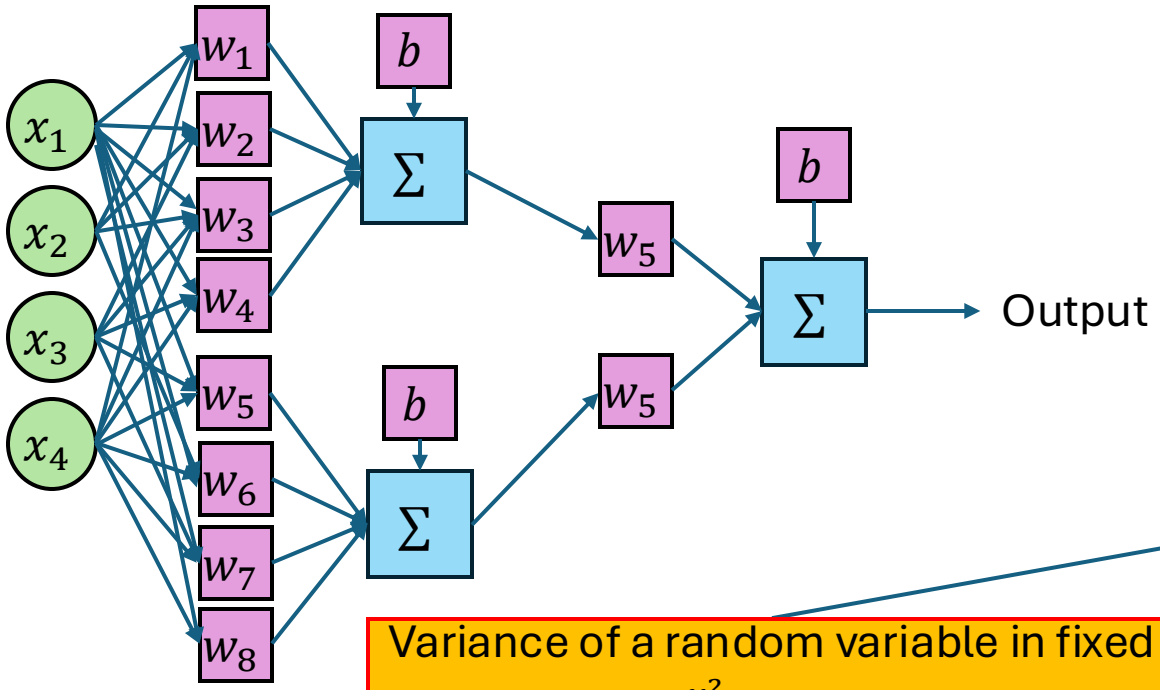
But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

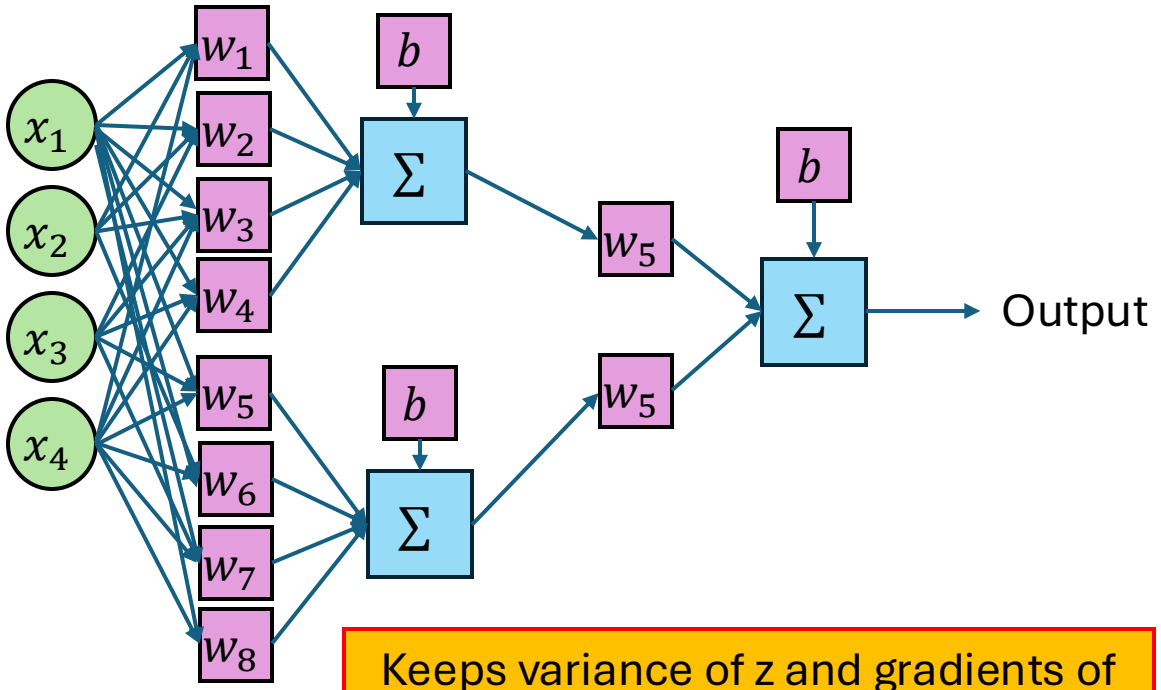
Idea #2: Xavier (Glorot) initialization:

Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Normal: Initialize each weight with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$

Variance of a random variable in fixed range [-x, x] is $\frac{x^2}{3}$ (easy to derive from definition of variance)

Network Initialization



Keeps variance of z and gradients of weights the same for each layer at initialization.

Idea #1: Uniform random weights between -1 and 1 (works fine)

But… what if there are many many weights in a layer? The scale of the output can grow.

Idea #2: Xavier (Glorot) initialization:

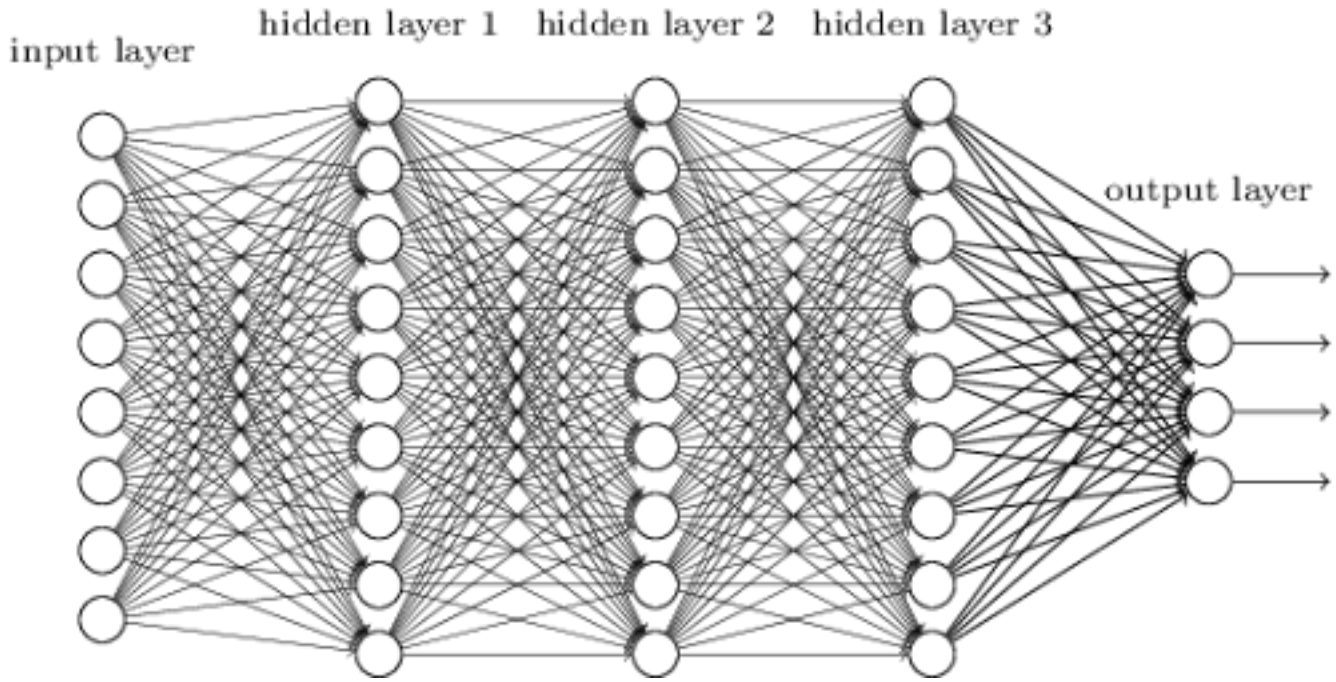
Uniform: Initialize each weight uniformly at random in the range [-x, x] with $x = \sqrt{\frac{6}{n_{in}+n_{out}}}$

Normal: Initialize each weight with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{in}+n_{out}}}$

Understanding the difficulty of training deep feedforward neural networks. ***Xavier Glorot, Yoshua Bengio***

Hidden Layers

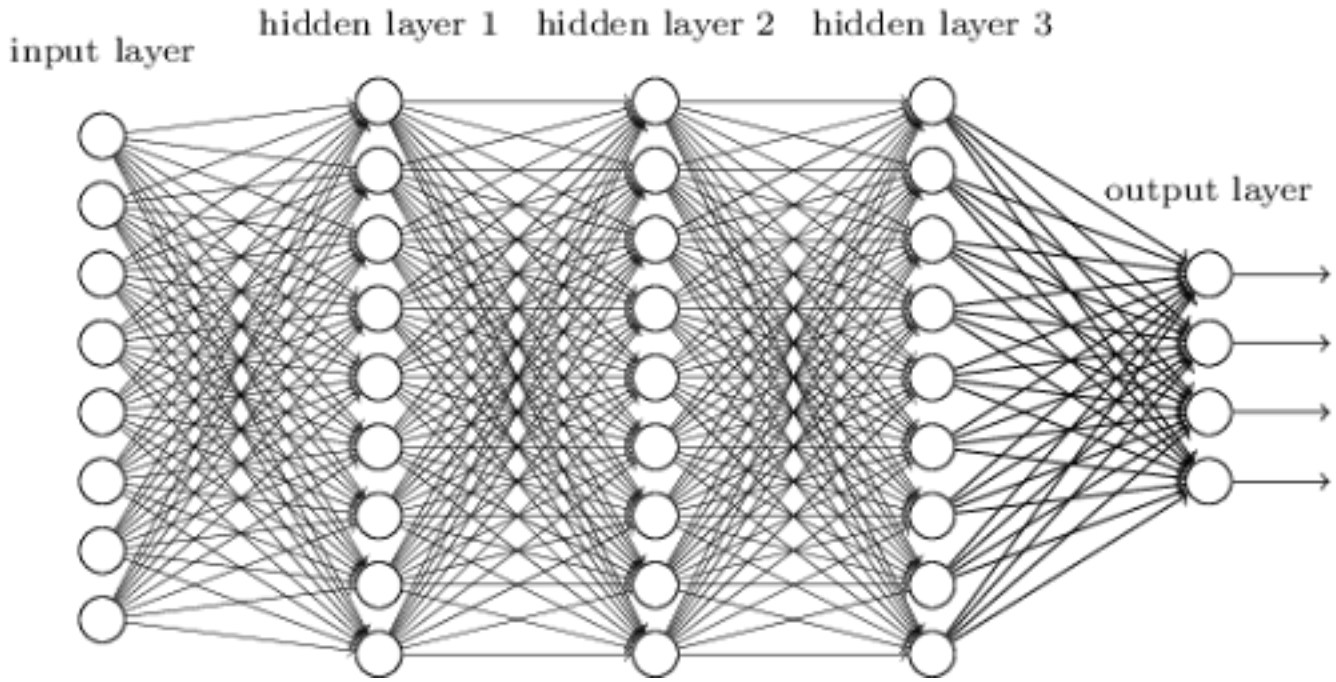
- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?



Hidden Layers

- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?

How complex is the problem you are trying to solve?

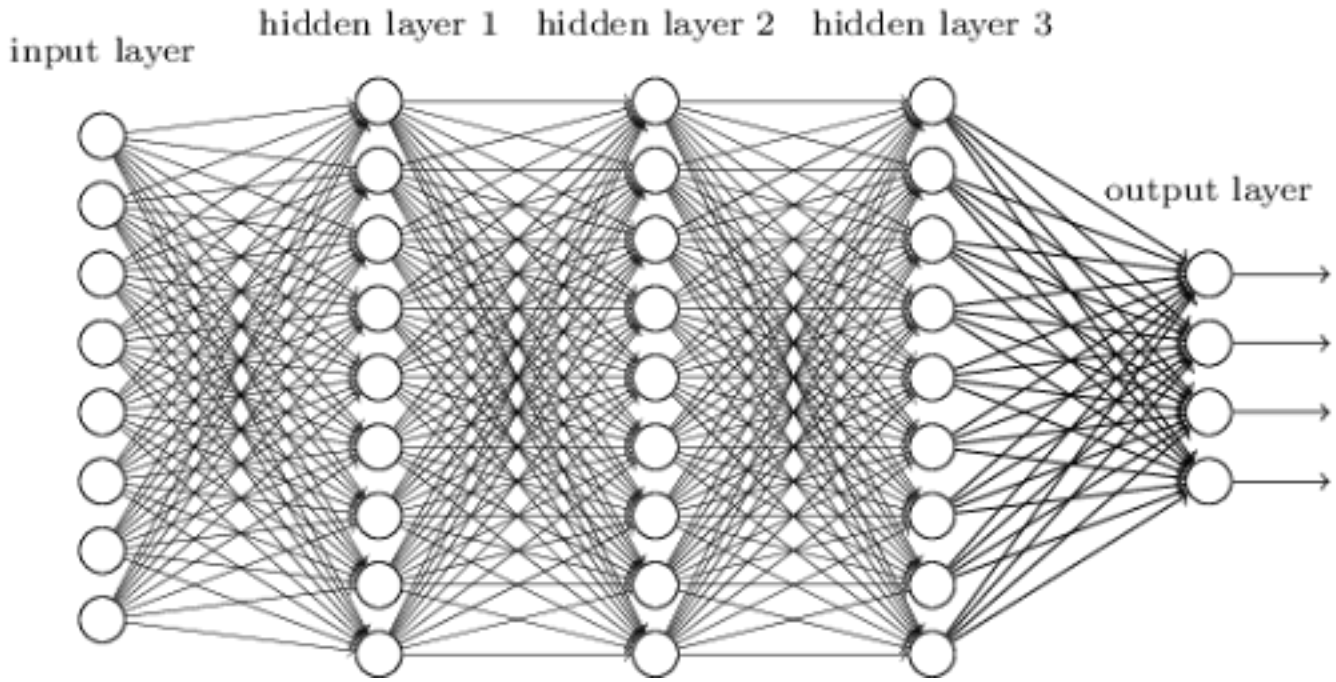


Hidden Layers

- How deep (# hidden layers) should your network be?
- How wide (# neurons in a layer) should your network be?

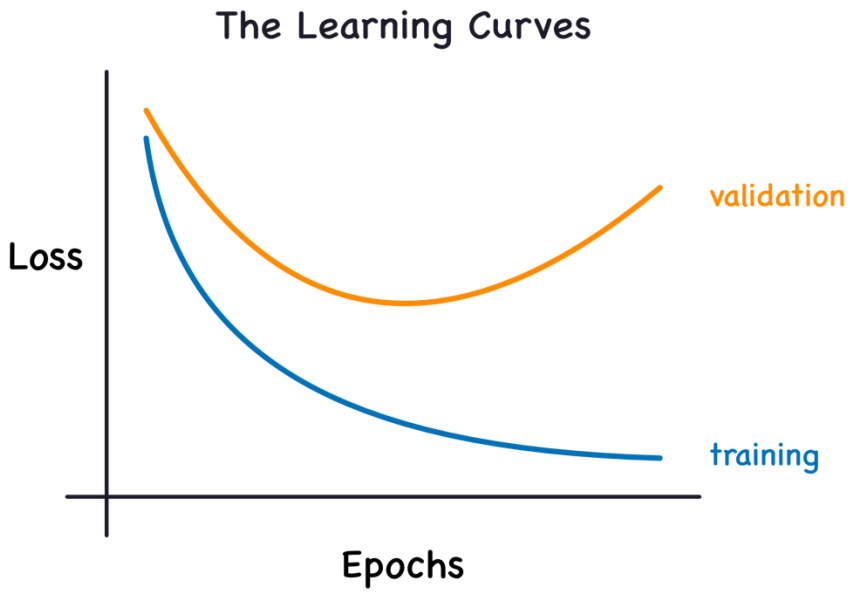
How complex is the problem you are trying to solve?

Process of (informed) trial and error. How do you know if one hyper-parameter setting is better than another?



How to use the Validation Set

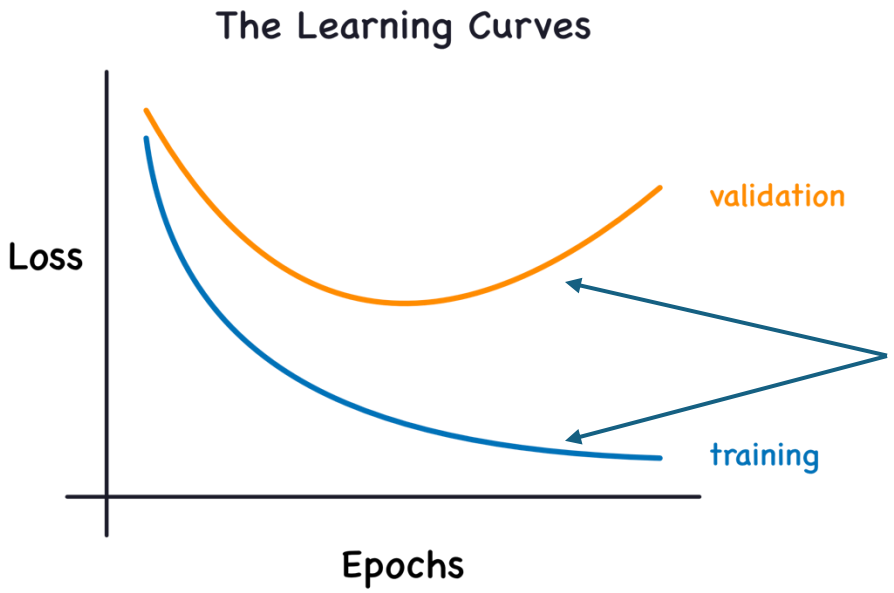
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

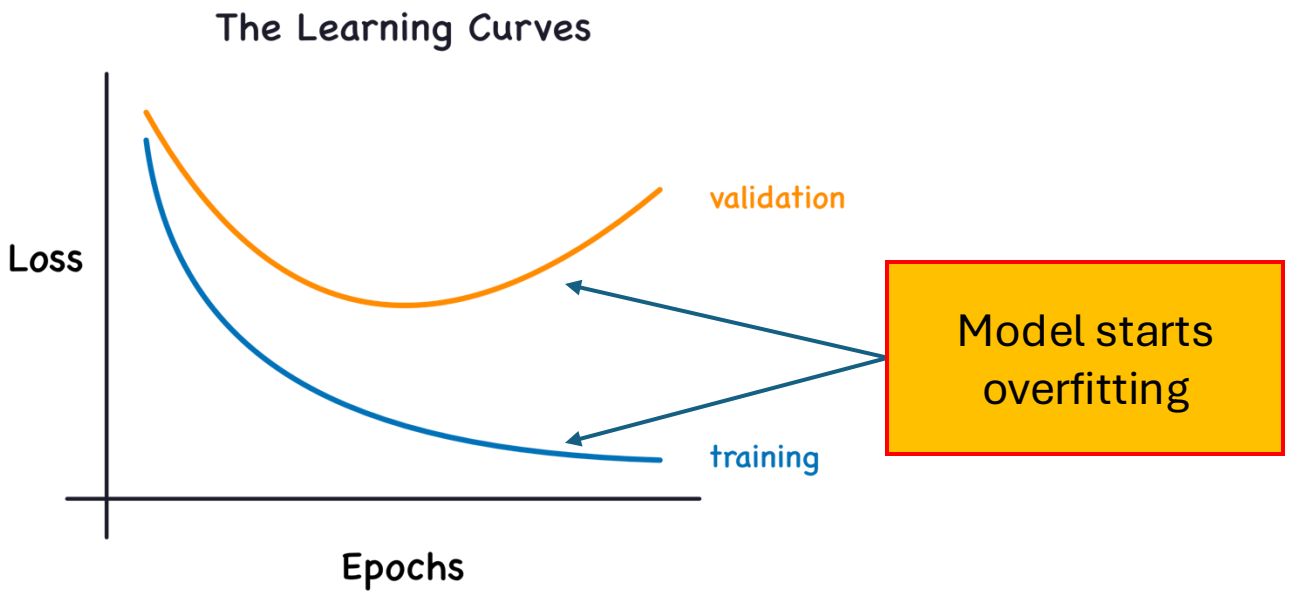
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

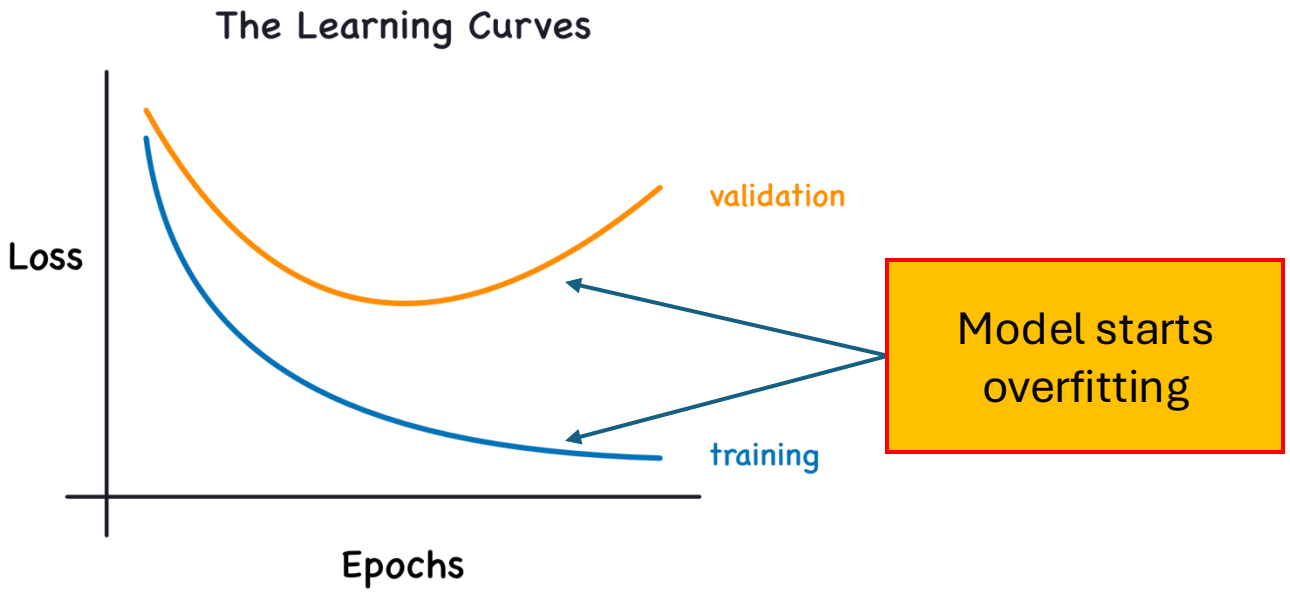
(In theory)



https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



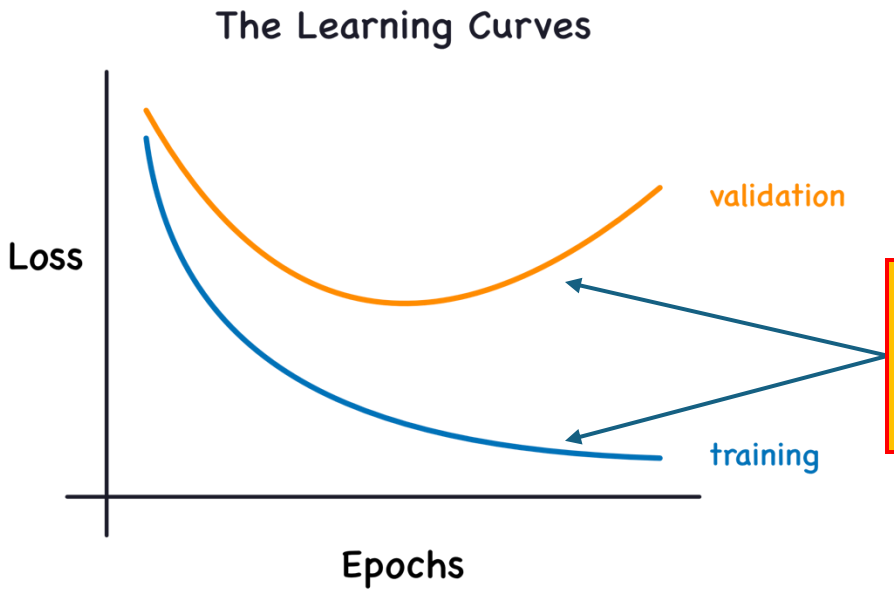
Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

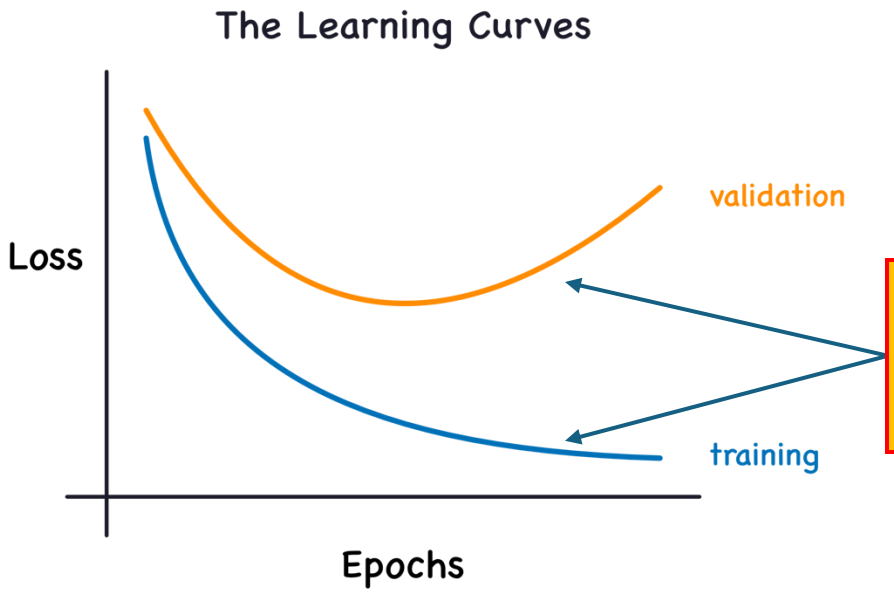
1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

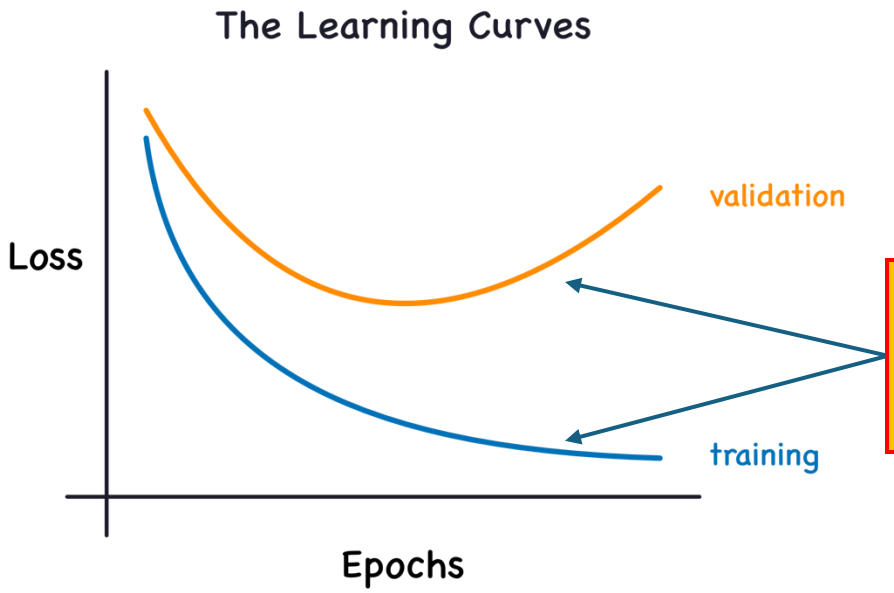
What if your validation loss is much higher than training loss?

Your model has overfit, try **reducing** its size

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

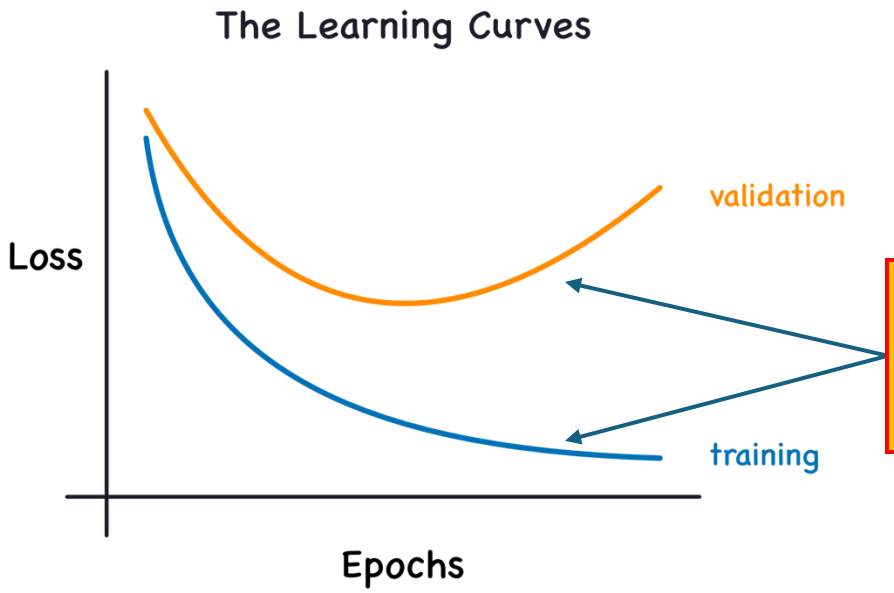
Your model has overfit, try **reducing** its size

What if your validation loss and training loss are both high?

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

How to use the Validation Set

(In theory)



Model starts overfitting

Early Stopping Algorithm

1. Track training loss and validation loss
2. If validation loss starts to increase, terminate

What if your validation loss is much higher than training loss?

Your model has overfit, try **reducing** its size

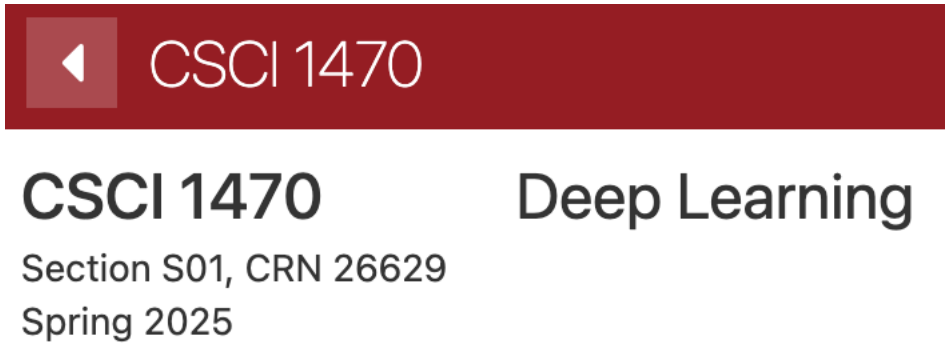
What if your validation loss and training loss are both high?

