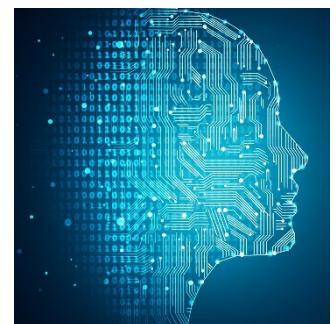


Machine Learning Neural Networks



Kevin Moon (kevin.moon@usu.edu)

STAT/CS 5810/6655





Overview

Neural networks are a class of ML algorithms that can be applied to solve just about any problem

For many problems, NNs achieve state-of-the-art performance, e.g., object recognition, speech recognition, machine translation

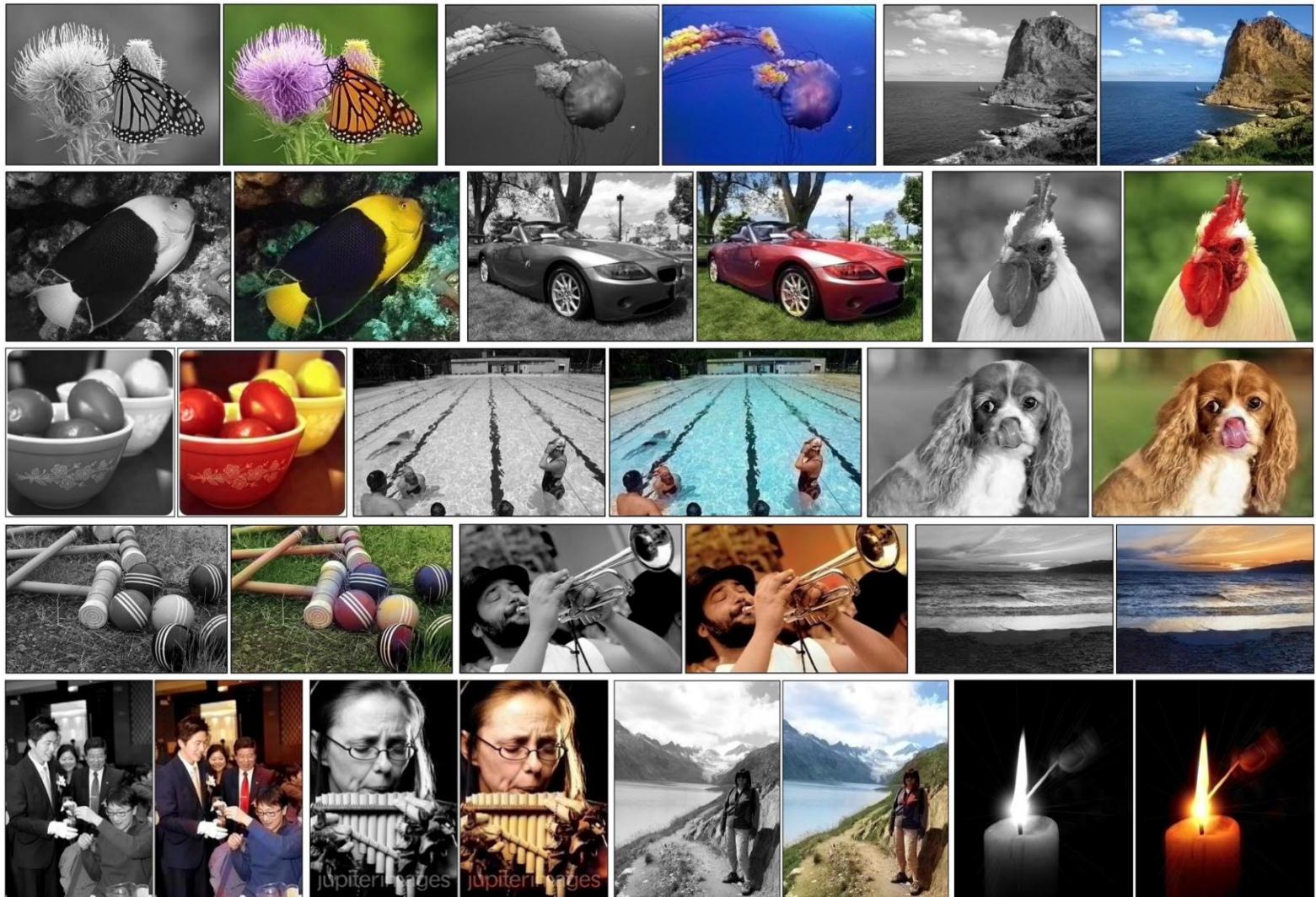
Architectures

- Multilayer perceptron (supervised learning in general)
- CNN (supervised learning with image data)
- Autoencoder (dimensionality reduction, denoising)
- Recurrent neural network (time series analysis, e.g., audio, text)
- Generative adversarial network (synthesis)
- Self-organizing maps (clustering)
-



Success of Neural Networks

Image colorization (Zhang et al., 2016)





Success of Neural Networks

Image colorization (Zhang et al., 2016)



Colorized classical photographs by Ansel Adams

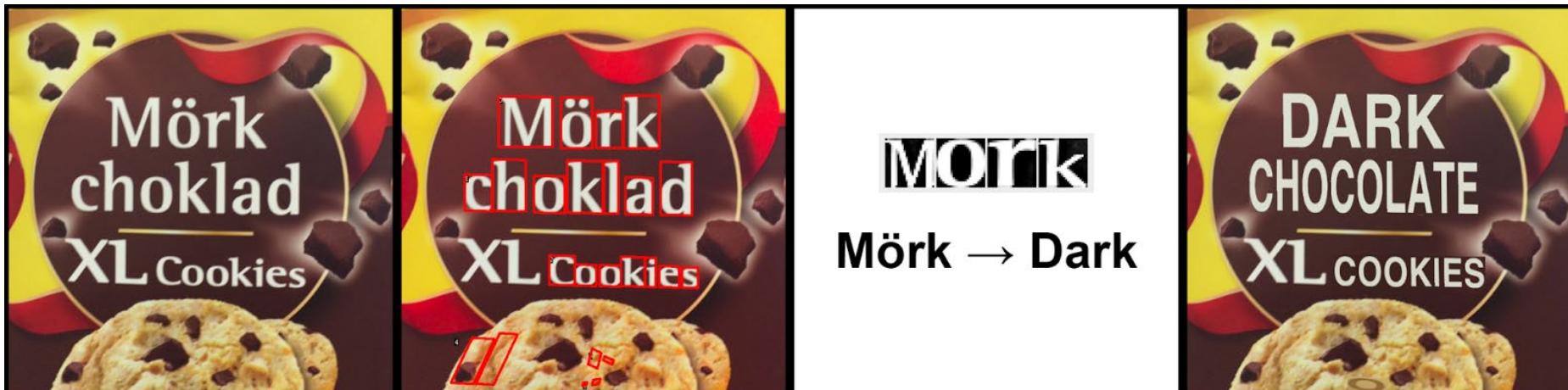


Success of Neural Networks



Real-time visual translation on smartphones

1. Find the letters
2. Recognize the letters
3. Translate
4. Render the translation in the same style

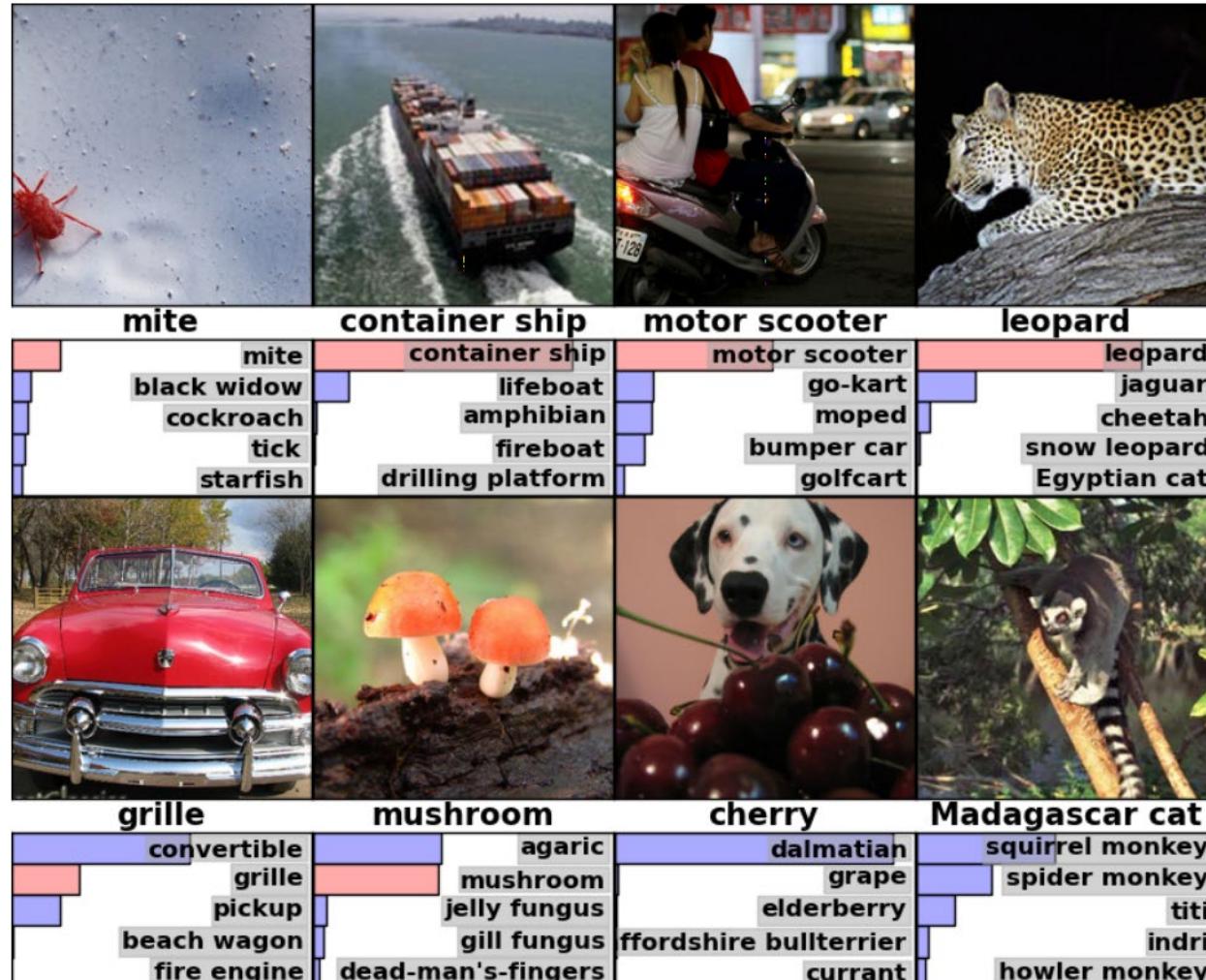


Google blog, 2015



Success of Neural Networks

Object classification/detection in images (Krizhevsky et al., 2012)





Success of Neural Networks



Automatic image caption generation (Karpathy & Fei-Fei, 2015)



"man in black shirt is playing guitar."



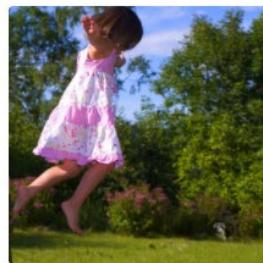
"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."



"little girl is eating piece of cake."



"baseball player is throwing ball in game."



"woman is holding bunch of bananas."



"black cat is sitting on top of suitcase."



Success of Neural Networks

Automated essay writing with ChatGPT

Reset Thread

Dark Mode

OpenAI Discord

Learn More

Log out

 Can you write me a 4 paragraph academic essay comparing and contrasting the theories of nationalism of Benedict Anderson and Ernest Gellner?

 Benedict Anderson and Ernest Gellner both have theories about the origins and nature of nationalism, but their perspectives differ in key ways. Anderson believes that nationalism is a cultural construct, arising from the development of print capitalism and the creation of imagined communities. Gellner, on the other hand, sees nationalism as a product of modernization and industrialization, with nations being formed as a means of organizing society and providing a shared identity.

