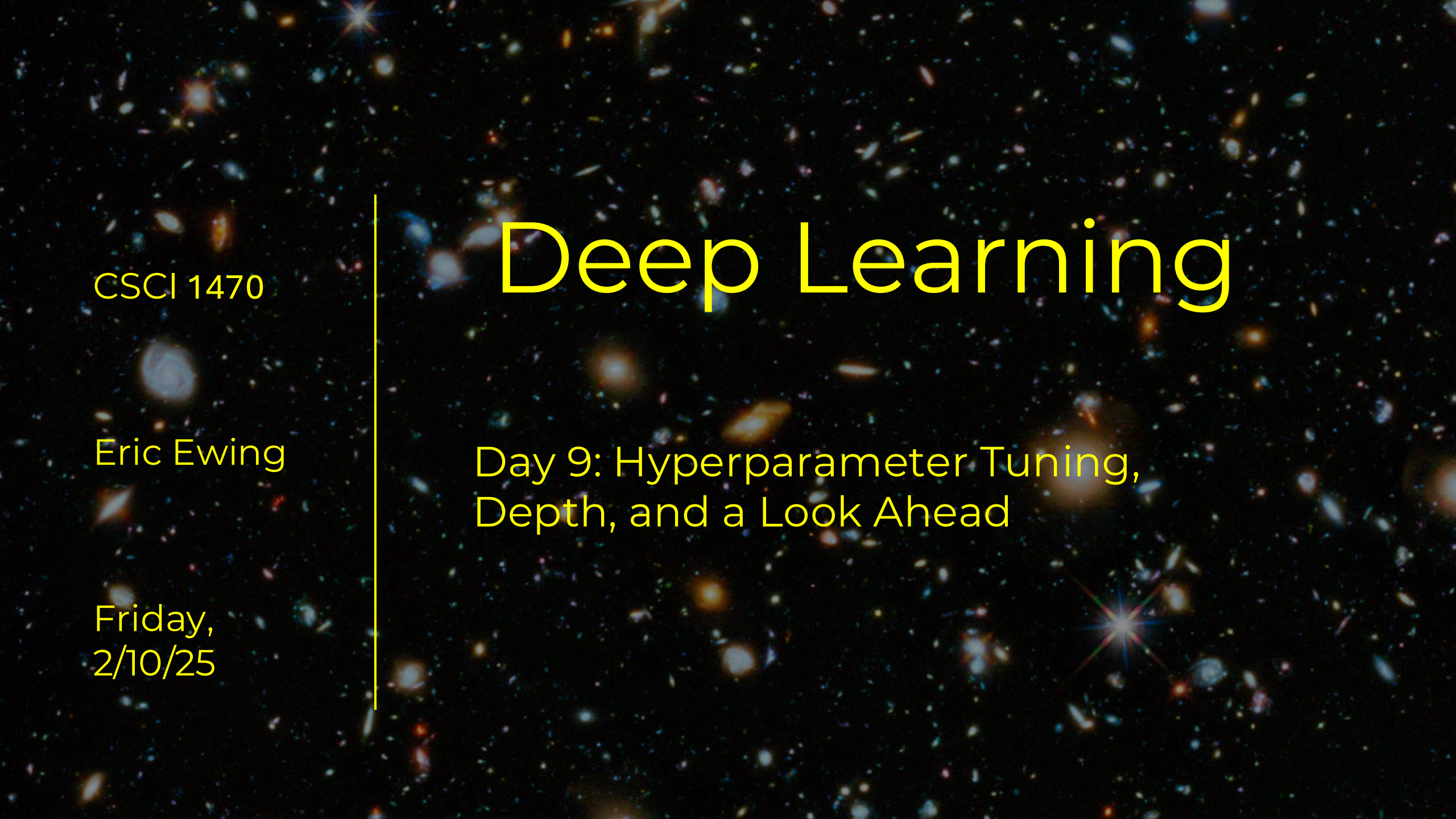


CSCI 1470

Eric Ewing

Friday,
2/10/25

Deep Learning

Day 9: Hyperparameter Tuning, Depth, and a Look Ahead

Goals

(1) Hyperparameter Tuning

    (i) A quick journey into topology

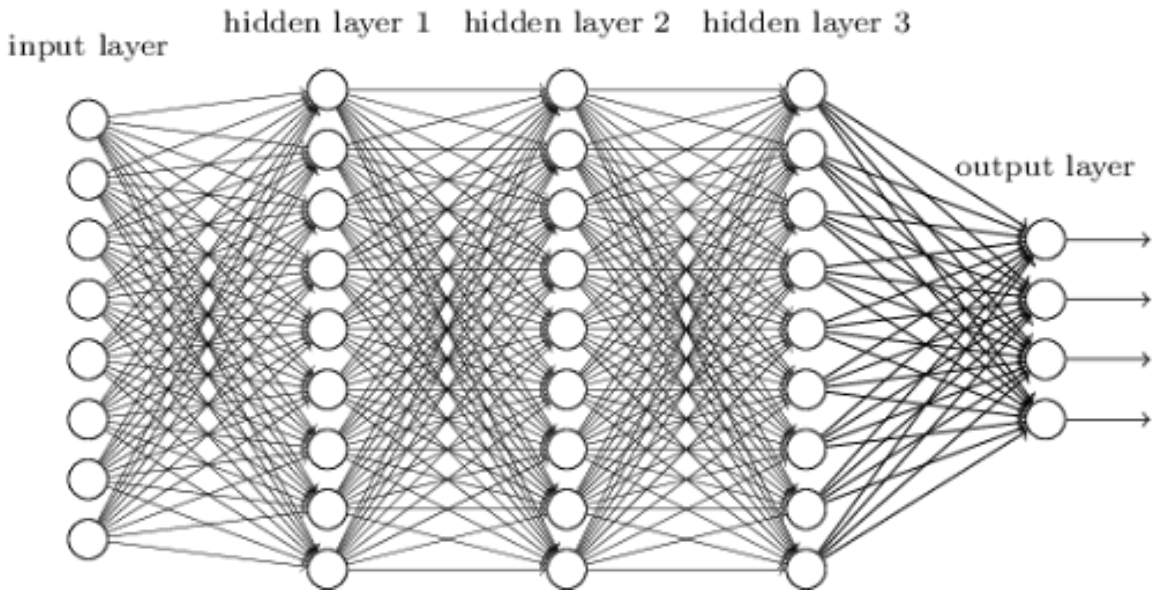
(2) AutoML

(3) Looking Ahead

    (i) What can we not solve with MLPs?

Recap

How many layers? How big should each layer be?



DL Frameworks (e.g., Tensorflow) use autograd computation graphs to calculate gradients with backprop

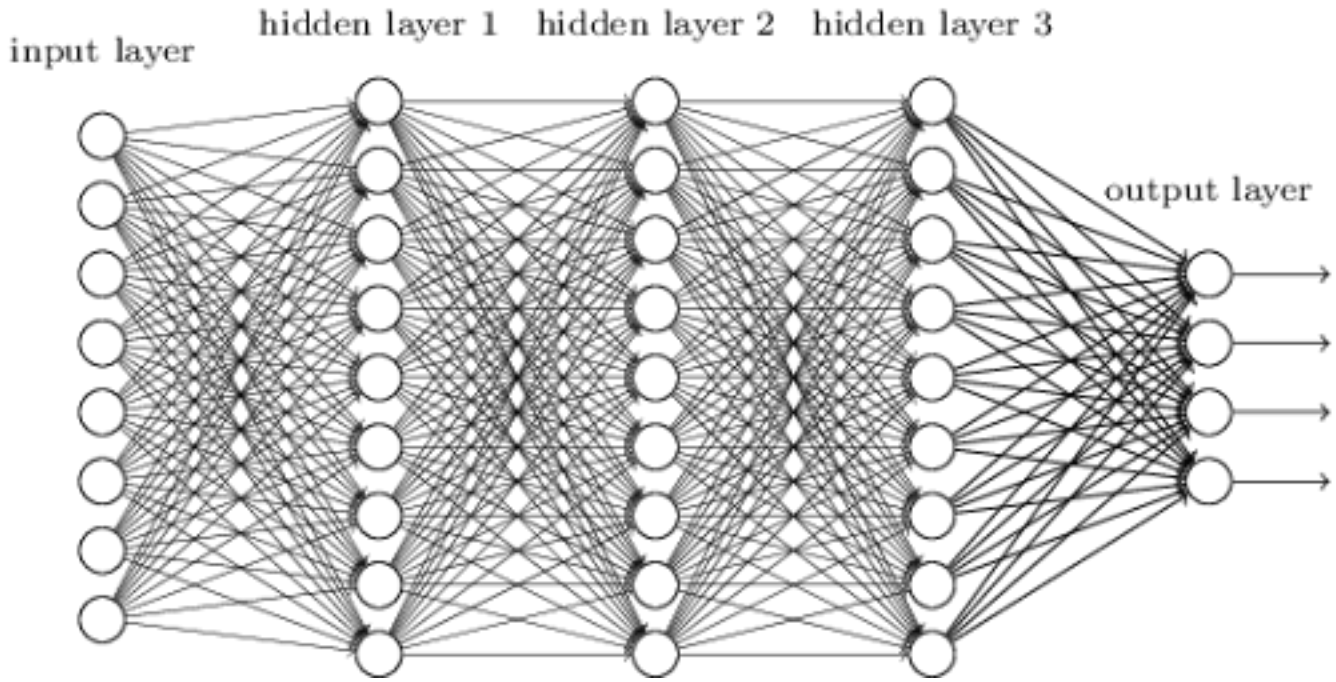
Hyperparameters are the parameters of learning that **we** have control of