Your model has underfit, try **increasing** its size

https://wandb.ai/mostafaibrahim17/ml-articles/reports/A-Deep-Dive-Into-Learning-Curves-in-Machine-Learning--Vmlldzo0NjA1ODY0

Is adding more width or depth better?

Is adding more width or depth better?



Theoretical Approaches to Understanding Depth

Proofs:

- Are there functions that deep networks can represent better than shallow networks (with similar numbers of neurons)?

Conceptual Understanding:

- Neural Networks and Manifolds for representation learning

Benefits of depth in neural networks

“For any positive integer k, there exist neural networks with $\Theta(k^3)$ layers, $\Theta(1)$ nodes per layer, and $\Theta(1)$ distinct parameters which can not be approximated by networks with $O(k)$ layers unless they are exponentially large — they must possess $\Omega(2^k)$ nodes.”

Matus Telgarsky “Benefits of depth in neural networks”, JMLR

Benefits of depth in neural networks

"For any positive integer k, there exist neural networks with $\Theta(k^3)$ layers, $\Theta(1)$ nodes per layer, and $\Theta(1)$ distinct parameters which can not be approximated by networks with O(k) layers unless they are exponentially large — they must possess $\Omega(2^k)$ nodes."

There exist functions that shallow networks cannot represent as efficiently as deep networks

Matus Telgarsky "Benefits of depth in neural networks", JMLR

Benefits of depth in neural networks

"For any positive integer k, there exist neural networks with $\Theta(k^3)$ layers, $\Theta(1)$ nodes per layer, and $\Theta(1)$ distinct parameters which can not be approximated by networks with O(k) layers unless they are exponentially large — they must possess $\Omega(2^k)$ nodes."

There exist functions that shallow networks cannot represent as efficiently as deep networks

How well does theory match real world applications? Are these functions pathological?

Matus Telgarsky "Benefits of depth in neural networks", JMLR

Depth-Width Tradeoffs in Approximating Natural Functions with Neural Networks

Itay Safran
Weizmann Institute of Science
itay.safran@weizmann.ac.il

Ohad Shamir
Weizmann Institute of Science
ohad.shamir@weizmann.ac.il

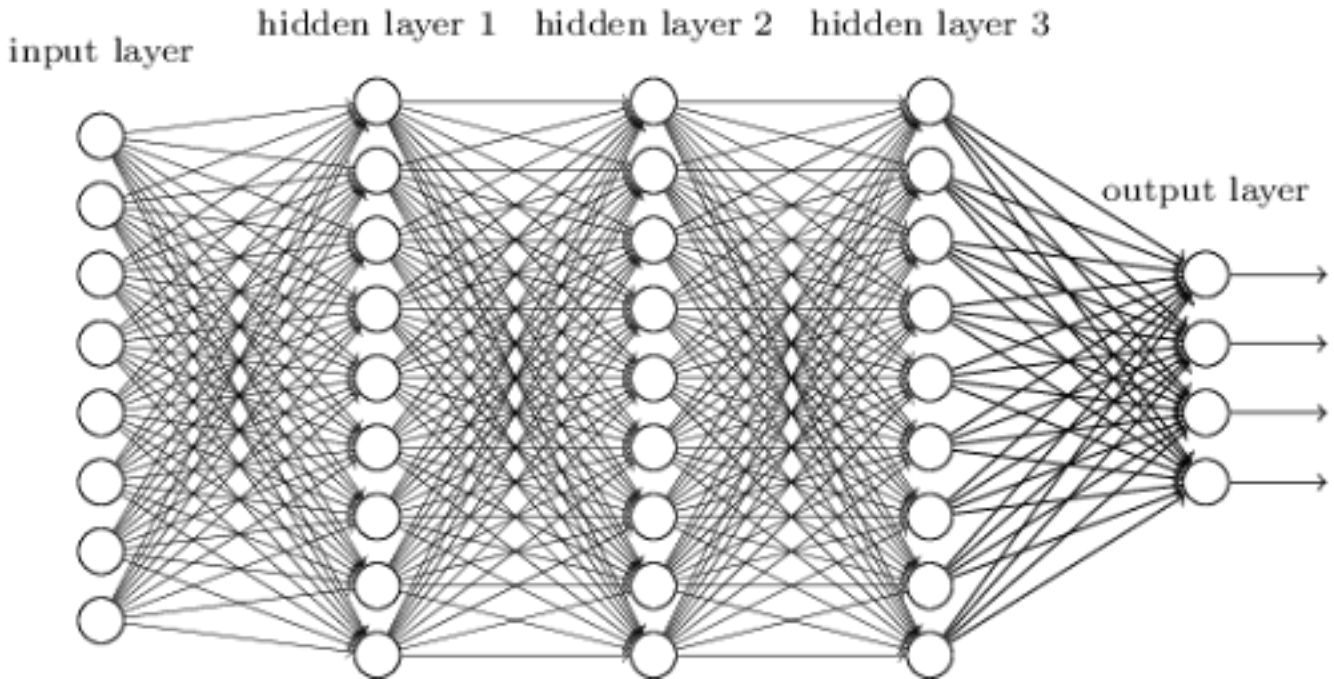
With the same number of total parameters, deep networks can learn more complex functions.

Recall that NNs are compositions of functions for which we are learning parameters:

$$f(g(h\left(i(j(x))\right)$$

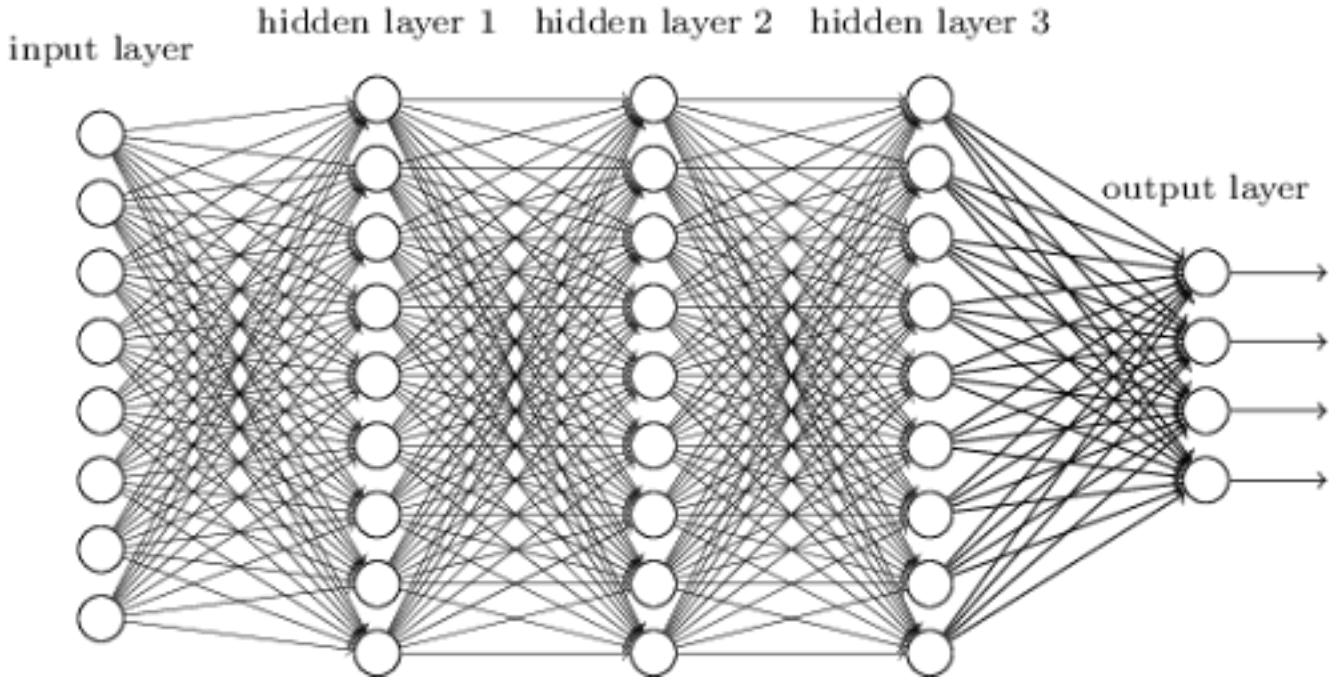
It's better (in general) to have more functions composed than it is to have more complex functions

- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?

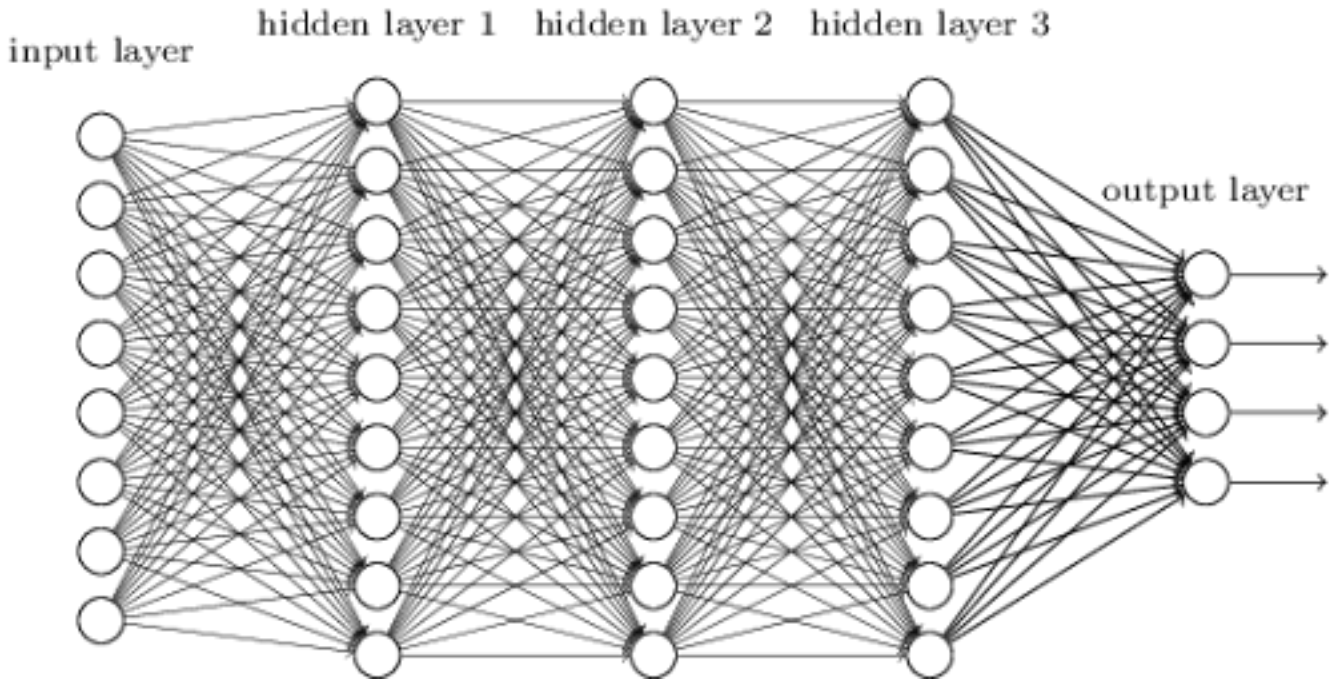


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?

$W_1 \in \mathbb{R}^{10\times10}$
$W_2 \in \mathbb{R}^{10\times10}$
$W_3 \in \mathbb{R}^{10\times10}$
$W_4 \in \mathbb{R}^{10\times4}$
Total = 340

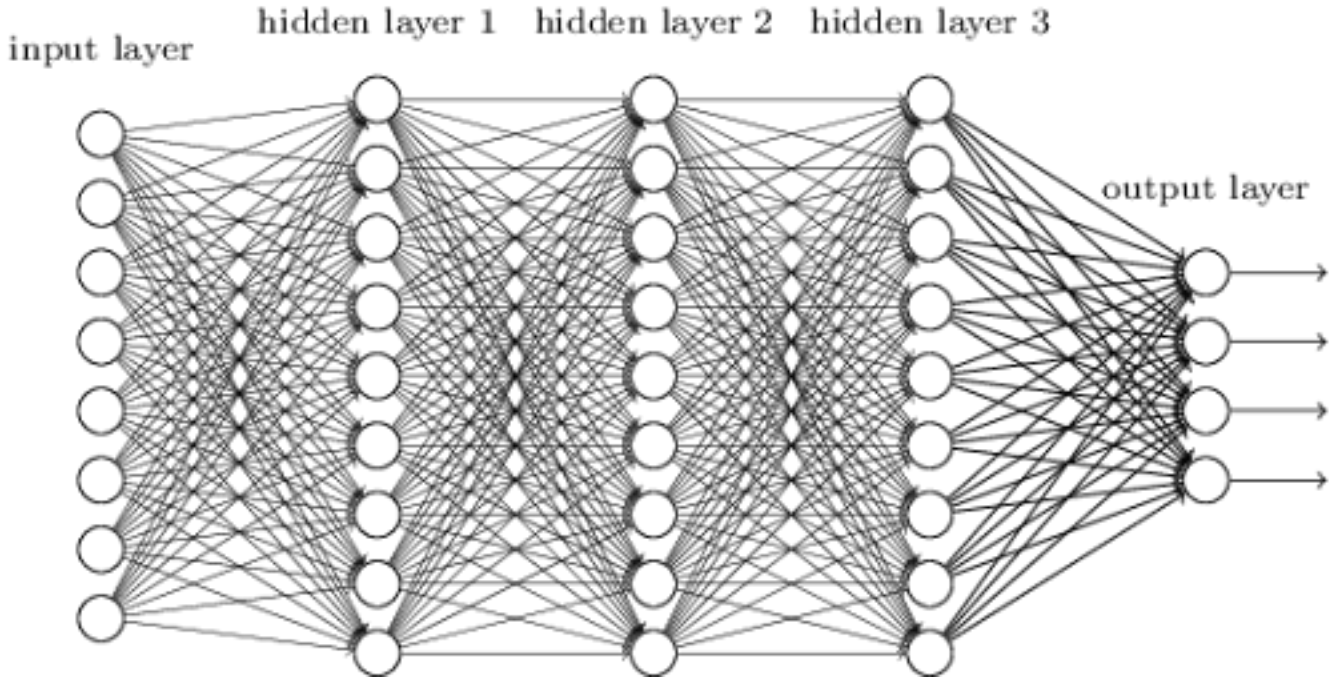


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the width of each hidden layer?

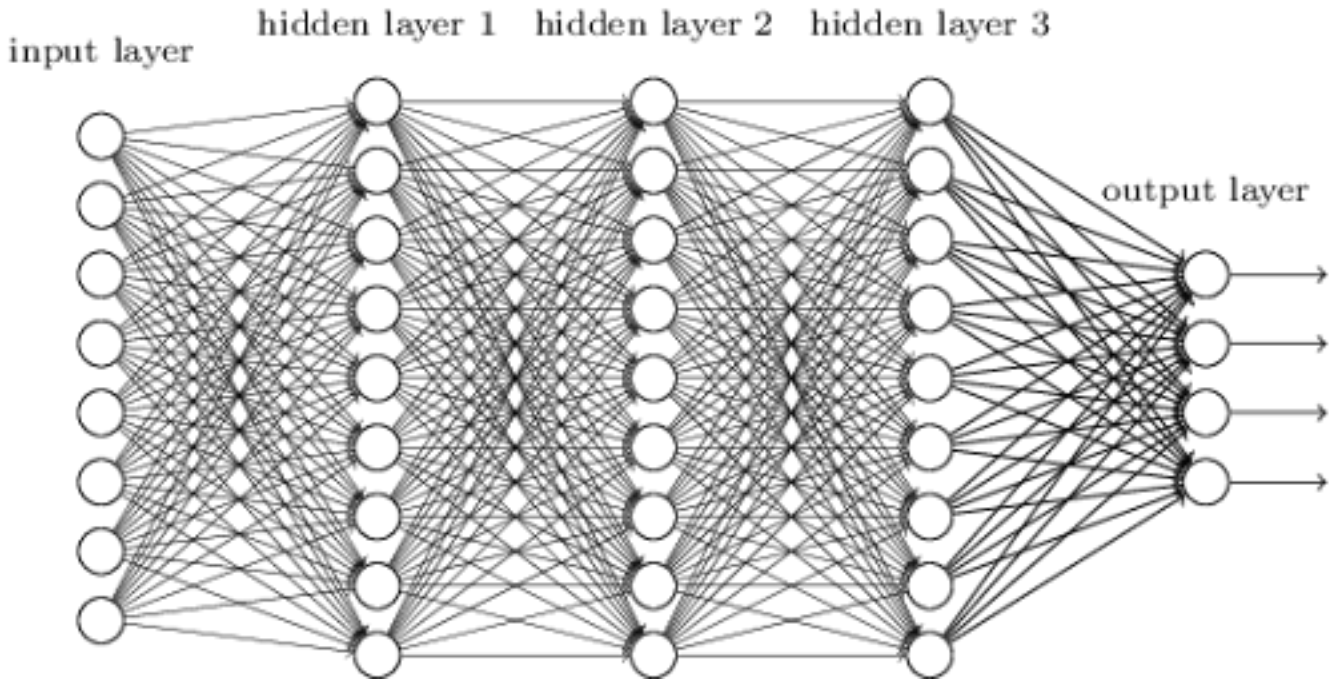


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the width of each hidden layer?

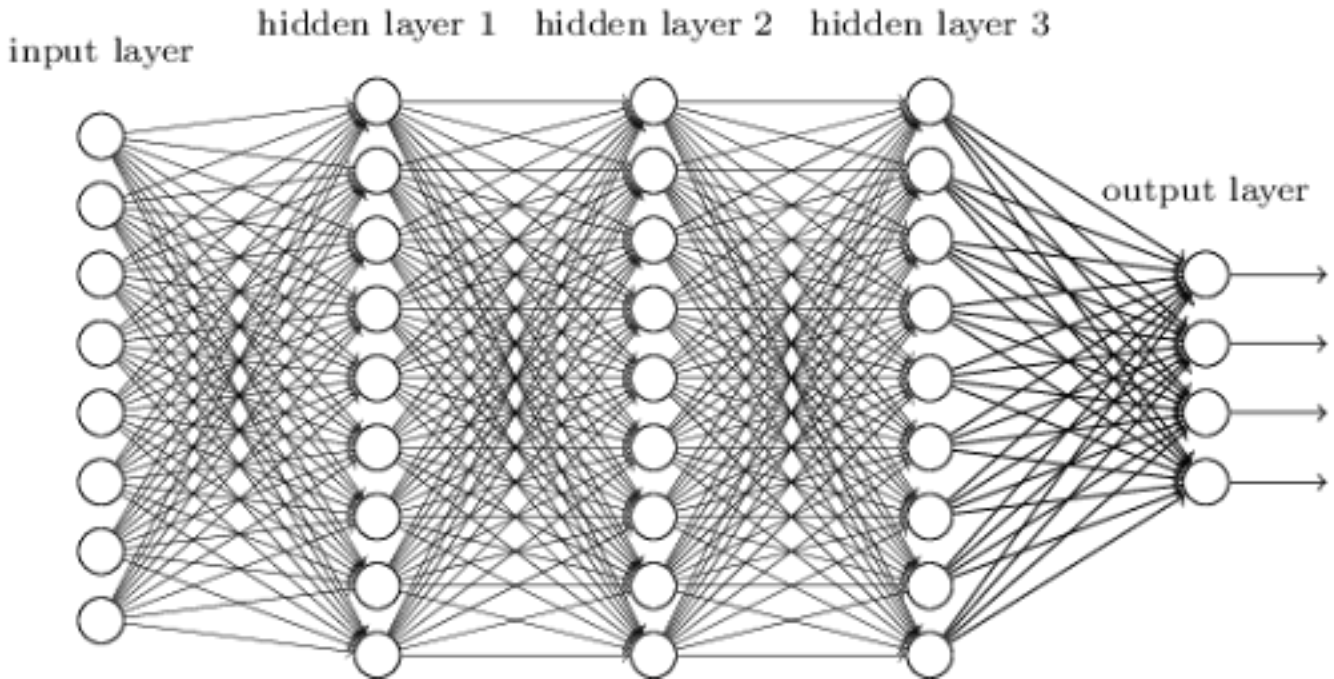
$W_1 \in \mathbb{R}^{10\times20}$
$W_2 \in \mathbb{R}^{20\times20}$
$W_3 \in \mathbb{R}^{20\times20}$
$W_4 \in \mathbb{R}^{20\times4}$
Total =1080



- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the depth?



- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the depth?



$W_1 \in \mathbb{R}^{10\times10}$
$W_2 \in \mathbb{R}^{10\times10}$
$W_3 \in \mathbb{R}^{10\times10}$
$W_4 \in \mathbb{R}^{10\times10}$
$W_5 \in \mathbb{R}^{10\times10}$
$W_6 \in \mathbb{R}^{10\times10}$
$W_7 \in \mathbb{R}^{10\times4}$
Total = 640

The Manifold Hypothesis

The Manifold Hypothesis

Manifold: A space that appears locally like Euclidean space

The Manifold Hypothesis

Manifold: A space that appears locally like Euclidean space

Locally, the surface of the earth appears like a flat plane in $\mathbb{R}^2$, while the earth itself is a sphere(-ish) in $\mathbb{R}^3$

The Manifold Hypothesis

Manifold: A space that appears locally like Euclidean space

Locally, the surface of the earth appears like a flat plane in $\mathbb{R}^2$, while the earth itself is a sphere(-ish) in $\mathbb{R}^3$

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

The Manifold Hypothesis

Manifold: A space that appears locally like Euclidean space

Locally, the surface of the earth appears like a flat plane in $\mathbb{R}^2$, while the earth itself is a sphere(-ish) in $\mathbb{R}^3$

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

Even though we may have d features in your data, it may require many fewer features to fully represent.

MNIST and Manifolds

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

MNIST images are 28x28 or 784 pixels total.

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

MNIST images are 28x28 or 784 pixels total.

If we restrict our pixels to only being black or white (0 or 1), then there are 2^{784} possible images we can create.

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

MNIST images are 28x28 or 784 pixels total.

If we restrict our pixels to only being black or white (0 or 1), then there are 2^{784} possible images we can create.

How many of these images are digits?

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

MNIST images are 28x28 or 784 pixels total.

If we restrict our pixels to only being black or white (0 or 1), then there are 2^{784} possible images we can create.

How many of these images are digits?

Our high-dimensional data is very sparse in high dimensions, perhaps there is a lower dimensional space where it can be better represented.

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

MNIST and Manifolds

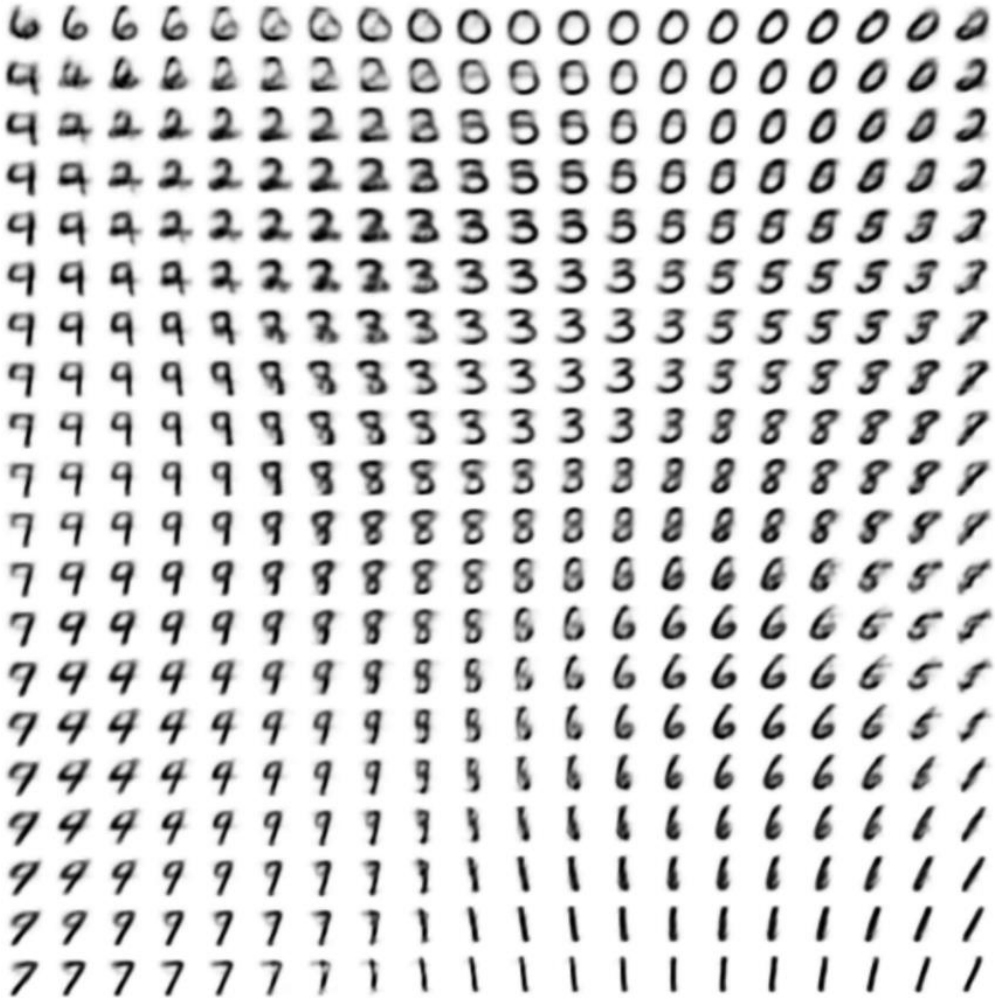
Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

MNIST images are 28x28 or 784 pixels total.

If we restrict our pixels to only being black or white (0 or 1), then there are 2^{784} possible images we can create.

How many of these images are digits?

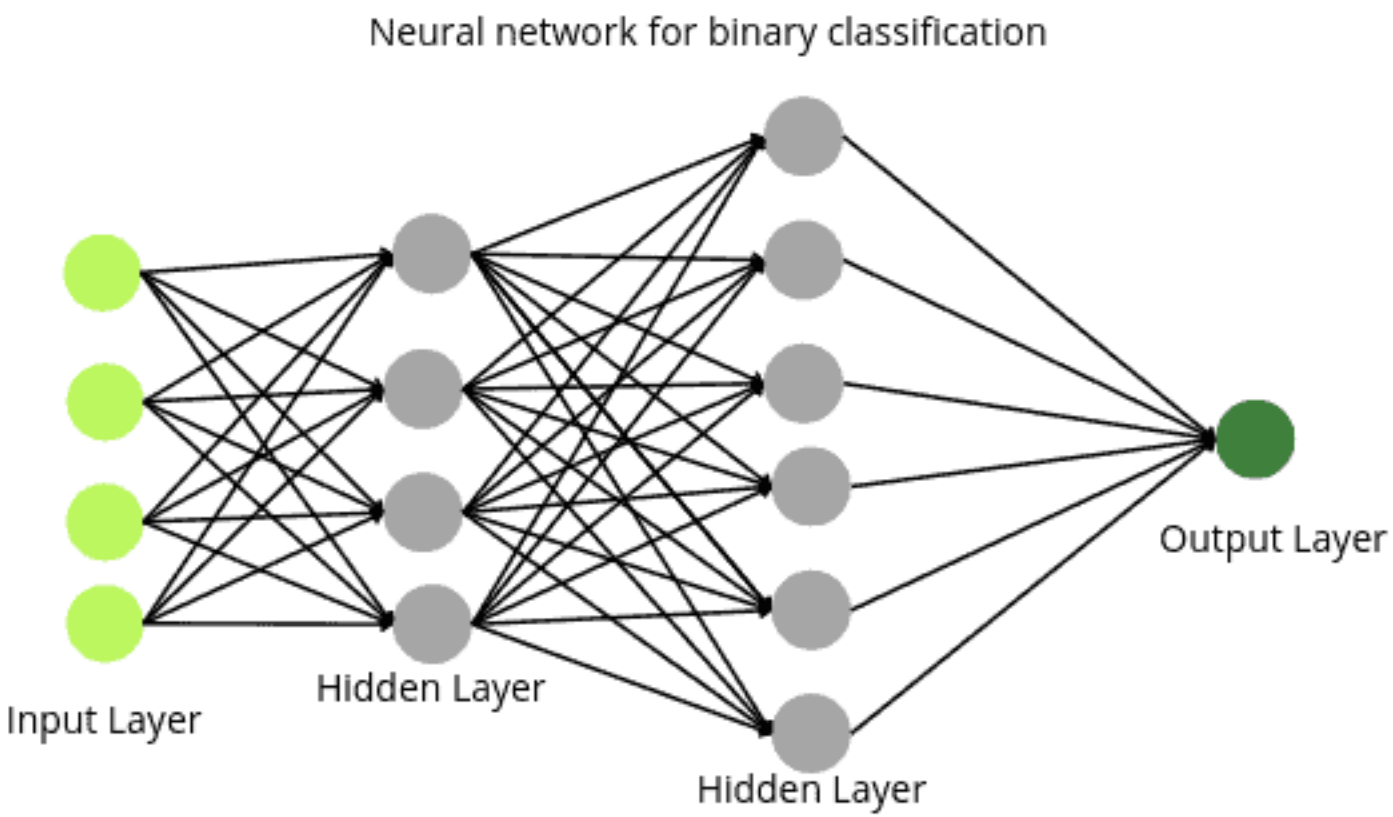
Our high-dimensional data is very sparse in high dimensions, perhaps there is a lower dimensional space where it can be better represented.