One key difference between the two theories is the role of culture. For Anderson, culture is central to the creation of nationalism, as the spread of print media allows for the creation of imagined communities that transcend physical boundaries. This allows for the development of a national identity based on shared cultural experiences and practices. In contrast, Gellner downplays the importance of culture in the formation of nationalism, arguing that it is primarily a function of economic and political developments. He sees nations as being formed in response to the need for a shared identity and sense of belonging in an industrialized society.

Another point of contrast is the emphasis on agency. Anderson's theory suggests that people actively participate in the construction of national identity, through their engagement with print media and other cultural practices. In contrast, Gellner's theory emphasizes the role of external factors in shaping national identity, with individuals being passive recipients of the national identity imposed on them by the forces of modernization.

Overall, while both Anderson and Gellner provide valuable insights into the origins of nationalism, their theories offer different perspectives on the role of culture and agency in the formation of national identity. Anderson emphasizes the cultural construction of nationalism, while Gellner sees it as a product of modernization and industrialization. Both theories offer valuable insights into the complex process of national identity formation, and can be useful in understanding the dynamics of nationalism in different historical contexts.

< 3 / 3 >



Success of Neural Networks

Automated programming with ChatGPT



Find the bug with this code:

```
...
for (var i = 0; i < 5; i++) {
    setTimeout(() => console.log(i), 1000)
}
...
```



The code contains a bug that will cause it to always print the number 5 to the console, rather than the expected sequence of numbers from 0 to 4. This is because the `setTimeout` function creates a closure that captures the value of the `i` variable, but the `i` variable is changed before the callback function is executed.



To fix this bug, the code can be modified to create a new variable in the loop that is passed to the `setTimeout` callback function, rather than using the loop variable directly. This will ensure that each callback function captures the correct value of the loop variable, and the expected output will be produced. Here is an example of how the code could be fixed:

Copy code

```
for (var i = 0; i < 5; i++) {
    // Create a new variable and pass its value to setTimeout
    const value = i;
    setTimeout(() => console.log(value), 1000)
}
```

This code will print the numbers 0 to 4 to the console, as expected. Alternatively, the `let` keyword can be used in place of `var` to declare the loop variable, which will automatically create a new variable for each iteration of the loop and avoid the need to create a new variable manually. This is a common pattern for avoiding closure-related bugs in JavaScript.



Success of Neural Networks

AI generated art



Best Dalle2 Pics
@Dalle2Pics

...

A raccoon playing tennis at Wimbledon in the 1990s
[#dalle2 #dalle](#)



3:33 AM · May 14, 2022



Success of Neural Networks

Automatic game playing

Alpha Go Zero

nature
International Journal of science

Access provided by Yale University

Altmetric: 2188 Citations: 1 [More detail >](#)

Article

Mastering the game of Go without human knowledge

David Silver , Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature 550, 354–359 (19 October 2017)
doi:10.1038/nature24270
[Download Citation](#)
[Computational science](#) [Computer science](#) [Reward](#)

Received: 07 April 2017
Accepted: 13 September 2017
Published online: 18 October 2017

Alpha Zero

[arXiv.org > cs > arXiv:1712.01815](#)

Search or
(Help | Advanced)

Computer Science > Artificial Intelligence

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis

(Submitted on 5 Dec 2017)

The game of chess is the most widely-studied domain in the history of artificial intelligence. The strongest programs are based on a combination of sophisticated search techniques, domain-specific adaptations, and handcrafted evaluation functions that have been refined by human experts over several decades. In contrast, the AlphaGo Zero program recently achieved superhuman performance in the game of Go, by tabula rasa reinforcement learning from games of self-play. In this paper, we generalise this approach into a single AlphaZero algorithm that can achieve, tabula rasa, superhuman performance in many challenging domains. Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

Subjects: Artificial Intelligence (cs.AI); Learning (cs.LG)
Cite as: [arXiv:1712.01815 \[cs.AI\]](#)
(or [arXiv:1712.01815v1 \[cs.AI\]](#) for this version)

Submission history

From: David Silver [[view email](#)]
[v1] Tue, 5 Dec 2017 18:45:38 GMT (272kb,D)



Neural Network Overview

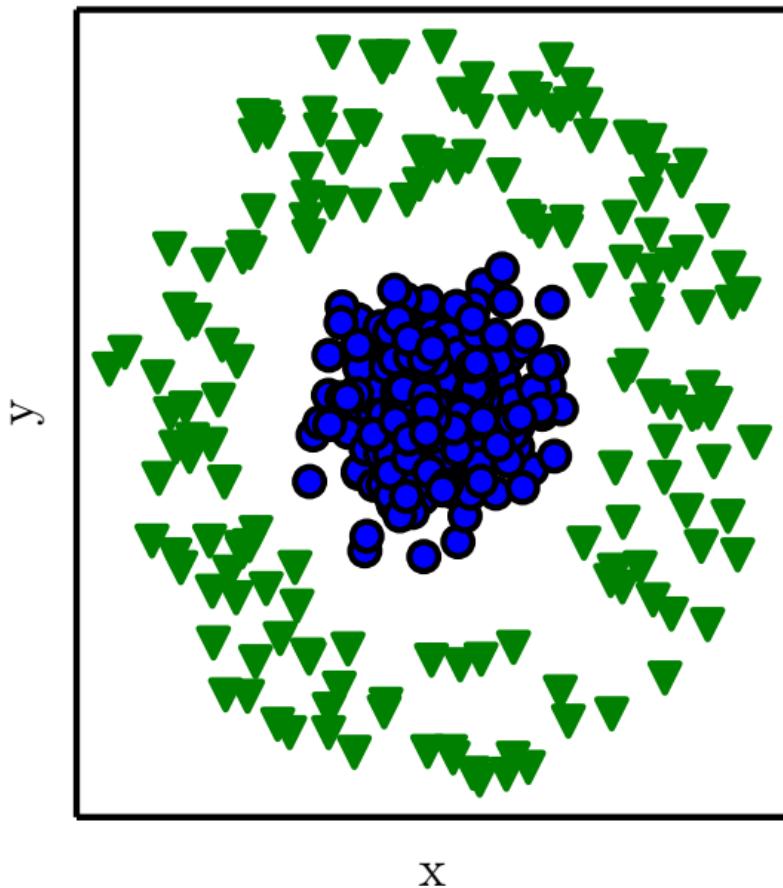
- Goal of a neural network is to learn a function
 - E.g., regression, classification, translation, data generation
- Neural networks are parametric
 - The number of weights and biases that can be learned is fixed for a given architecture
- Learning is performed by choosing the weights and biases that minimize a loss function
- Minimization uses (almost always) stochastic gradient descent (SGD)
- Gradients for SGD are calculated using backpropagation



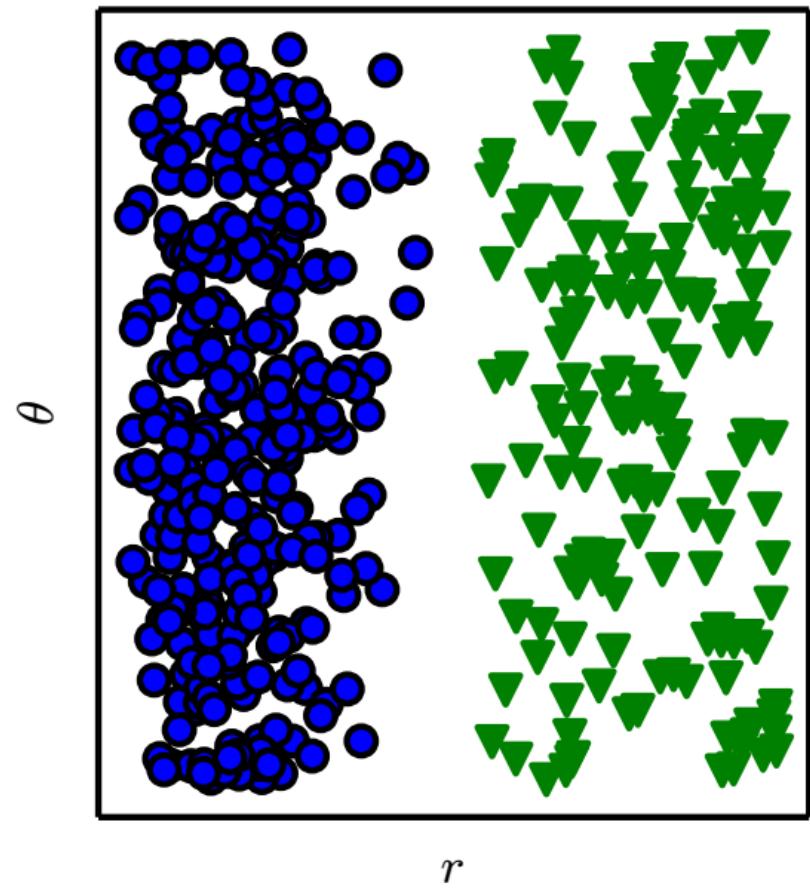
Representation Learning

Representations matter

Cartesian coordinates



Polar coordinates

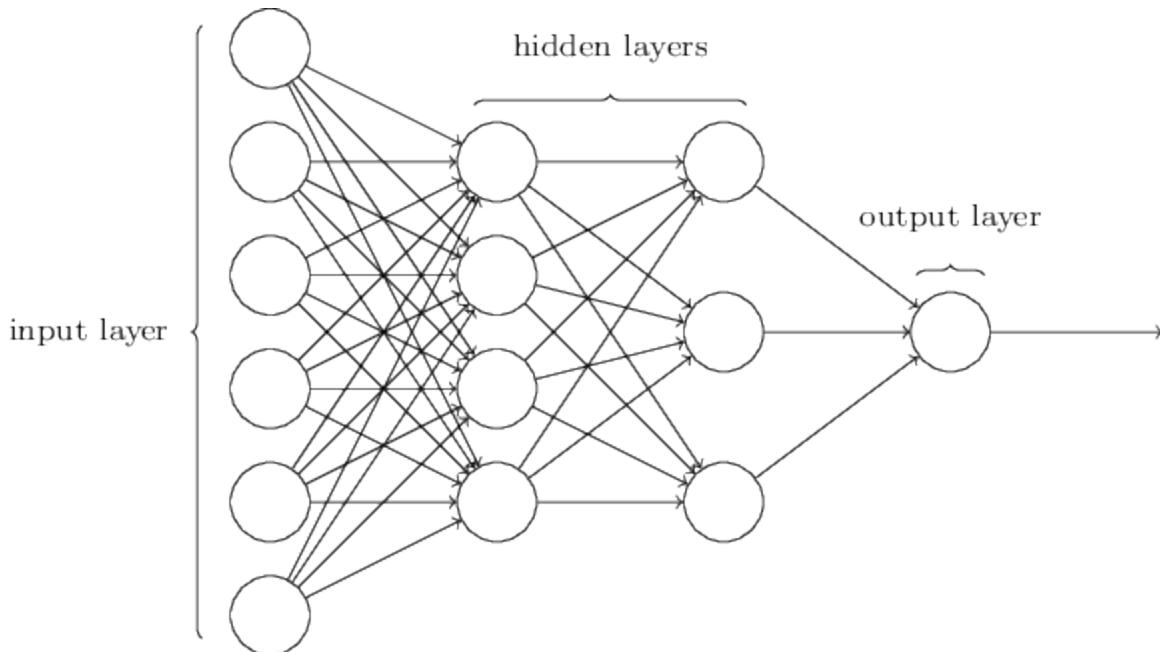


Goodfellow et al., 2016



Multilayer Perceptron

- A perceptron is basically a linear classifier or regression function
- Historically, “perceptron” implies a specific algorithm for learning the weights but that isn’t relevant for us today
- Let’s begin with regression
- A *multilayer perceptron* (MLP) is a nonlinear model for a regression function