Hidden Layers

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

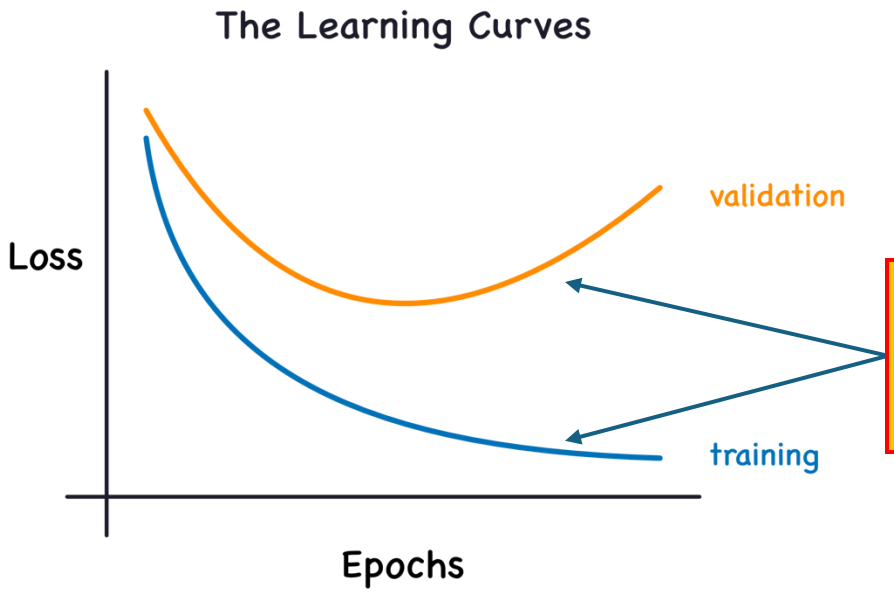
How complex is the problem you are trying to solve?

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)



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?

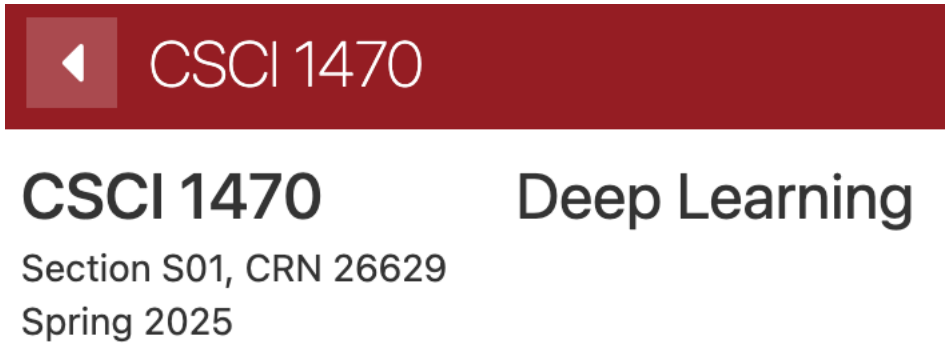
Your model has overfit, try **reducing** its size

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?



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."

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.

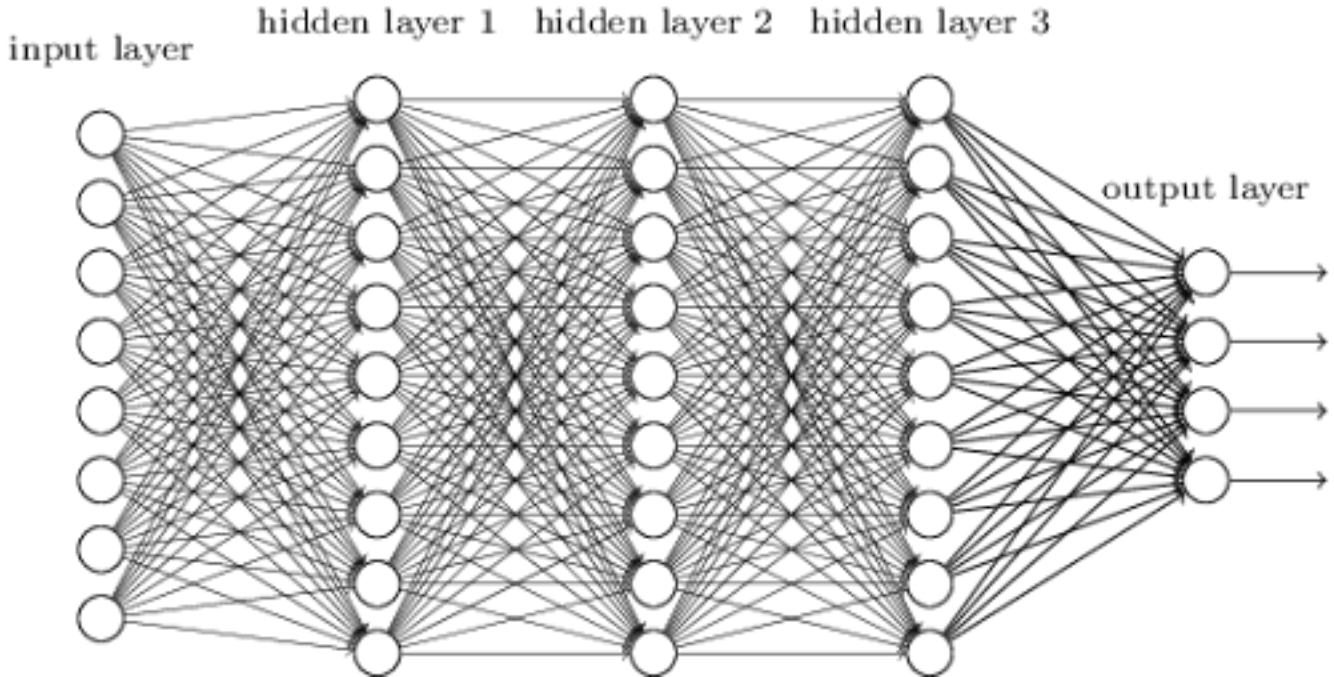
Recall that NNs are compositions of functions for which we are learning parameters:

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

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

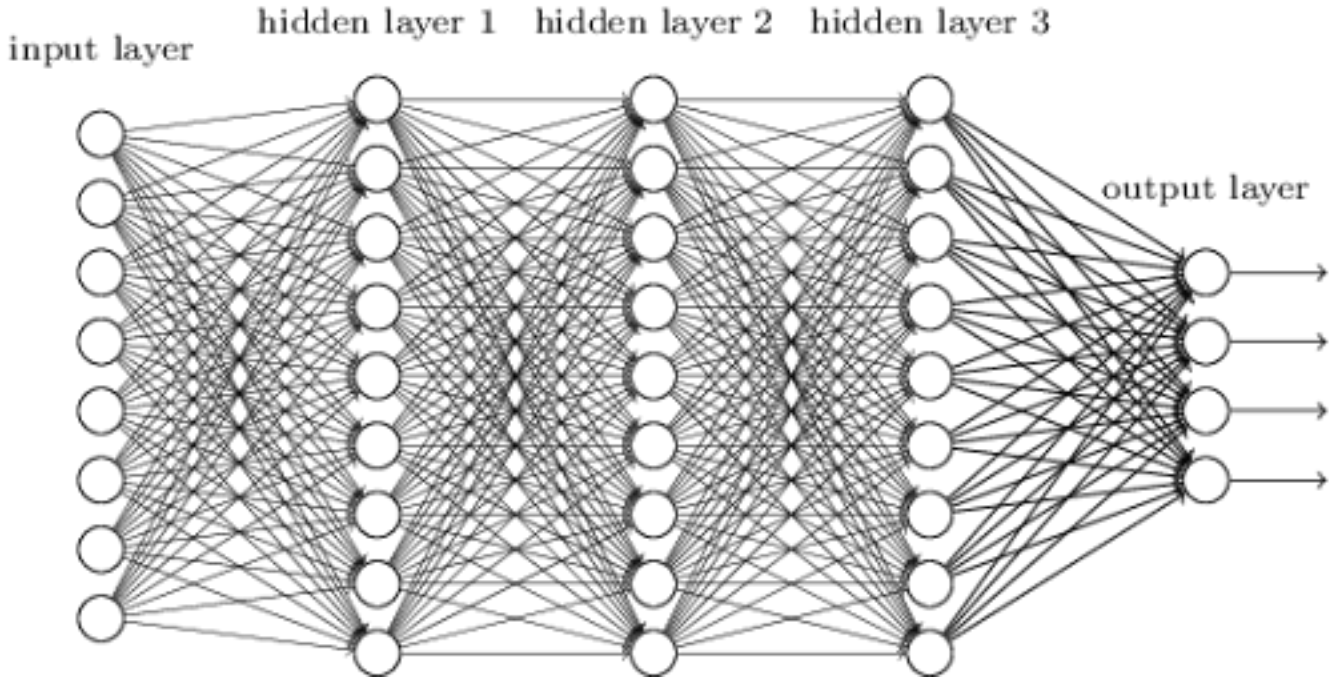
- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?

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

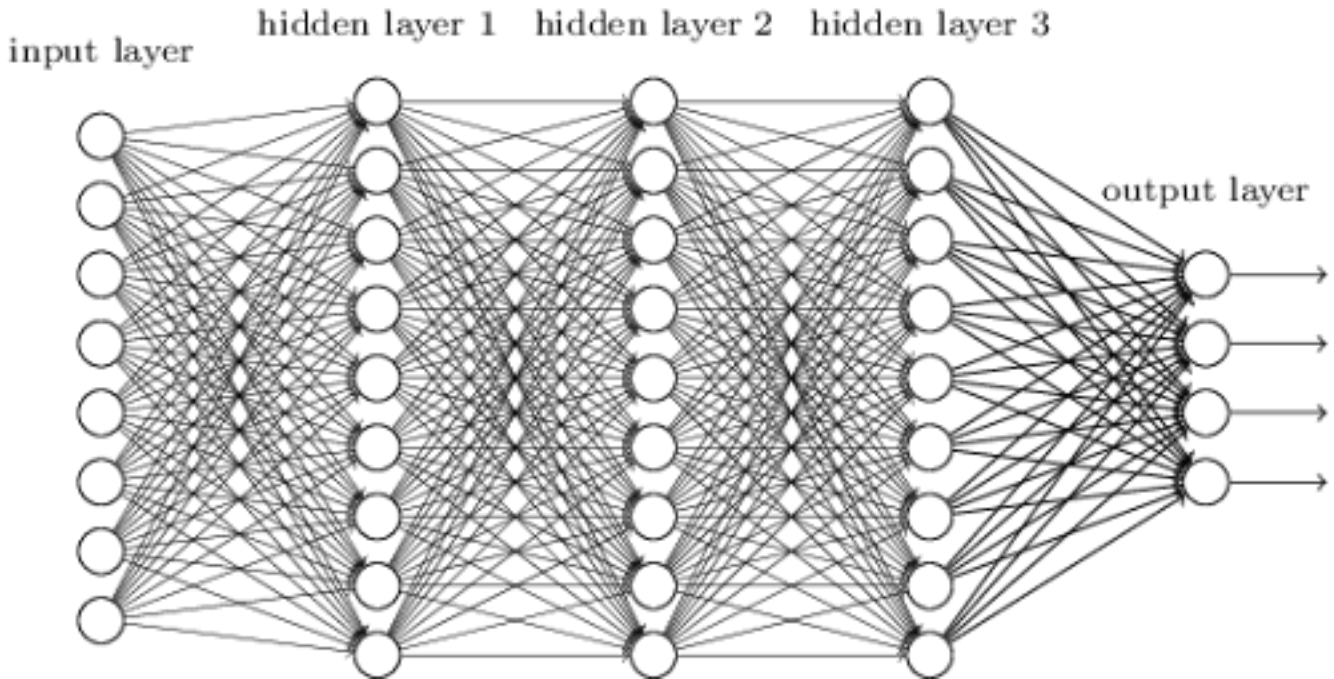


- If there are 10 inputs, 3 layers of 10 neurons, and 4 outputs, how many weights are there total?
- What if we double the width of each hidden layer?

$W_1 \in \mathbb{R}^{10\times20}$
$W_2 \in \mathbb{R}^{20\times20}$
$W_3 \in \mathbb{R}^{20\times20}$
$W_4 \in \mathbb{R}^{20\times4}$
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? of each hidden layer?



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

The Manifold Hypothesis

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

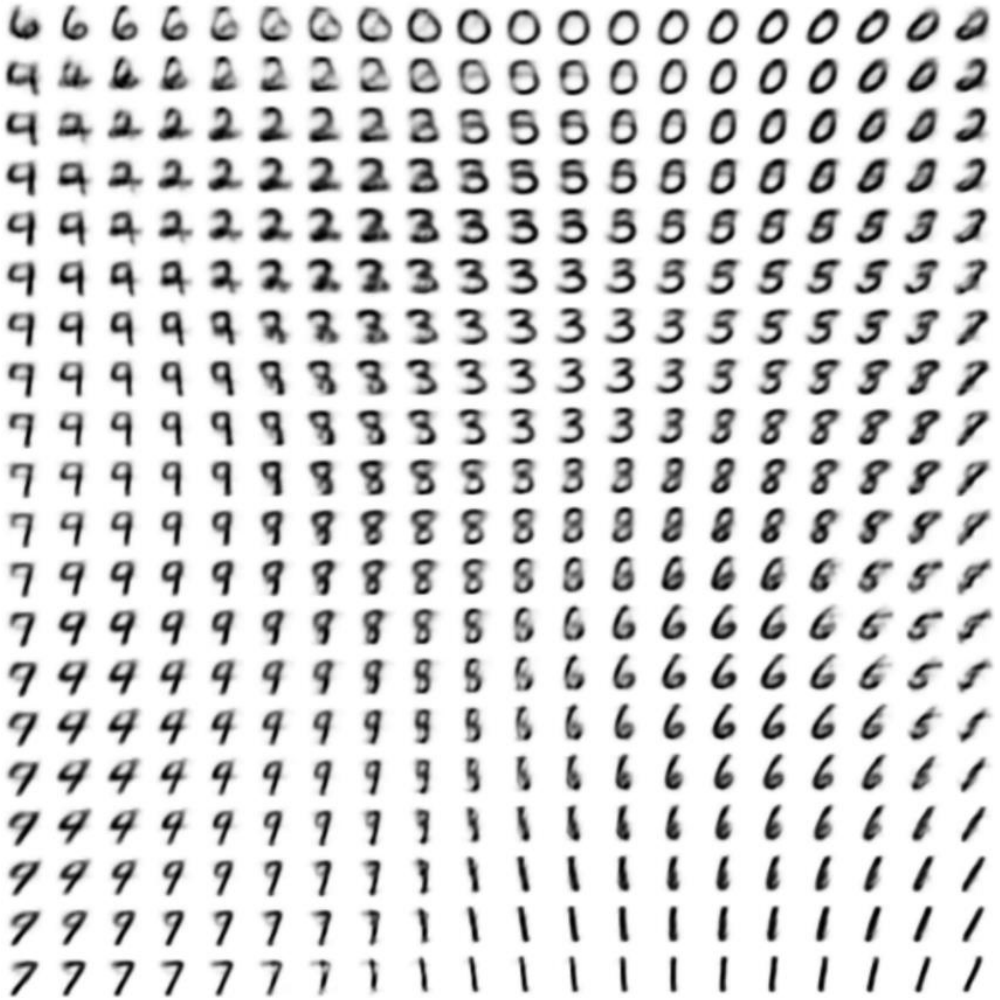
Hypothesis: real-world high-dimensional data lies on low-dimensional manifolds embedded within the high-dimensional space.