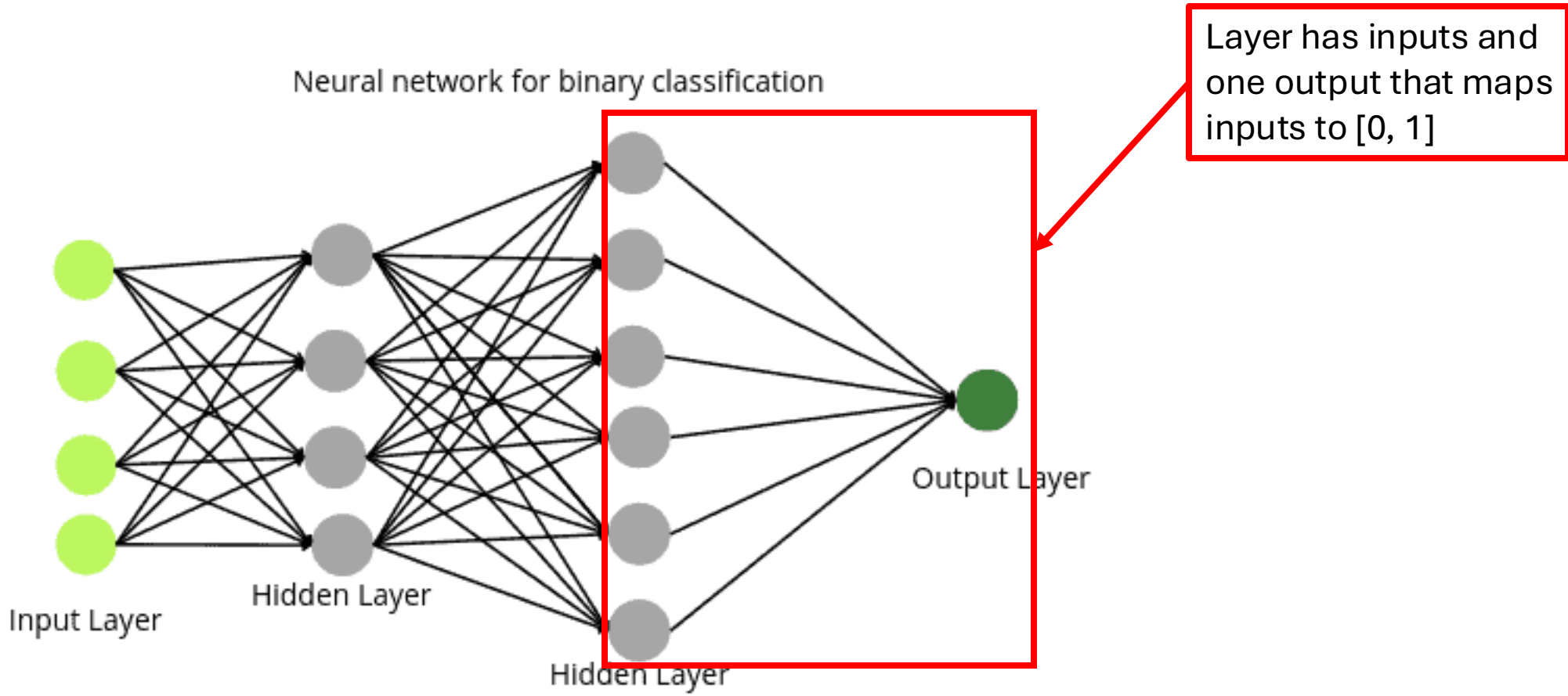
A learned manifold of MNIST

Viktor Toth and Lauri Parkkonen “Autoencoding sensory substitution”, 2019

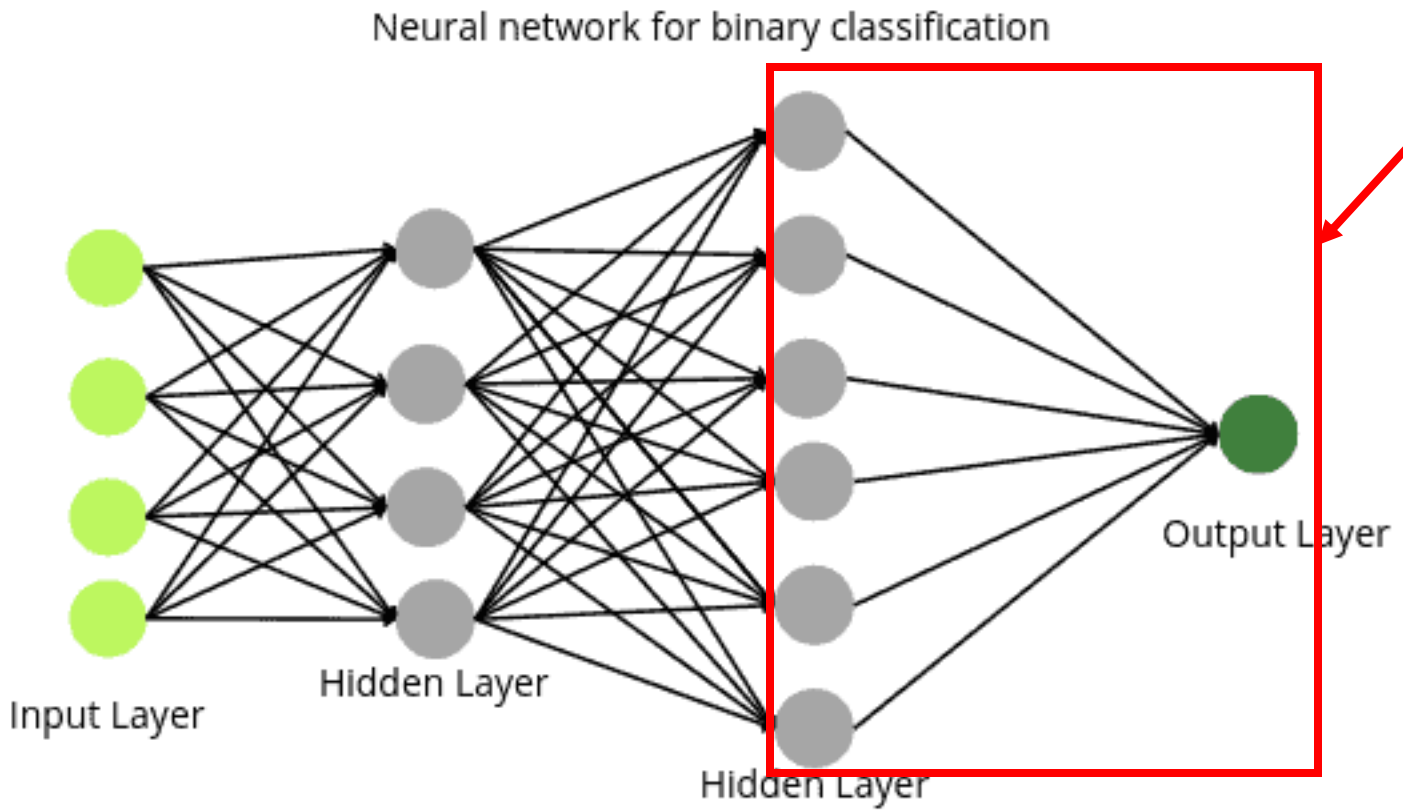
Deep Networks and Representation Learning



Deep Networks and Representation Learning



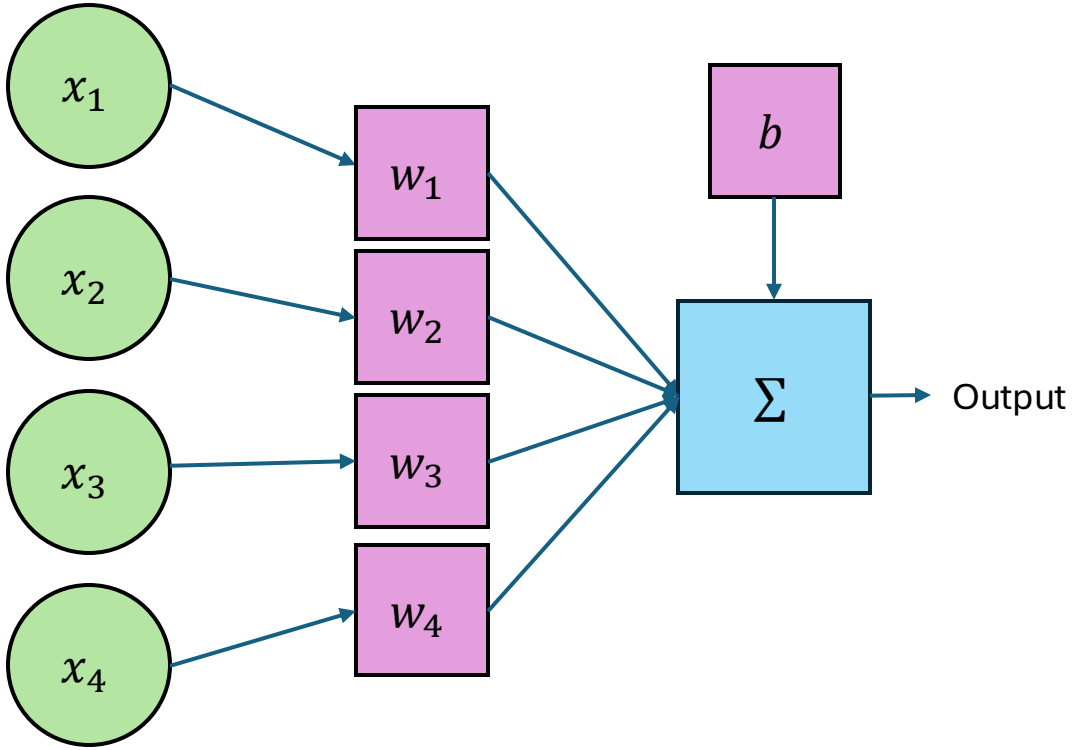
Deep Networks and Representation Learning

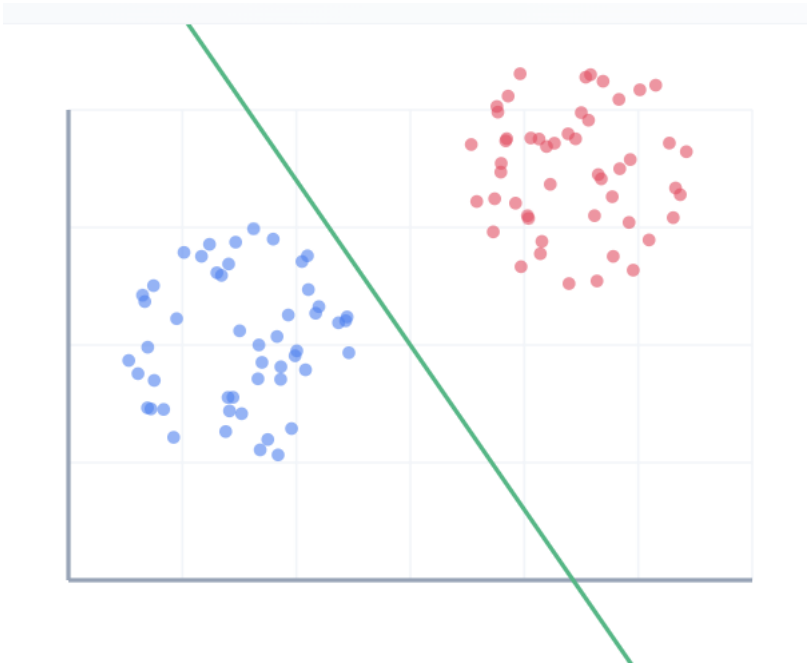


Layer has inputs and one output that maps inputs to [0, 1]

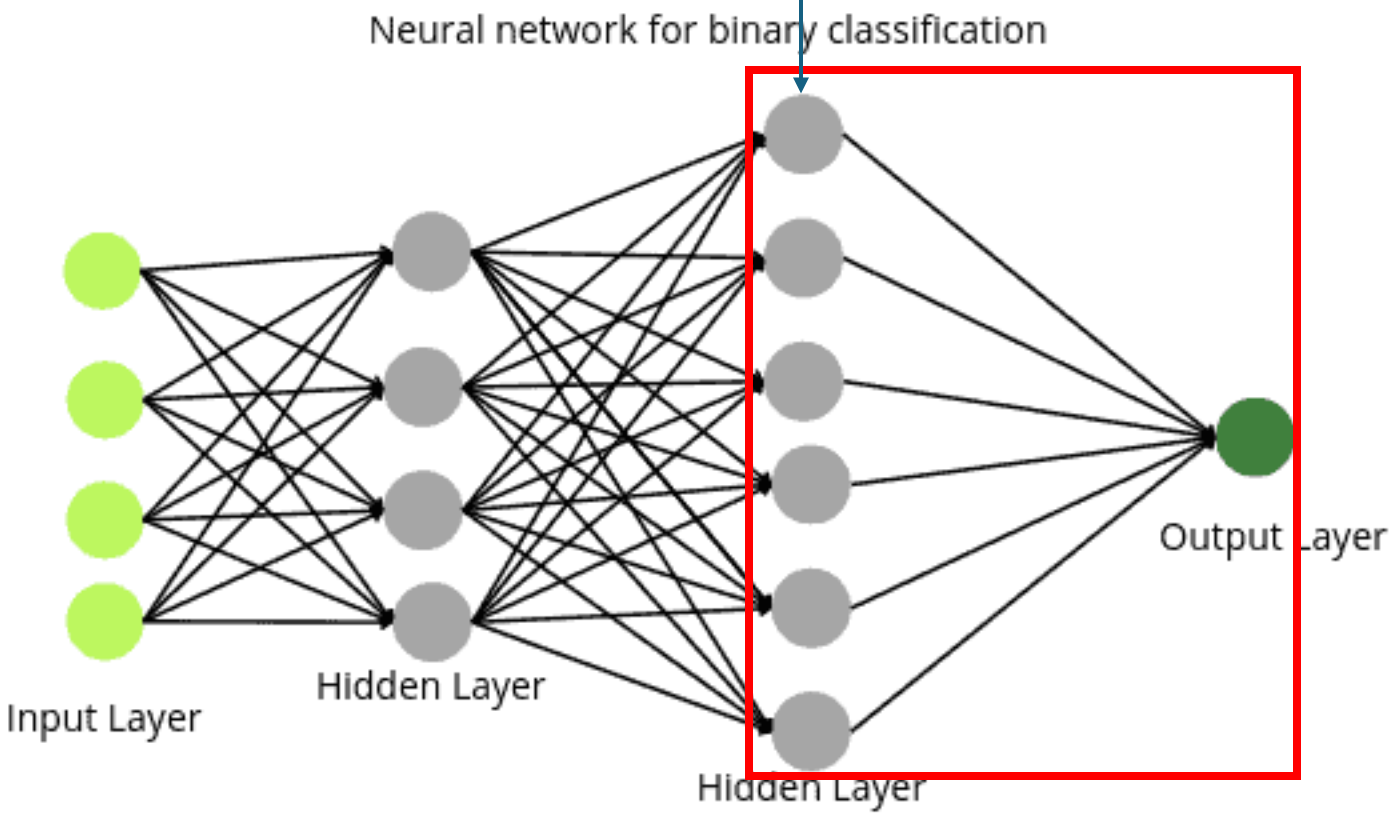
Looks a lot like a perceptron…

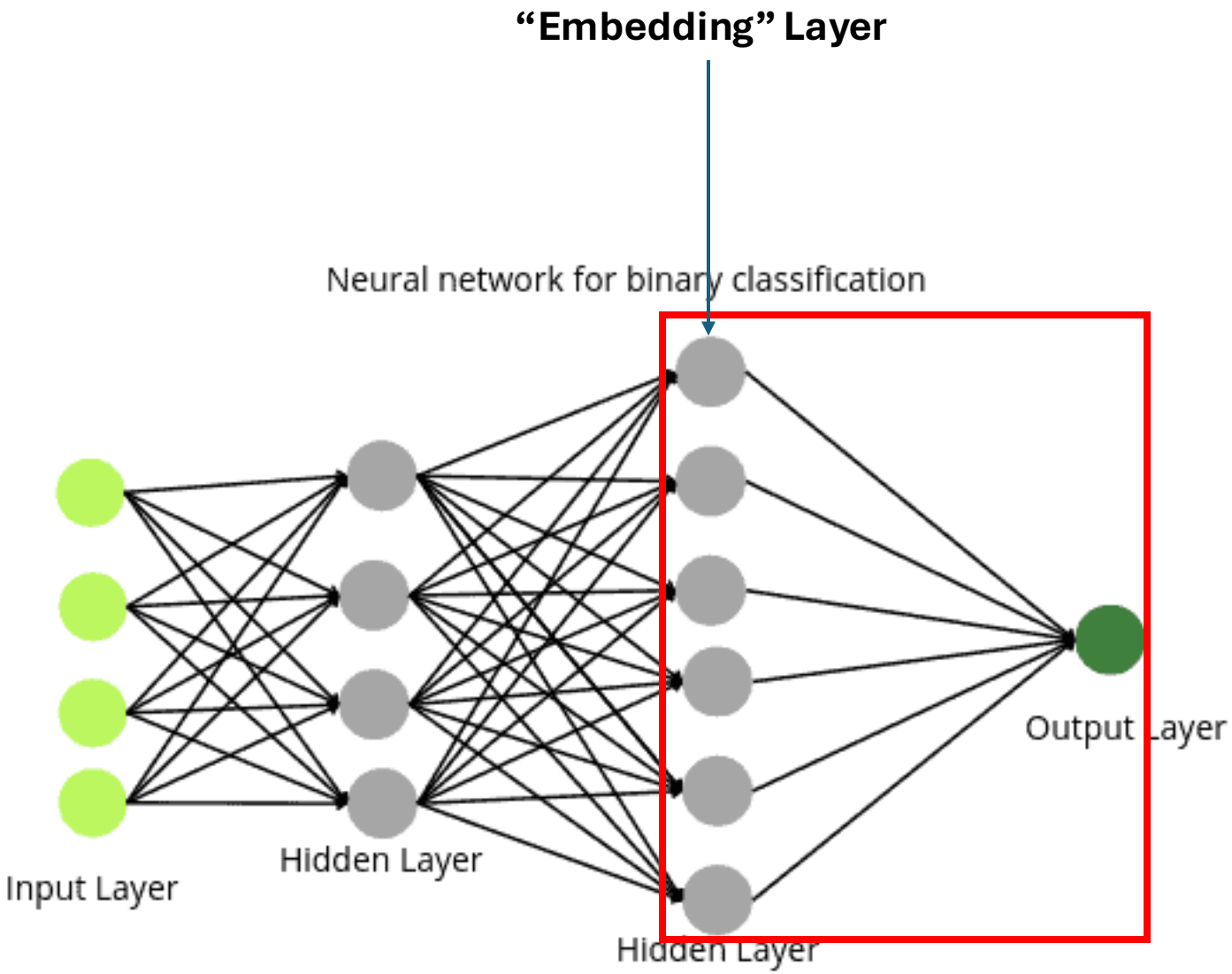
Perceptrons are Linear Separators



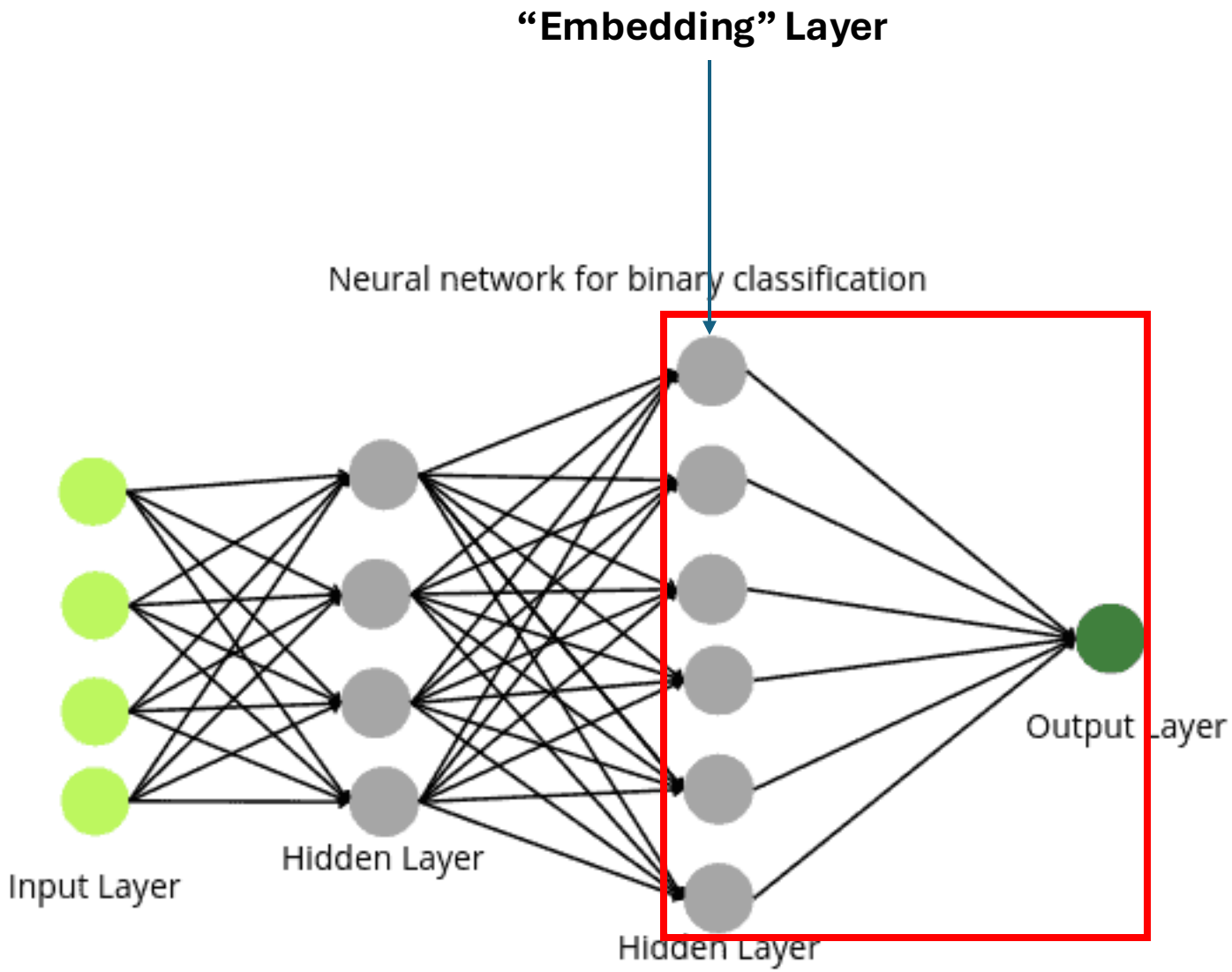


“Embedding” Layer





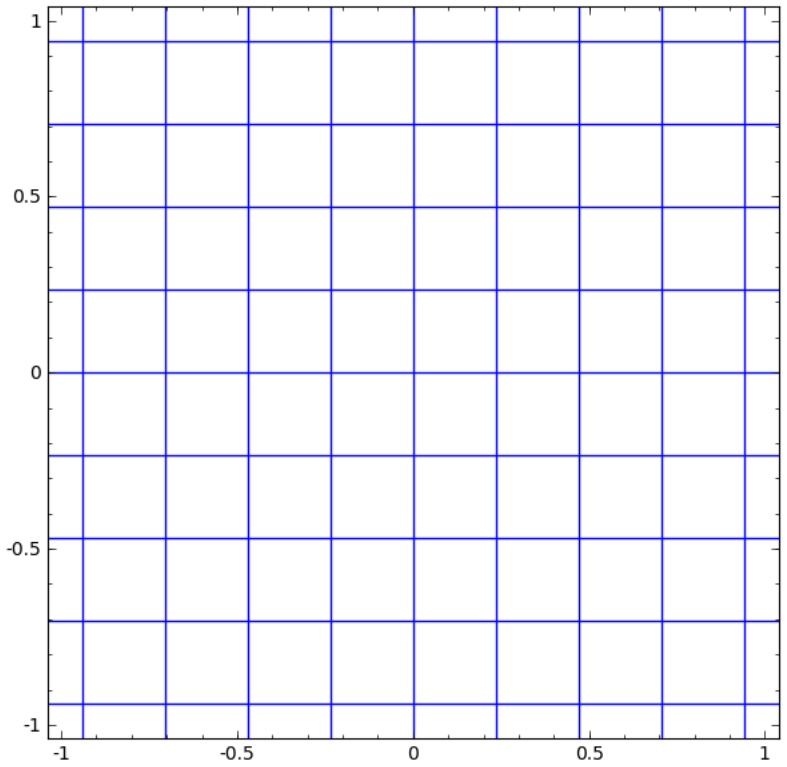
If the network can achieve 100% accuracy and the final layer is a linear separator (ala a perceptron), what does that imply about the embedding layer?



If the network can achieve 100% accuracy and the final layer is a linear separator (ala a perceptron), what does that imply about the embedding layer?

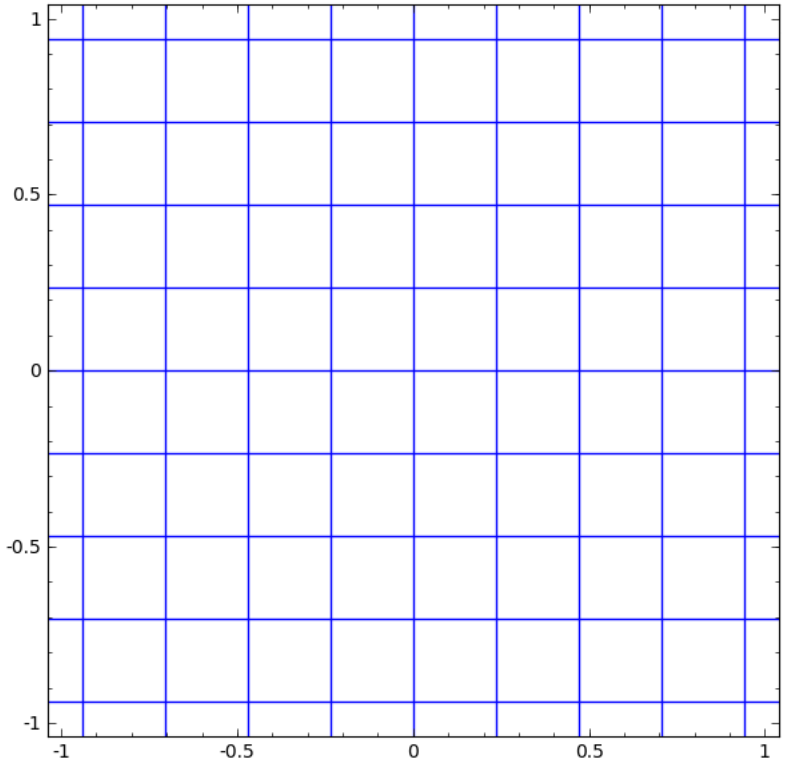
Neural Networks are learning to transform data into new learned "features" in the embedding layer. In the case of classification, the NN tries to learn linearly separable features.

A Linear Transformation applied to (x, y) coordinates

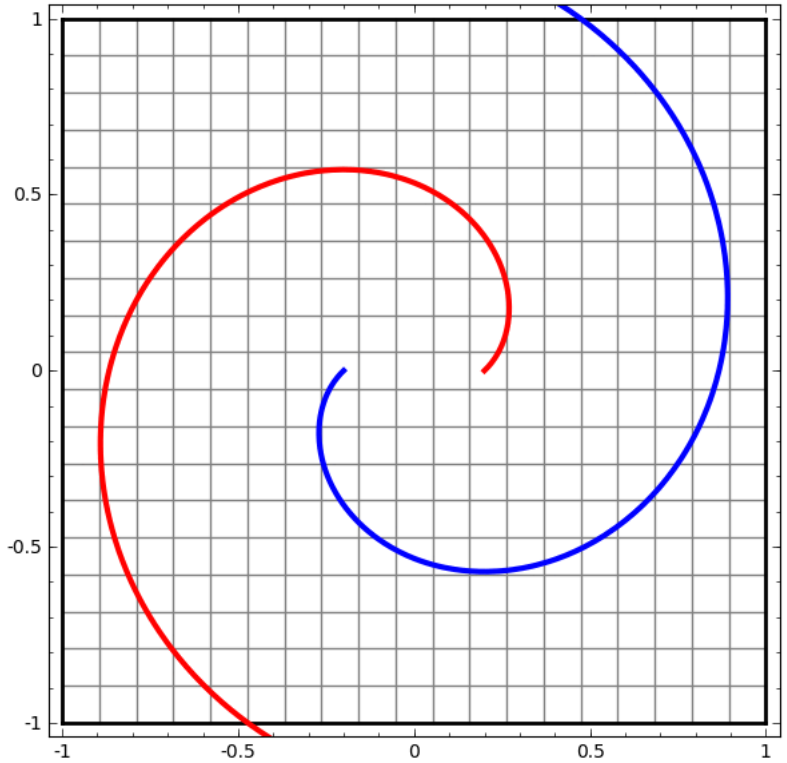


https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

A Linear Transformation applied to (x, y) coordinates



A series of linear transformations (4) applied to (x, y) coordinates to separate a spiral

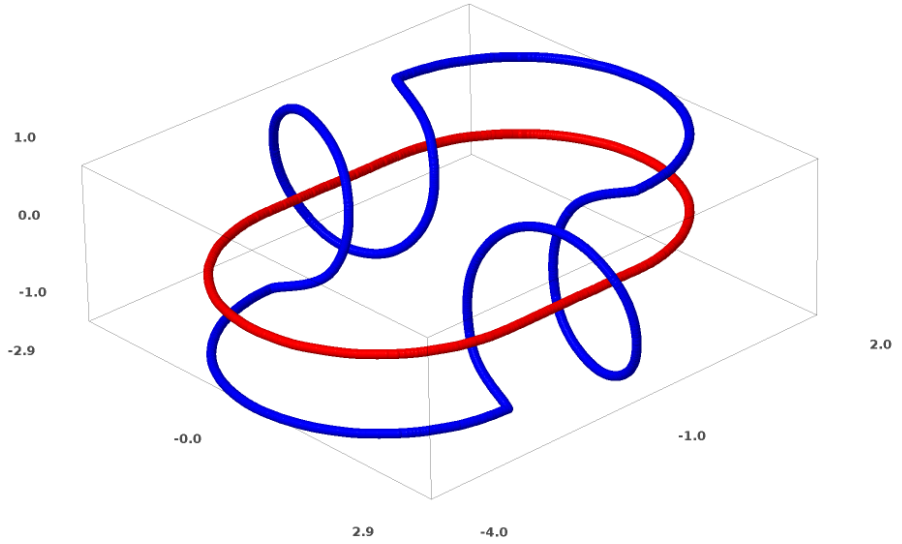


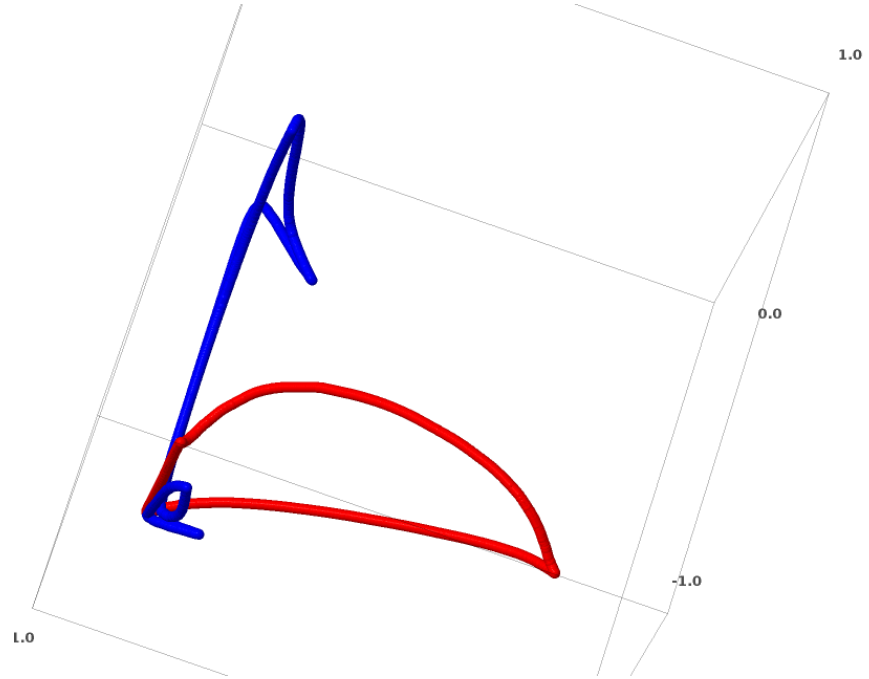
https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Manifold Hypothesis

Data may be hard to classify in its original form, but a series of transformations can transform it to a representation where classification is easy.