Notation



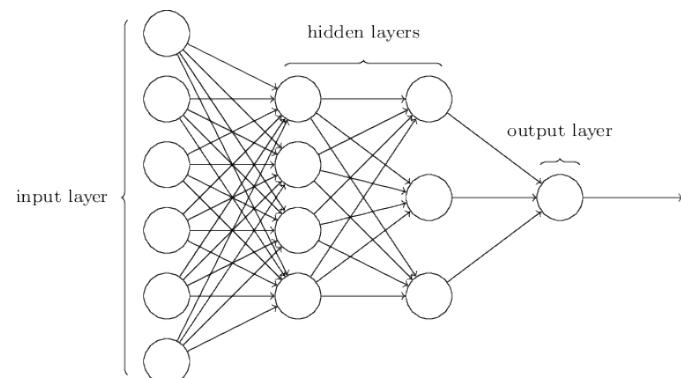
- Sub- and superscripts used differently than in previous lectures
- Input layer

$$\mathbf{z}^{(0)} = \mathbf{x}$$

- Hidden layer $1 \leq \ell < L$

$$\mathbf{a}^{(\ell)} = W^{(\ell)} \mathbf{z}^{(\ell-1)}$$

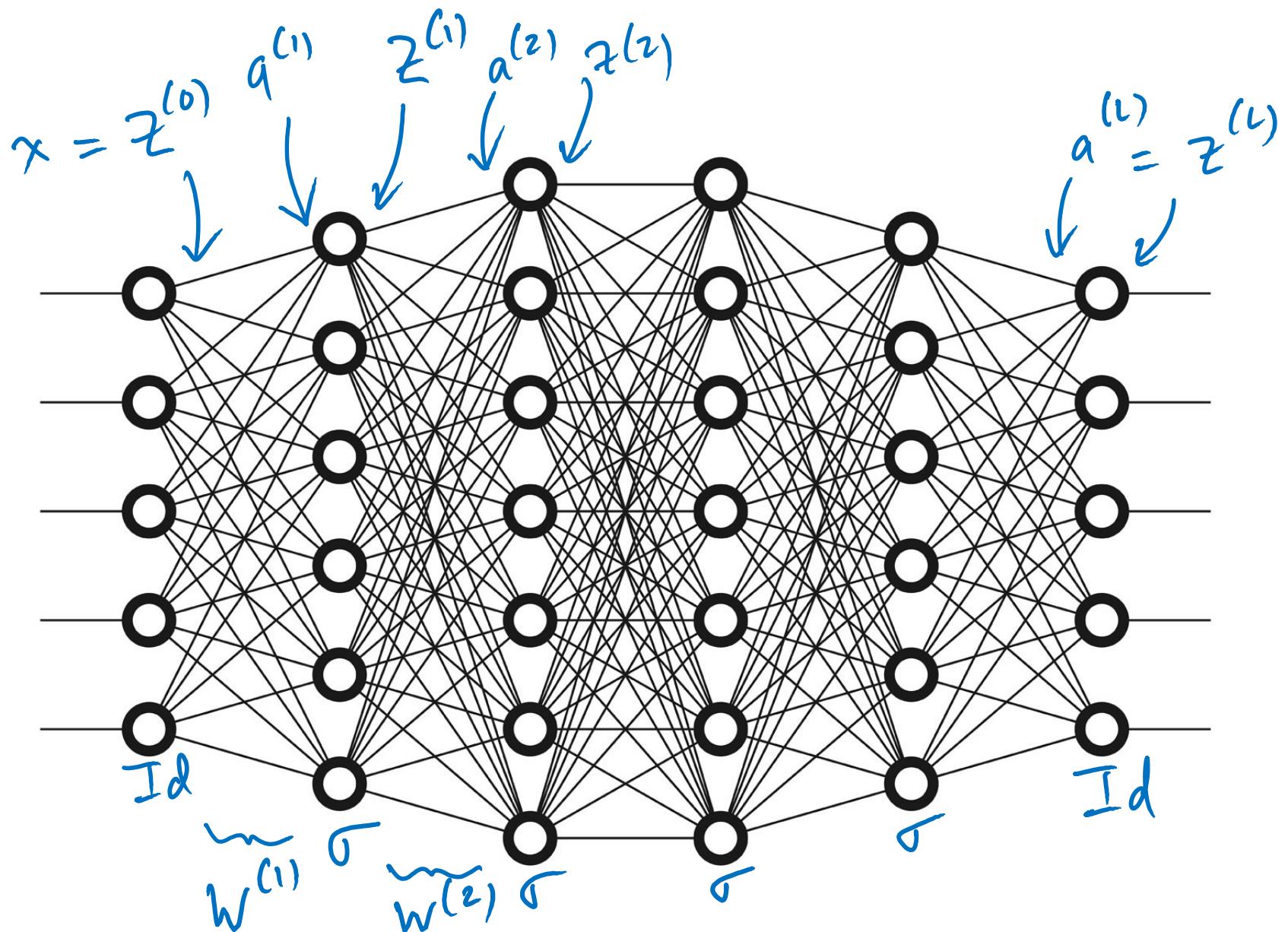
$$\mathbf{z}^{(\ell)} = \sigma(\mathbf{a}^{(\ell)}) \quad \text{applied elementwise}$$



- If bias desired, prepend a 1 to any $\mathbf{z}^{(\ell)}$, $0 \leq \ell < L$
 - Output layer
- $$\mathbf{z}^{(L)} = \mathbf{a}^{(L)} \quad \text{(identity activation function)}$$
- Evaluation of the output from the input is called *forward propagation*.



Illustration of Notation





Final Predictor: Regression

- Just take the output layer variables

$$f(\mathbf{x}) = \mathbf{z}^{(L)} = \mathbf{a}^{(L)}$$

- Easily accommodates multidimensional outputs
 - # of output layer nodes = # of predictor variables



Final Predictor: Classification

- M classes
- Represent classes by indicator vectors:

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- Softmax predictor on the output: if $\mathbf{a} \in \mathbb{R}^m$, then
- $$S(\mathbf{a}) = \left[\frac{e^{a_1}}{\sum_j e^{a_j}} \quad \dots \quad \frac{e^{a_m}}{\sum_j e^{a_j}} \right]^T$$
- Prediction: $S(\mathbf{a}^{(L)})$

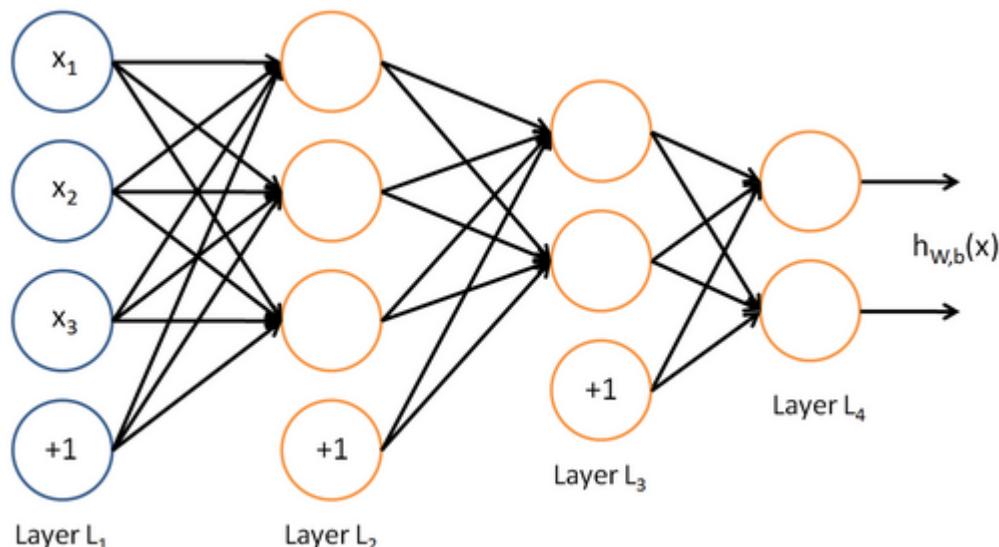


MLP Architectures

User needs to specify

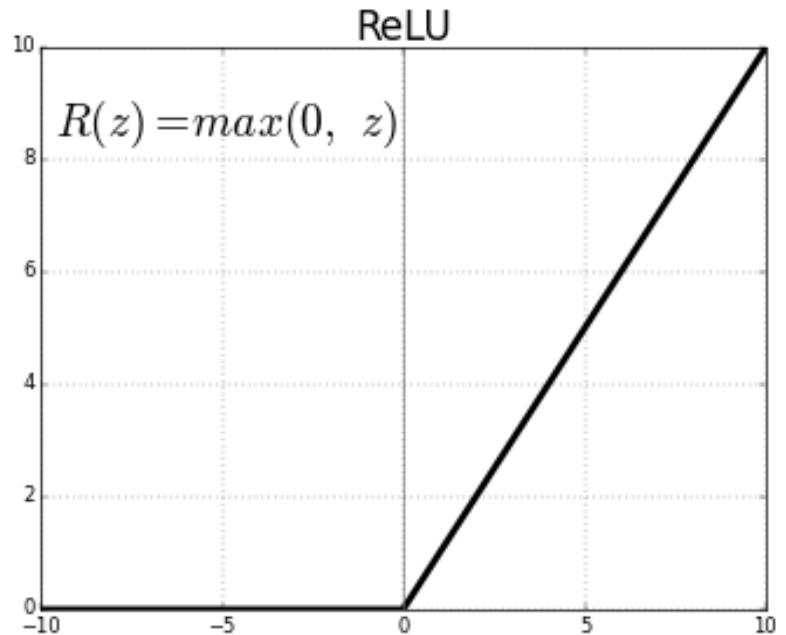
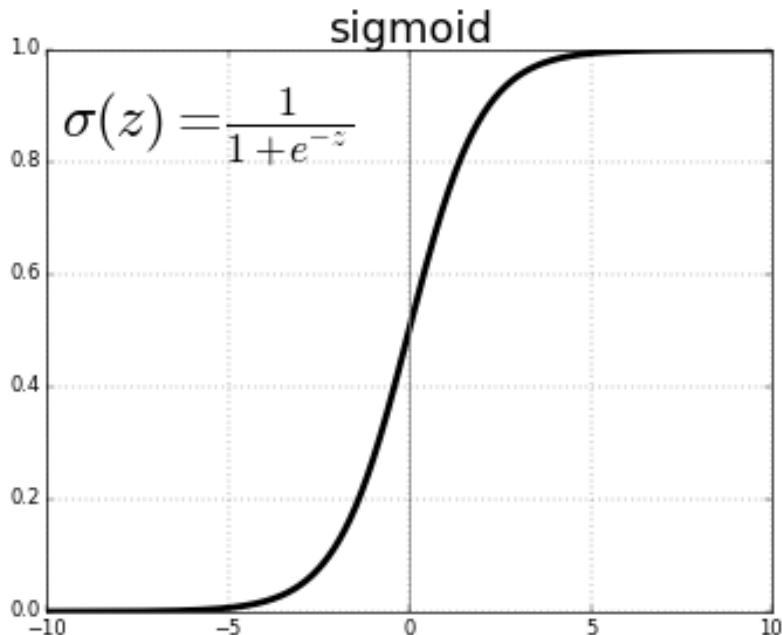
- Number of hidden layers
- Number of nodes per hidden layer
- Nonlinear activation function
- Bias at each layer?

These architectures are also referred to as “fully connected” or “feed-forward NNs”





Activation Functions



- Others such as leaky ReLU, tanh



Weight Matrix Notation

- If r and s are positive integers, let $w_{rs}^{(\ell)}$ denote the entry of $W^{(\ell)}$ at row r , column s



Group Exercise

1. Why is the softmax function on the final layer good for classification? Given the output vector of the softmax function on the final layer, how would you assign a class to an input?
2. Consider a MLP with one hidden layer and a single output node. Using a single expression write the output $f(\mathbf{x})$ as a function of the input \mathbf{x} and the weights $w_{ji}^{(\ell)}, \ell \in \{1,2\}$.