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?

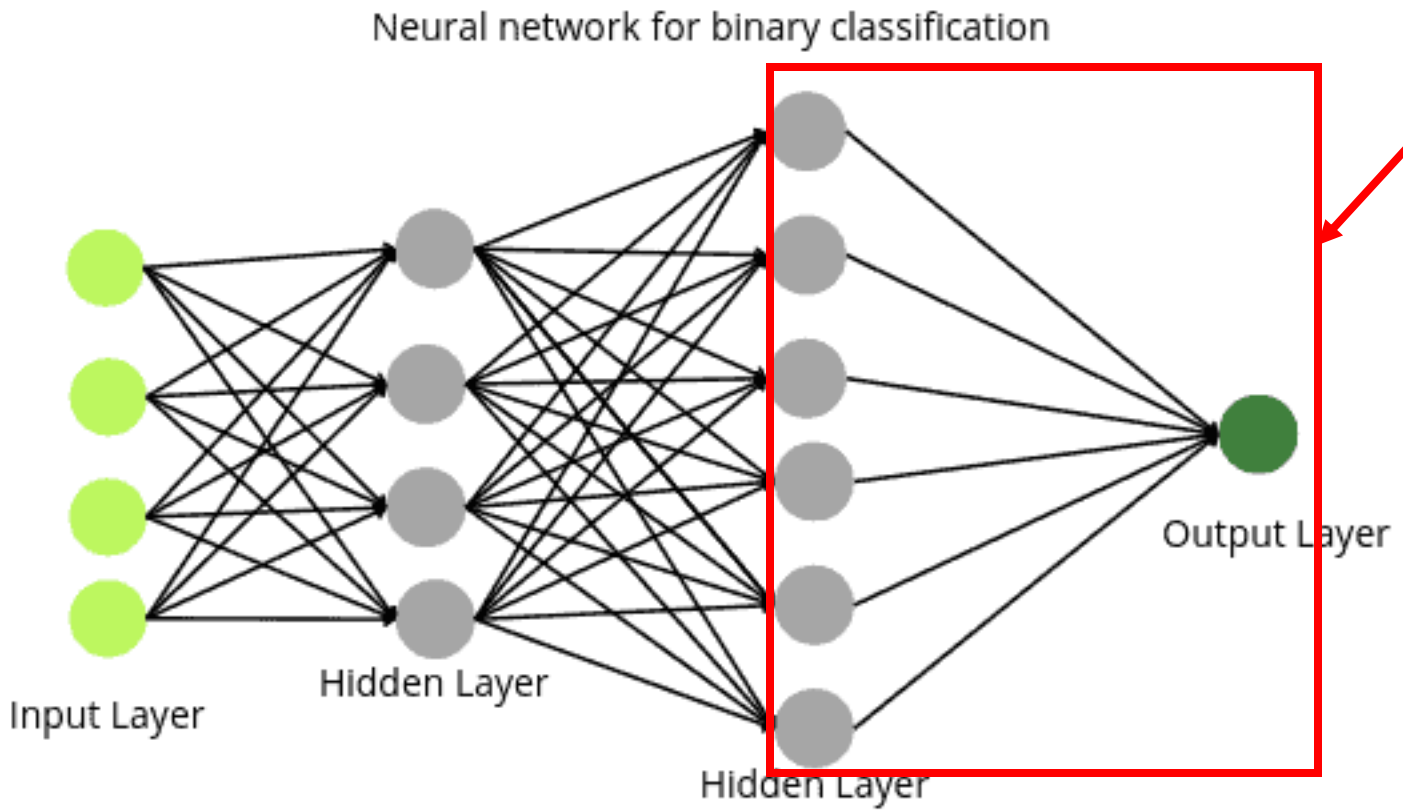
Our high-dimensional data is very sparse in high dimensions, perhaps there is a lower dimensional space where it can be better represented.



A learned manifold of MNIST

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

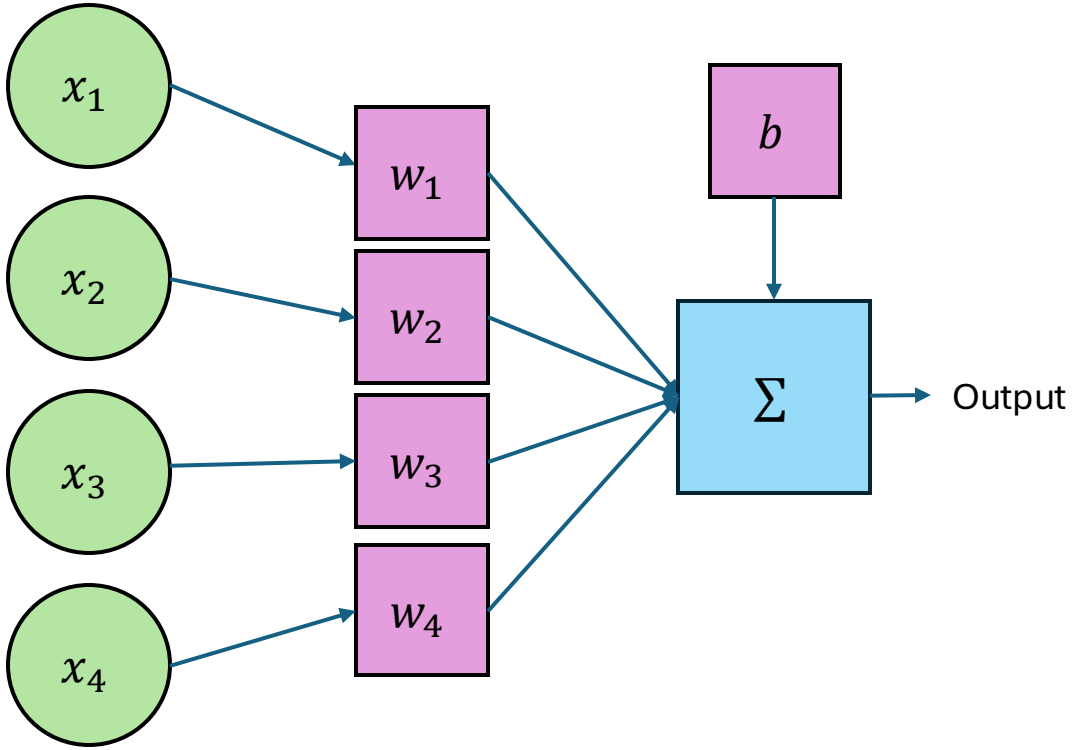
Deep Networks and Representation Learning

