Intro to Artificial Neural Networks (ANNs); Intro to Deep Learning

Week	TUES. Week 10
Assignment Due	
Assignment Done	☐
Due Date	
Notes Done	☑

Presentation

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/0a1fa7ac-891b-4bd5-91ad-40d0261b5b1c/L1-DeepLearningIntro.pdf

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/366aa92c-18a3-44da-9ccd-50fd852293f9/L2-Intro_ANNs.pdf

Questions about Homework

Is there a reason for selecting preexisting

- then the observation will always belong to that cluster

If randomly selecting an observation is a bad idea, how would you create a reference vector in the first place?

- One idea may be kinda like with hierarchical clustering tress:

    - Do bottoms-up merging till level k then look at groups made at level k and initialize reference vectors to be mean of groups created by merging trees process
- Another idea would be to look at the distribution of observations in the feature space and uniformly select locations in a certain space.

Class Notes

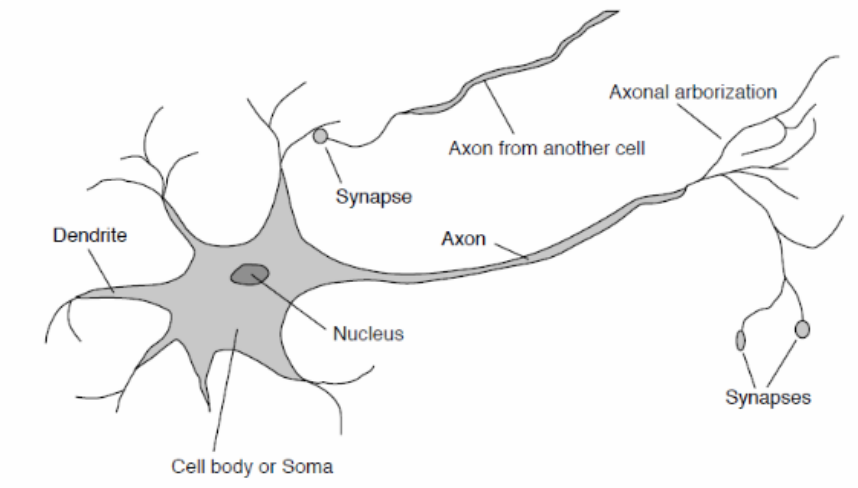
Final exam is in Disque, Room 103

- The exam *is* cumulative in nature → profs are creating exam still

Introduction to Artificial Neural Networks

Logistic regression can be thought of as the simplest form of an **artificial neural network**:

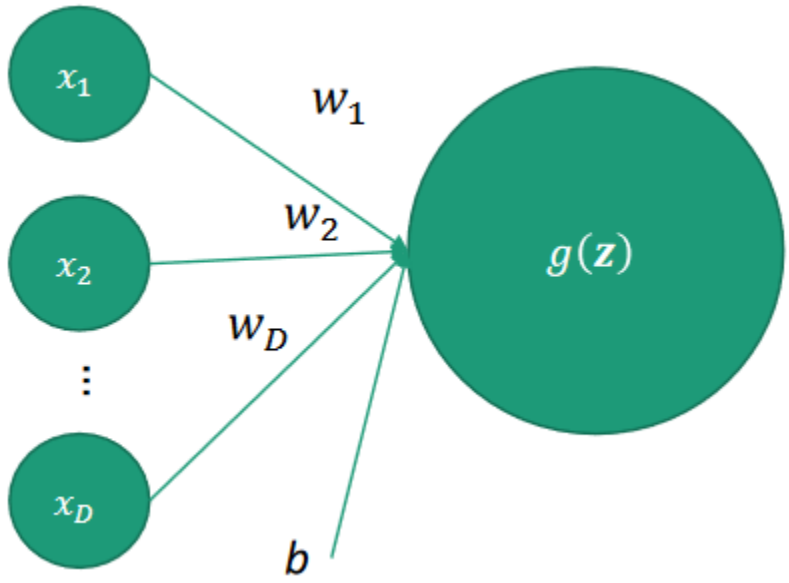
- With ANN, in a way, we're chaining logistic regression units together.
- The term **artificial neural network** comes from idea of the process of making a decision involves making smaller simple decisions.
    - Idea behind learning: Receive a stimulus, in response we sent a signal that is combined from simple processing units that are applied → from vast numbers of neurons that are interconnected
    - Is somewhat biologically inspired