Neural Networks may be knot “untanglers”



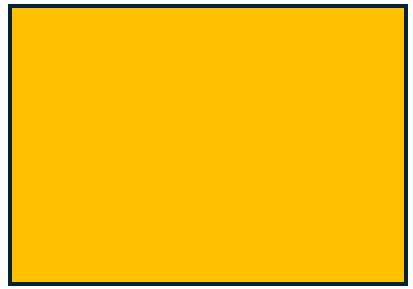


https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

What “Shape” Should your Network Be?

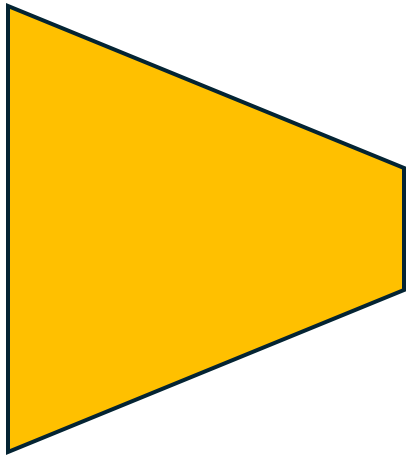
What “Shape” Should your Network Be?

Each layer is same size



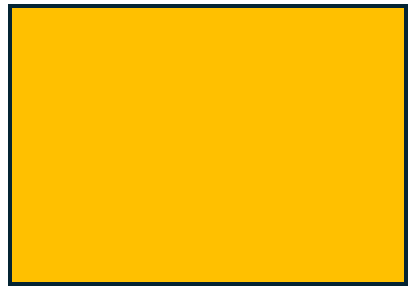
What “Shape” Should your Network Be?

Start with
largest layer



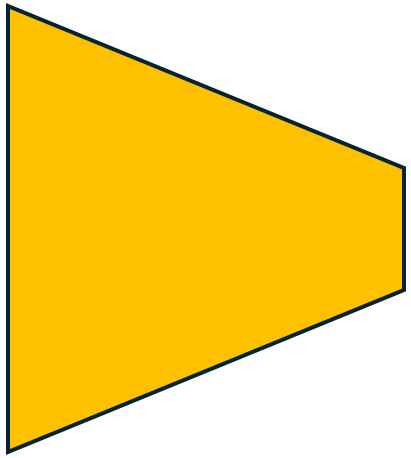
End with
smallest layer

Each layer is same size



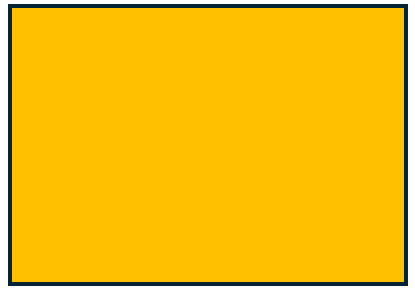
What “Shape” Should your Network Be?

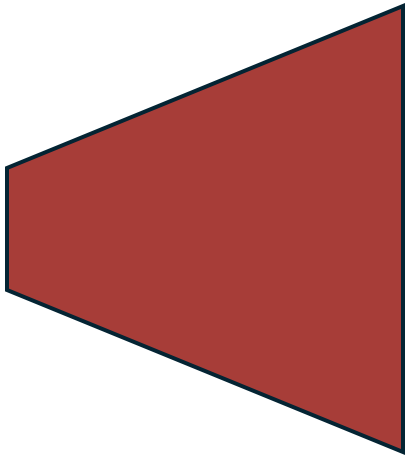
Start with
largest layer



End with
smallest layer

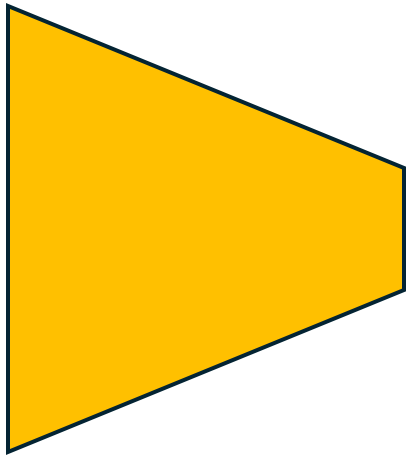
Each layer is same size





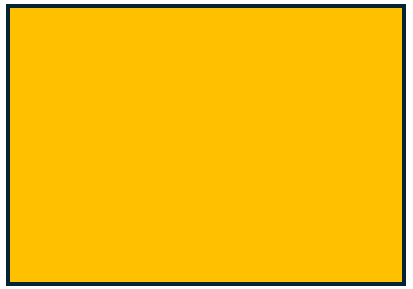
What “Shape” Should your Network Be?

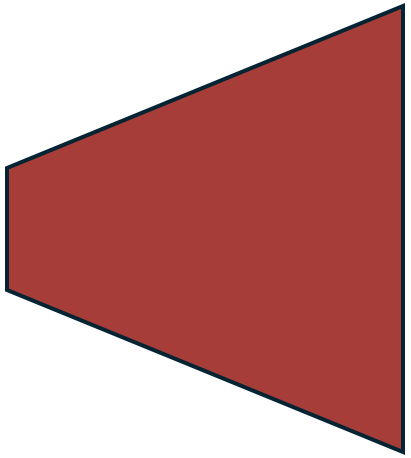
Start with
largest layer



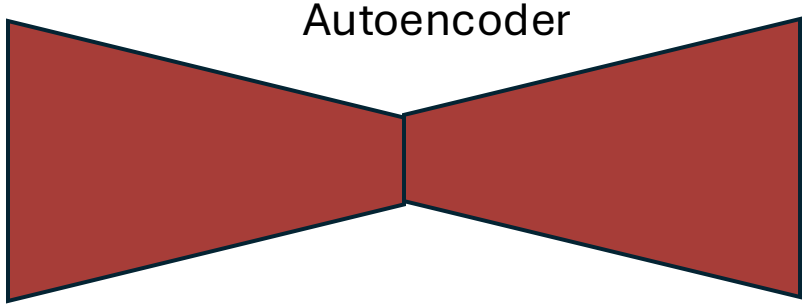
End with
smallest layer

Each layer is same size



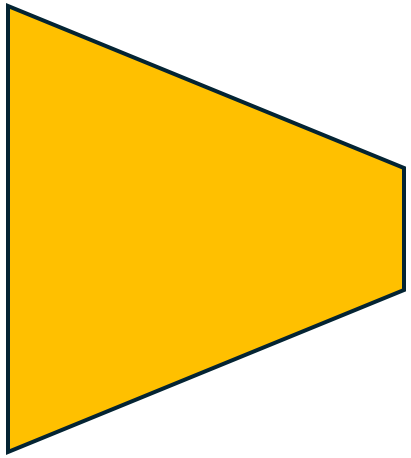


Autoencoder



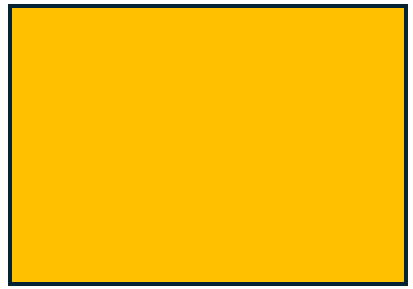
What “Shape” Should your Network Be?

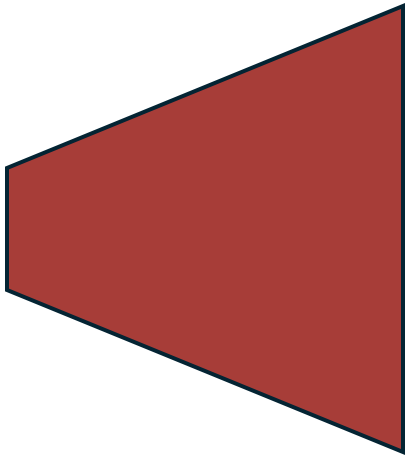
Start with
largest layer



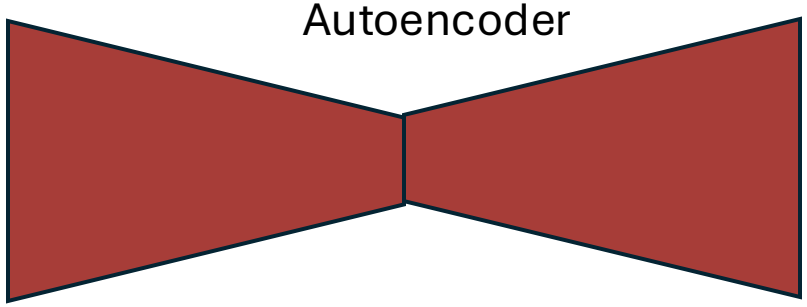
End with
smallest layer

Each layer is same size

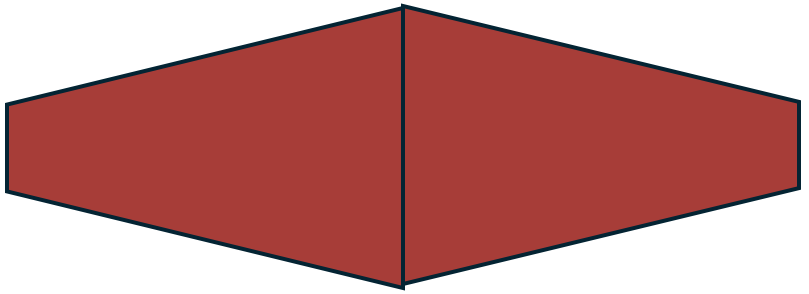




Autoencoder



U-net-style?



So How Many Layers/How Large Should the be?

- Final embedding needs to be expressive enough to represent your data in meaningful learned features
- Layer(s) before your embedding layer should be complex enough to transform your data into the embedding features.
- You are unlikely to need more than three sequential hidden linear layers for most common tasks

Overparameterization

Overparametization: Using more parameters than necessary for a ML problem.

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

~10,000 parameters in network

Overparameterization

Overparametization: Using more parameters than necessary for a ML problem.

Most of the time, networks use many more parameters than *necessary*.

In general, it's impossible to know the fewest amount of parameters that could solve a problem.

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

~10,000 parameters in network

Overparameterization

(This paper doesn't use SGD or backprop, but another optimization method)

PLAYING ATARI WITH SIX NEURONS

Giuseppe Cuccu
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

Julian Togelius
Game Innovation Lab
Tandon School of Engineering
New York University, NY, USA
julian@togelius.com

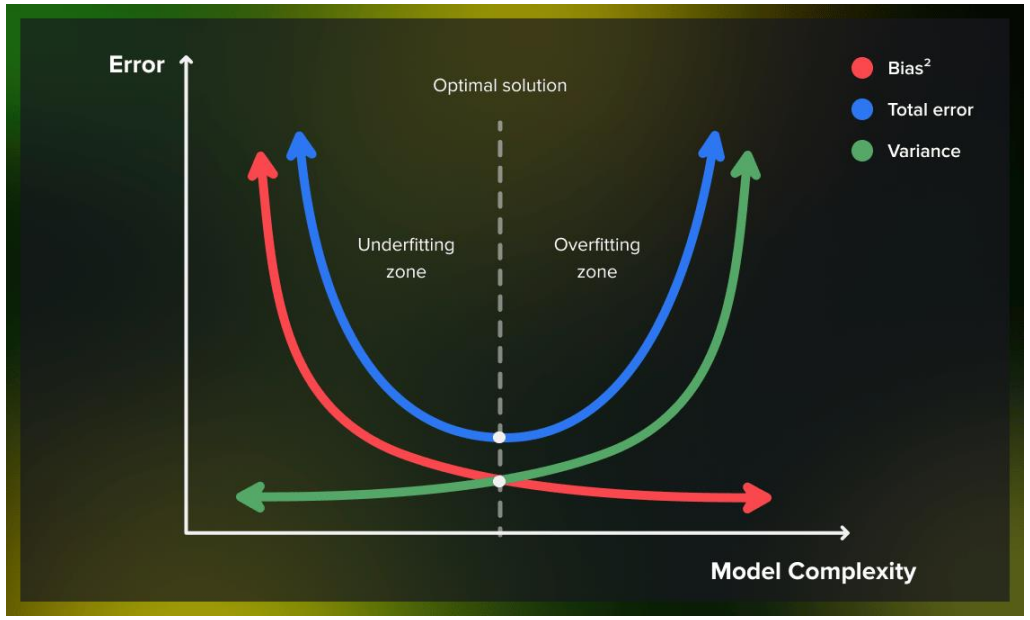
Philippe Cudré-Mauroux
eXascale Infolab
Department of Computer Science
University of Fribourg, Switzerland
name.surname@unifr.ch

ABSTRACT

Deep reinforcement learning, applied to vision-based problems like Atari games, maps pixels directly to actions; internally, the deep neural network bears the responsibility of both extracting useful information and making decisions based on it. By separating the image processing from decision-making, one could better understand the complexity of each task, as well as potentially find smaller policy representations that are easier for humans to understand and may generalize better. To this end, we propose a new method for learning policies and compact state representations separately but simultaneously for policy approximation in reinforcement learning. State representations are generated by an encoder based on two novel algorithms: Increasing Dictionary Vector Quantization makes the encoder capable of growing its dictionary size over time, to address new observations as they appear in an open-ended online-learning context; Direct Residuals Sparse Coding encodes observations by disregarding reconstruction error minimization, and aiming instead for highest information inclusion. The encoder autonomously selects observations online to train on, in order to maximize code sparsity. As the dictionary size increases, the encoder produces increasingly larger inputs for the neural network: this is addressed by a variation of the Exponential Natural Evolution Strategies algorithm which adapts its probability distribution dimensionality along the run. We test our system on a selection of Atari games using tiny neural networks of only 6 to 18 neurons (depending on the game's controls). These are still capable of achieving results comparable—and occasionally superior—to state-of-the-art techniques which use two orders of magnitude more neurons.

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



https://serokell.io/blog/bias-variance-tradeoff

A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

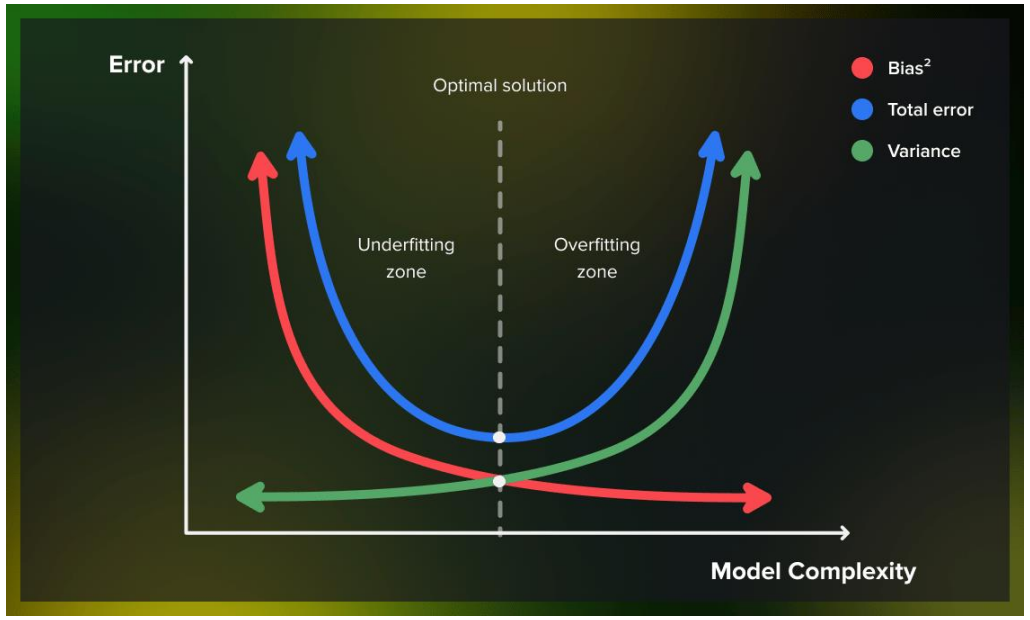
Abstract

The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



If you are overfitting, reduce model complexity (smaller width/fewer layers). If underfitting, add more model complexity.

https://serokell.io/blog/bias-variance-tradeoff

A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

Abstract

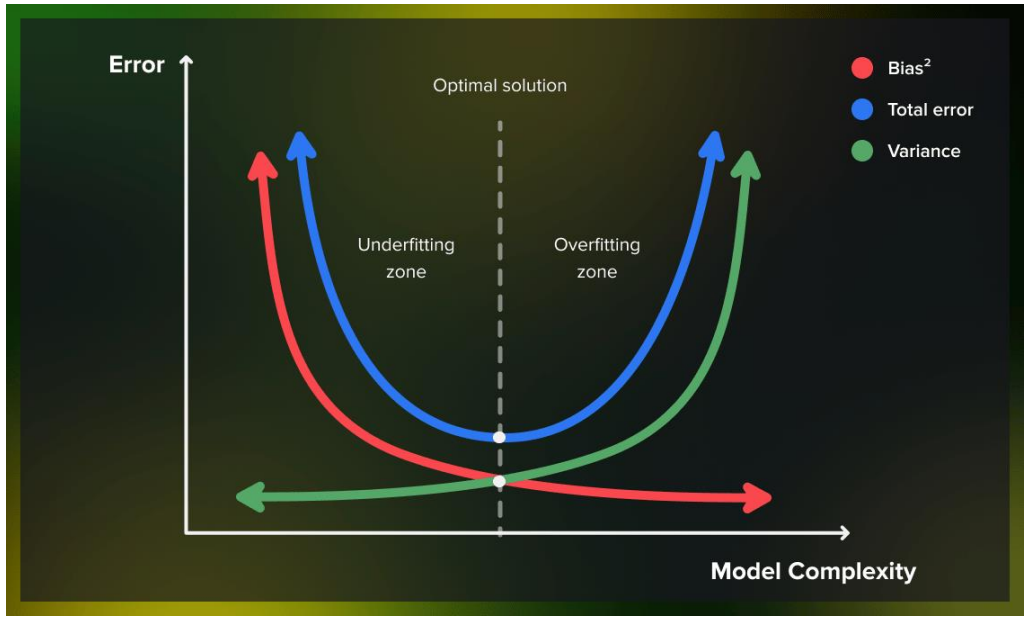
The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

(We will cover other techniques for managing overfitting soon)

Overparameterization

Bias-Variance Tradeoff
(Traditional Understanding)



If you are overfitting, reduce model complexity (smaller width/fewer layers). If underfitting, add more model complexity.

https://serokell.io/blog/bias-variance-tradeoff

A Farewell to the Bias-Variance Tradeoff?
An Overview of the Theory of Overparameterized Machine Learning

Yehuda Dar* **Vidya Muthukumar†** **Richard G. Baraniuk‡**

Abstract

The rapid recent progress in machine learning (ML) has raised a number of scientific questions that challenge the longstanding dogma of the field. One of the most important riddles is the good empirical generalization of *overparameterized* models. Overparameterized models are excessively complex with respect to the size of the training dataset, which results in them perfectly fitting (i.e., *interpolating*) the training data, which is usually noisy. Such interpolation of noisy data is traditionally associated with detrimental overfitting, and yet a wide range of interpolating models – from simple linear models to deep neural networks – have recently been observed to generalize extremely well on fresh test data. Indeed, the recently discovered *double descent* phenomenon has revealed that highly overparameterized models often improve over the best underparameterized model in test performance.

Understanding learning in this overparameterized regime requires new theory and foundational empirical studies, even for the simplest case of the linear model. The underpinnings of this understanding have been laid in very recent analyses of overparameterized linear regression and related statistical learning tasks, which resulted in precise analytic characterizations of double descent. This paper provides a succinct overview of this emerging *theory of overparameterized ML* (henceforth abbreviated as TOPML) that explains these recent findings through a statistical signal processing perspective. We emphasize the unique aspects that define the TOPML research area as a subfield of modern ML theory and outline interesting open questions that remain.

Optimizers

Optimizers

- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

Optimizers

- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

- How do you choose between them?

Optimizers

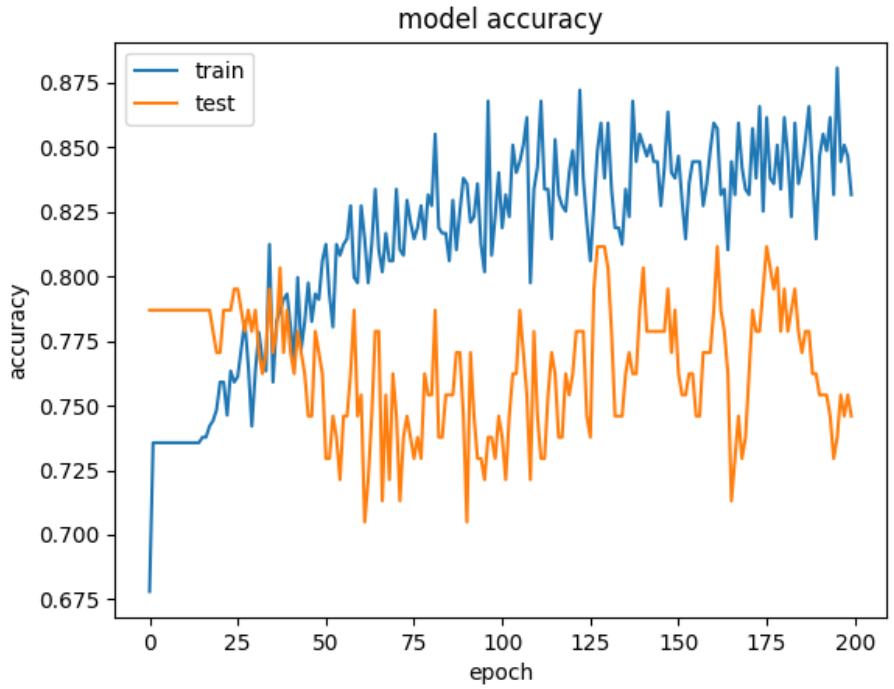
- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp,… the list is ever growing

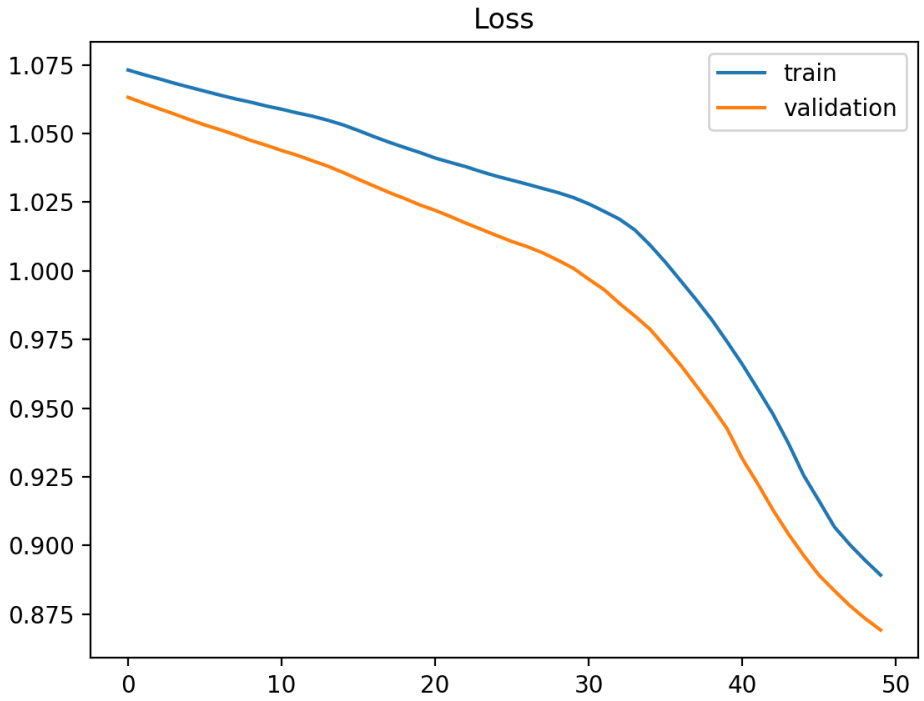
- How do you choose between them?

- Just use Adam.
  - The only downside is that it might work so well that you end up overfitting.
  - Suggested initial learning rate of 3e-4

Batch Size and Learning Rate

Having too small a batch or too high a learning rate can cause variance in training/validation loss – symptoms often look similar





General Tips

General Tips

- Don’t change too much at once.

General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance

General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

General Tips

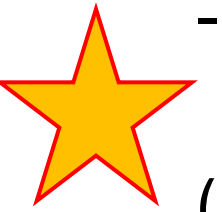
- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

(Use the scientific method)

General Tips

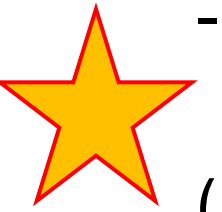
- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.

(Use the scientific method)



General Tips

- Don’t change too much at once.
- Keep track of parameters you’ve tested and track their performance
- Don’t just randomly guess parameters, apply critical thinking, come up with a hypothesis and test your hypothesis.



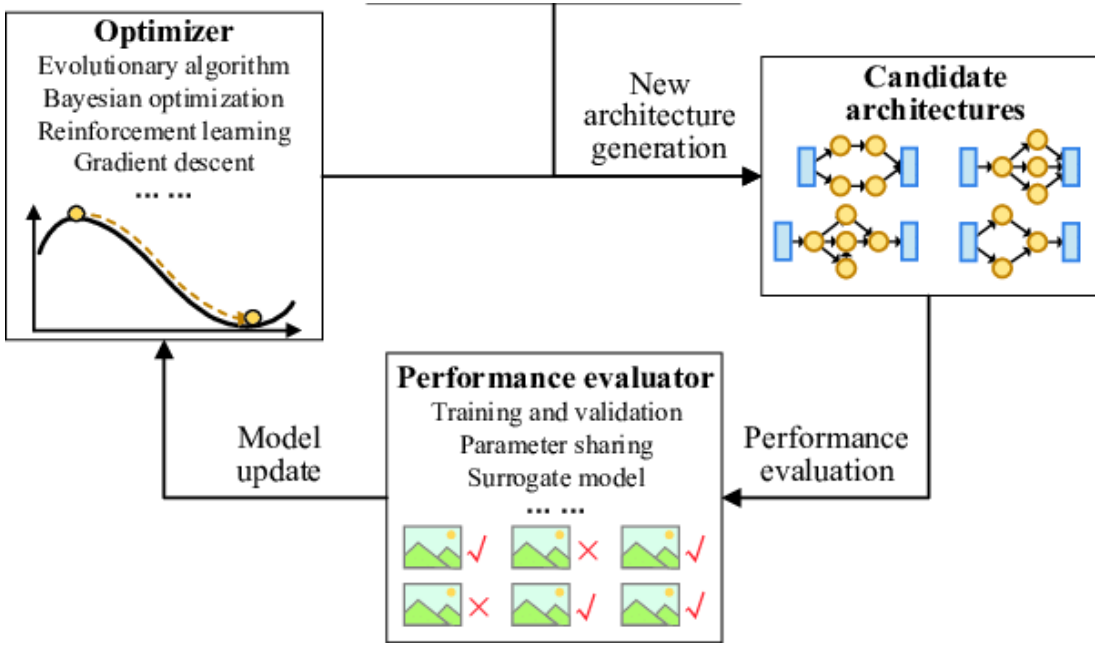
(Use the scientific method)

Andrej Karpathy: A recipe for training neural networks
https://karpathy.github.io/2019/04/25/recipe/

AutoML

Neural Architecture Search (NAS)

Changing hyperparameters results in different performance, can we run an optimization algorithm on our hyperparameters?



Pros:

- No longer need human input
- May find better hyperparameters than humans

Cons:

- Takes a very long time…
- Hyperparameters are discrete and highly dependent (e.g., width/depth), it's a really hard optimization problem…

https://www.researchgate.net/publication/353166978_Action_Command_Encoding_for_Surrogate_Assisted_Neural_Architecture_Search

AutoML

Option #1: Grid search

Define a set of possible parameters (i.e., learning rates, width, depth, etc)

Try every combination of hyperparameters possible, pick setting with best validation set performance.

AutoML

Option #1: Grid search

Define a set of possible parameters (i.e., learning rates, width, depth, etc)

Try every combination of hyperparameters possible, pick setting with best validation set performance.

What are some downsides of grid search?

AutoML

AutoML

Option #2: Bayesian Optimization

AutoML

Option #2: Bayesian Optimization

We believe the performance of hyperparameters that are close together, should have similar results.

AutoML

Option #2: Bayesian Optimization

We believe the performance of hyperparameters that are close together, should have similar results.

Define a set of possible parameters (i.e., learning rates, width, depth, etc)

AutoML

Option #2: Bayesian Optimization

We believe the performance of hyperparameters that are close together, should have similar results.

Define a set of possible parameters (i.e., learning rates, width, depth, etc)

Try sets of possible hyperparameters, each with some probability.

AutoML

Option #2: Bayesian Optimization

We believe the performance of hyperparameters that are close together, should have similar results.

Define a set of possible parameters (i.e., learning rates, width, depth, etc)

Try sets of possible hyperparameters, each with some probability.

- The probability that you try a specific hyperparameter setting depends on the performance of nearby hyperparameter settings.

AutoML

Option #2: Bayesian Optimization

We believe the performance of hyperparameters that are close together, should have similar results.

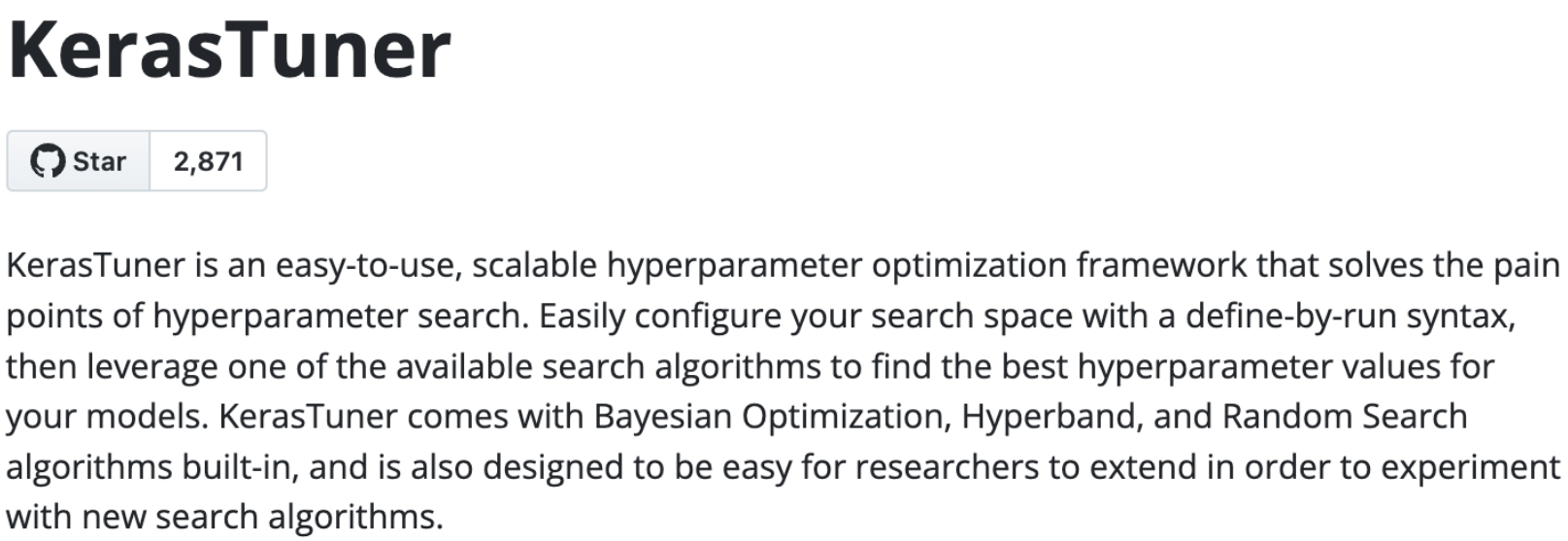
Define a set of possible parameters (i.e., learning rates, width, depth, etc)

Try sets of possible hyperparameters, each with some probability.