Training a neural network

- Given a specific architecture and training data, how do we learn the values of the weights and biases?
- Empirical risk minimization!
 - Need to choose an appropriate loss for the task
 - Can also include regularization terms



ERM: Regression Case

- Given training data $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$, need to learn

$$\boldsymbol{\theta} = (W^{(0)}, W^{(1)}, \dots, W^{(L-1)})$$

- Squared error loss:

$$R(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2$$

$$= \frac{1}{N} \sum_{n=1}^N R_n(\boldsymbol{\theta})$$

where $R_n(\boldsymbol{\theta}) = (y_n - f(\mathbf{x}_n))^2$.

- Note that f depends on $\boldsymbol{\theta}$ although the notation does not reflect this
- Gradient descent: How to calculate gradient?



Backpropagation

- To do gradient descent, we need an expression for the gradient of the loss function $R(\boldsymbol{\theta})$ wrt each component in $\boldsymbol{\theta}$
 - I.e., we need an expression for the gradient of the loss function wrt the weight matrix in each layer
- There is an efficient way to do this:
 - Write the gradient wrt the weights in the ℓ th layer in terms of the gradient wrt the weights in the $\ell + 1$ th layer
 - Calculating the entire gradient requires only one pass back through the network
 - This is called *backpropagation*
- See the resources or STAT/CS 6685 next semester for more details on backpropagation



Remarks

- Other losses are commonly used such as cross-entropy for classification
- Each hidden unit passes and receives information to and from only those units to which it is connected \Rightarrow parallel implementation
- In general, $R(\theta)$ is not convex and will have several local minima, so it is a good idea to try different initializations
- Regularization is very important to avoid overfitting and often many strategies are combined
 - Some techniques: early stopping, weight penalties, dropout, batch normalization
- Stochastic gradient descent is almost always used
- Feature learning perspective

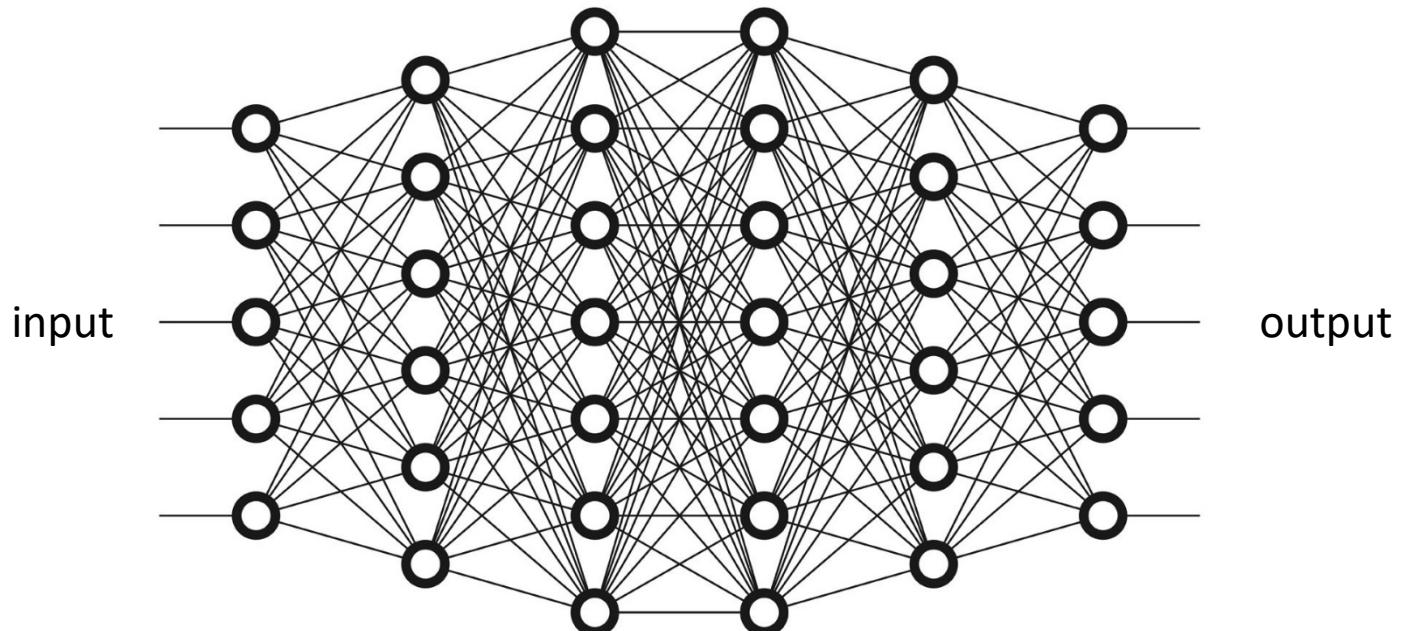
Recent trends

Moving beyond the 1980's



Till now...

- We have focused purely on the MLP



- Fully connected
- Feed-forward



Next

- Convolutional neural networks
- Autoencoders
- Recurrent neural networks
- Deep learning
- Attention and transformers
- Generative adversarial networks (GANs)
- Diffusion models
- Graph neural networks



Convolutions

- Basically a sliding window
- Key property: translation invariant – shifting the input just shifts the output by the same amount
- Figure assumes a *stride* (shift increment) of 1, but larger strides are possible

0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0

1	3	1
3	2	3
1	3	1

Feature map

0	0x1	0x3	0x1	0	0	0
0	1x3	1x2	1x3	1	0	0
0	1x1	0x3	0x1	1	0	0
0	1	1	1	1	0	0
0	1	0	0	1	0	0
0	1	1	1	1	0	0
0	0	0	0	0	0	0

Applying feature map

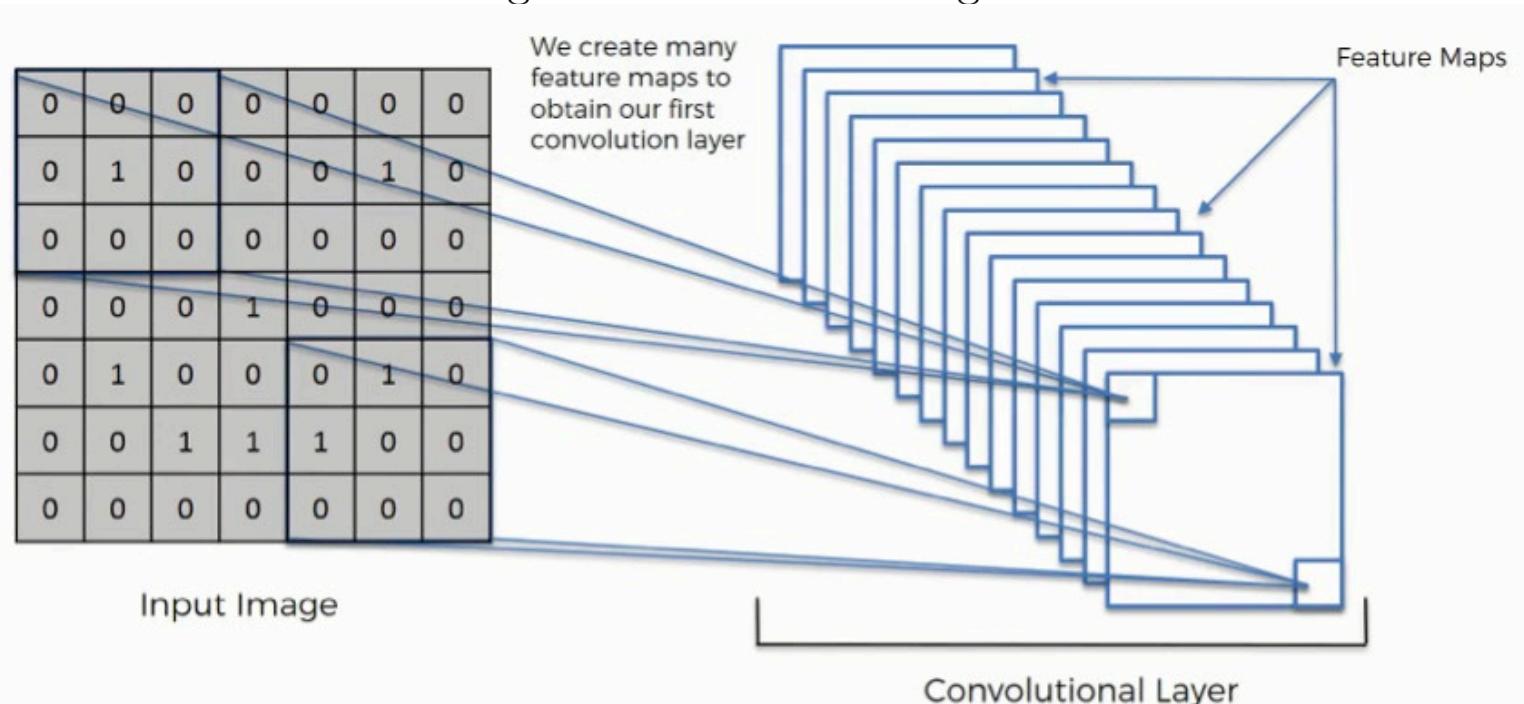
8	9						

Output: convolved feature



Convolutional Layers

- Several filters per layer
- Filter coefficients are the weights to be learned
- # of weights in layer = (filter size) \times (# of filters in layer)
- Useful for data with temporal or spatial structure, e.g.,
- Not fully connected + shared weights \implies far fewer weights





Examples of Filters/Feature maps

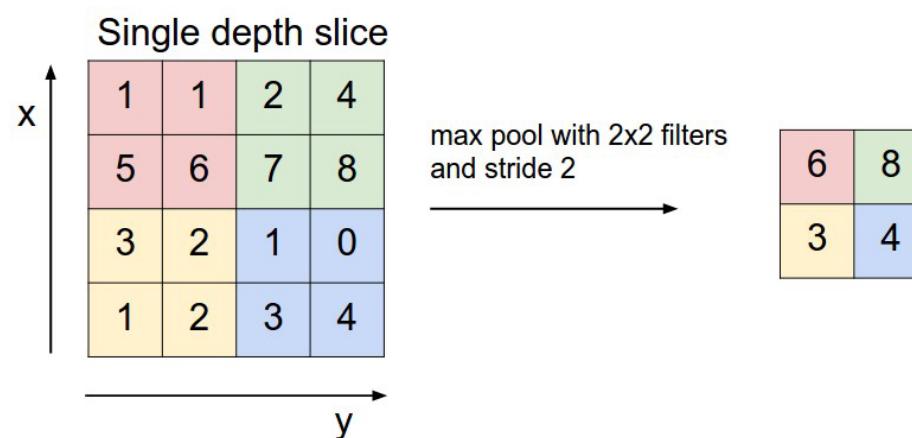
Operation	Filter	Convolved Image	
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$		
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$		
Edge detection	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$		
Edge detection	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		
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 (approximation)			$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$



Pooling/Downsampling Layer



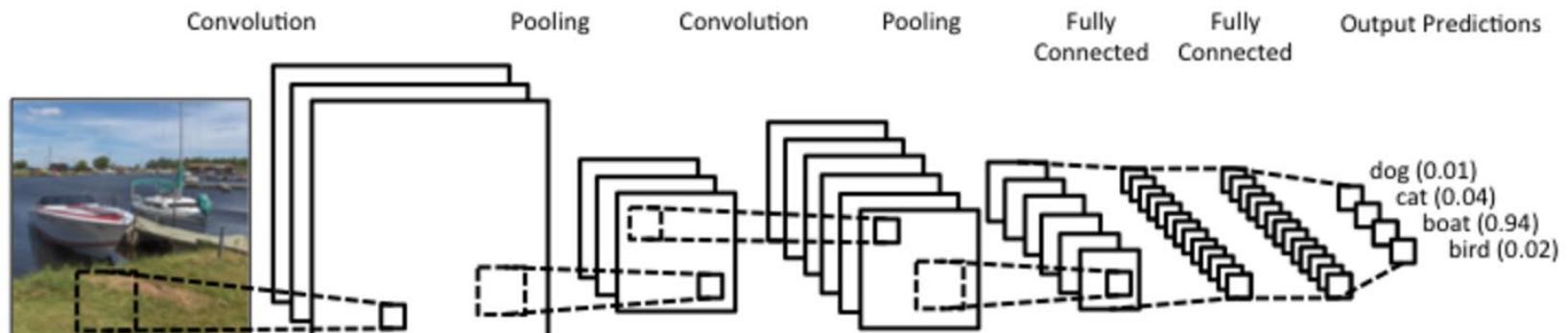
- Commonly combined with convolutional layers
- Objective is to shrink the layers leading to fewer weights to learn in subsequent layers
- No weights to learn in pooling layer
- Common implementation for images: *max pooling*, 2×2 window, stride of 2
- Other pooling operators: average pooling, sum pooling (but max tends to work best)