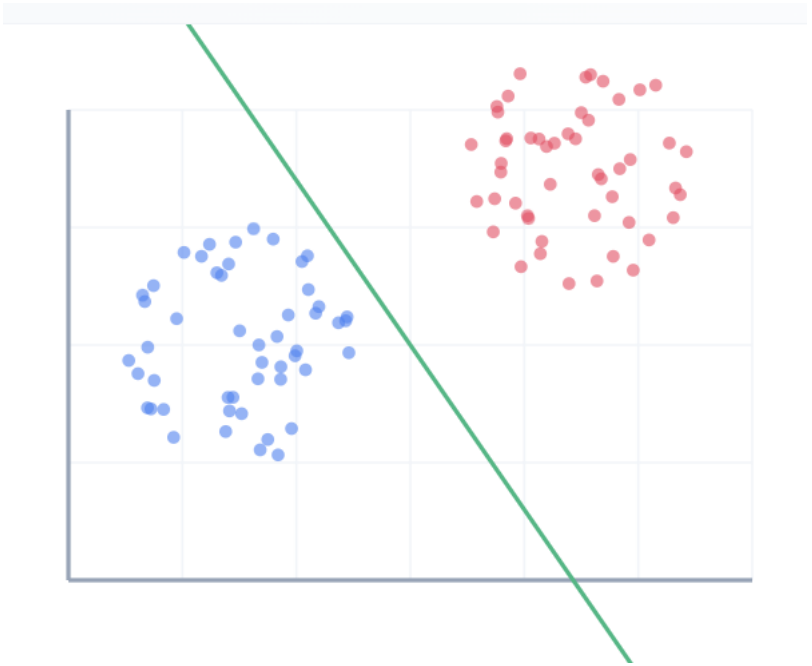


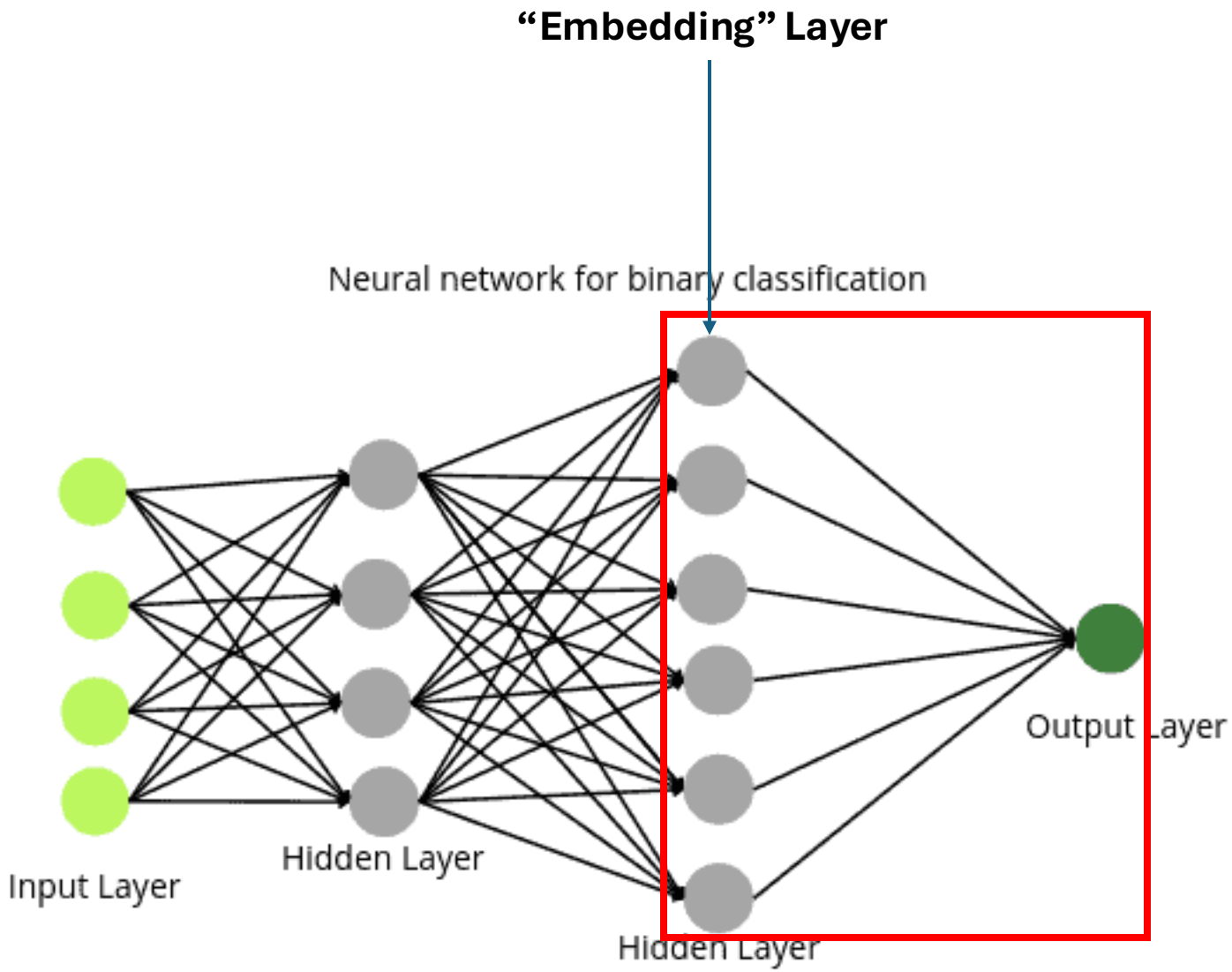
Layer has inputs and one output that maps inputs to [0, 1]

Looks a lot like a perceptron…

Perceptrons are Linear Separators



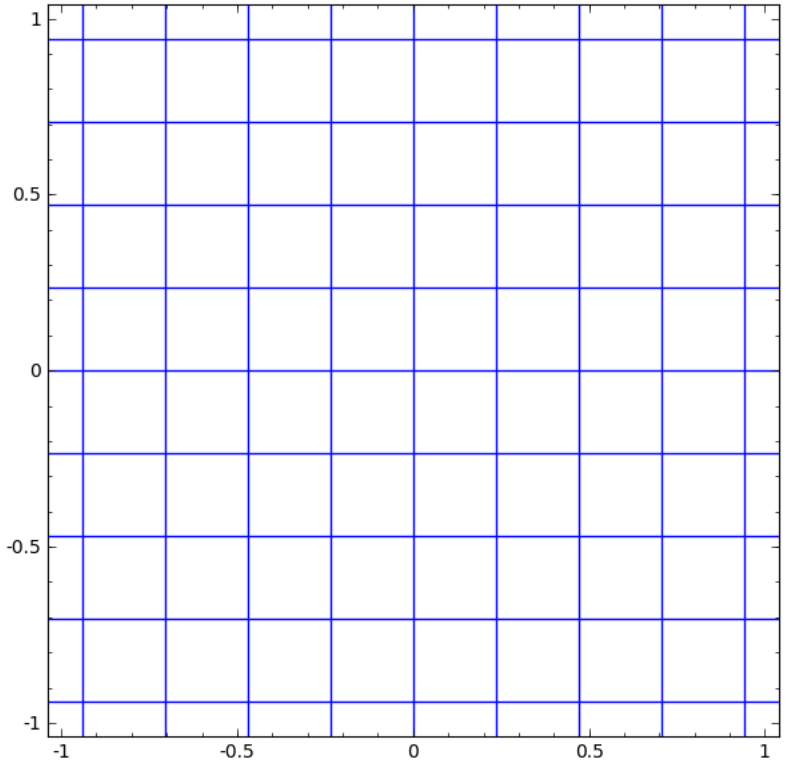




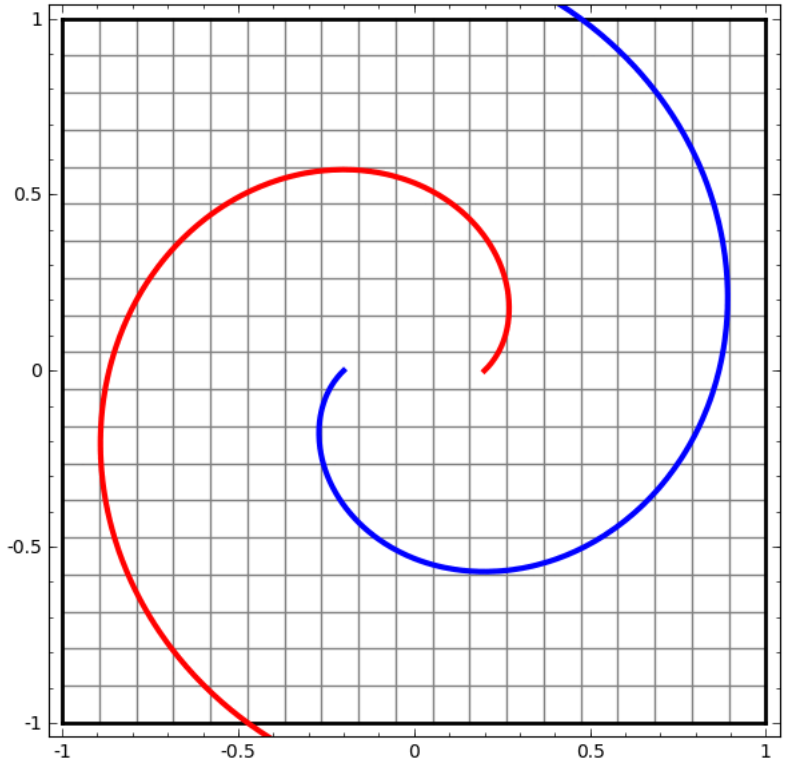
If the network can achieve 100% accuracy and the final layer is a linear separator (ala a perceptron), what does that imply about the embedding layer?