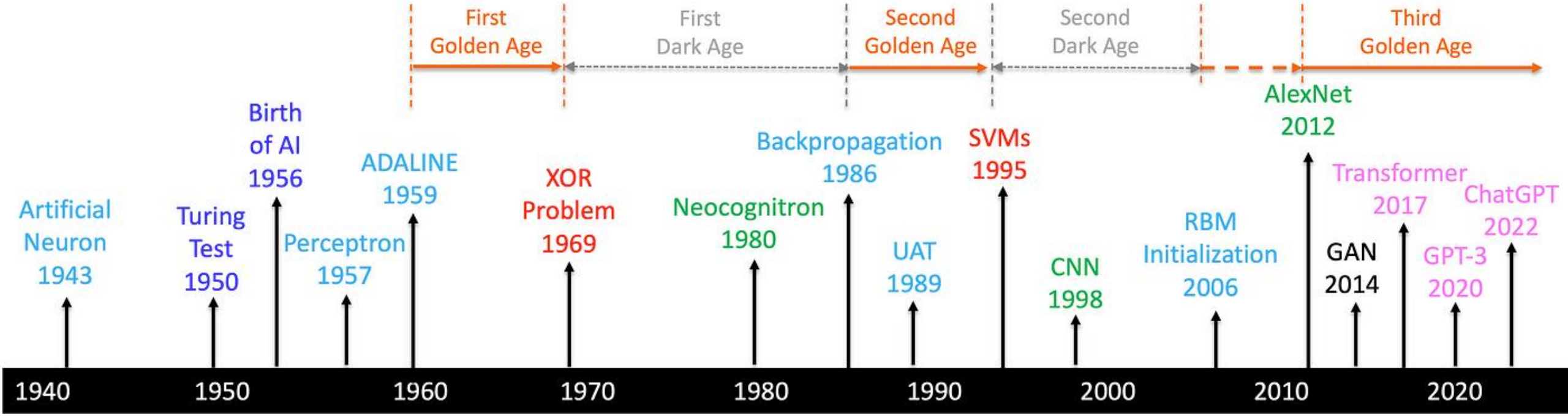
- The probability that you try a specific hyperparameter setting depends on the performance of nearby hyperparameter settings.
- Also track uncertainty of hyperparameters (i.e., settings you have not tried something close to before)

AutoML

Keras tuner is compatible with Tensorflow, Pytorch, and Jax and has various automatic hyperparameter tuning methods



A Brief History of AI with Deep Learning




McCulloch-Pitts

Rosenblatt

Widrow-Hoff

Minsky-Papert

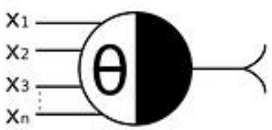
Rumelhart, Hinton et al.

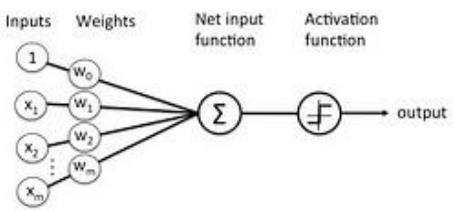
LeCun

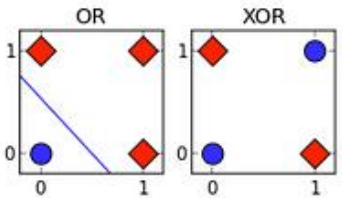
Hinton-Ruslan

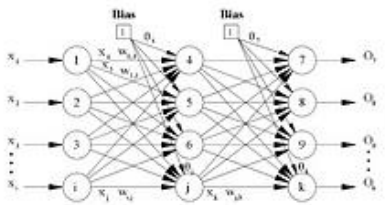
Krizhevsky et al.

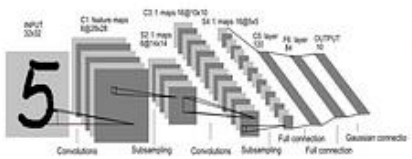
Vaswani

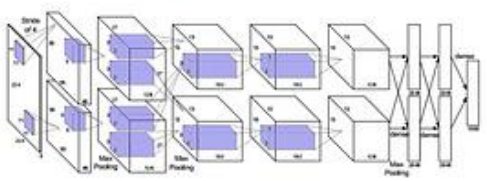












https://medium.com/@lmpo/a-brief-history-of-ai-with-deep-learning-26f7948bc87b

What has happened in the last 15 years?

What has changed?

1. Power and efficiency of compute (GPUs)
2. Availability of data (the internet)
3. New Architectures (e.g., CNNs, Transformers)

Issues with MLPs

1. Resource Intensive
2. Difficult to incorporate certain types of information
3. (and more)

Issues with MLPs

1. Resource Intensive
2. Difficult to incorporate certain types of information
3. (and more)

GPUs to the rescue!

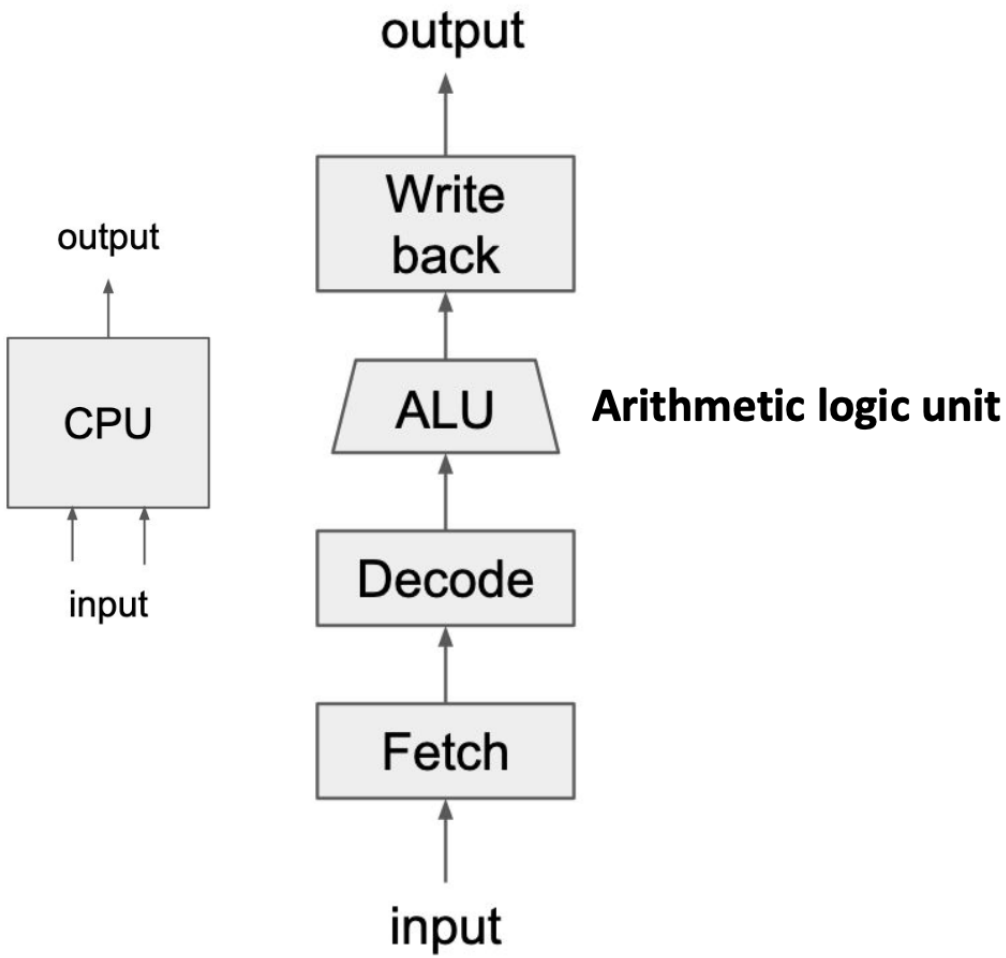
- *Graphics* Processing Units
- GPUs are really good at computing mathematical operations in parallel!
- Matrix multiplication == many **independent** multiply and add operations

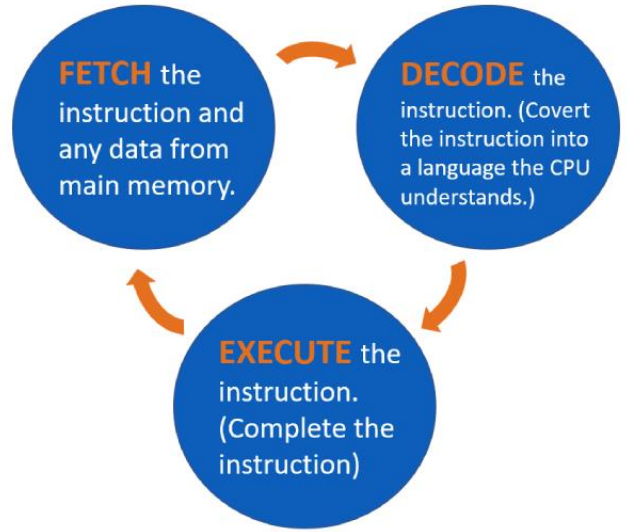
  Easily parallelizable

  GPUs are great for this!

Image courtesy: https://global.aorus.com/blog-detail.php?i=878

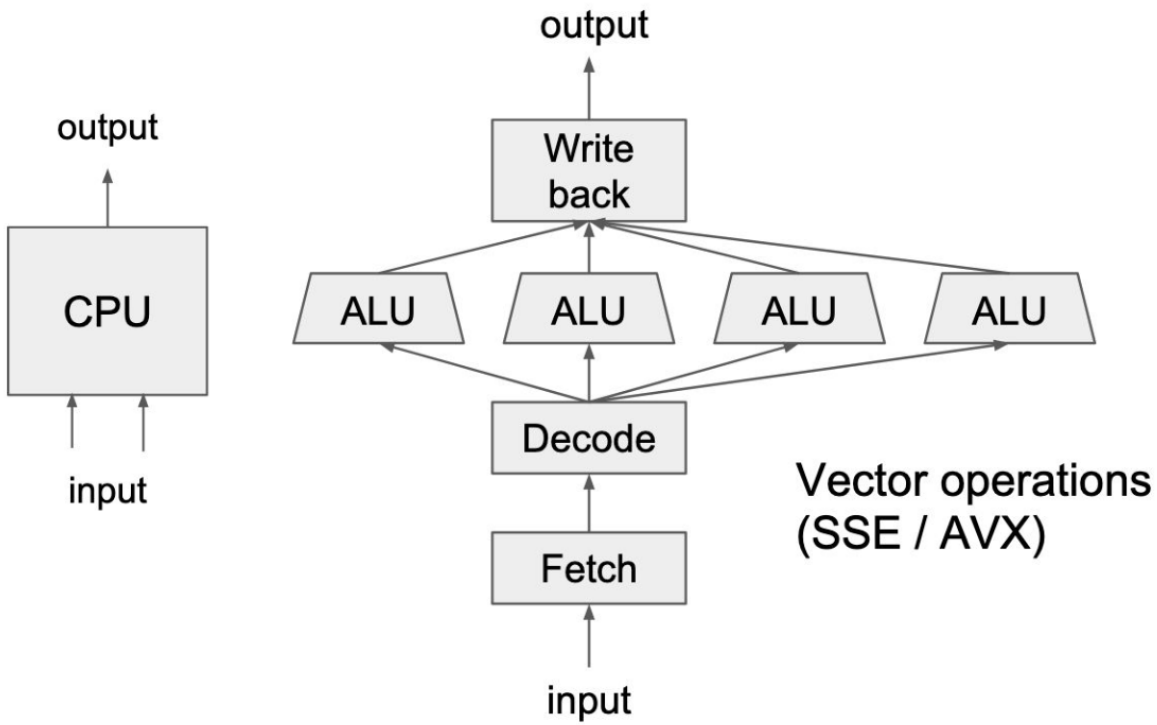
CPU v/s GPU

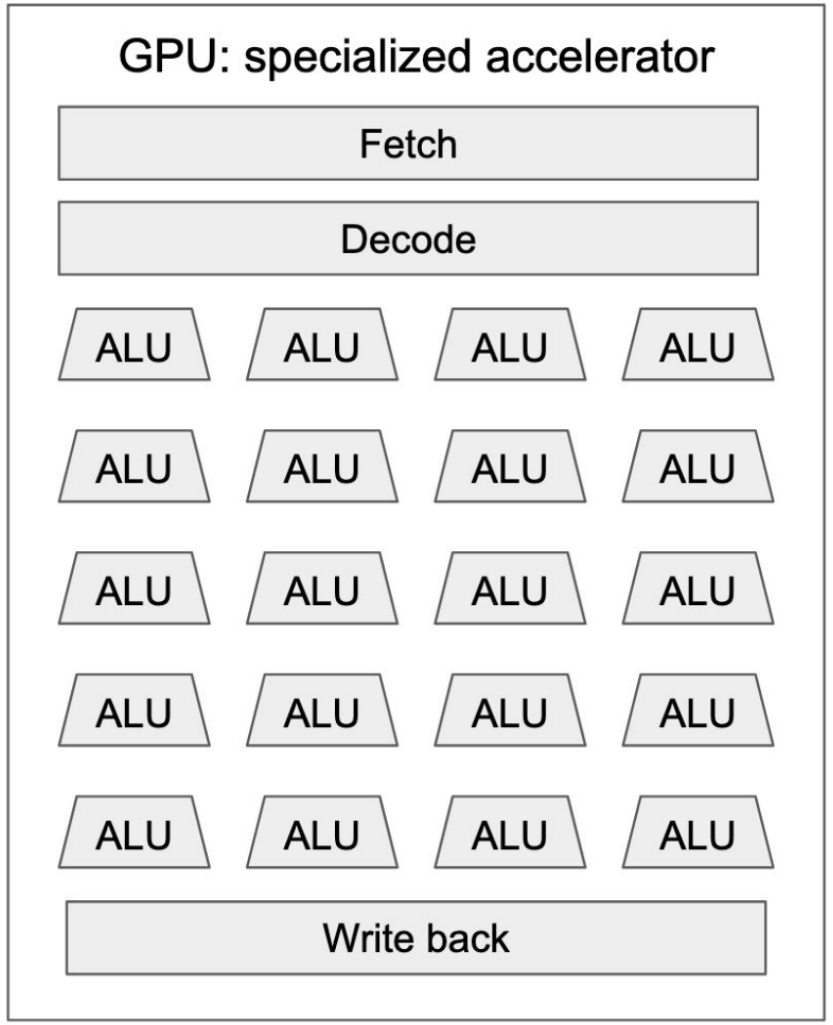




https://dlsys.cs.washington.edu/pdf/lecture5.pdf

CPU v/s GPU





https://dlsys.cs.washington.edu/pdf/lecture5.pdf

GPU-Parallel Acceleration

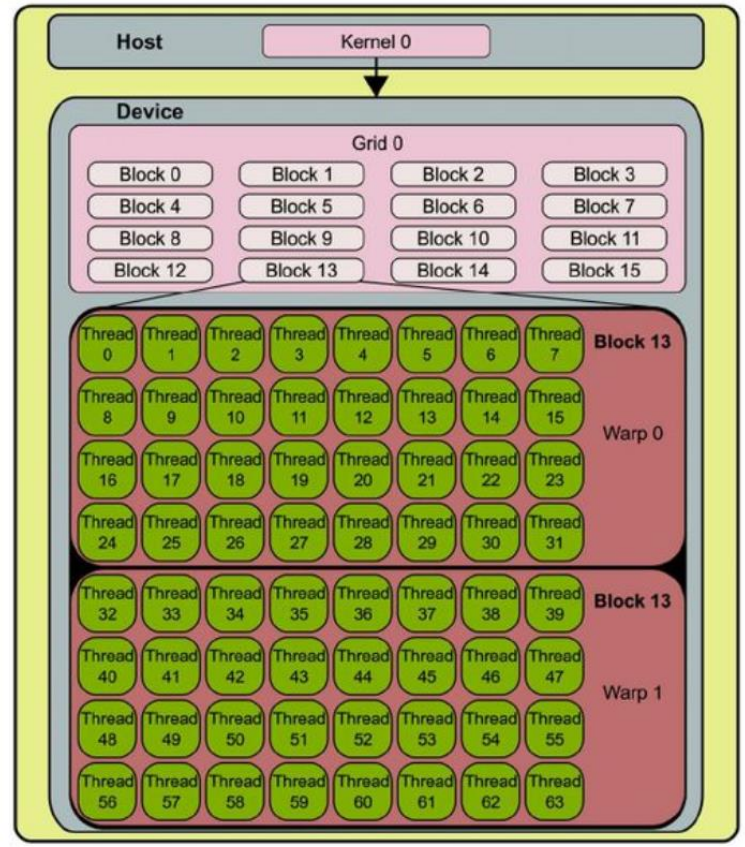
Compute Unified Device Architecture is a parallel computing platform and application programming interface (API)

- User code (***kernels***) is compiled on the ***host*** (the CPU) and then transferred to the ***device*** (the GPU)
- Kernel is executed as a ***grid***
- Each grid has multiple ***thread blocks***
- Each thread block has multiple ***warps***

A warp is the basic schedule unit in kernel execution

A warp consists of 32 threads

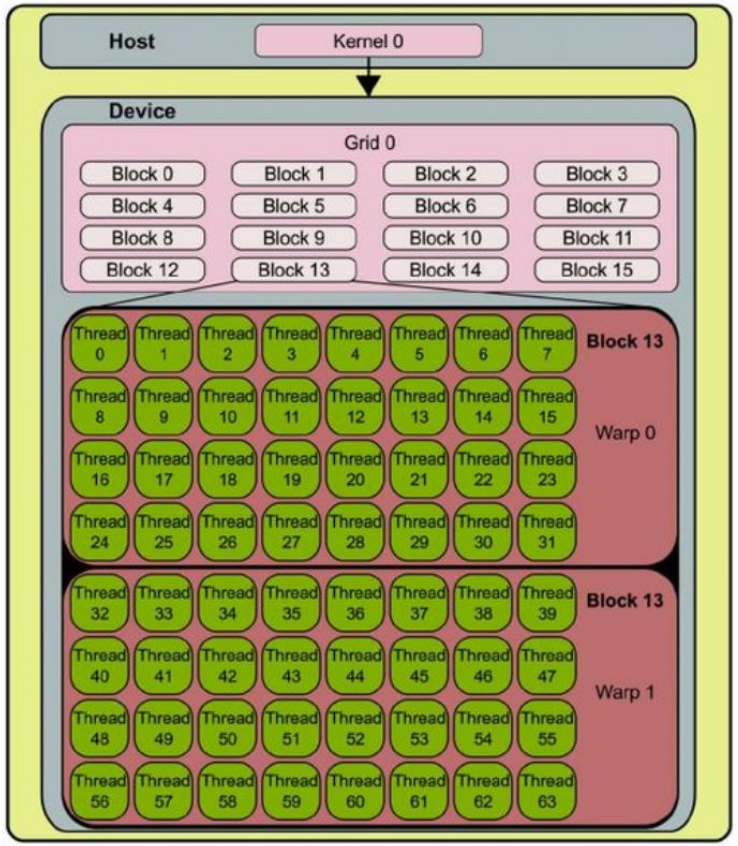
CUDA compute model



https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs

GPU-Parallel Acceleration

CUDA compute model



- Programmer decides how they want to parallelize the computation across grids and blocks
  - Modern deep learning frameworks take care of this for you

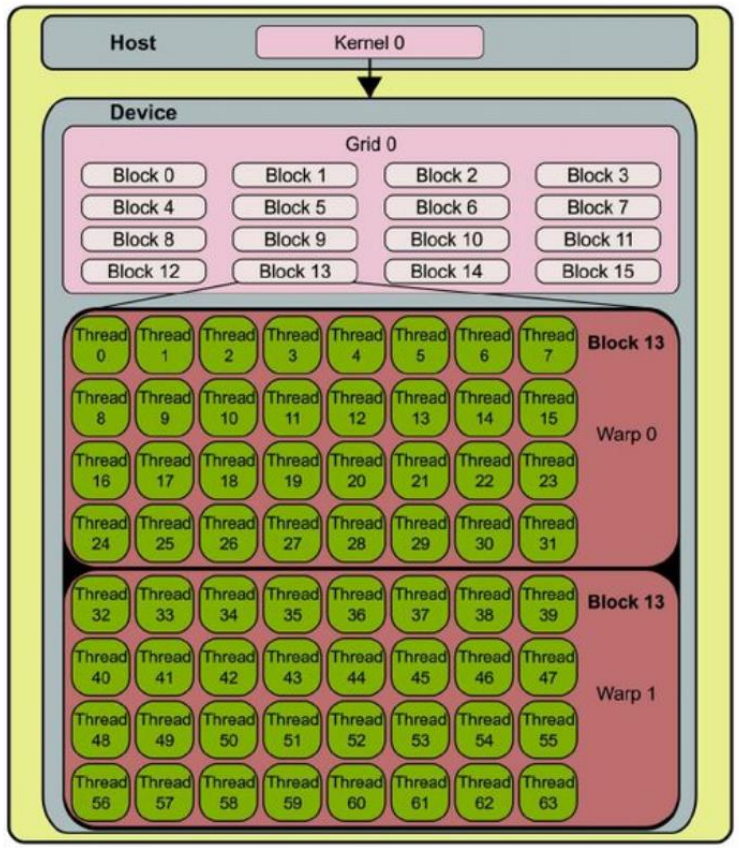
- CUDA compiler figures out how to schedule these units of computation on to the physical hardware

https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs

GPU-Parallel Acceleration

Any questions?

CUDA compute model



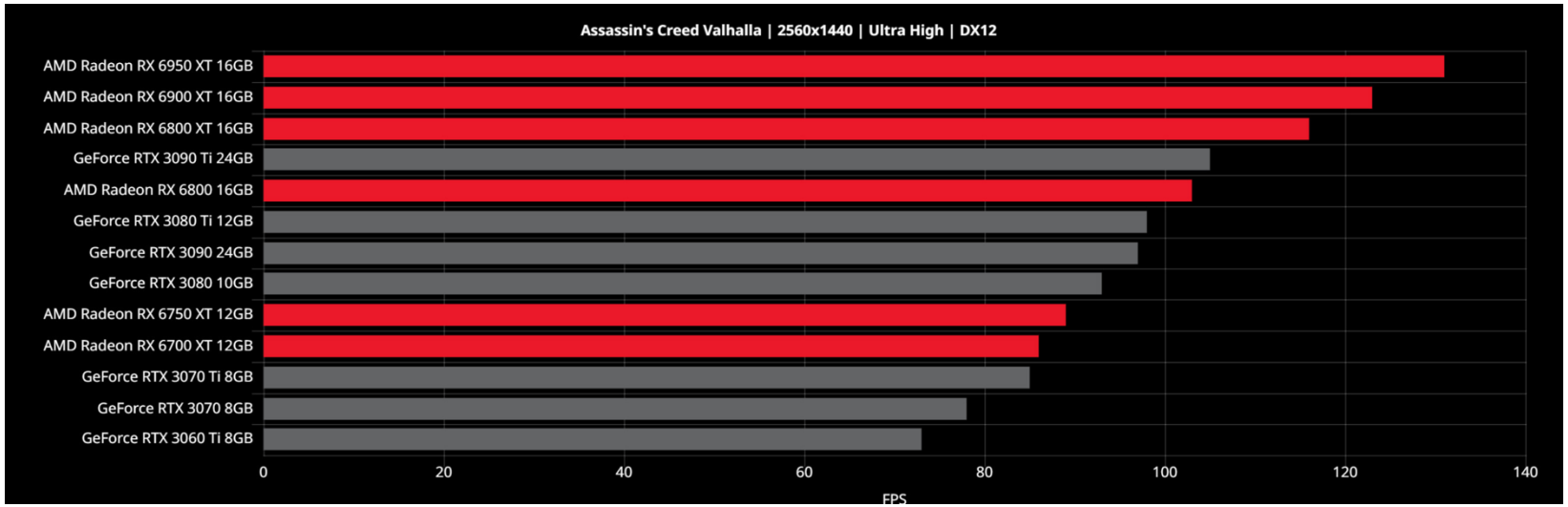
- Upshot: order of magnitude speedups!
- Example: training CNN on CIFAR-10 dataset

Device	Speed of training, examples/sec
2 x AMD Opteron 6168	440
i7-7500U	415
GeForce 940MX	1190
GeForce 1070	6500

From: https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c

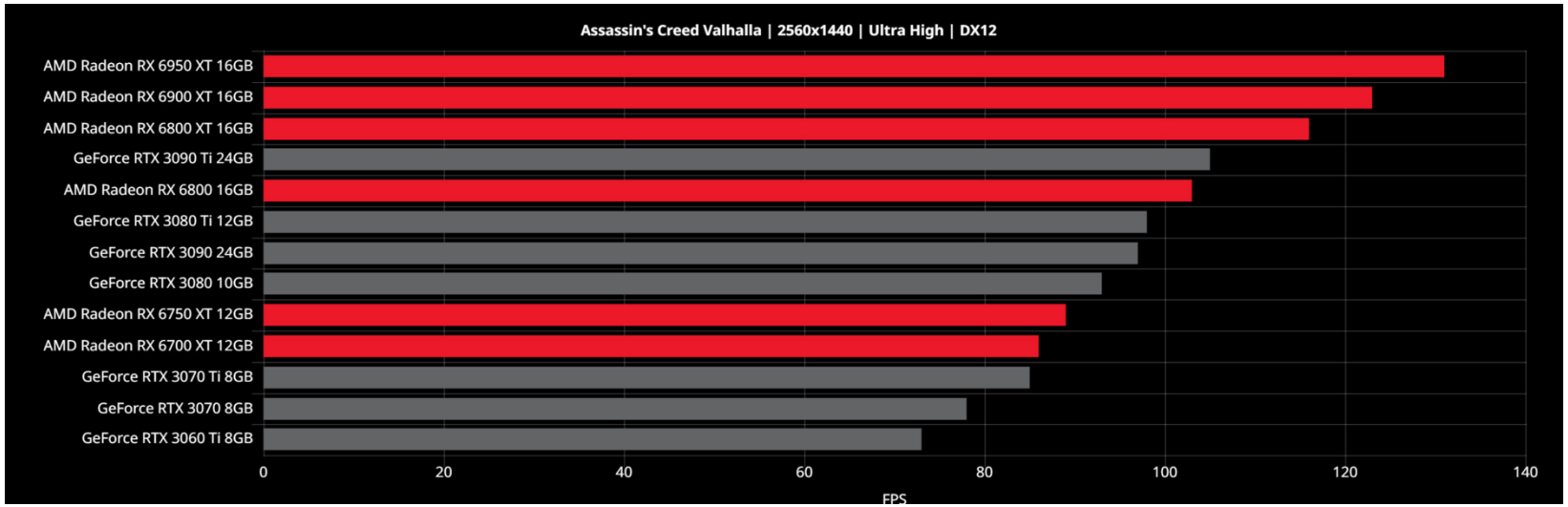
https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs

AMD GPUs are competitive for gaming and graphics, why not for AI?



(With a benchmarking tool made by AMD)

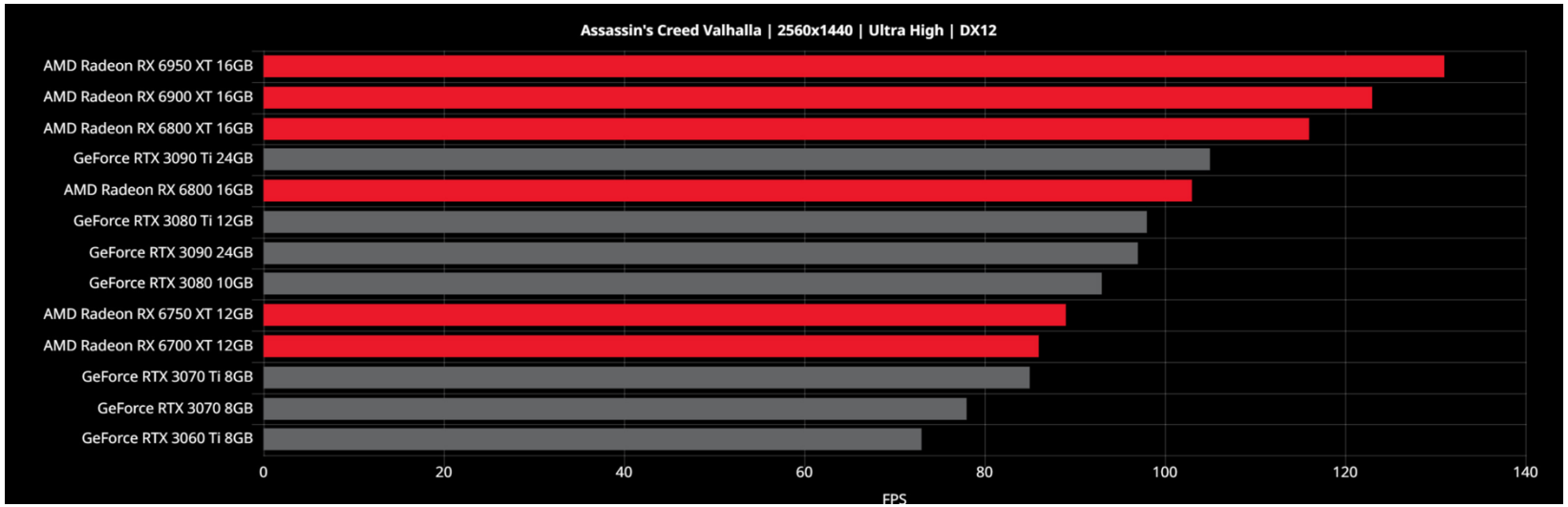
AMD GPUs are competitive for gaming and graphics, why not for AI?



(With a benchmarking tool made by AMD)

- CUDA is far better than competitors (AMD)
  - Easier to use
  - Better optimization
- AMD makes GPUs for graphics, NVIDIA makes GPUs for AI

AMD GPUs are competitive for gaming and graphics, why not for AI?