Convolutional Neural Networks

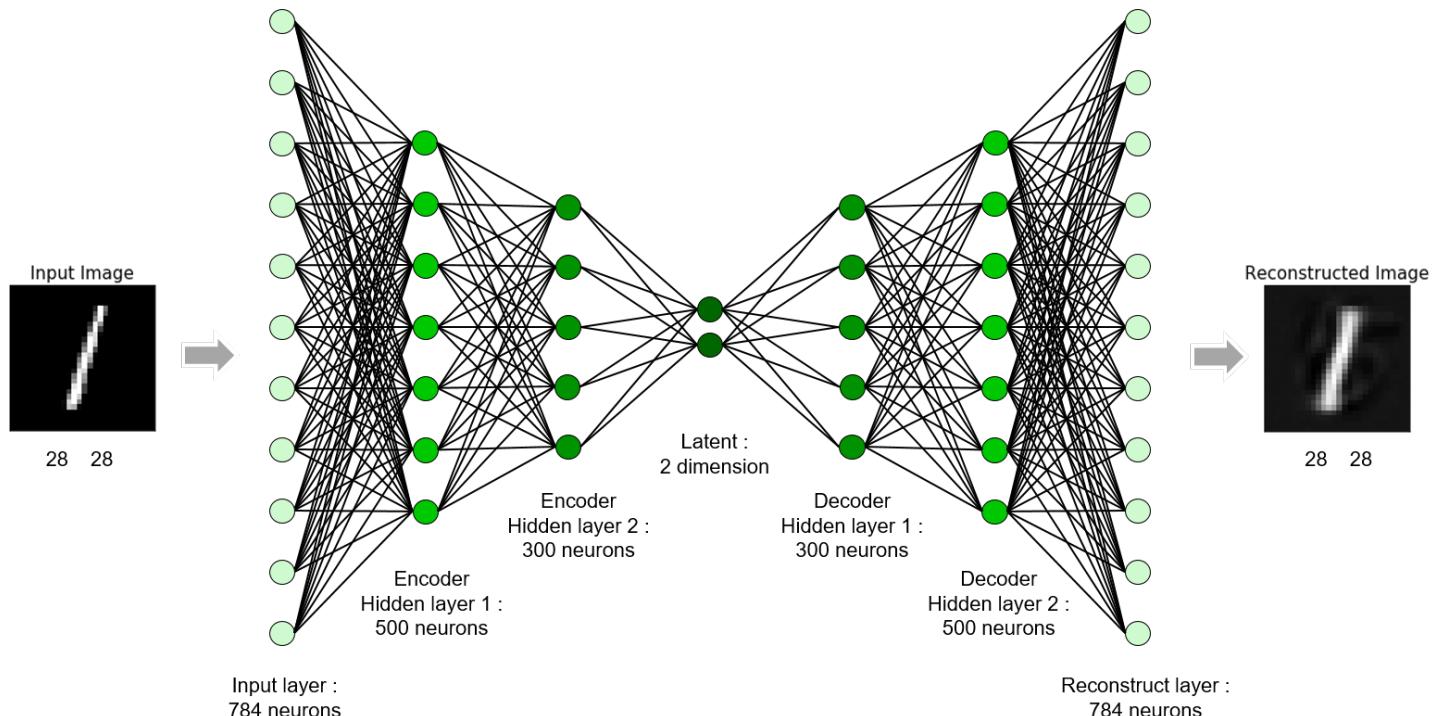
- Combination of convolutional, pooling, and fully connected layers
- Backprop extends naturally to CNNs
- Major breakthrough: LeNet5 (Yann LeCun et. al, 1998) for hand-written digit recognition
- Now used in Facebook's face recognition system and many other object recognition systems





Autoencoders

- These are neural networks for dimensionality reduction / feature extraction
- Idea: Try to reconstruct original data through a bottleneck
- Extracted features = bottleneck layer

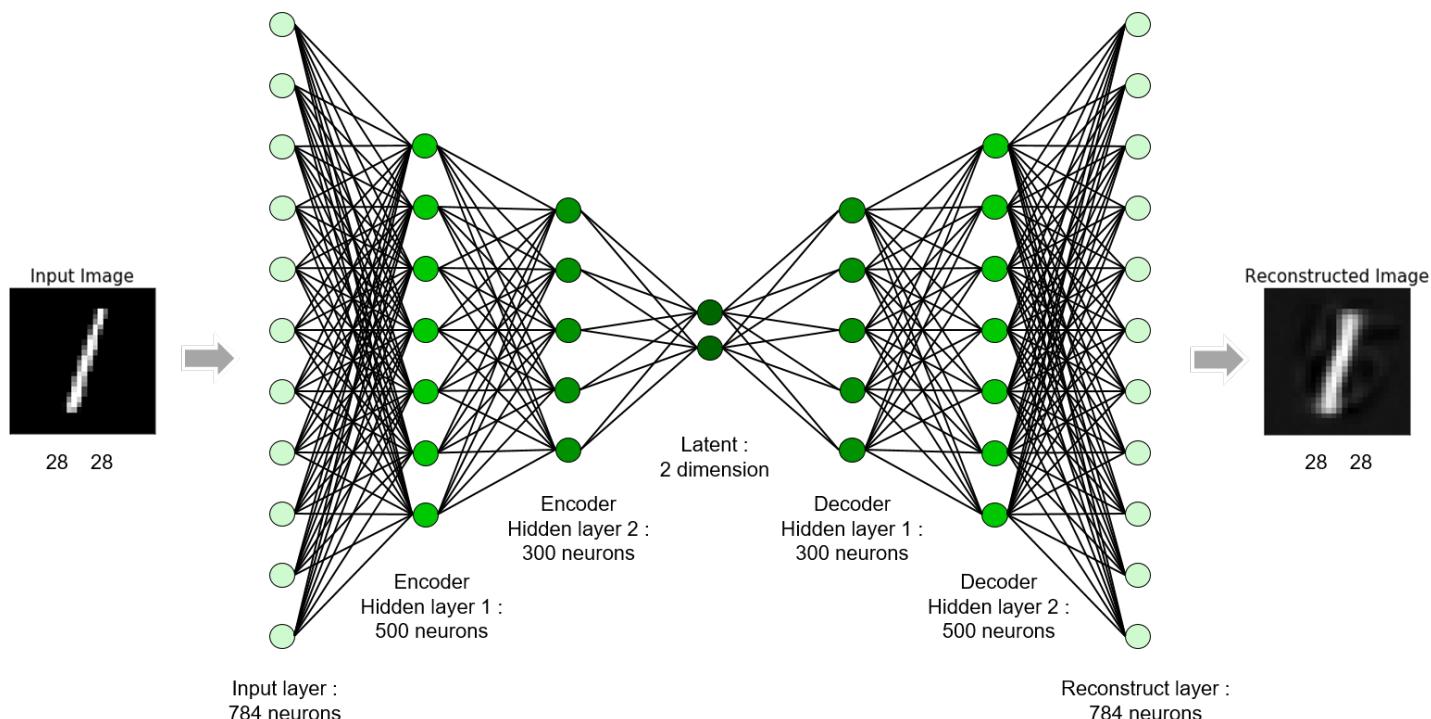




Group Exercise

Consider unlabeled training data $\mathbf{x}_1, \dots, \mathbf{x}_n \in \mathbb{R}^d$, and let $f_{\theta}(\mathbf{x}) \in \mathbb{R}^d$ denote the output of the autoencoder, with weights θ .

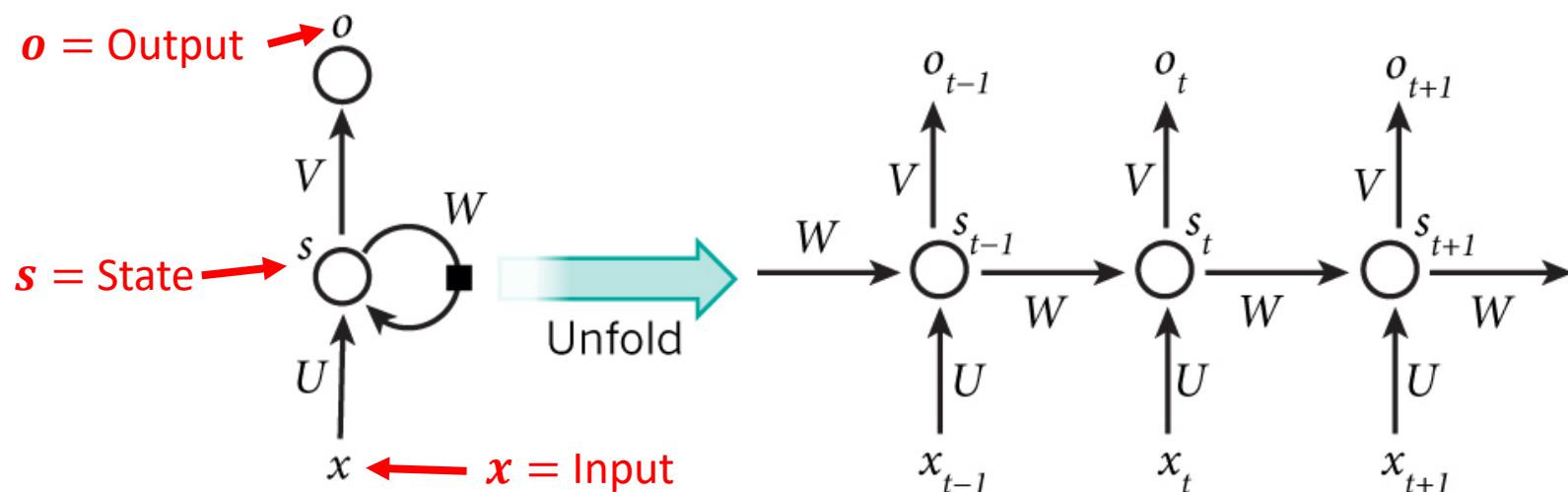
1. Propose an objective function for learning θ .