In its simplest form, an **ANN** is a sequence of systems that computer a weighted sum of inputs ($z = xw + b$) and applied some function to that weighted sum $g(z)$

- Think of neuron taking in several signals and combines them, doing simple processing of the new value and sending that out to other connected things.
- From a machine learning standpoint, we can think of logistic regression as the simplest form of ANN.
- We take our input data, send it down (think of as applying a weight) the branches of a neuron, adding in a bias, where we have a computed value and then apply some sort of simple function to that computed value.
    - In this case of logistic regression, we took the combined weighted signal and applied the sigmoid function.
- With ANN, the simple functions applied to the weighted signal are referred to as **activation functions** → several options exist here
    - ***Logistic regression*** uses the ***logistic sigmoid activation function***.
- Also commonly used in ANN is a ***hyperbolic tangent activation function***, which is similar to logistic sigmoid, but takes an input and outputs a value -1 to 1.
- Another one is referred to as real-loom, existing as a piecewise function:
    - Anything greater than 1 is given the actual value that it is
    - Anything less than 1 is given the value of 0.

While ultimately, we're taking our incoming data, weighting it, adding bias, and applying an activation function, with ANN, we think of it graphically as a ***directed graph***:

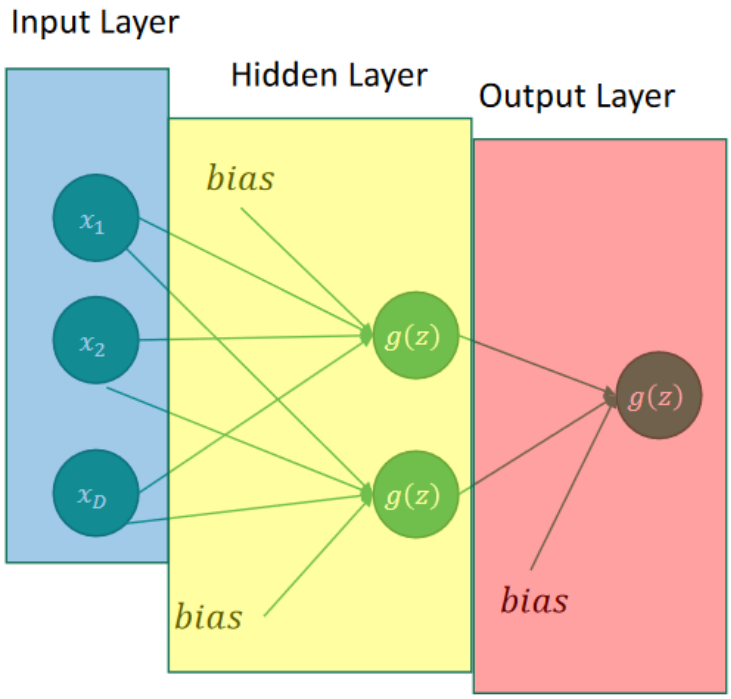


- All inputs can be considered nodes and weights are edges s.t. $xw + b$ is computed by weighting the inputs and adding a bias value.
  - That weighted value is then processed by the activation function, $g(z)$
  - Often will see individual features are nodes and weights are edges.
- We have often seen weights have typically been a column vector.
  - With ANN, we're allowed to have more than one output, meaning our weight matrix can have more than one column, which gives us multiple output values.
  - $W = \begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$

All we did in previous slide is frame logistic and linear regression is frame them in framework of ANN with features being nodes, weights being edges, which gives us an output:

- To have an ***ANN***, we must stack these regressions on top of one another.

- In traditional shallow artificial neural network, we have 3 layers:



  - **Input layer** → simply the original features
  - **Hidden layer** → does weighted sum of input features and applies an activation function to give us an output
  - **Output layer** → weighs output of hidden layer and adds those weighted outputs of the hidden layer and processes it through a final activation function to give us our final output $\hat{y}$
    - The activation function chosen at the end for the output is dependent on what we want to do:
      - With regression, its common to use a linear activation function.
      - With binary classification, its common to have the logistic sigmoid activation function at end.
      - With multiclass classification, its common to have a softmax activation function at the end.
      - Most common: tan H and relu are most common captivation functions.