(With a benchmarking tool made by AMD)

- CUDA is far better than competitors (AMD)
  - Easier to use
  - Better optimization
- AMD makes GPUs for graphics, NVIDIA makes GPUs for AI

CUDA is Still a Giant Moat for NVIDIA

Despite everyone's focus on hardware, the software of AI is what protects NVIDIA

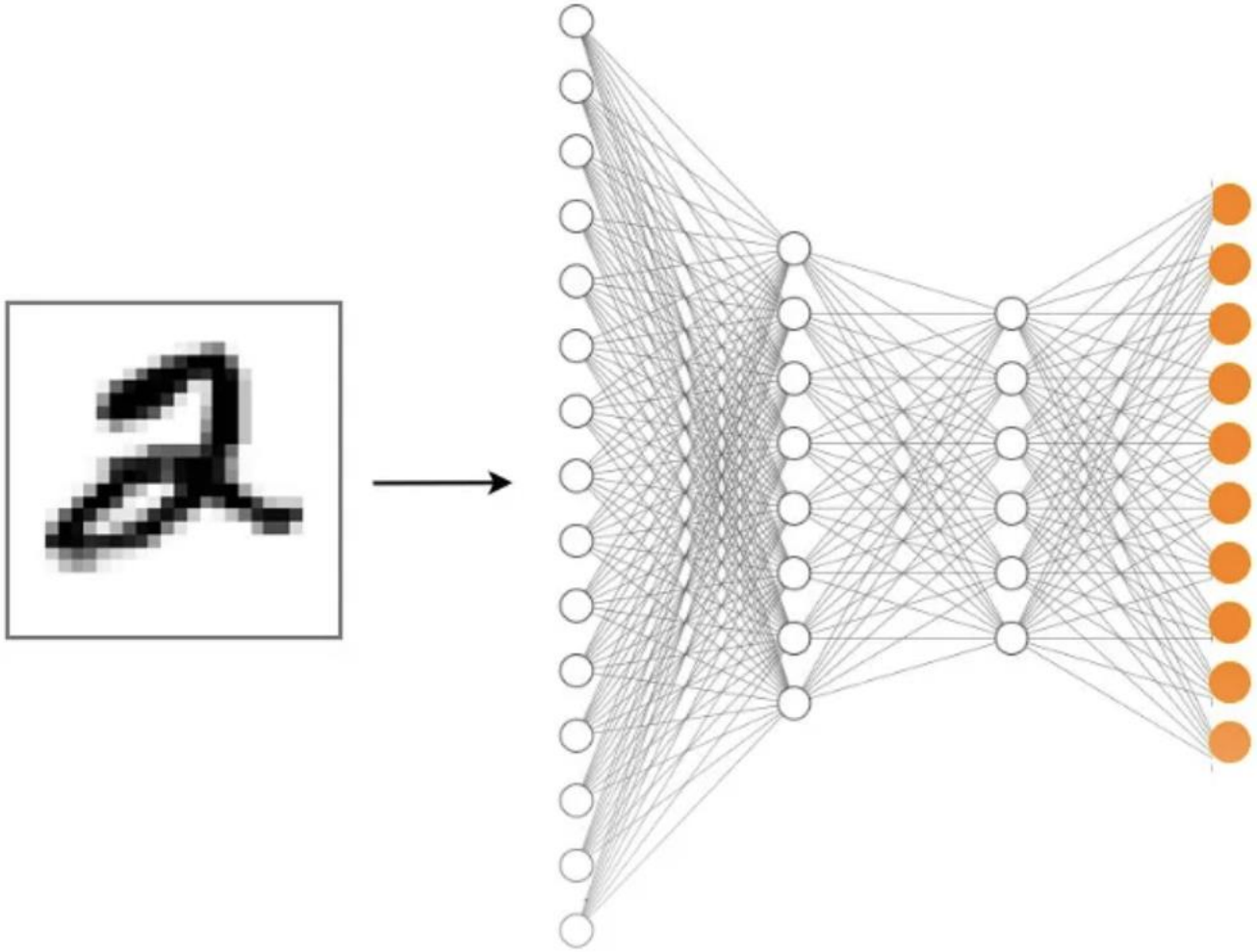
JAMES WANG

MAR 23, 2024

Issues with MLPs

1. Resource Intensive
2. Difficult to incorporate certain types of information

MLPs and Spatial Reasoning



https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

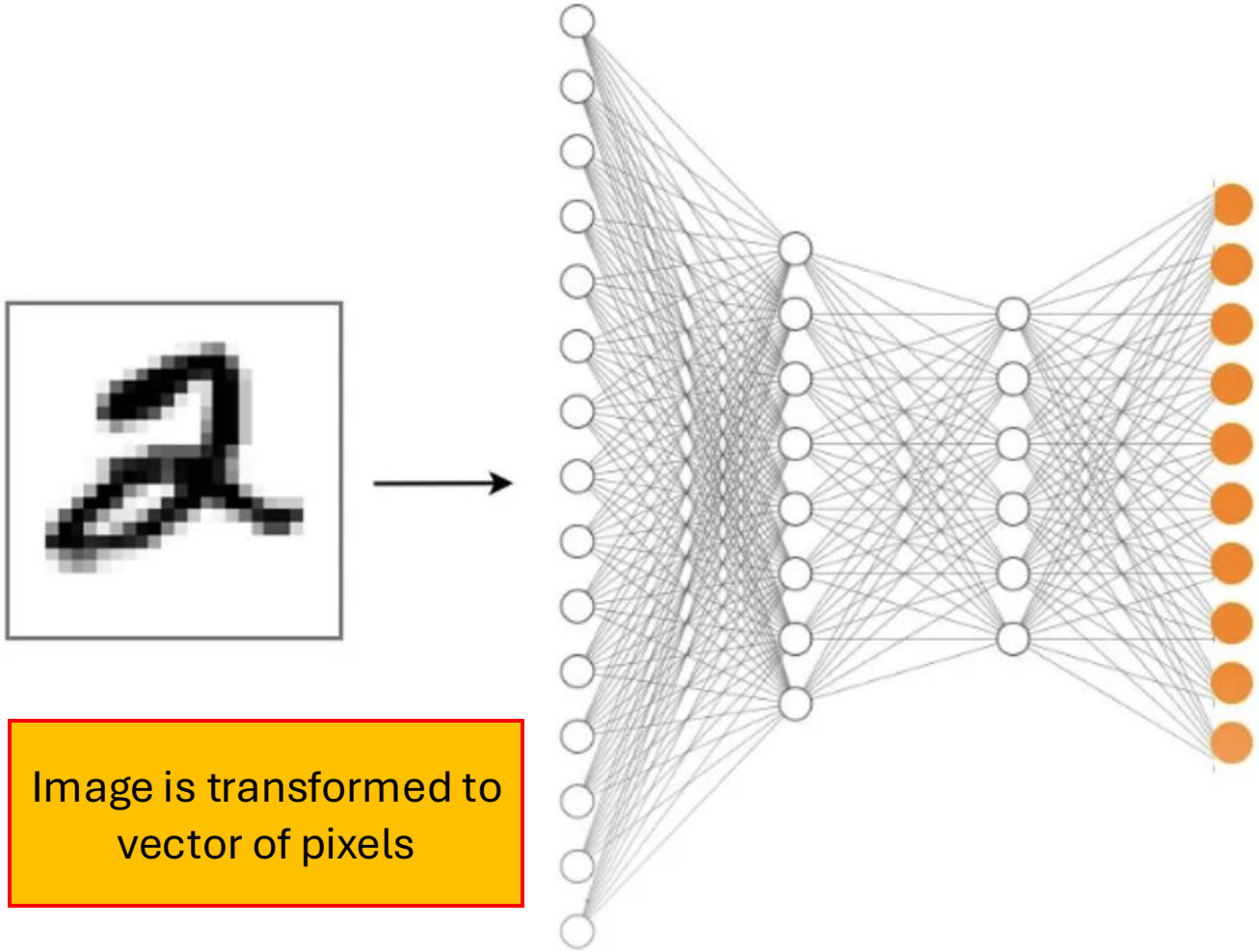


Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

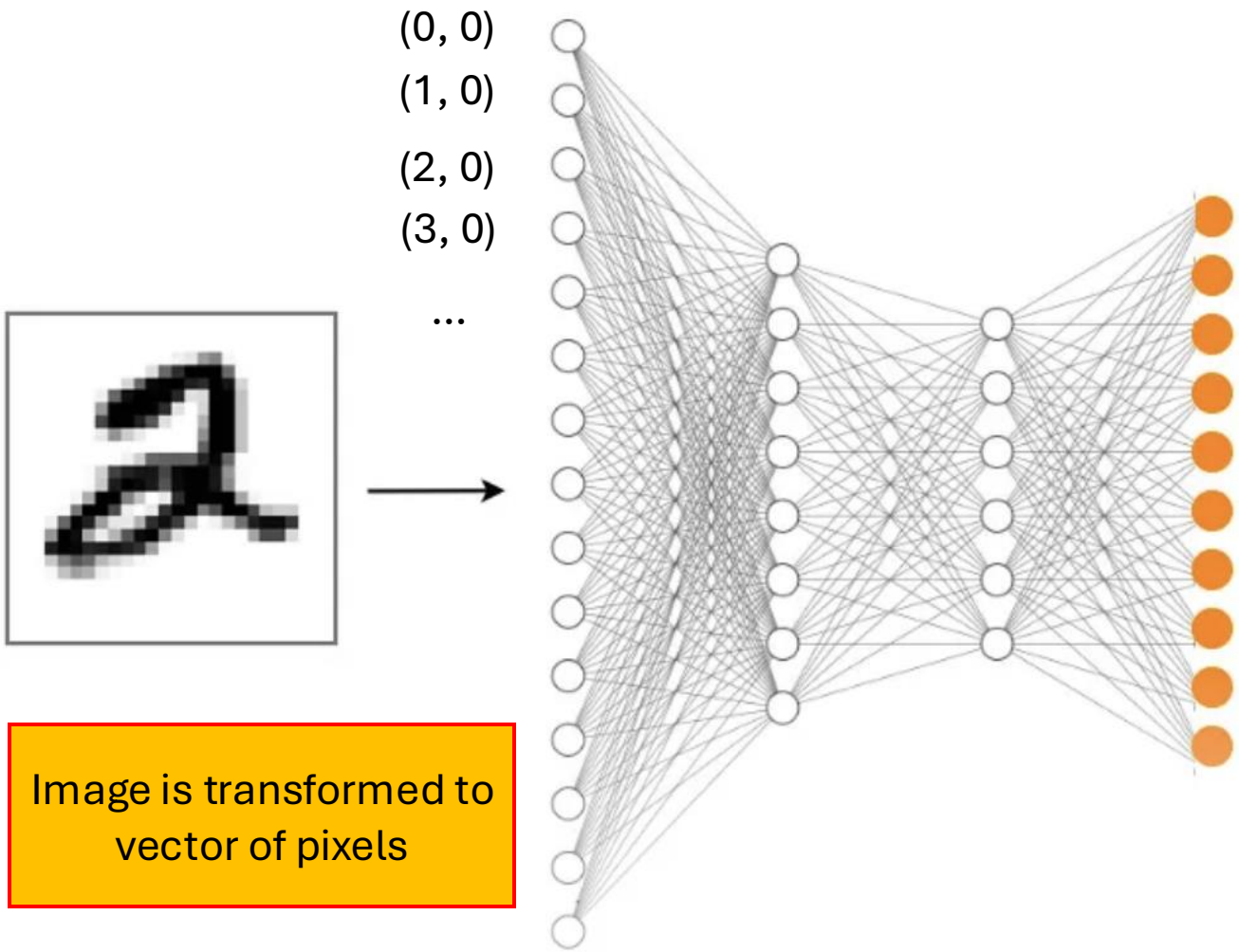


Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

What would happen if we permuted the ordering of the pixels?

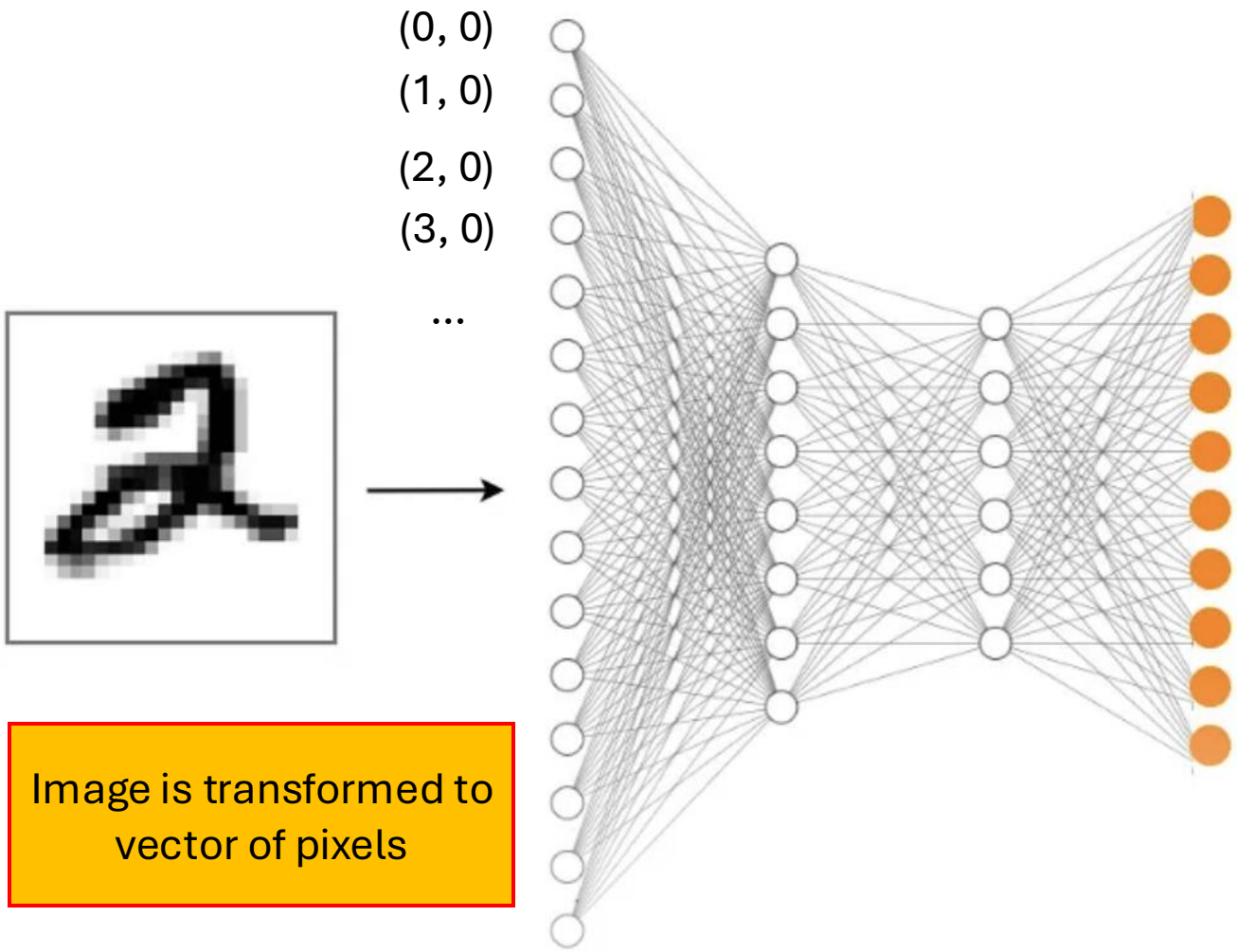


Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

What would happen if we permuted the ordering of the pixels?

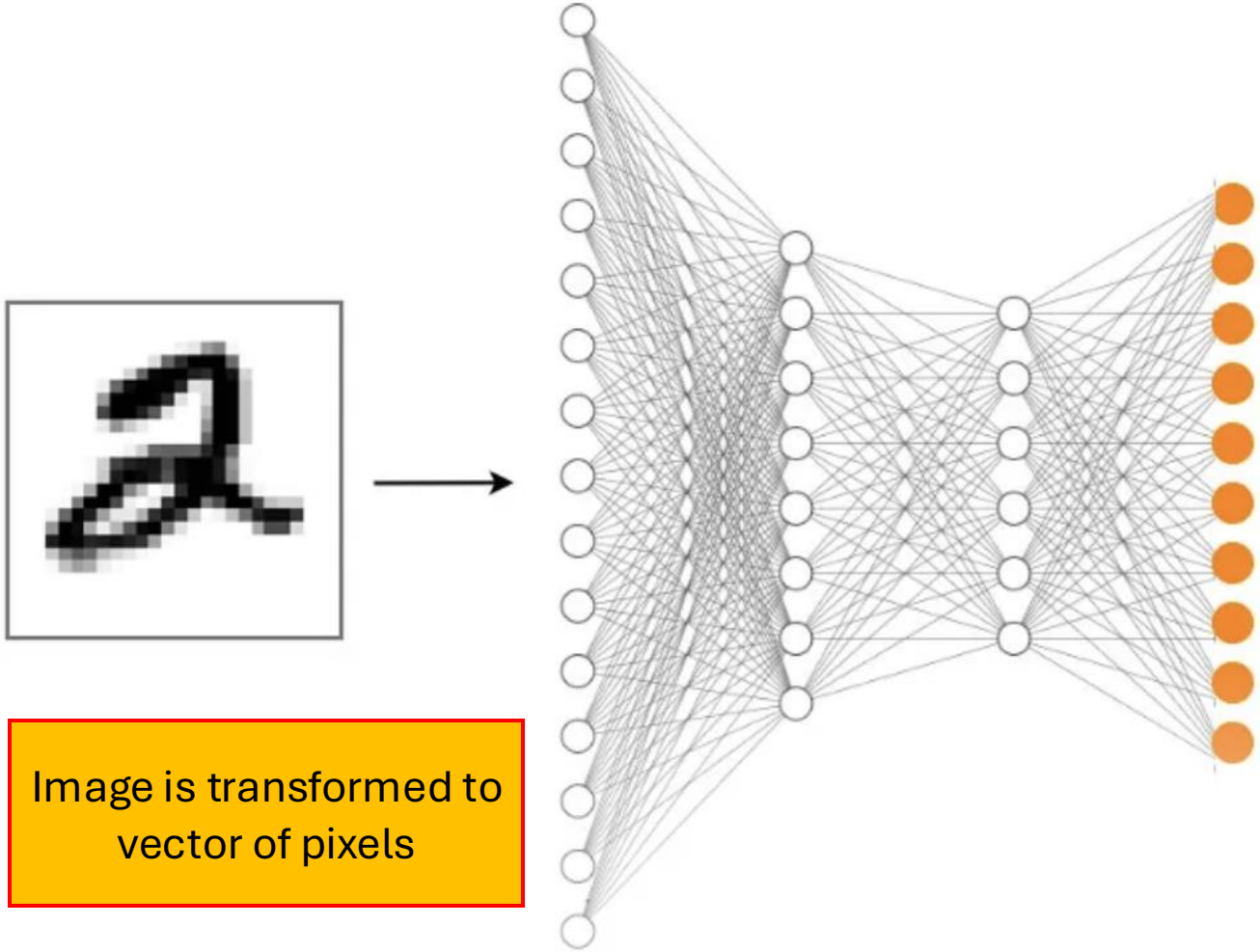


Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

What would happen if we permuted the ordering of the pixels?

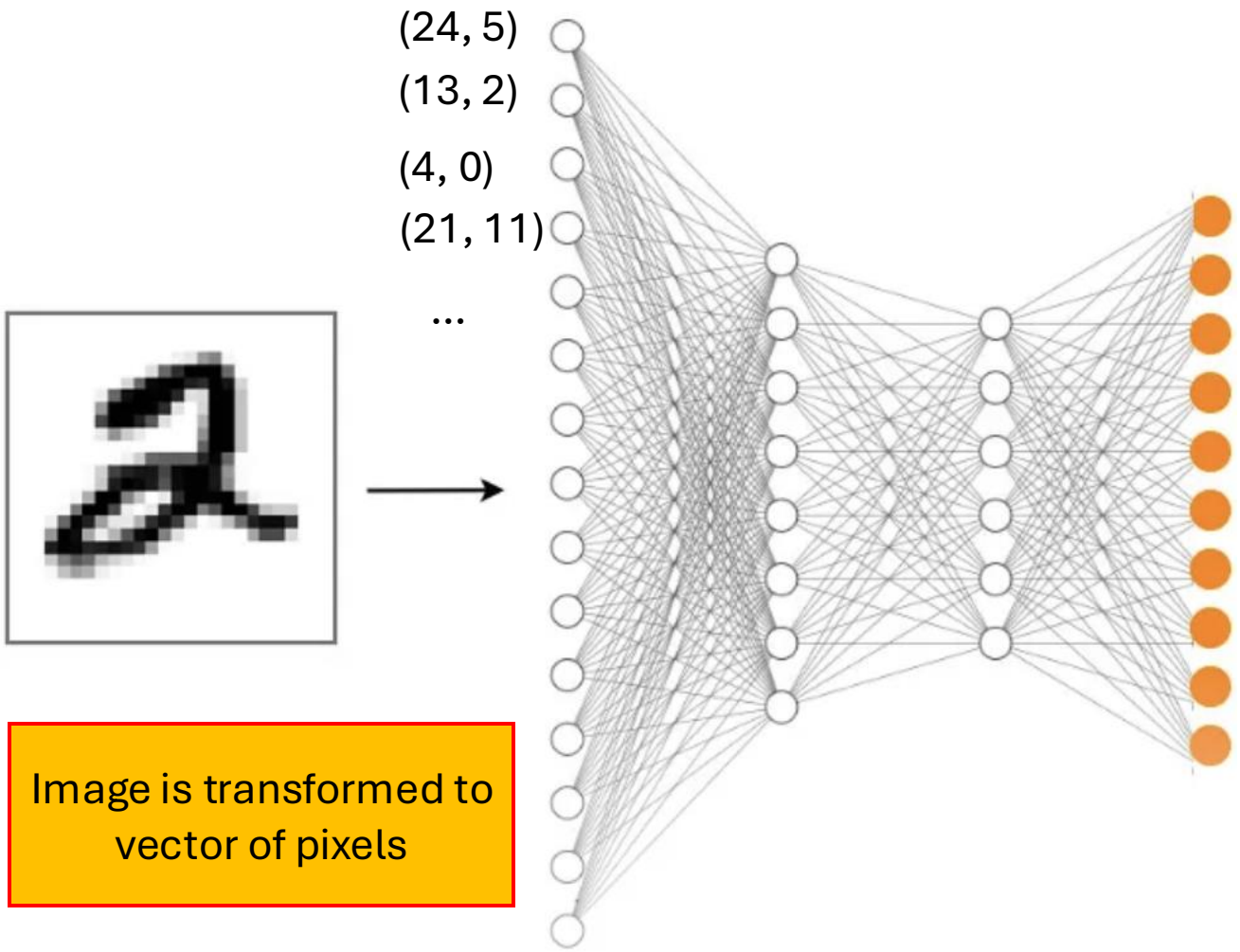
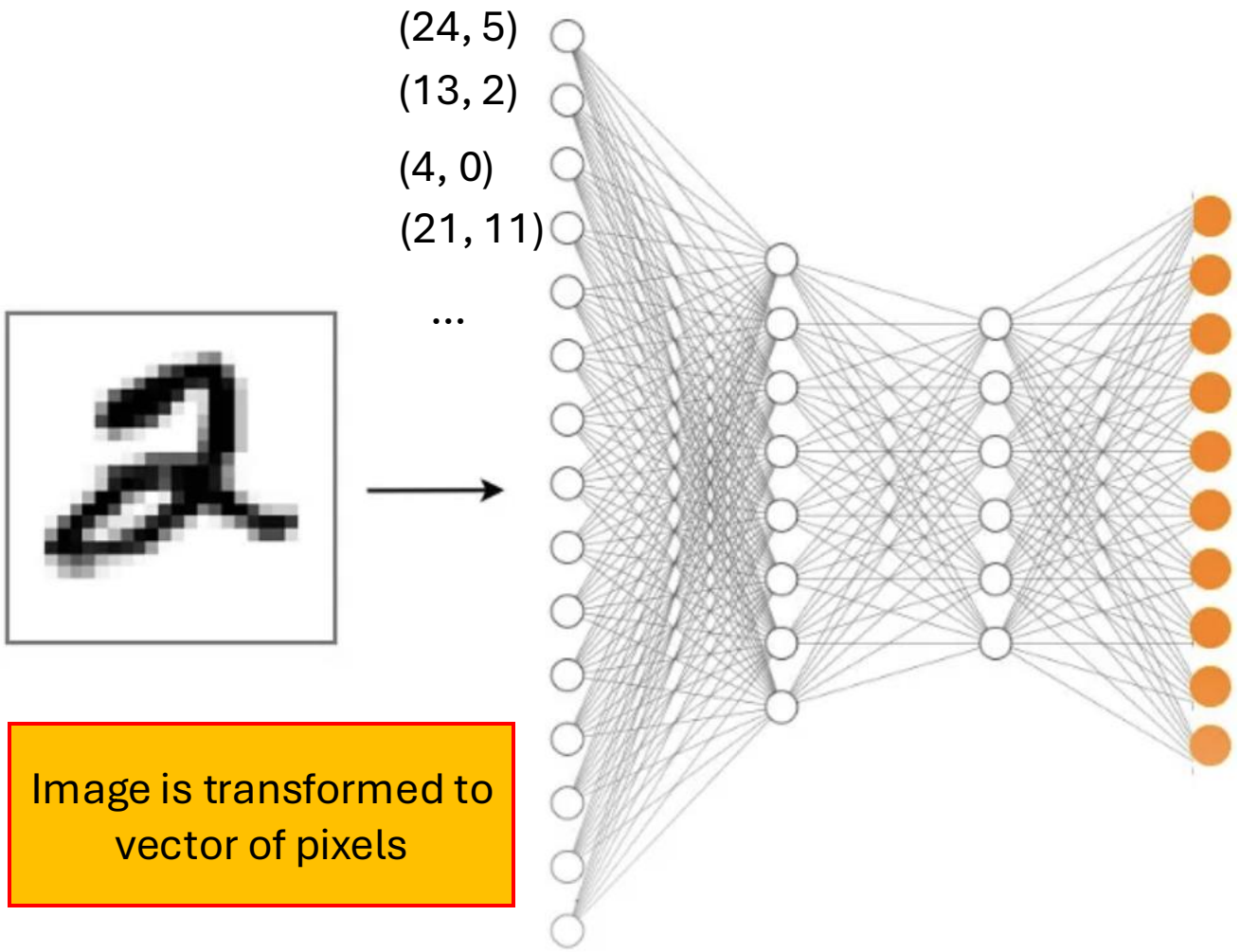


Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning



What would happen if we permuted the ordering of the pixels?

Will the training of the neural network differ?

Image is transformed to vector of pixels

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

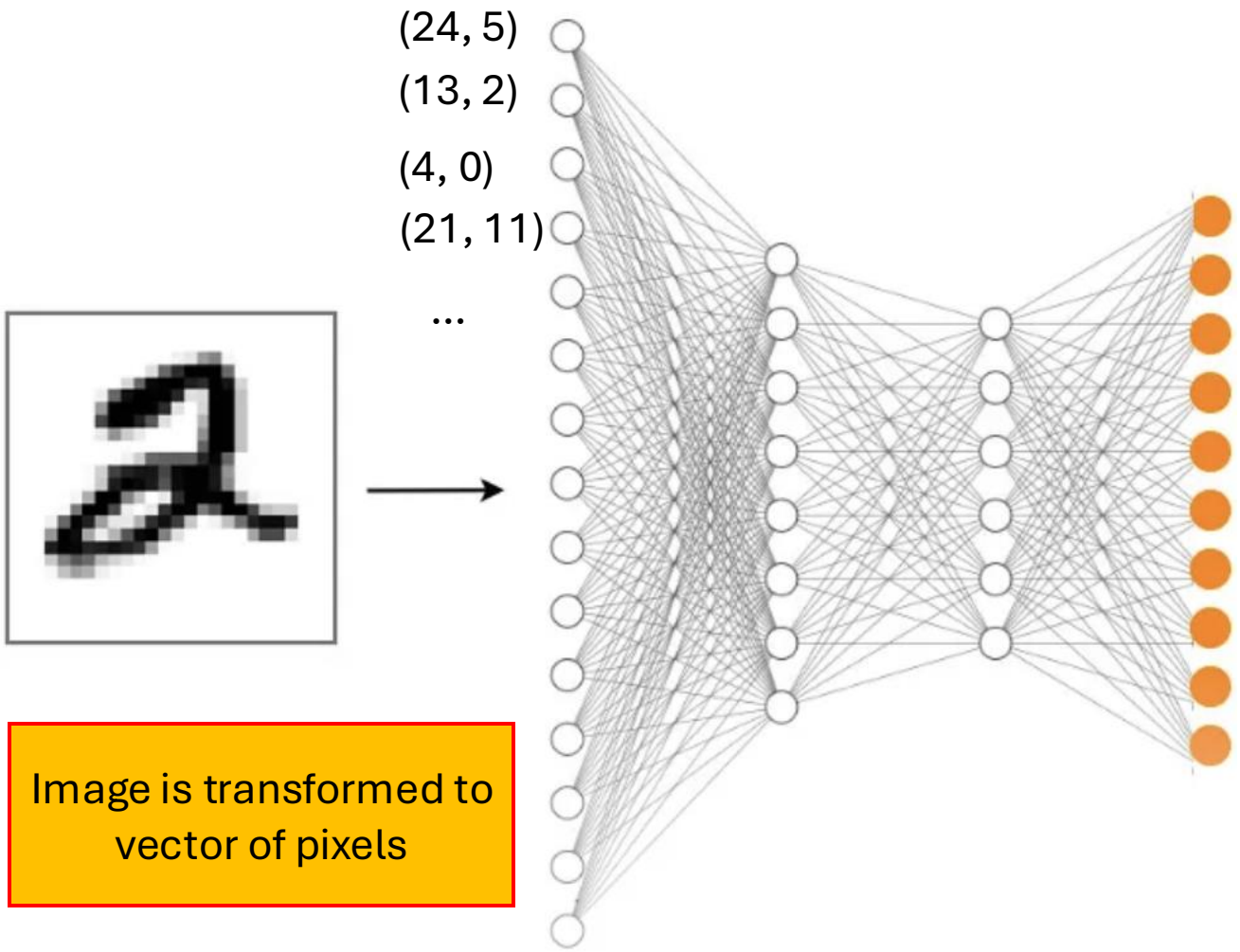


Image is transformed to vector of pixels

What would happen if we permuted the ordering of the pixels?

Will the training of the neural network differ?