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



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

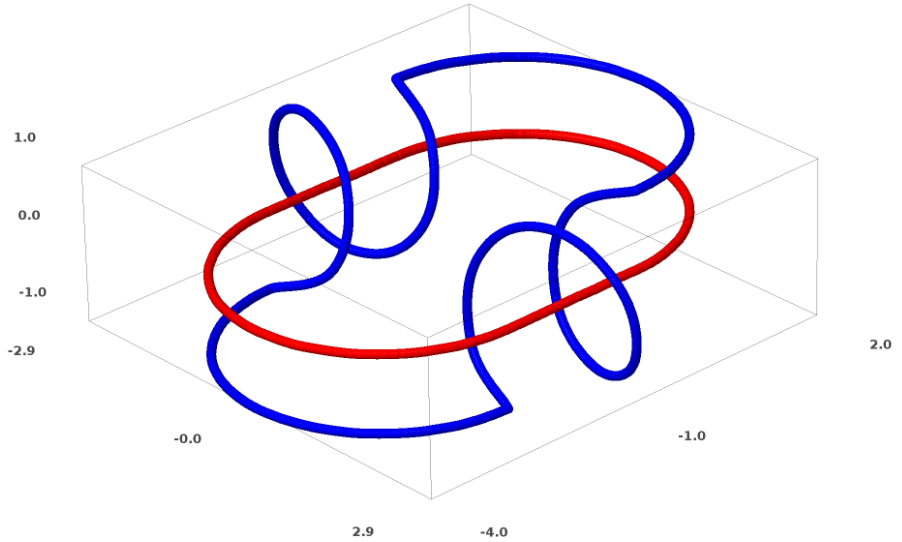


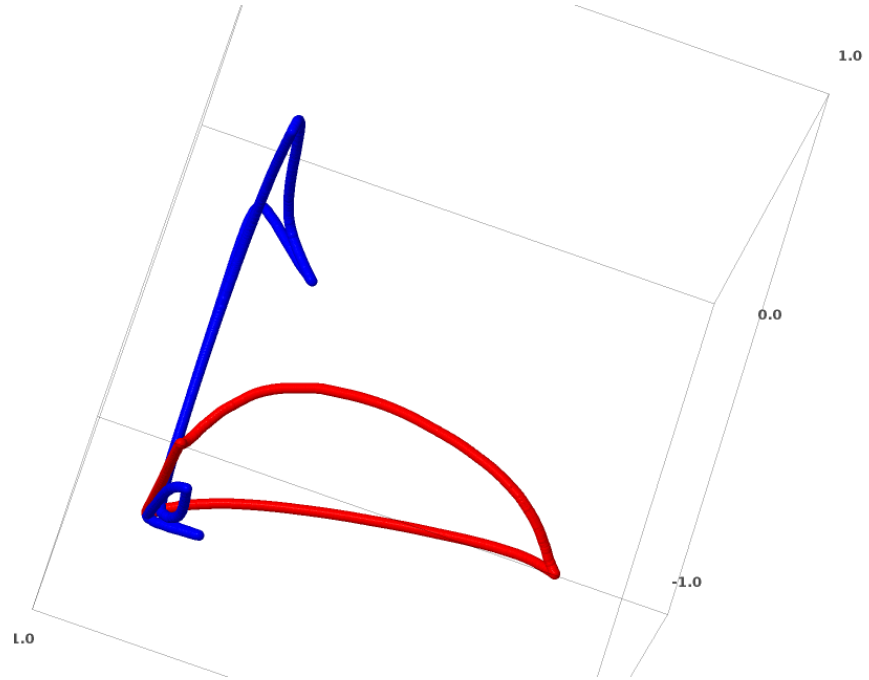
https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

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”



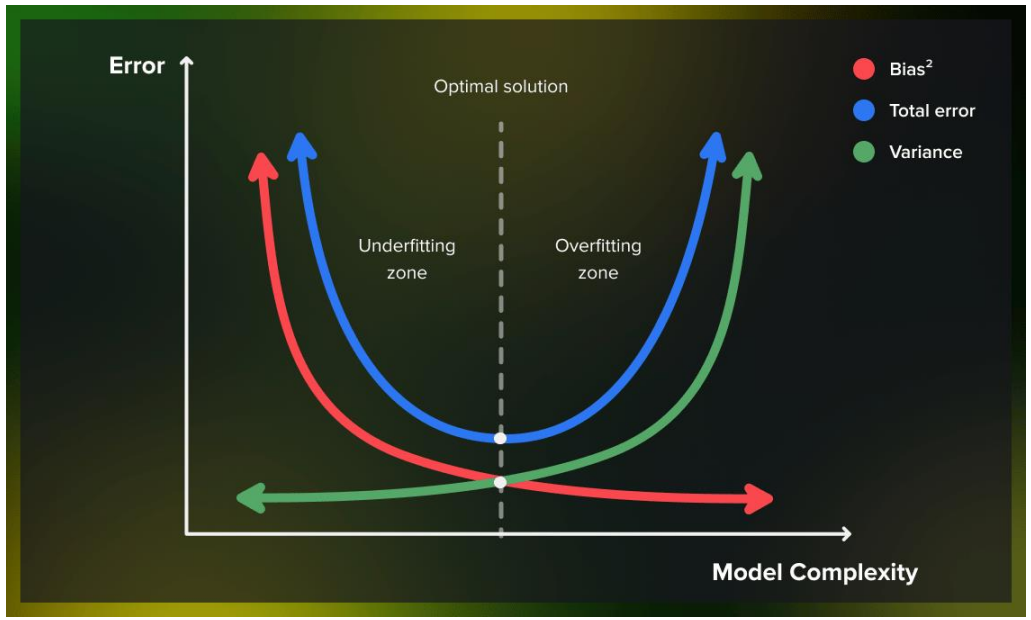


https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

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 linear layers
- Try constant layer width (e.g., 64 neurons for each hidden layer)
- Try “funnel” shape (e.g., 64->32->16->output)

Model Complexity



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

Model complexity: ~how many parameters are in the model.

A polynomial regression with 10 parameters is more complex than a linear regression.

Neural Networks do not necessarily follow this trend.

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 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

This paper uses Evolutionary Strategies (ES) to learn embeddings.

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

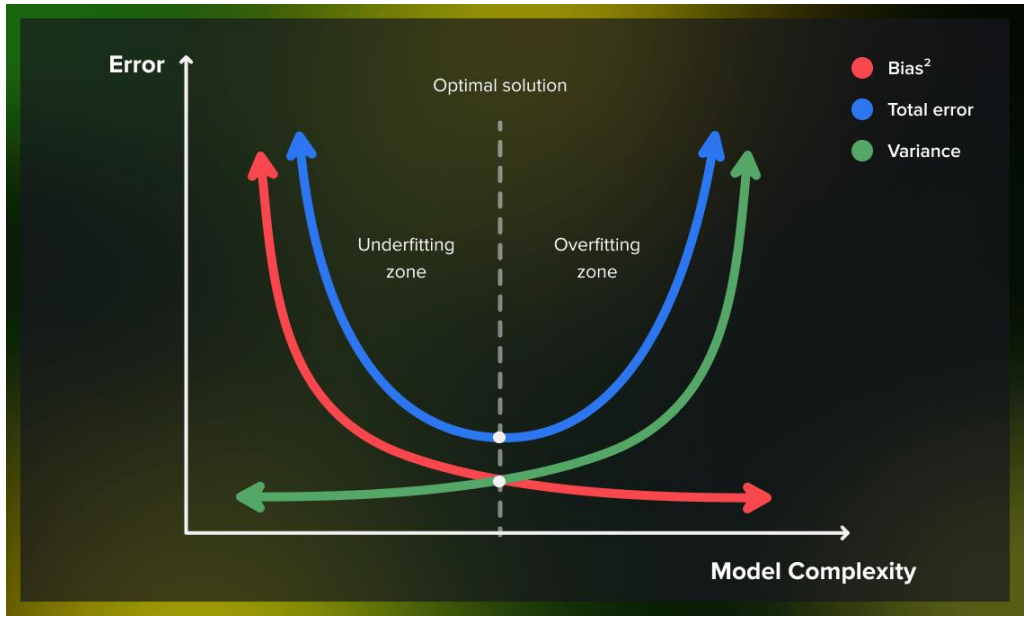
ABSTRACT

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.