The motivation to go from simple logistic and linear regression to ANN is wanting the ability to model complexity of a system:

- Remember: There is a real equation that, when given an observation, can give us the target value we're looking for.
  - However, the crux of the problem is that we don't know it.
    - To approximate it, we need a sufficiently complex model, which singular/simple logistic/linear regression may not not be enough for.
  - It may be enough to give a linear######

In this formulation, we can let $W^{(h)}$ and $b^{(h)}$ be the weights and biases of the hidden layer.

- To ultimately get to our value $\hat{y}$, we can take our input and weight them by the hidden input weights and offset them by our hidden input biases.
  - Then we can apply our activation function.
    - ############

Numeric Example

Imagine we have our data:

$$X = \begin{bmatrix} 1 & 2 & 0 \\ -1 & 4 & 2 \end{bmatrix}, W^{(h)} = \begin{bmatrix} 1 & -1 \\ -1 & -1 \\ 2 & 1 \end{bmatrix}, B^{(h)} = [-3, 4]$$

Our activation function is: $g(z) = \begin{cases} 0 & z \leq 0 \\ z & z > 0 \end{cases}$

The result from our output layer ends up being $Xw^{(h)} + b^{(h)} = \begin{bmatrix} -4 & 1 \\ -4 & 3 \end{bmatrix}$

In applying our activation function, we get: $g(Xw^{(h)} + b^{(h)}) = \begin{bmatrix} 0 & 1 \\ 0 & 3 \end{bmatrix}$

Now, we take the result from the hidden layer and apply our weights and biases for the output layer accordingly, then applying that activation function for the output layer, where the weights, biases, and activation functions are:

$$W^{(o)} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, b^{(h)} = [2], g^{(o)} = \frac{1}{1+e^{-z}}$$

The result from our output layer, upon applying the weights and biases, ends up being

$$Hw^{(o)} + b^{(o)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

In applying our activation function, we get: $g(Xw^{(h)} + b^{(h)}) = \begin{bmatrix} \frac{1}{1+e^{-1}} \\ \frac{1}{1+e^{1}} \end{bmatrix}$

Learning an ANN

The difficulty we have with ANNs is determining our optimal weights to make our decisions to figure out, as we already have done with linear and logistic regression.