No! MLPs do not use spatial information, it does not matter which order the pixels are fed in so long as it is the same ordering for every input

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

MLPs and Spatial Reasoning

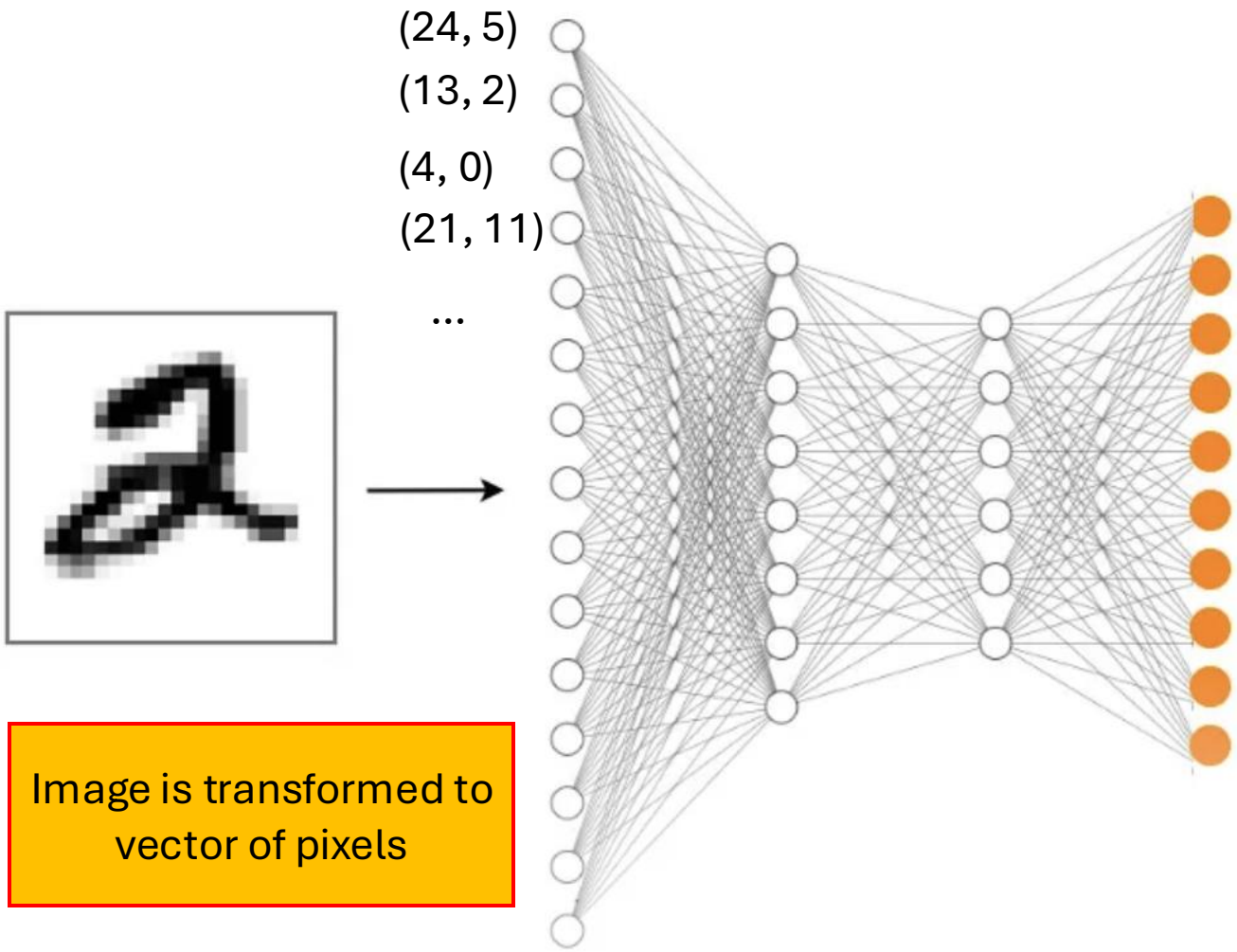


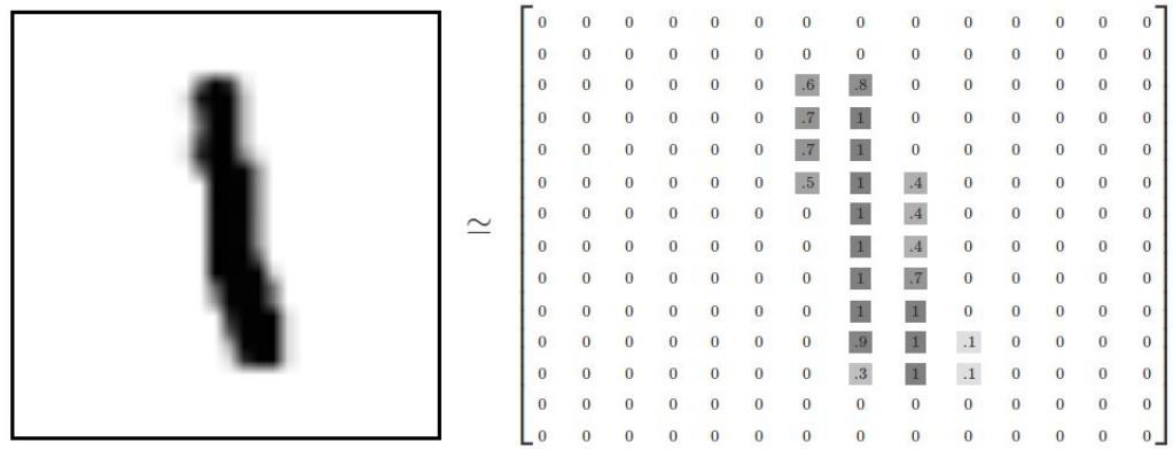
Image is transformed to vector of pixels

Isn't this actually a hard problem that we are trying to learn?

https://medium.com/towards-data-science/creating-a-multilayer-perceptron-mlp-classifier-model-to-identify-handwritten-digits-9bac1b16fe10

Limitations of Full Connections for MNIST

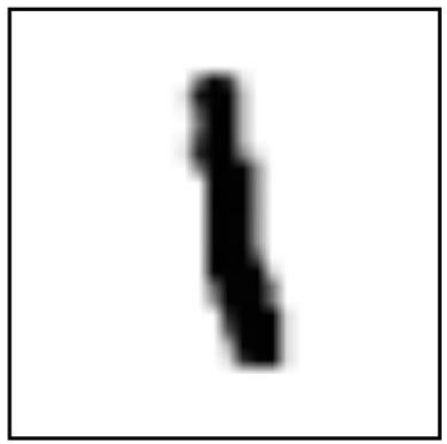
Suppose we've got a well-trained MNIST classifier...



#1 encoded as □

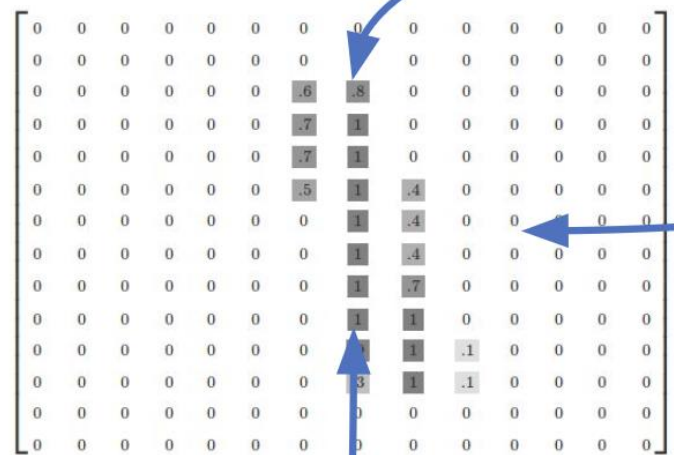
Limitations of Full Connections for MNIST

Suppose we've got a well-trained MNIST classifier...



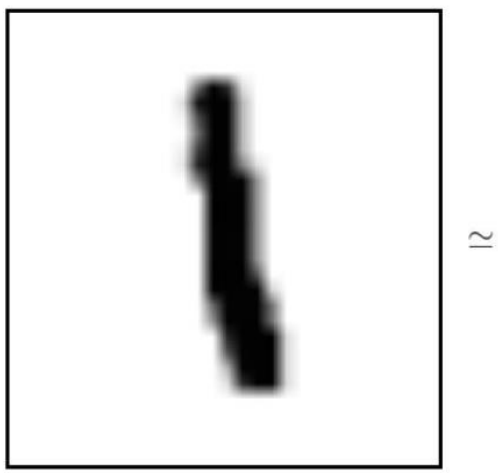
#1 encoded as □

≈

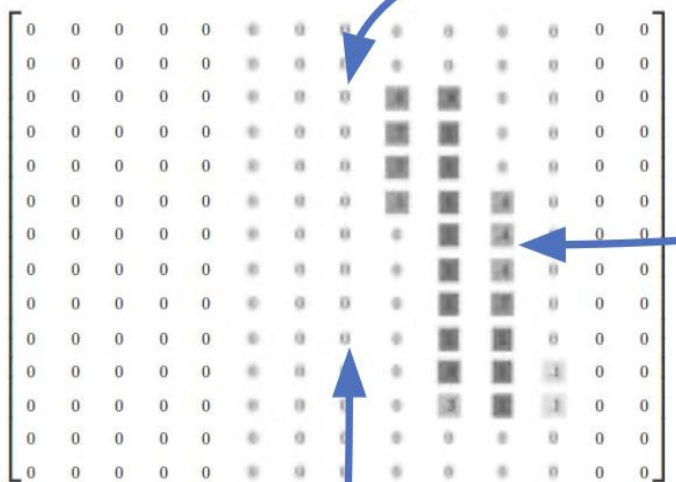


Limitations of Full Connections for MNIST

If we shift the digit to the right, then a different set of weights becomes relevant → etwork might have trouble classifying this as a 1...



#1 encoded as



this pixel gets weight 0.6

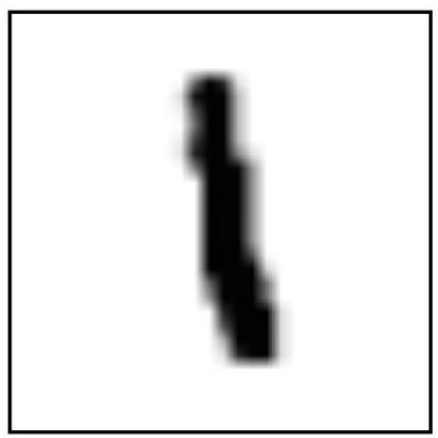
this pixel gets weight 0.1

this pixel gets weight 0.9

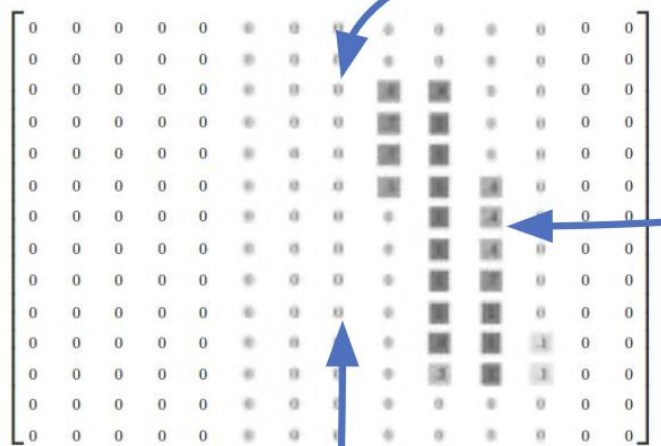
Can you tell this is a 1?

This would ***not*** be a problem for the human visual system

Our eyes don't look at absolute intensity values...



#1 encoded as



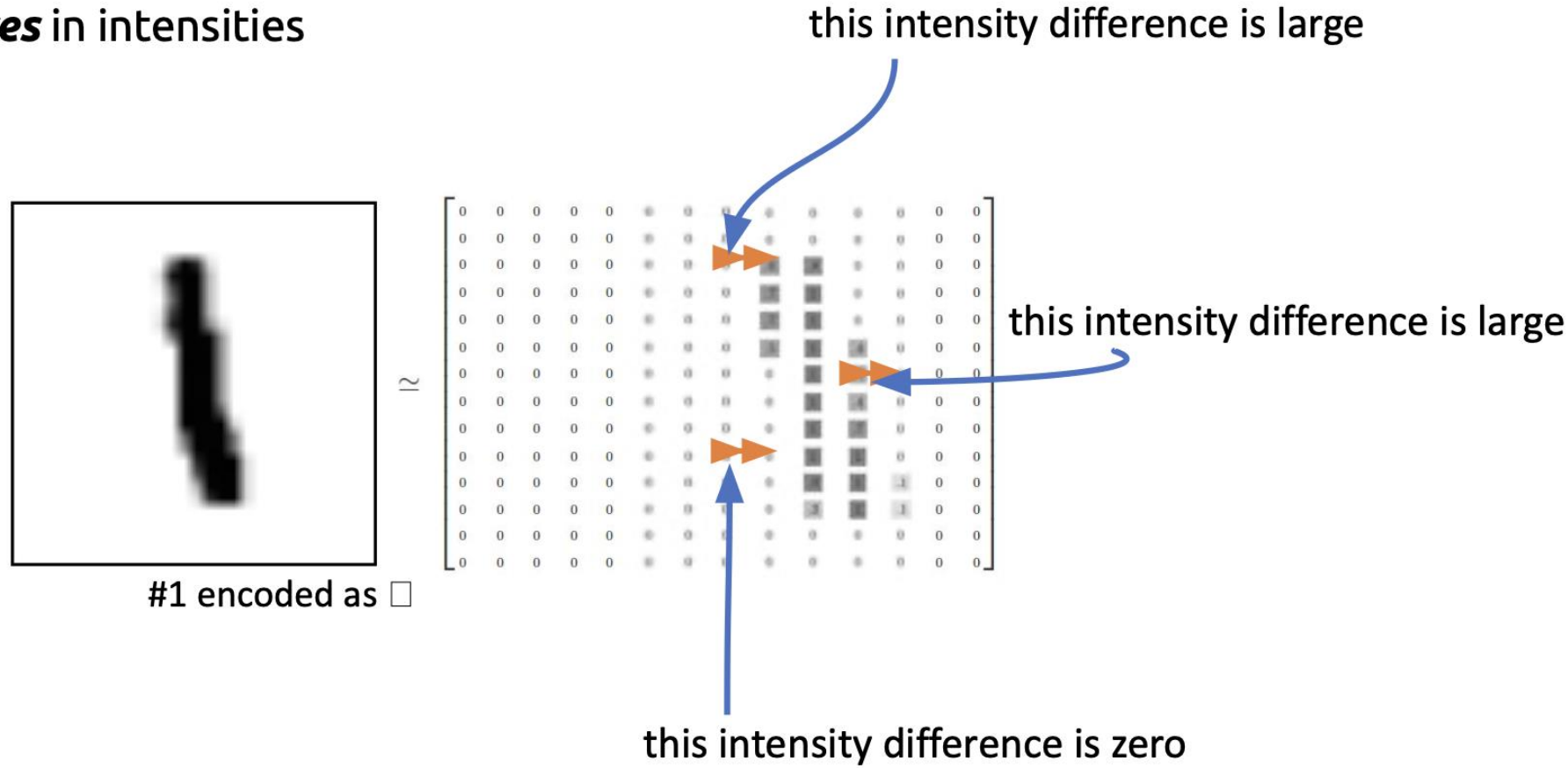
this pixel has a low intensity

this pixel has a high intensity

this pixel has a low intensity

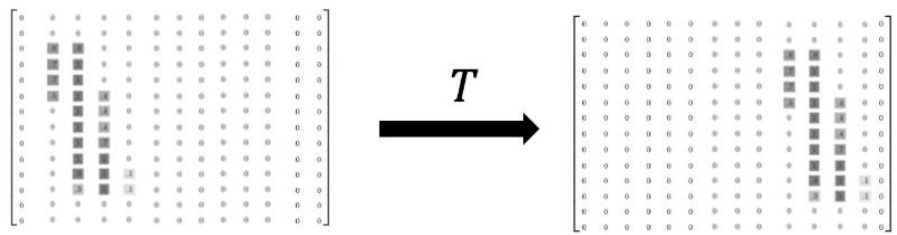
This would ***not*** be a problem for the human visual system

...but rather ***local differences*** in intensities



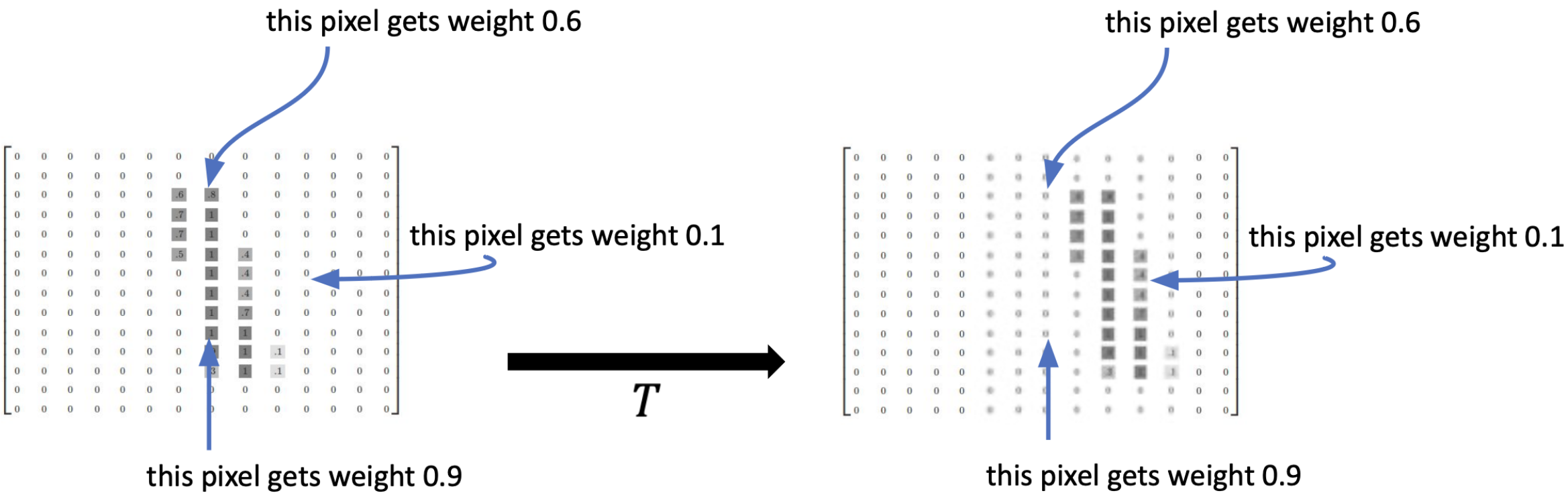
Translational Invariance

- To make a neural net f robust in this same way, it should ideally satisfy ***translational invariance***: $f\big(T(x)\big) = f(x)$, where
  - x is the input image
  - T is a translation (i.e. a horizonal and/or vertical shift)



$$f\left(\quad\right) \overset{?}{=} f\left(\quad\right)$$

Fully Connected Nets are ***not*** Translationally Invariant

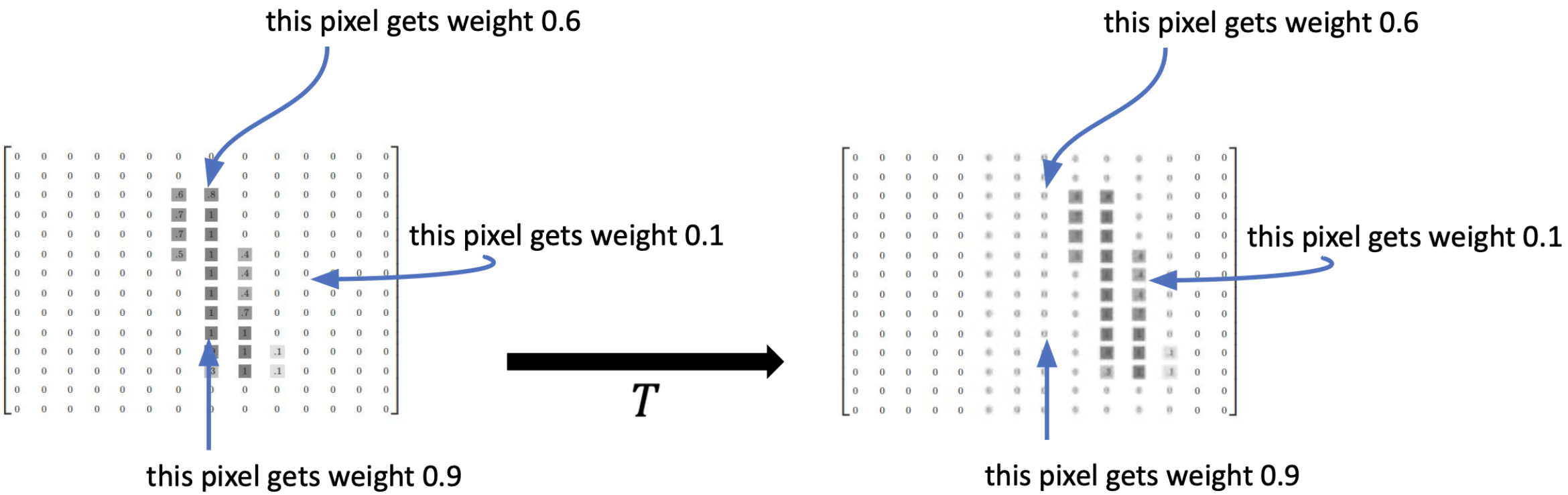


Sum of these three: $0.6 \cdot 0.8 + 0.1 \cdot 0 + 0.9 \cdot 1 = 1.38$

Sum of these three: $0.6 \cdot 0 + 0.1 \cdot 0.4 + 0.9 \cdot 0 = 0.4$

Fully Connected Nets are ***not*** Translationally Invariant

How to make the network translationally invariant?



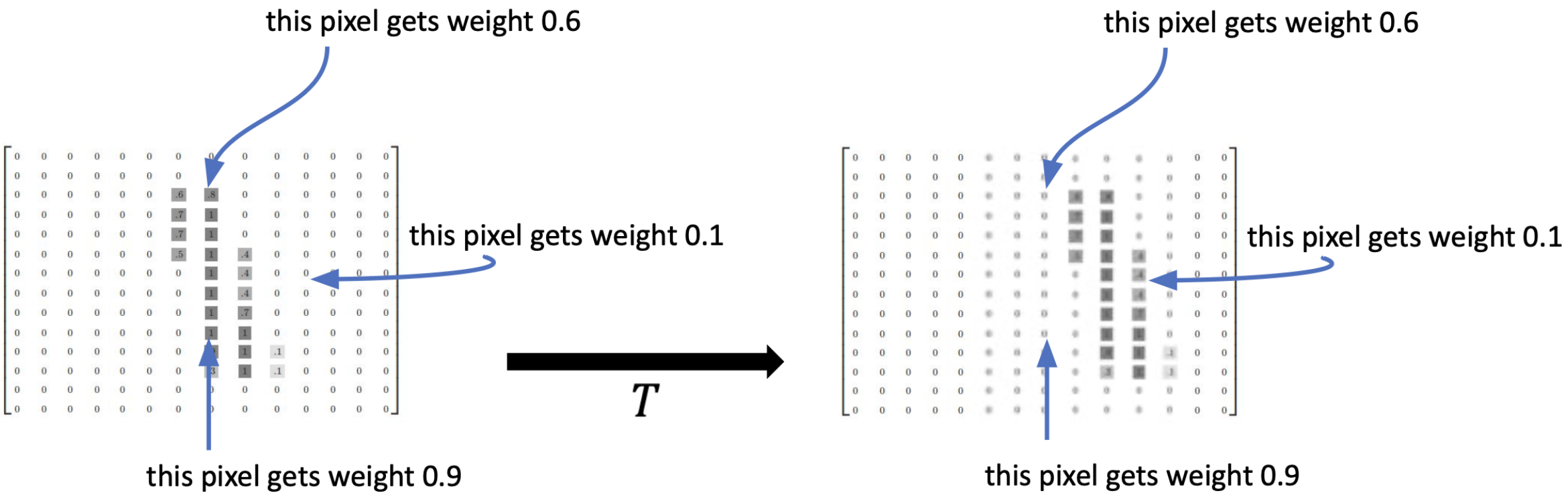
Sum of these three: $0.6 \cdot 0.8 + 0.1 \cdot 0 + 0.9 \cdot 1 = 1.38$

Sum of these three: $0.6 \cdot 0 + 0.1 \cdot 0.4 + 0.9 \cdot 0 = 0.4$

Fully Connected Nets are ***not*** Translationally Invariant

How to make the network translationally invariant?

Focus on local differences/patterns

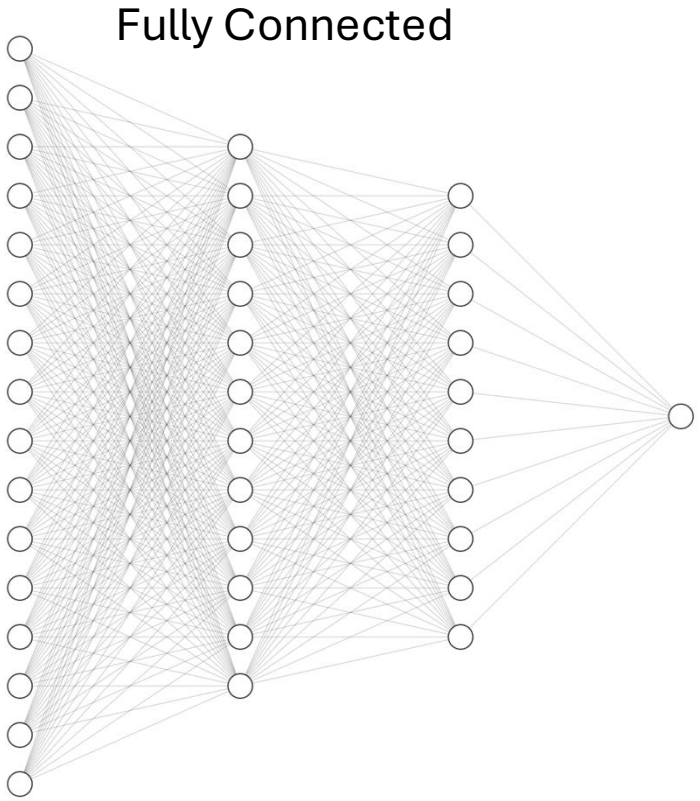


Sum of these three: $0.6 \cdot 0.8 + 0.1 \cdot 0 + 0.9 \cdot 1 = 1.38$

Sum of these three: $0.6 \cdot 0 + 0.1 \cdot 0.4 + 0.9 \cdot 0 = 0.4$

MLPs and Spatial Reasoning

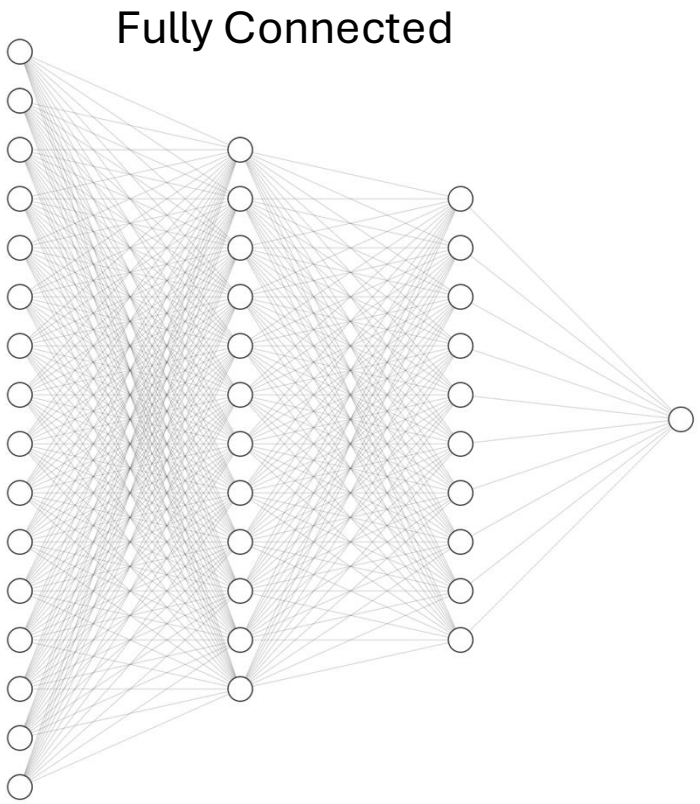
MLPs (also called fully-connected networks) have weights from every pixel to every neuron



MLPs and Spatial Reasoning

How can we change a fully-connected network to account for spatial information?

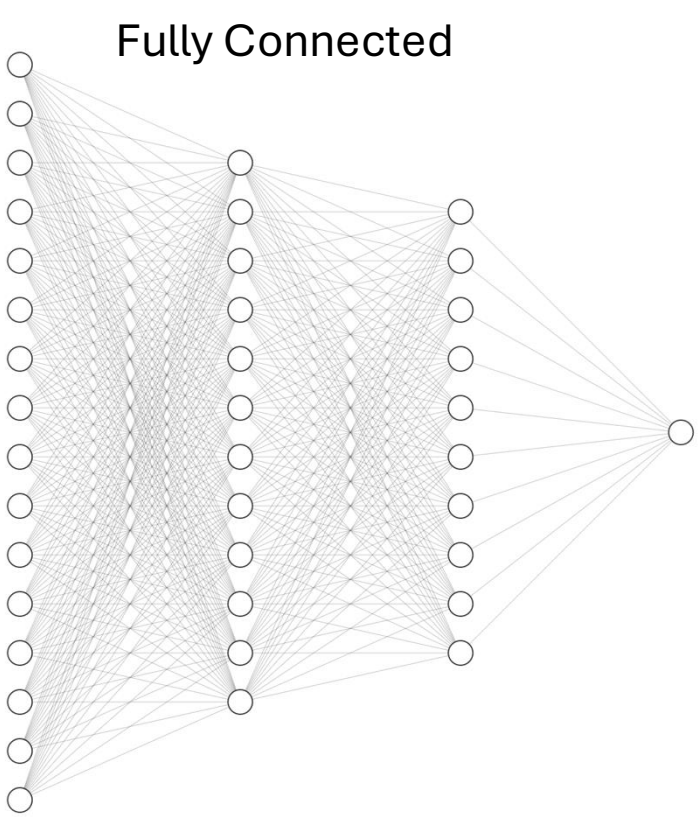
MLPs (also called fully-connected networks) have weights from every pixel to every neuron

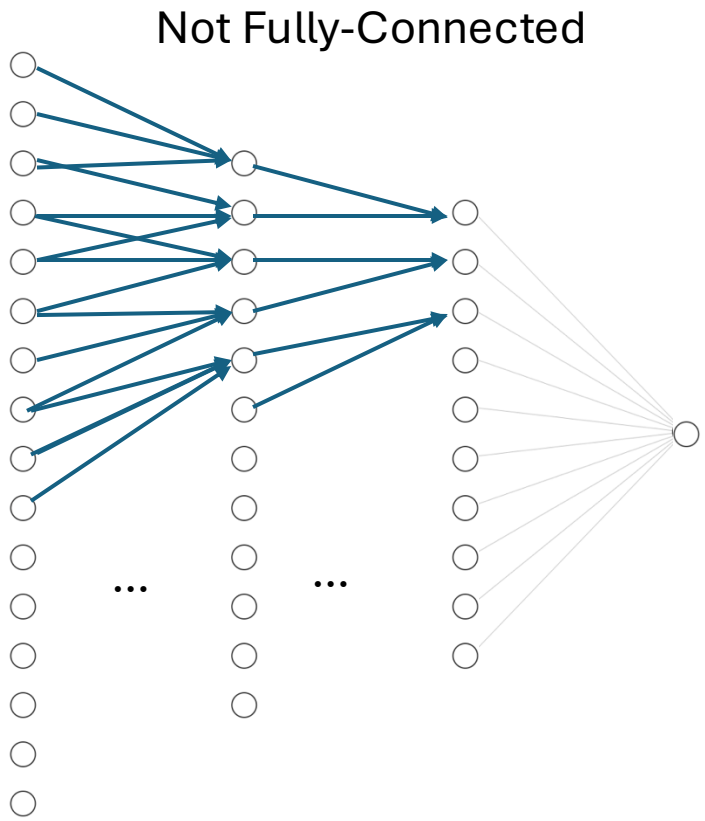


MLPs and Spatial Reasoning

How can we change a fully-connected network to account for spatial information?