Recurrent Neural Networks

- For making sequential predictions, where next prediction depends on previous ones
- Uses: speech-to-text, machine translation
- Idea:
 - The same task is being performed repeatedly
 - Therefore the same predictor should be used at each step
 - The state variable encodes recent history (“memory”)

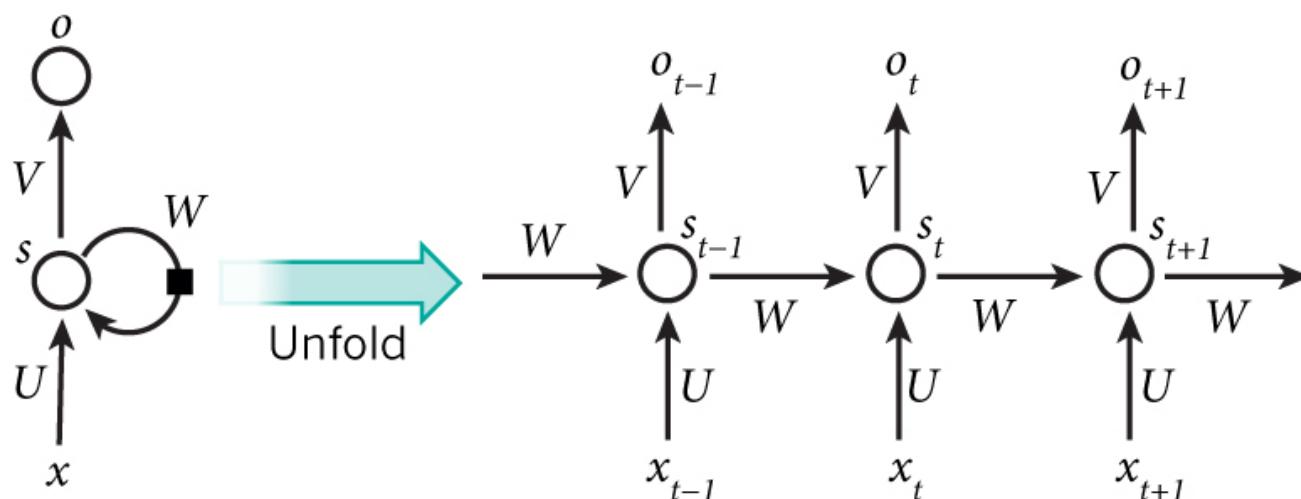




Recurrent Neural Networks

- **Training:** Use same weights U, V, W at each time step, since it's the same task being performed repeatedly
- This is a type of neural network where the output is related to the input by

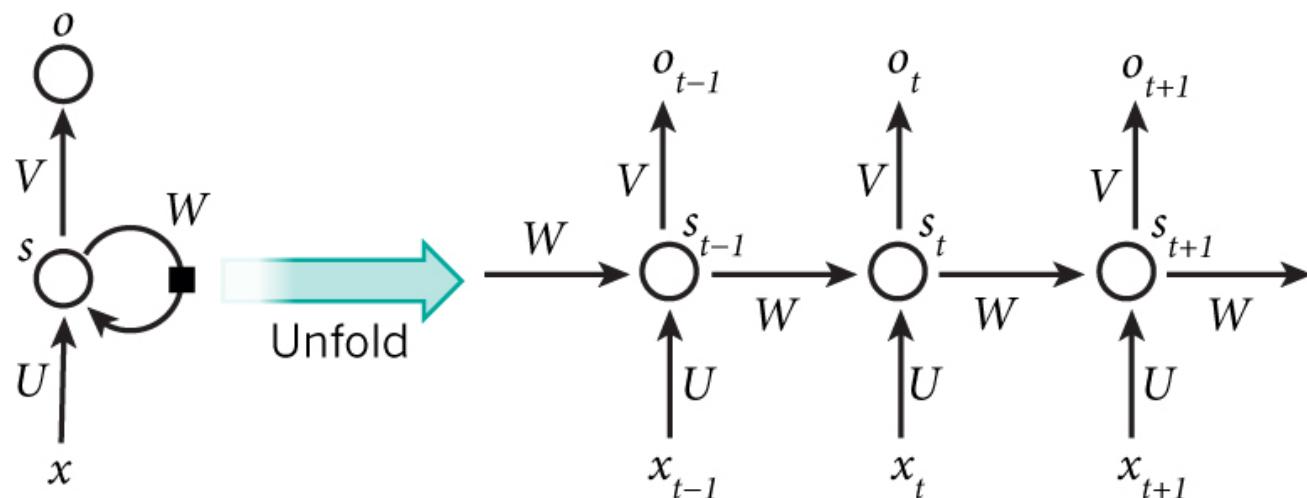
$$\begin{aligned}\mathbf{o}_t &= V\mathbf{s}_t \text{ (regression),} \\ \mathbf{o}_t &= \text{softmax}(V\mathbf{s}_t) \text{ (class.)} \\ \mathbf{s}_t &= \sigma(U\mathbf{x}_t + W\mathbf{s}_{t-1})\end{aligned}$$





RNNs: Practical Matters

- **Training:** Backpropagation on the unfolded network
- Building in longer time dependencies make RNNs hard to train
- Long short term memory (LSTM) networks: versions of RNNs that have longer-term memory that can be trained fairly efficiently
 - Overtaken recently by transformers and attention mechanisms
- Used in Google Translate, Apple and Google speech recognition, auto-replies, chatbots, etc.





Large Neural Networks (Deep Learning)

- CNNs have been around since the 1990s, autoencoders and RNNs since the 1980s,
- For many years, neural networks were not considered state of the art for many prediction tasks
- Larger neural networks needed for greater accuracy, but they can be difficult to train
 - Vanishing gradients problem (large gradients at output layer are reduced to small gradients at early layers)
 - More (bad) local minima, backprop gets stuck
 - Not enough data to prevent overfitting
- Then neural networks experience a renaissance that began around 2006



Deep Learning

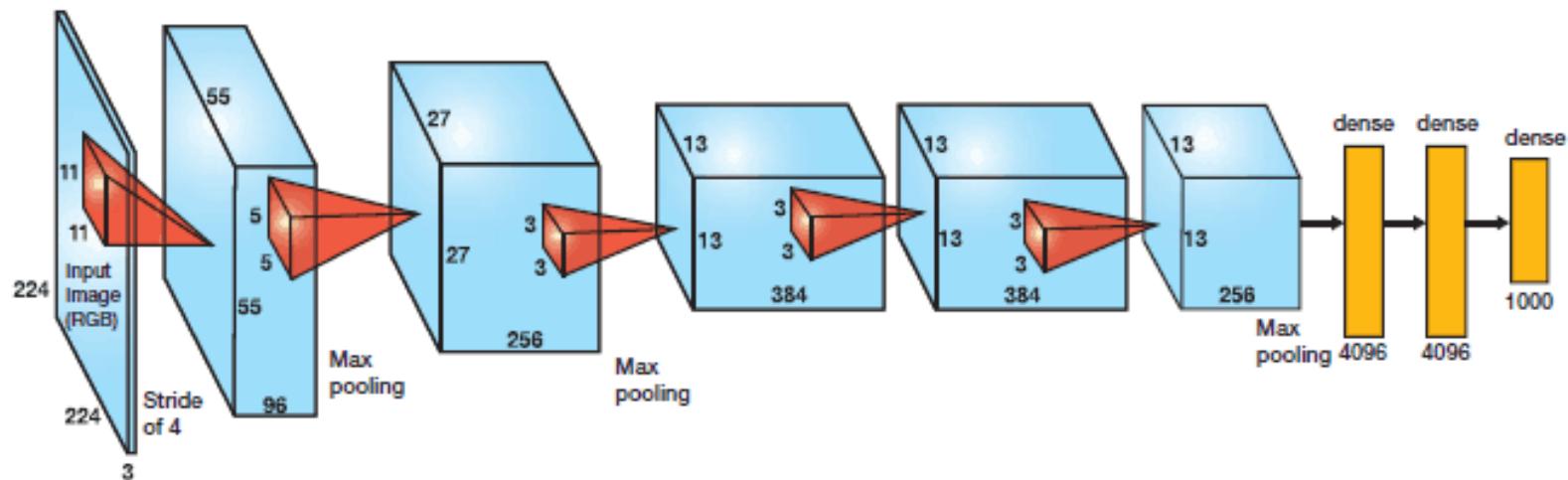
Enabled by

- Graphics processing units (GPUs): parallel floating-point calculators with 100s-1000s of cores
- Large, public databases such as ImageNet (Fei Fei Li, 2009) which has over 14 million *labeled* examples and 20 thousand classes of objects
- New training strategies
 - Dropout
 - Greedy layerwise training
 - Modifications of SGD (e.g., Adam)
 - Batch normalization
- Rectified linear units (helps with vanishing gradient problem)
- Improved architectures



AlexNet

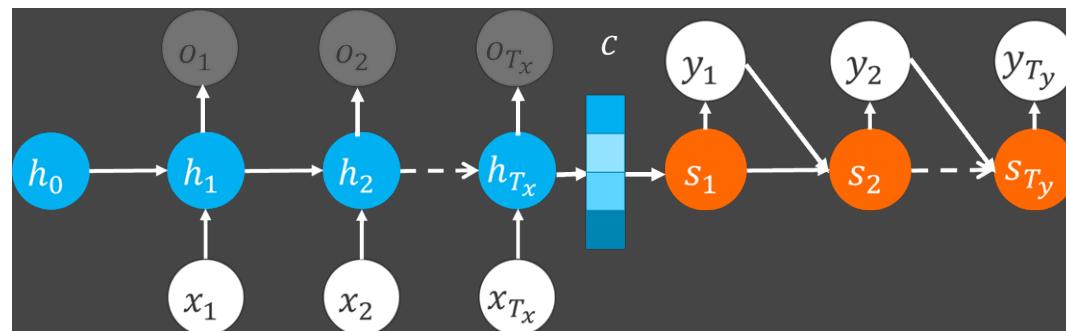
- The big breakthrough
- Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton (2012)
- Reduced error rate on ImageNet from 26% to 16%
- Used GPUs, dropout, ReLU, which have since become standard





Attention

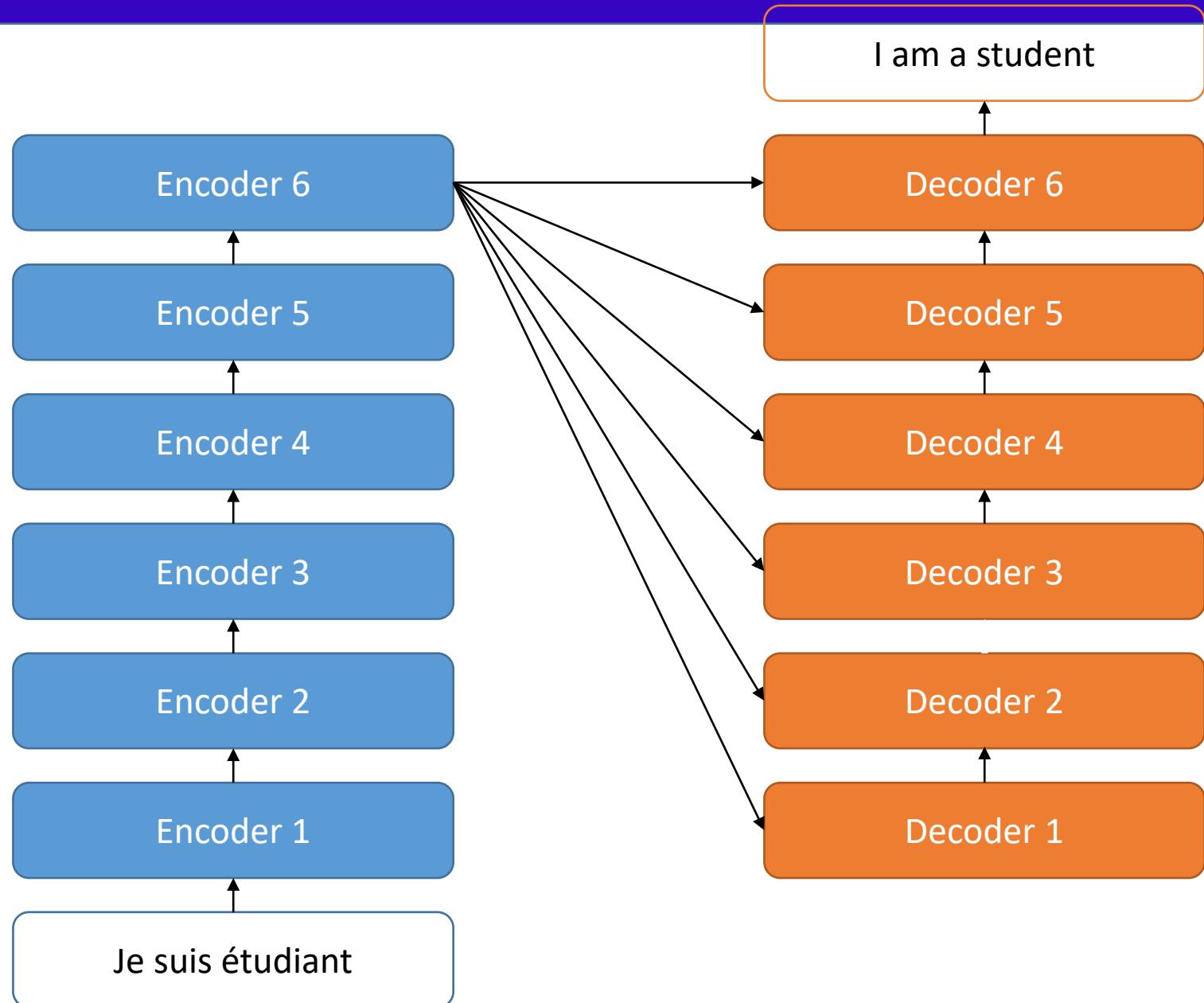
“Weil der Hund nicht satt war, hat er den Schüssel geleckt



“The dog licked the bowl because it was not full.”