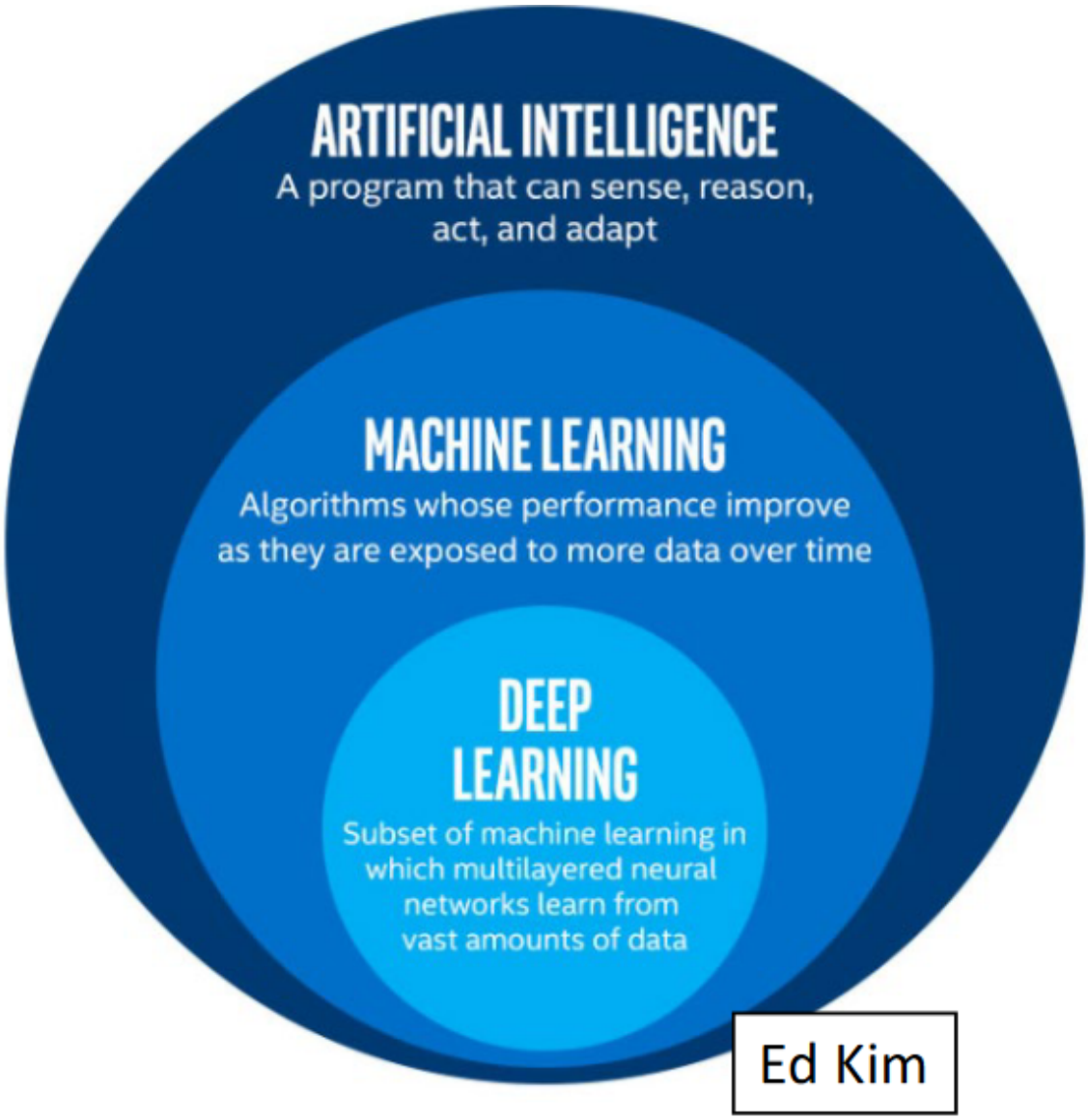
- Before, with linear/logistic regression, we arrived at initial weights (through randomization), push data to get $\hat{y}$ values and evaluated by an objective function.
    - At some times when we couldn't get a direct solution, we would compute the gradient of our objective function based on our estimate and use this to determine the partial derivatives of the different weights and biases that we're searching for.
    - We begin with $\frac{\partial J}{\partial \hat{y}}$ at the output of putting our initial weights through the system and push it backwards to then update our estimates of the weights and biases.
        - Technically we did this with logistic regression assignment, but complicates as we have more and more layers to consider.

The details of forward/backwards propagation and ability to generalize, modularize, abstract all types of complex architectures is explored in CS615 course.

Introduction to Deep Learning

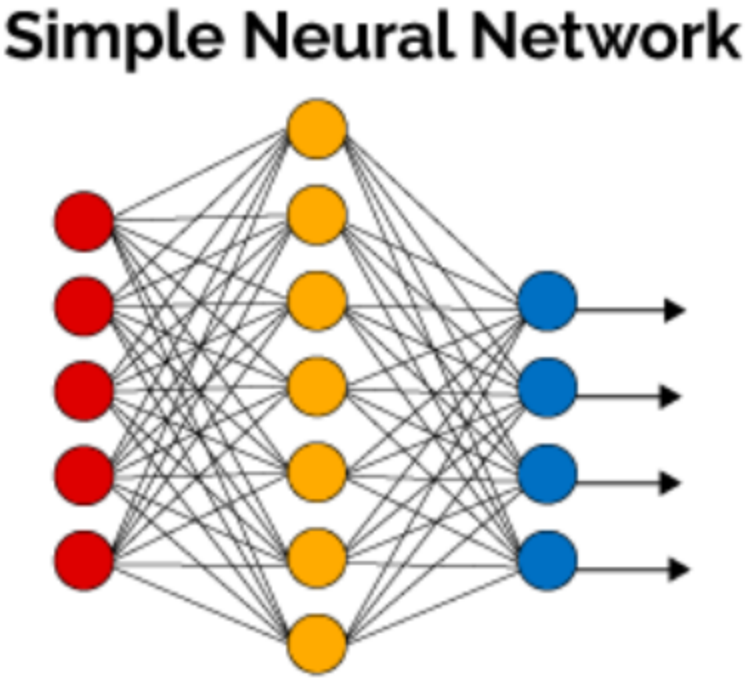
Most traditional machine learning algorithms (stuff we've explored in this course) tend to do best with a *good representation* of the data (b/c of human design of input features);