(We will cover other techniques for managing 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

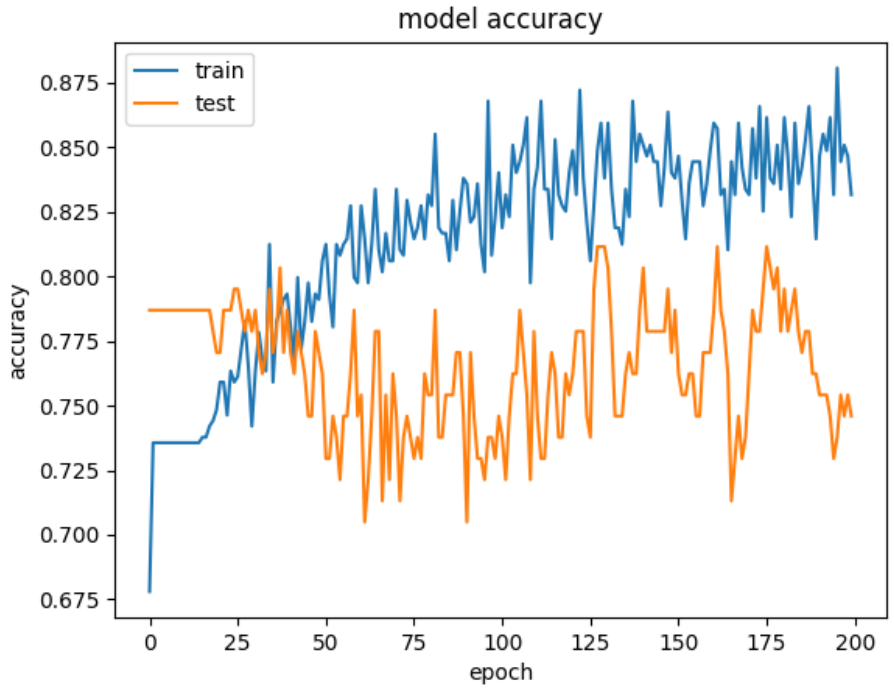
- SGD, SGD + Momentum, SGD + Adaptive Momentum (Adam), RMSProp, Shampoo… the list is ever growing

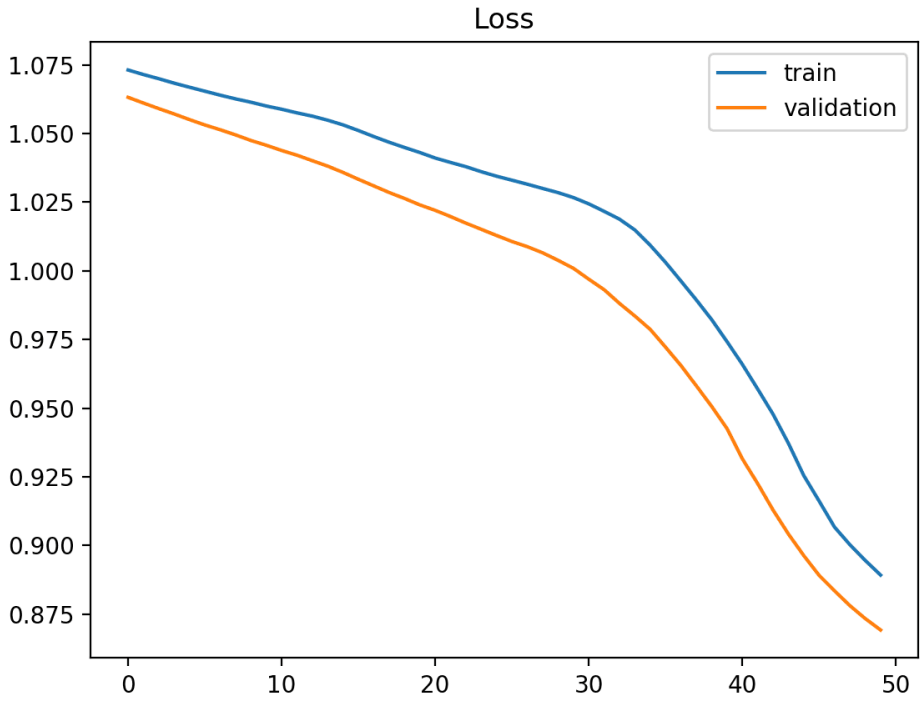
- How do you choose between them?

- Just use Adam (at least to start)
  - “Safest” of the optimizers and most forgiving if learning rate is bad
  - 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

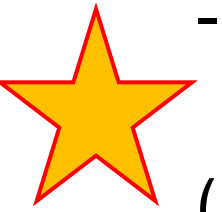
Having too small a batch or too high a learning rate can cause variance in training/validation loss – symptoms often look similar





General Tips

- 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.



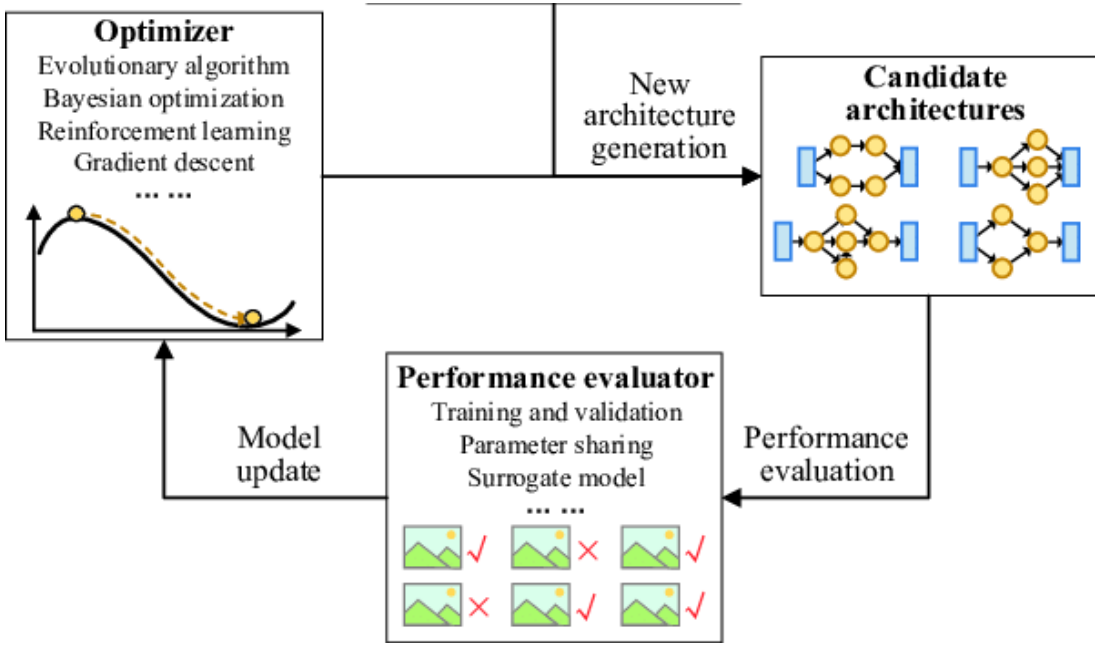
(Use the scientific method)

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.

What are some downsides of grid search?