Transformers





BERT (2019)

- BERT-large consists of 24 encoding layers (no separate decoder), each with 16 attention heads. Each hidden layer consists of 1024 nodes. 345M total parameters.
- Uses Next Sentence Prediction and Masked Language Modeling.
- Can be used for many downstream tasks:
 - Single sequence classification
 - Sentence Pair Classification
 - Question Answering
 - Single sentence tagging
- Trained on 3 billion words.



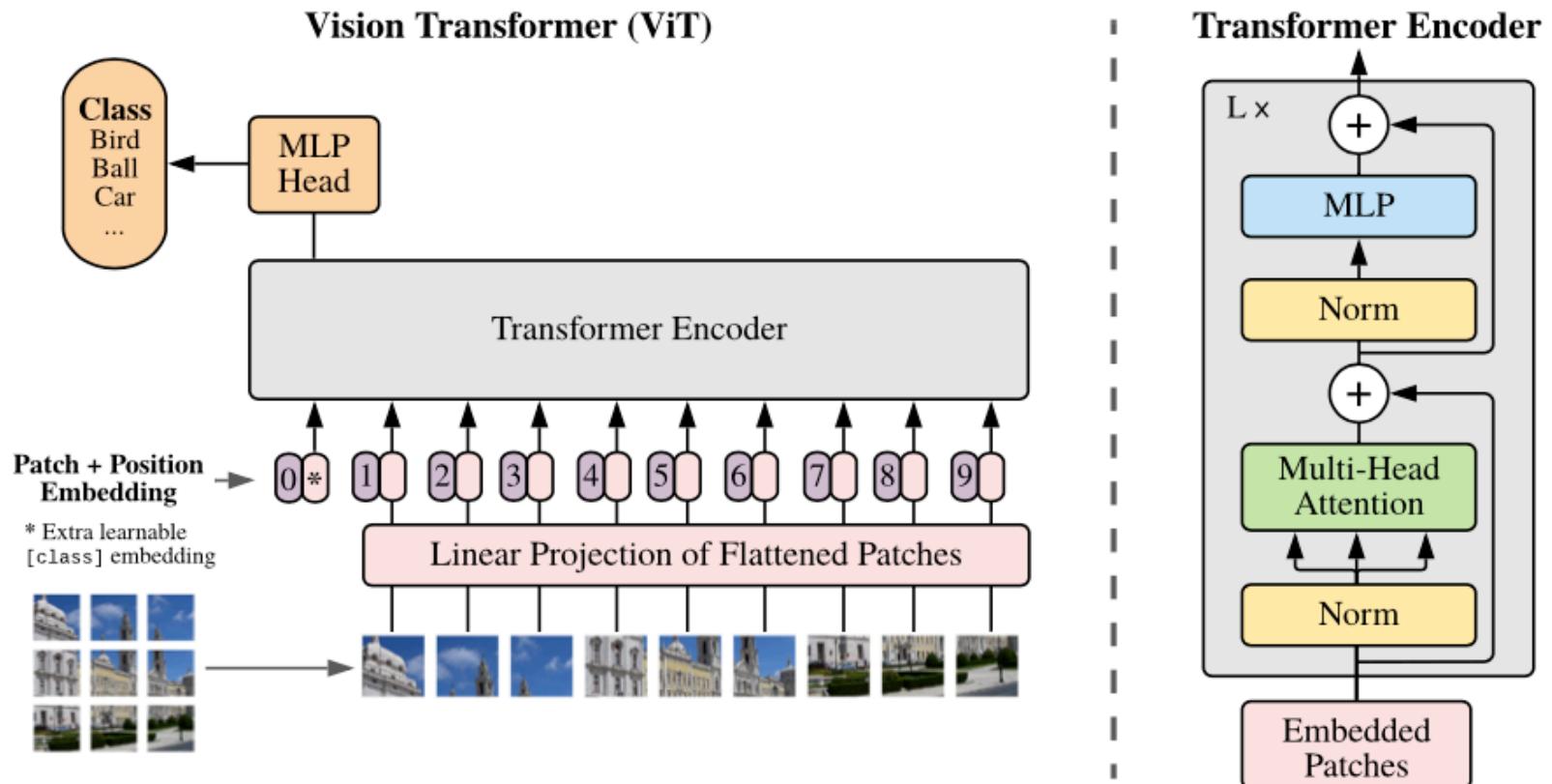
OpenAI GPT-3 (2020)

- Uses 96 attention blocks each with 96 attention heads with 128 nodes in each hidden layer. 175B parameters total.
- Cost to train would be \$4.6M on a V100.
- Performs next word prediction.
- Achieves SOTA results on many zero-shot tasks.
- Trained on CommonCrawl, Webtext2, Books1, Books2, and Wikipedia

Brown, Tom B., et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165



Vision Transformer (2020)



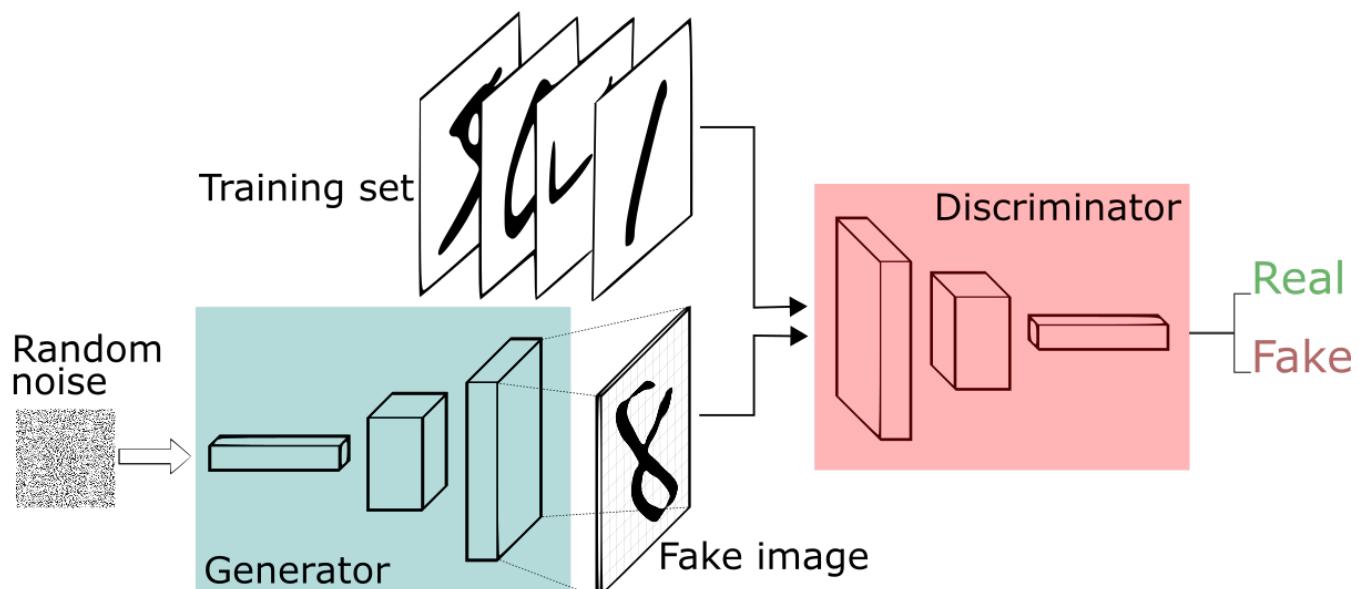
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby



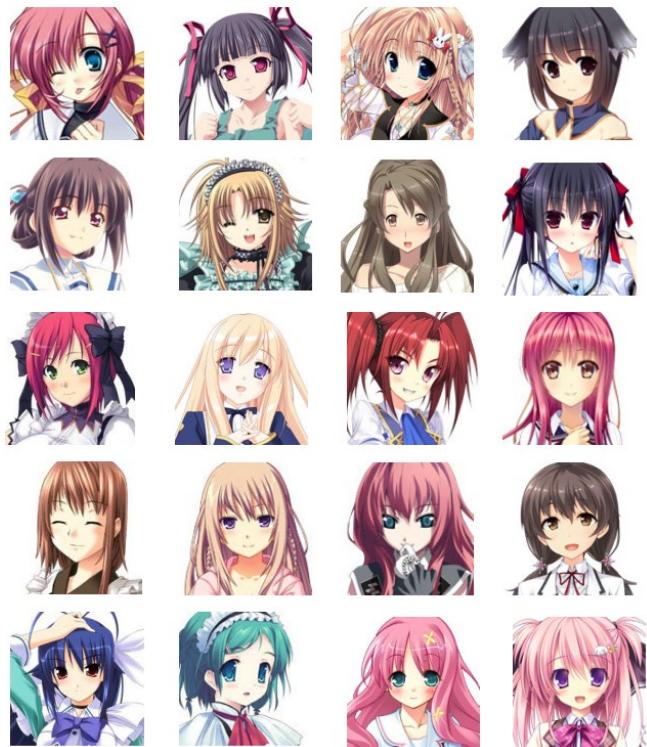
Generative Adversarial Networks (GANs)

- Used to simulate new realizations of a class of objects
- Discriminator network and generator network compete against each other
- Train by alternating optimization of classification empirical risk (fix one network while optimizing the other)





GAN for Anime Character Generation



Real training images

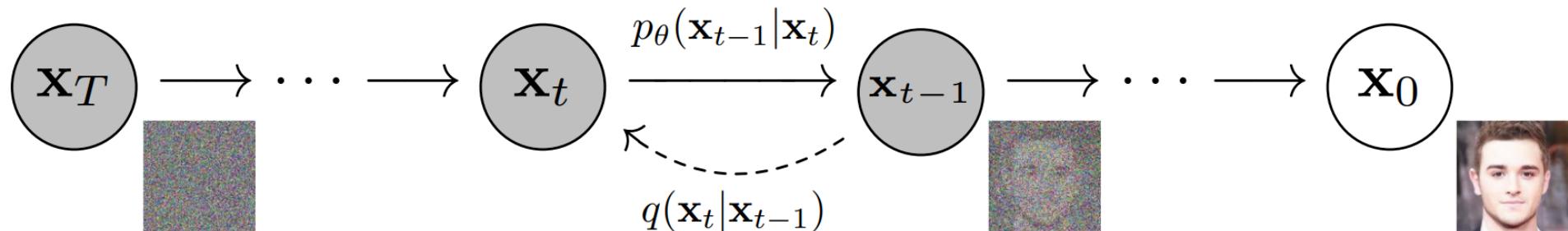


Generated images



Diffusion Models

- Rather than learn the structure in an unstable fashion via a GAN, Diffusion learns how to denoise an image into something meaningful.
- Trained by feeding in noisy images at different levels of noising and then must predict the structure of the noise at that step.
- The noise can then be subtracted from the image and the process repeats until all noise has been “removed” from the image.
- Allows the structure to be defined early on and details can be refined as the process proceeds.





DALL-E 2



A painting of a husky Labrador mix in the style of Alphonse Mucha.