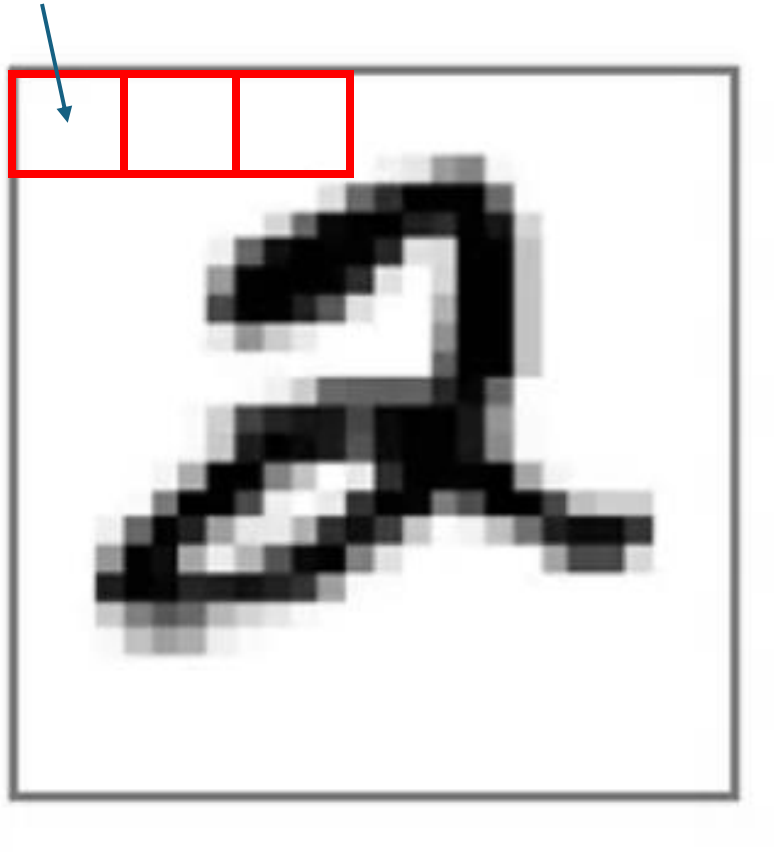
MLPs (also called fully-connected networks) have weights from every pixel to every neuron





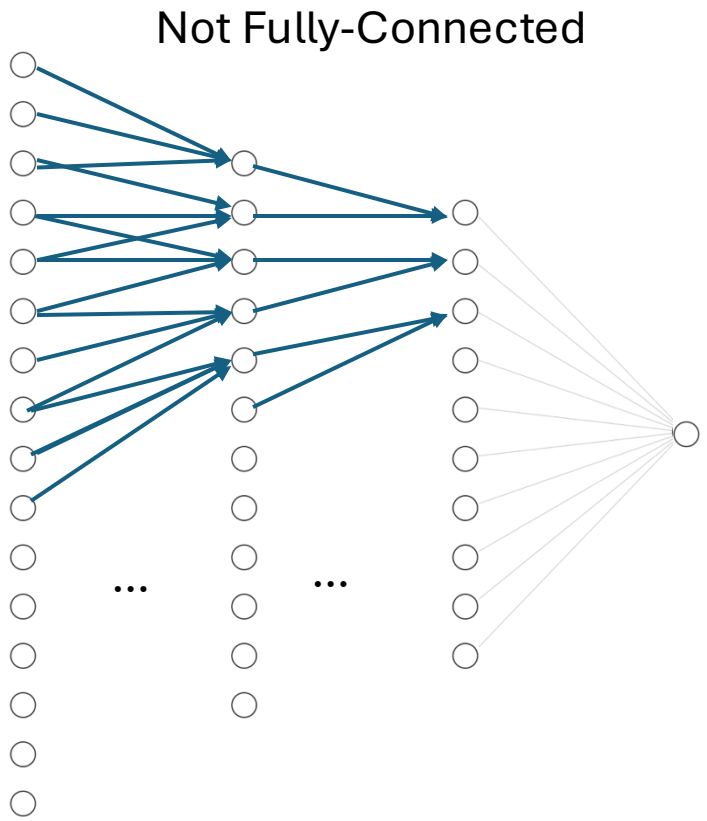
MLPs and Spatial Reasoning

Patches: Pixels close to each other



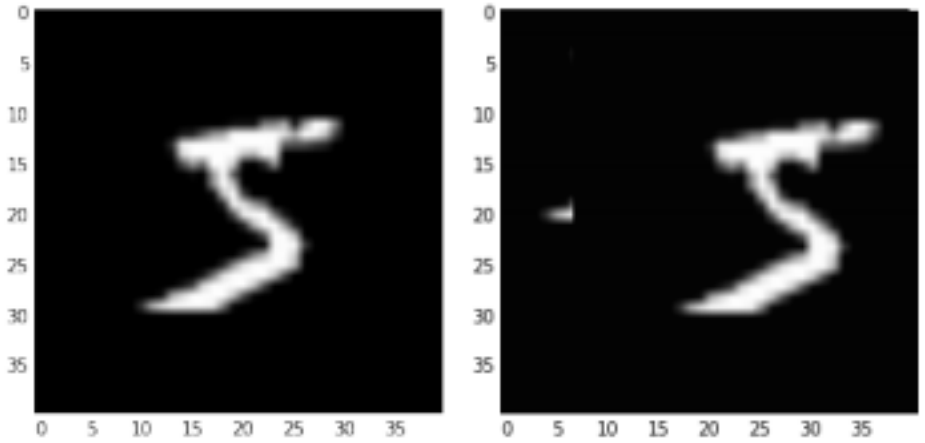
Advantages of Not Fully Connected Layers

- Fewer weights → Faster?
- The outputs of neurons are “features” for local “patches”
- Incorporates spatial information (pixels that are close together matter)



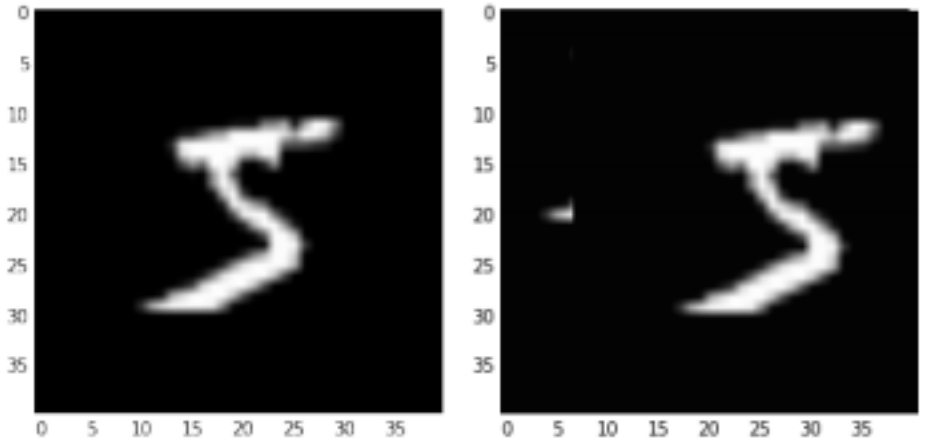
Disadvantages of Not Fully Connected Layers

- What happens if the image is Translated?
- The patches on the right side were never trained with 5’s in that side.



Disadvantages of Not Fully Connected Layers

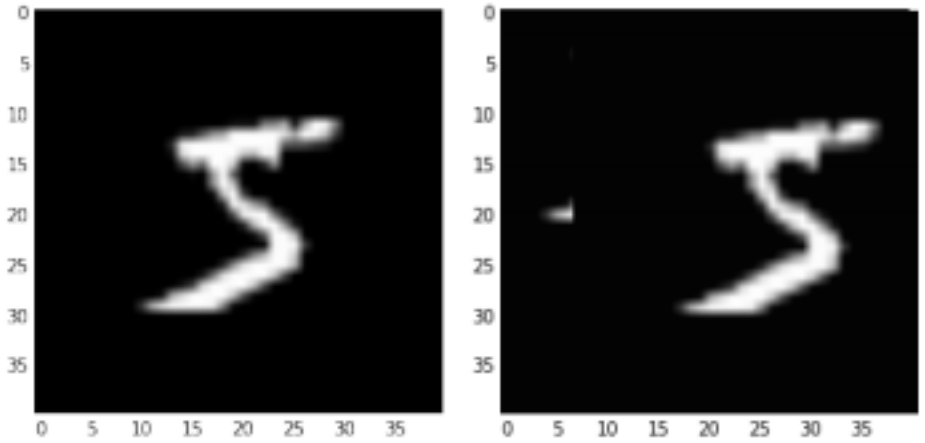
- What happens if the image is Translated?
- The patches on the right side were never trained with 5's in that side.



Even though we include spatial ***information***, we still don't have spatial ***reasoning***. *(Can't recognize a shifted 5 is still a 5)*

Disadvantages of Not Fully Connected Layers

- What happens if the image is Translated?
- The patches on the right side were never trained with 5's in that side.



Even though we include spatial ***information***, we still don't have spatial ***reasoning***. *(Can't recognize a shifted 5 is still a 5)*

What if we used the same weights for each patch? (Weight Sharing)

The Main Building Block: Convolution

Convolution is an operation that takes two inputs:

(1) An image (2D – B/W)

(2) A filter (also called a kernel)

1	1	1
0	0	0
-1	-1	-1

2D array of numbers; could be any values

What Convolution Does (Visually)

image

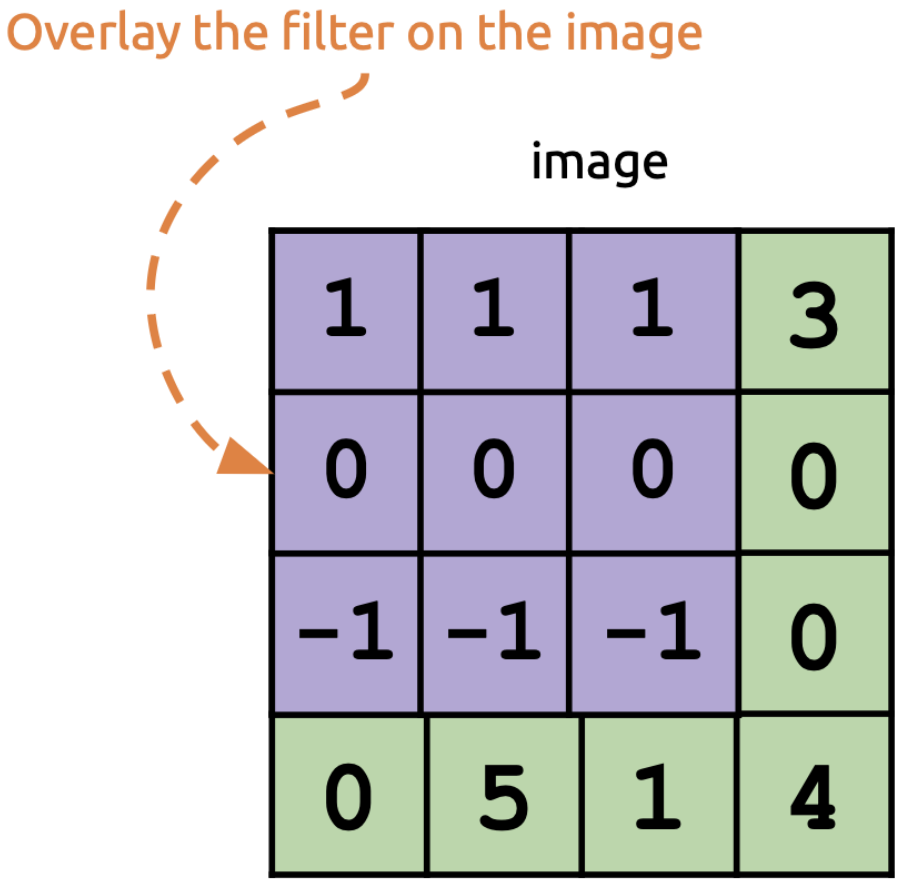
2	0	1	3
7	1	1	0
0	2	5	0
0	5	1	4

filter/kernel

1	1	1
0	0	0
-1	-1	-1

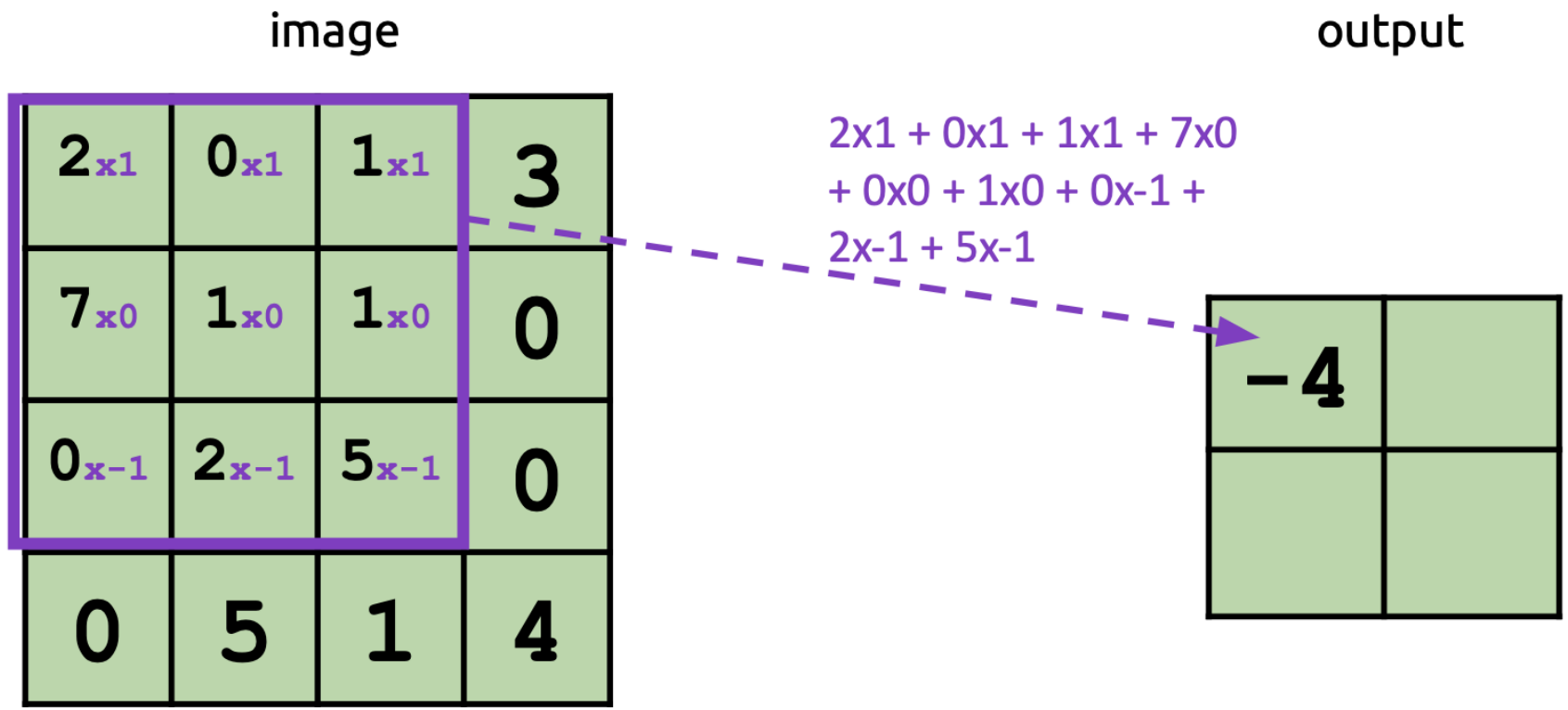
(We use this symbol for convolution)
(The verb form is “convolve”)

What Convolution Does (Visually)



What Convolution Does (Visually)

Sum up multiplied values to produce output value



What Convolution Does (Visually)

Move the filter over by one pixel

image

1	1	1	3
0	0	0	0
-1	-1	-1	0
0	5	1	4

output

-4	

What Convolution Does (Visually)

Move the filter over by one pixel

image

2	1	1	1
7	0	0	0
0	-1	-1	-1
0	5	1	4

output

-4	

What Convolution Does (Visually)

Repeat (multiply, sum up)

image

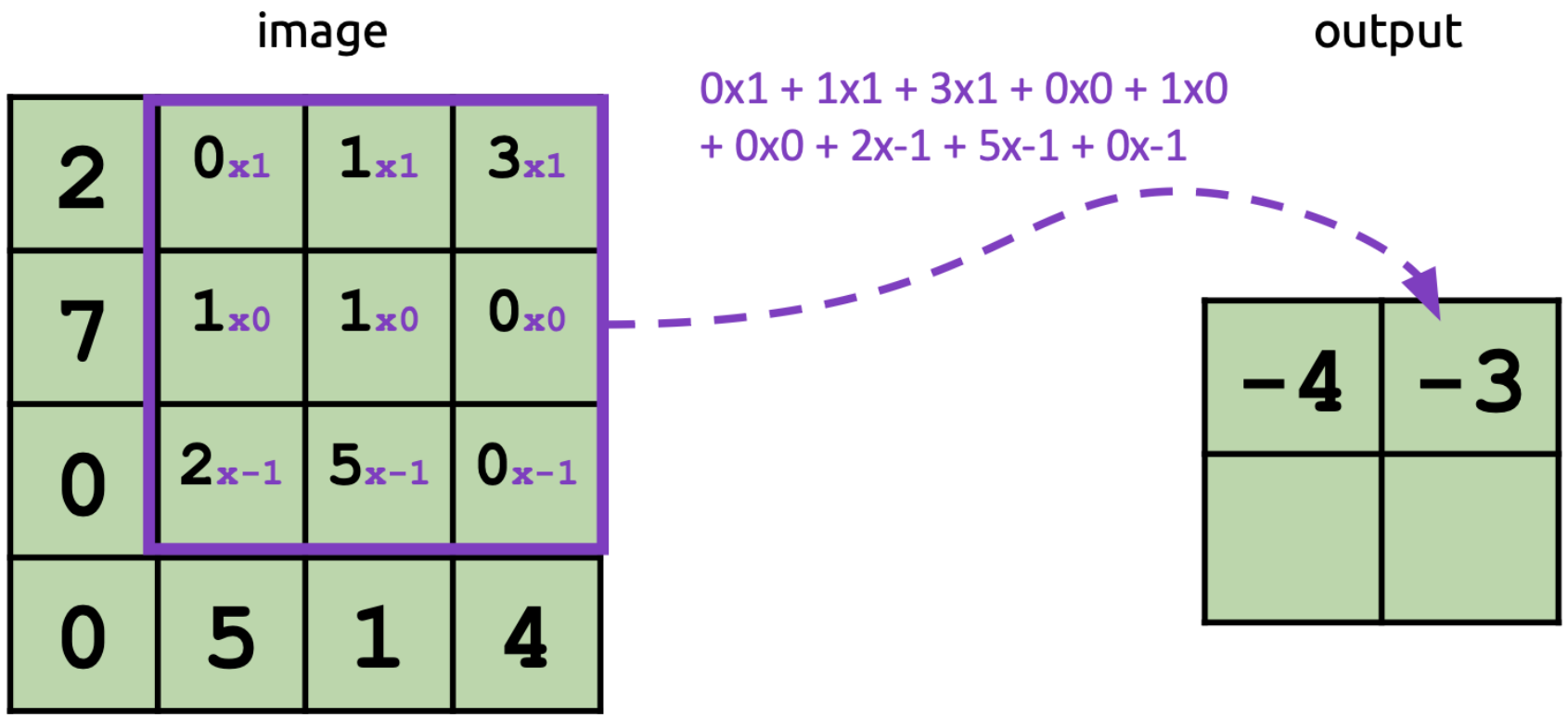
2	0_{x1}	1_{x1}	3_{x1}
7	1_{x0}	1_{x0}	0_{x0}
0	2_{x-1}	5_{x-1}	0_{x-1}
0	5	1	4

output

-4	

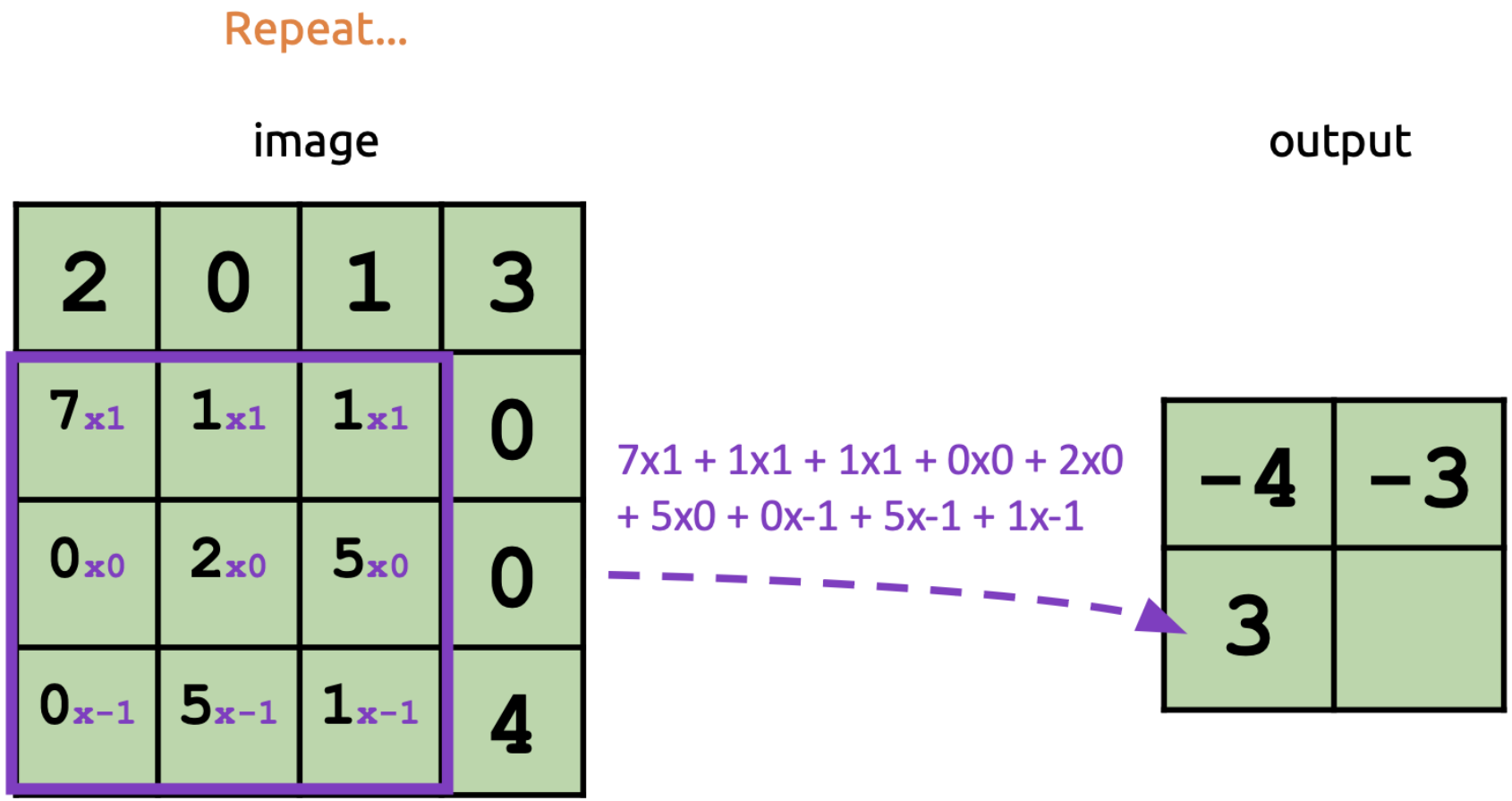
What Convolution Does (Visually)

Repeat (multiply, sum up)

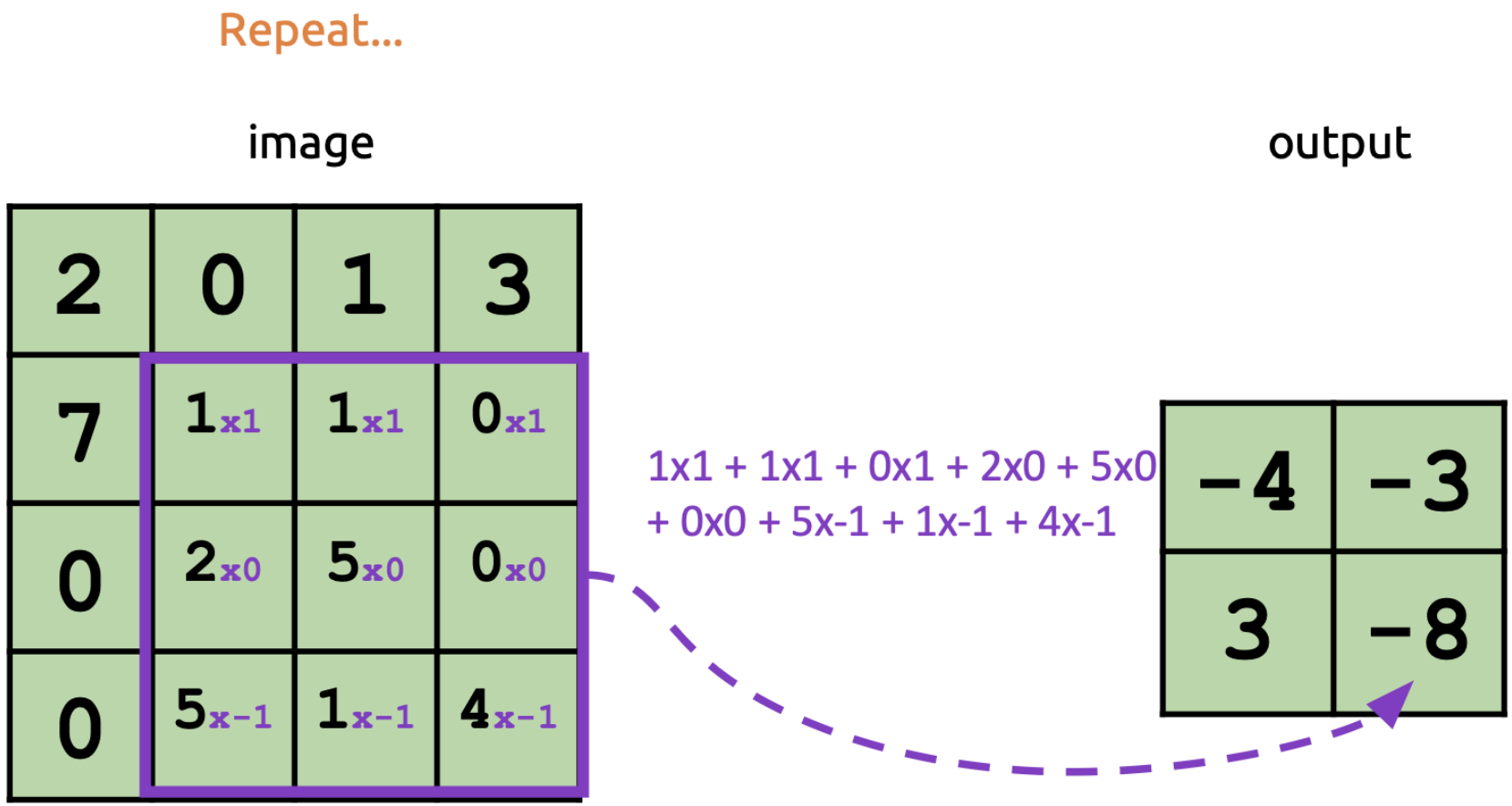


What Convolution Does (Visually)

Repeat...

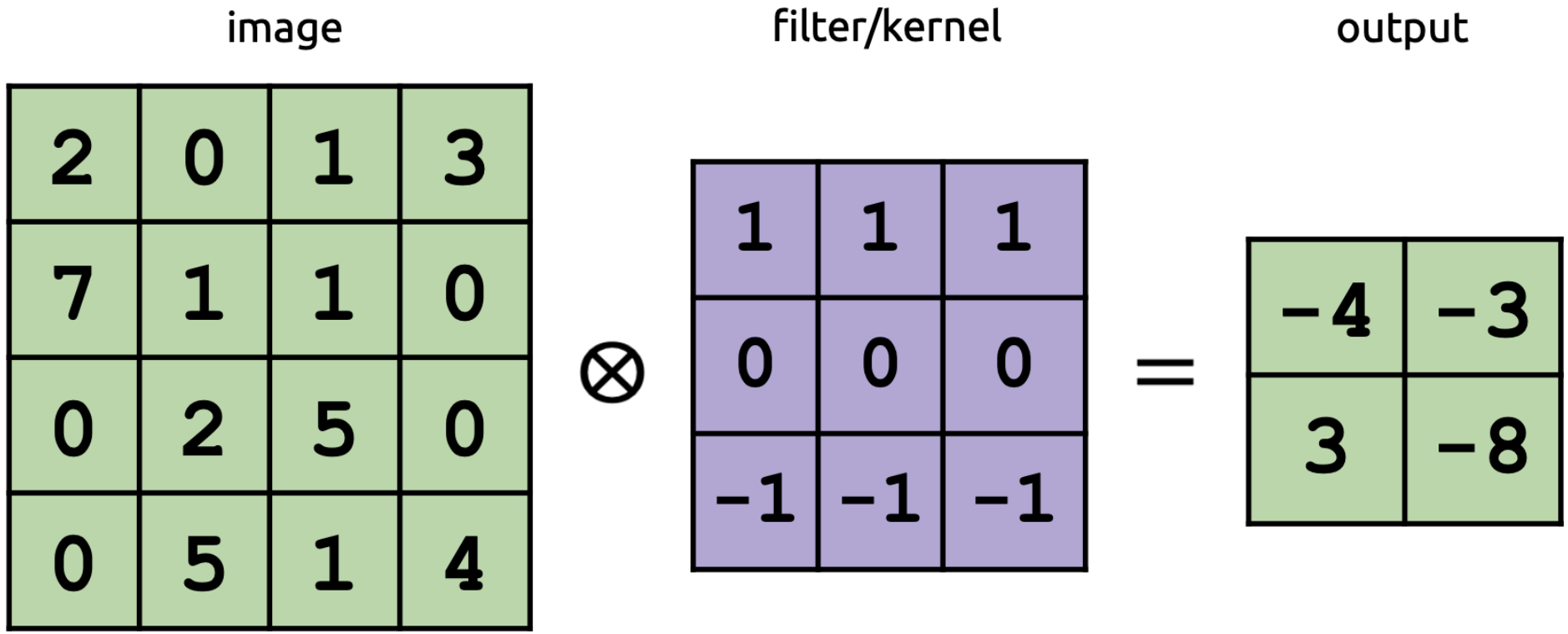


What Convolution Does (Visually)



What Convolution Does (Visually)

In summary:



Handmade Kernels and Filters

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Identity kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

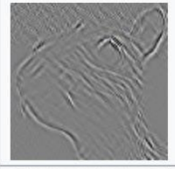
Sharpen kernel

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Box blur

$$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Gaussian blurr kernel

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Source: Wikipedia

What Comes Next?

Can we learn a filter for our images rather than “hand crafting” one?