AutoML

Option #2: Bayesian Optimization

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

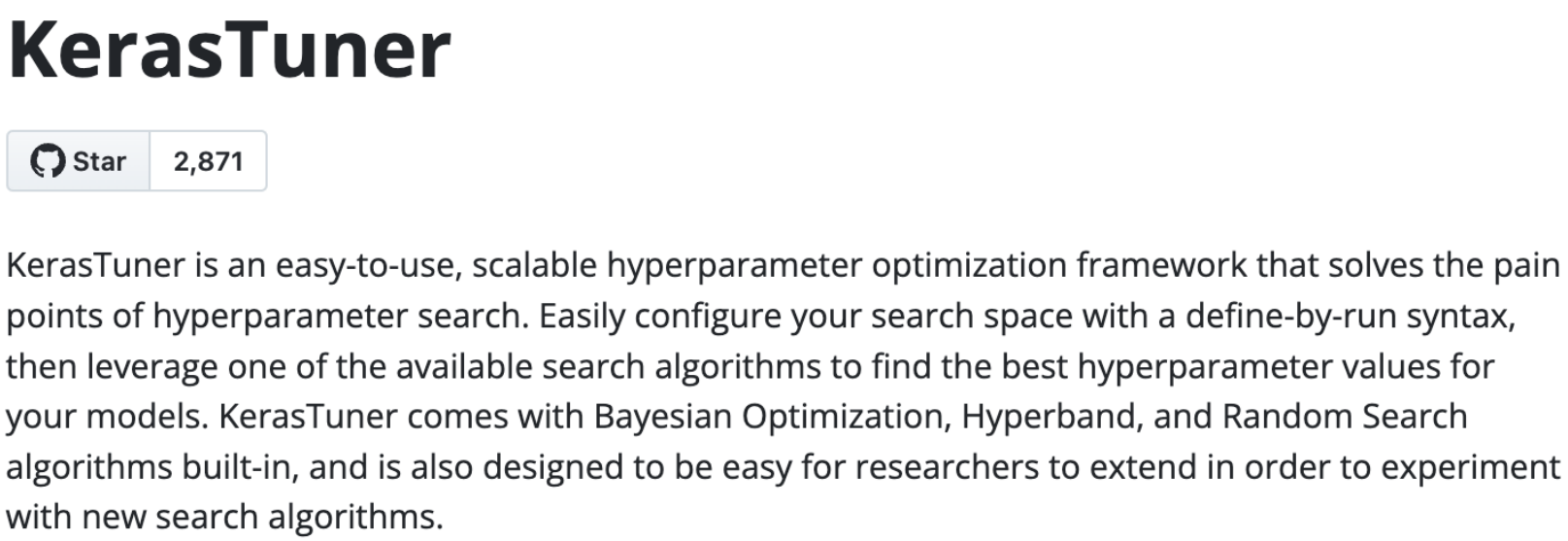
Define a set of possible parameters (i.e., learning rates, width, depth, etc)

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

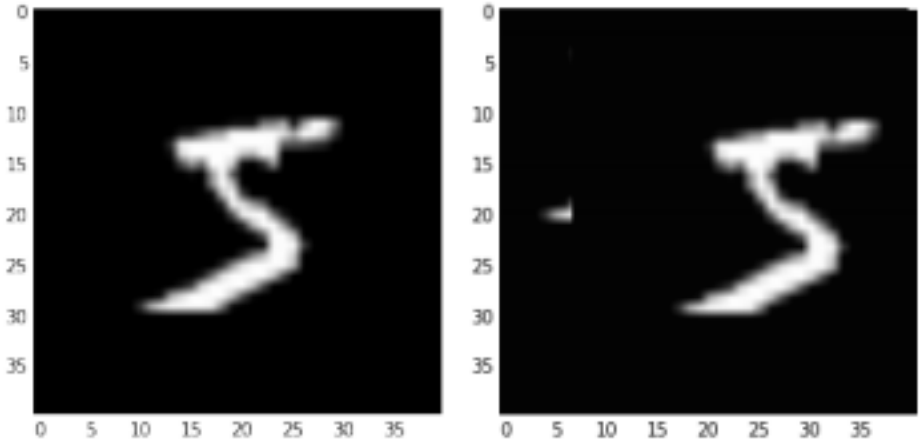


Looking Forward

Up until this point, we've covered Neural Networks generally referred to as MLPs, feed forward networks, or networks made up of "Linear Layers"

Up next: Convolutional Neural Networks

What happens if our input image is shifted?



Looking Forward

MLPs have a fixed number of inputs and outputs

Problem: We collect patient medical data from doctor's visits to make predictions about a disease prognosis.

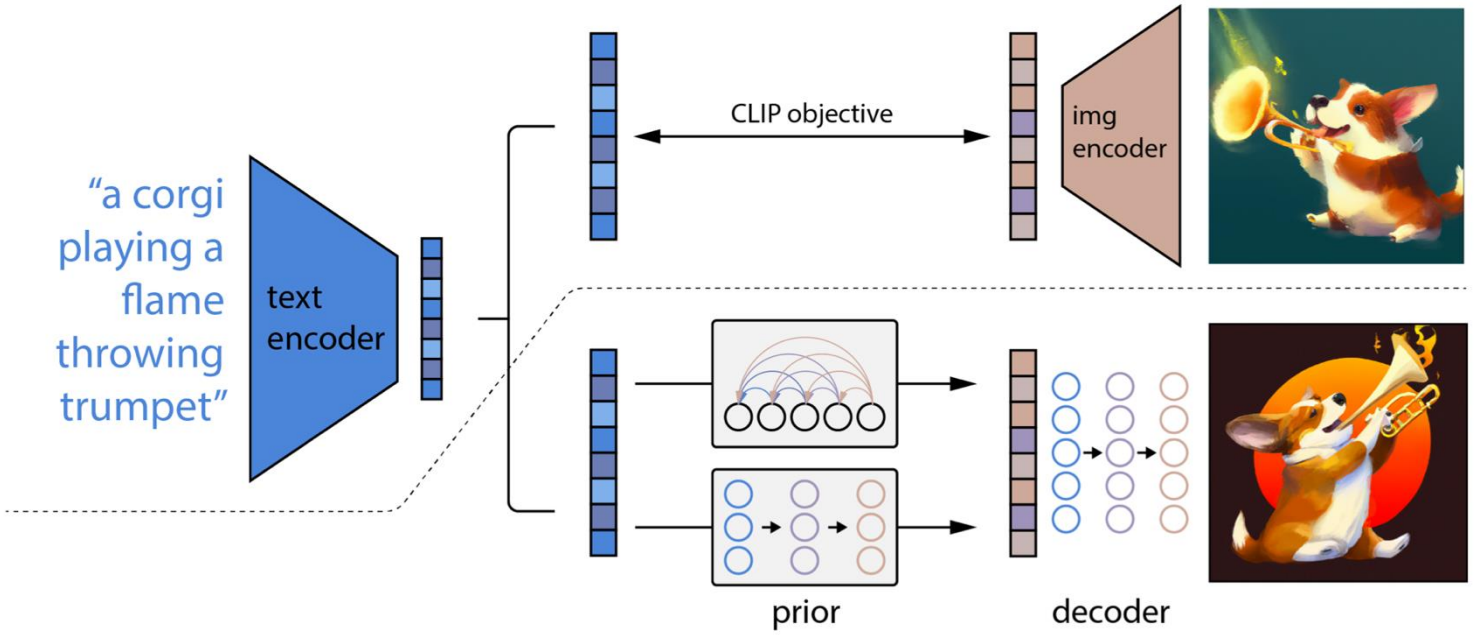
Patients visit the doctor multiple times, but not the same number of times.

How can we make predictions when given a sequence of inputs?

Solution: Recurrent Neural Networks and Attention

Looking Forward

Ok, so given we can take in inputs of sequences (perhaps, a sentence), how can we use that input to *generate* something rather than perform classification/regression?



https://nathanfradet.com/posts/text-to-image/

Recap

Deep MLPs tend to be more “efficient” than wide MLPs

The number of neurons and size of neurons are determine a model’s complexity. Models with high complexity will *tend* to overfit, low complexity underfit.

Understanding how MLPs learn parameters is essential for hyperparameter tuning.

The rest of this course will build on the foundation of MLPs to introduce flexibility to our learned systems