<https://openai.com/dall-e-2/>

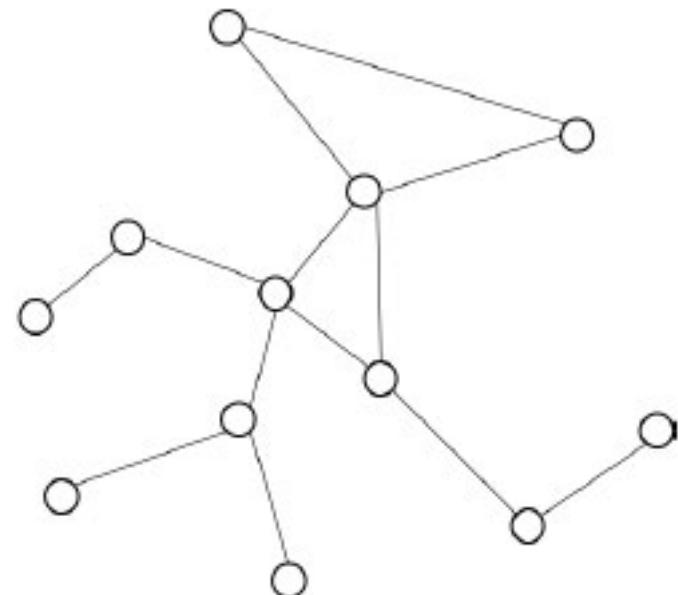


A snowy landscape in Southern Utah with a university sitting at the mouth of a canyon, dslr, 35mm.



Graph Neural Networks

- Can we extend convolution to other data types?
- What about graphs?
 - Images are a grid graph



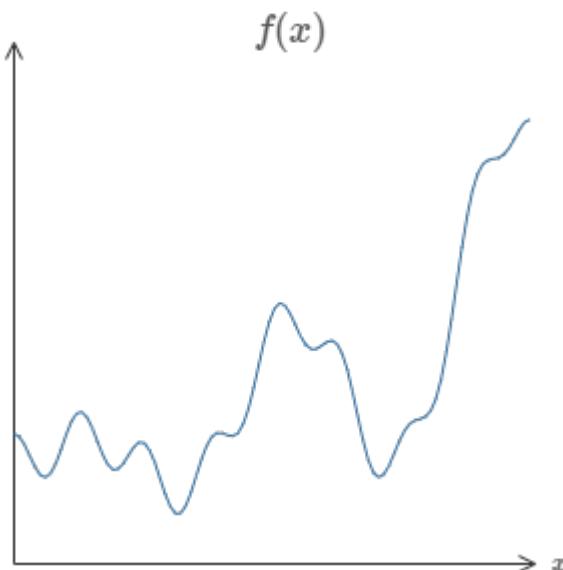
- Yes!



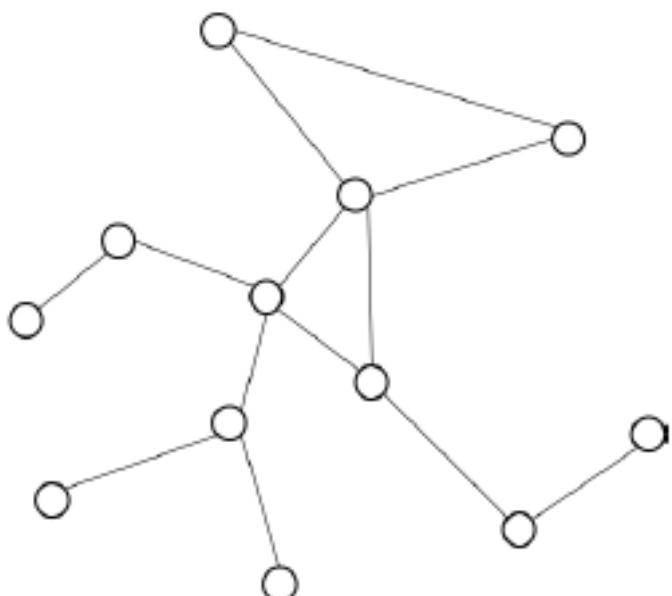
Graphs



- This is not the kind of graph we're talking about:



- This is a graph:

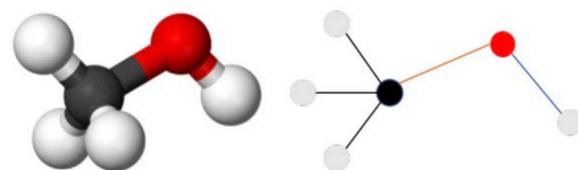
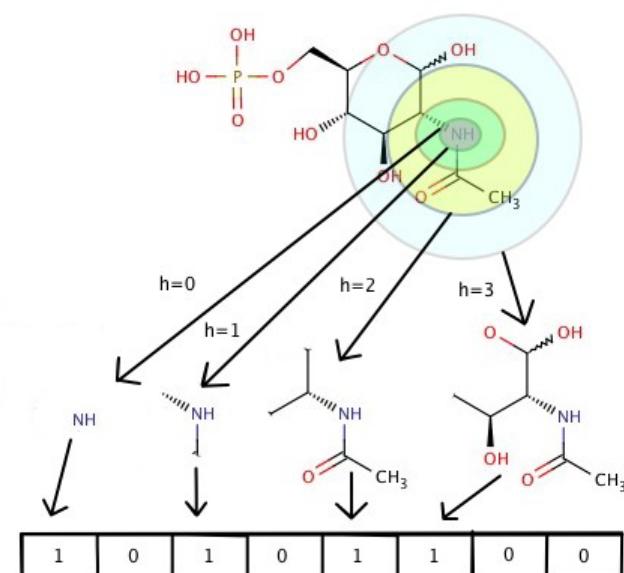


- This is a “plot”



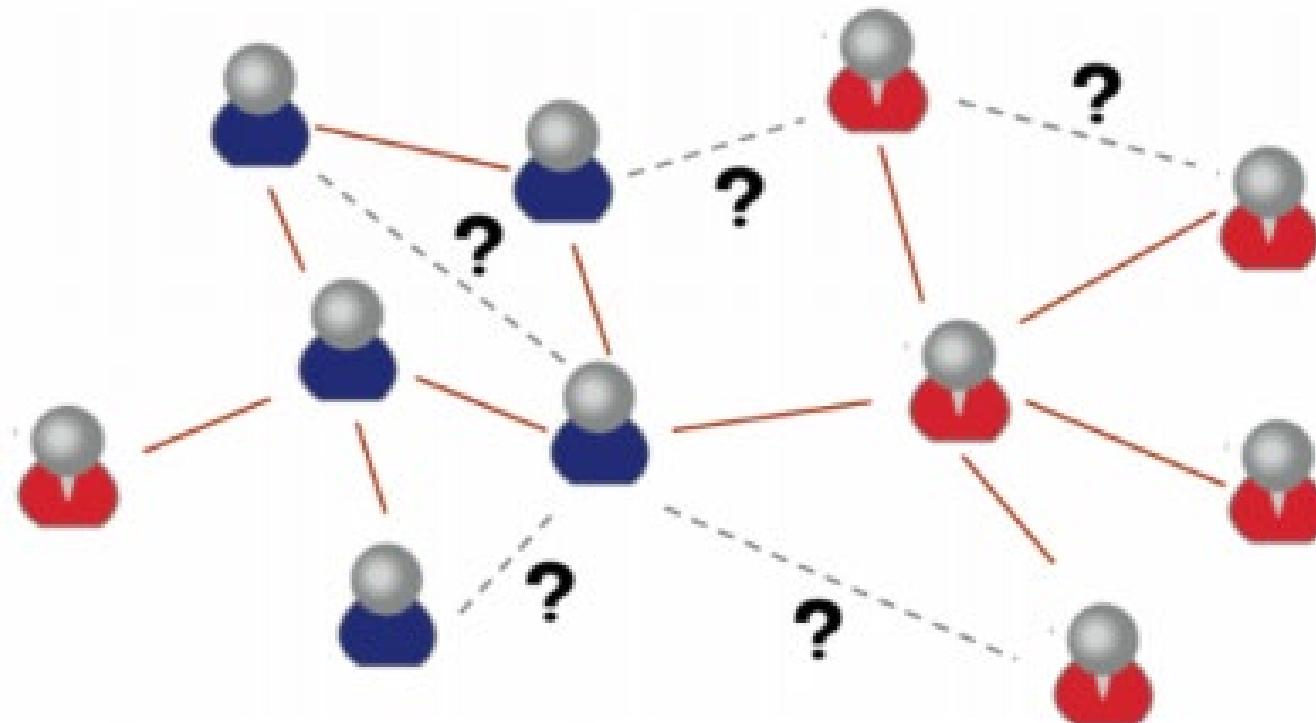
Applications

- Predicting molecular properties by encoding molecules as graphs



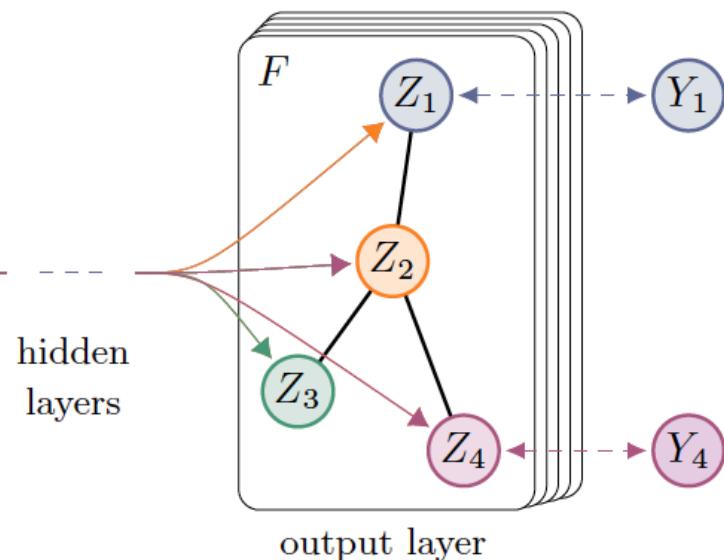
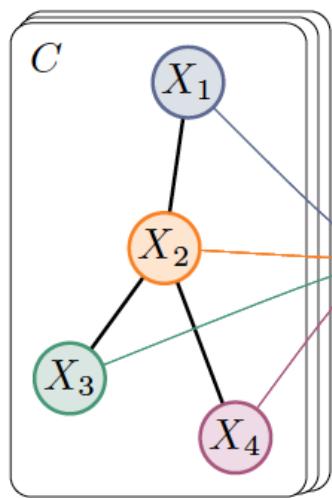


Social networks, link prediction

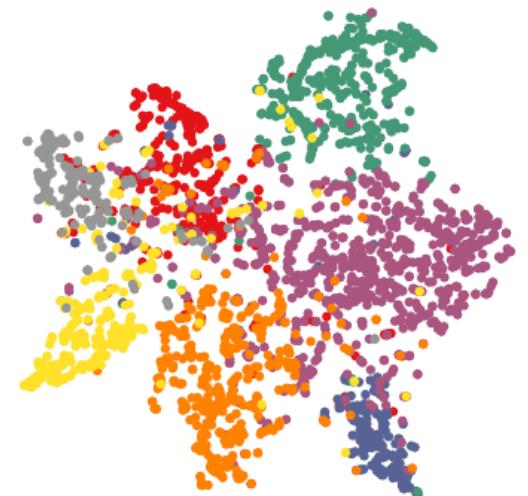




Citation Network, Graph Classification



(a) Graph Convolutional Network



(b) Hidden layer activations



Deep Learning in Practice

- Many existing platforms: Tensorflow, Theano, Caffe, PyTorch, Keras
...
- Rather than designing your own architecture, just using an existing architecture with pretrained weights, and update the weights for your data set.
- (Deep) neural networks are most successful when the architecture is tailored to structure inherent in the data, e.g., spatial structure (CNNs), temporal structure (RNNs).
- There is little theory to explain the success of deep learning (although people are working on it). It is largely an engineering achievement.



Further reading

- <http://neuralnetworksanddeeplearning.com/>
- <https://www.deeplearningbook.org/>
- STAT 5685/6685 (Spring 2025)
- ESL Chapter 11