- Problem with this: ML is used in several domains s.t. they *each* must determine the best representation of the input of the ML algorithms, which is costly.
    - Idea: Why can't we feed data in whatever raw form its in and allow machine learning algorithms figure out how to extract a meaningful representation to then use as input for traditional ML algorithms?
- Early stages of deep learning are involved with how to extract useful representations for later use in traditional machine learning.



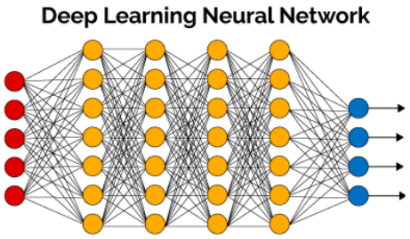
A simple ANN is actually a bit of a deep network.

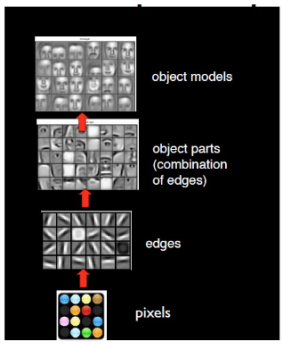
- You can think of the output of the hidden layer as an intermediate representation, which of course, is not that deep.

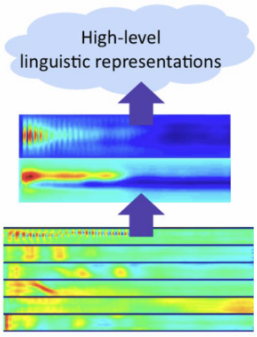


Idea: Perhaps the middle stages of a larger neural network is learning some relevant representation s.t. we get outputs that can be used as inputs for multiclass classification:

- Ultimately utilizing a useful intermediate representation from the raw data
    - Example: Imagine trying to do facial recognition
        - At the raw level, we have pixels → what can we form out of pixels?
            - Maybe we can form edges → these edges could form part of a face
                - From parts of a face, we could possibly form an entire face, as seen below.







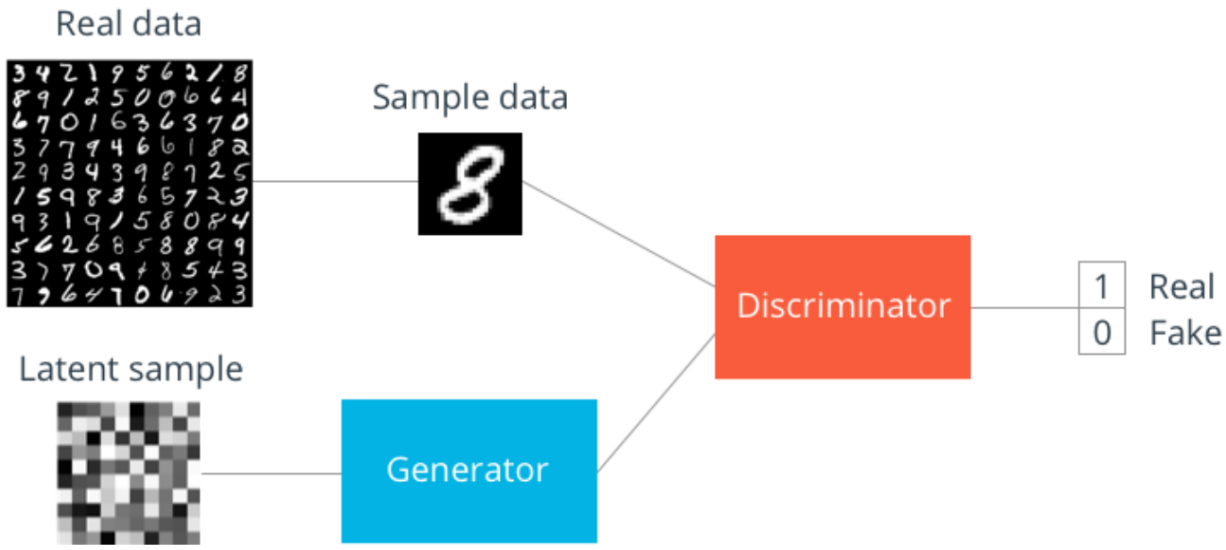
Additional Applications

Some types of deep learning that have seen success are:

- **Generative networks (GNs)** - have historical data and want network to generate new data that looks like real data
    - Ex: ##
    - Ideas that generated buzz before were **general adversarial networks (GANs)**.
- For image classification, the idea came for **convolutional neural networks (CNNs)**, specifically for image classification.
- For temporally ordered data, as in a sequence, **recurrent neural networks (RNNs)** been popular.

Generative Adversarial Networks (GANs)

With a GAN, we have general data and a discriminative network that tells us whether an object is real/fake.

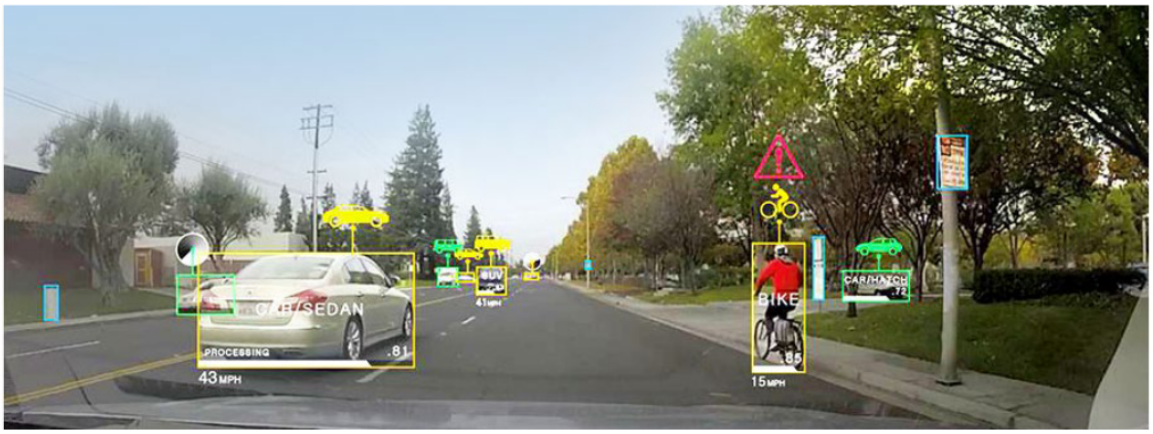


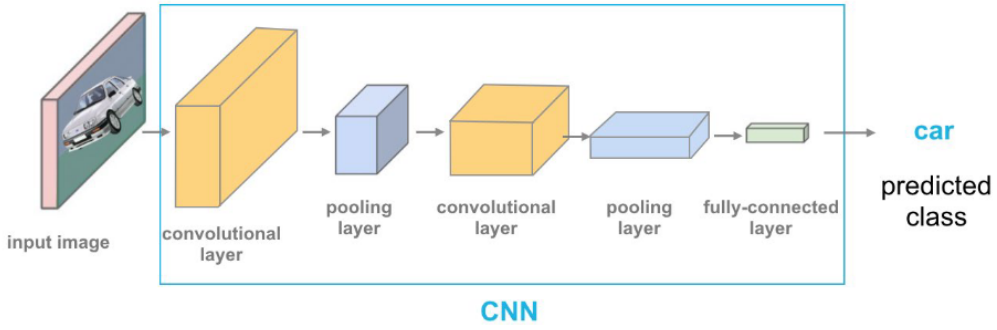
- We then have a generative network s.t. we give it random data and its goal/job is to confuse the discriminator.
    - We send the discriminator a bunch of real/fake data (where it knows the answer) and it'll compare what it thinks is the answer and what the actual answer is.
    - Upon the discriminator updating its network, the discriminator becomes better at detecting when things are fake.
        - This data on accuracy is then fed to the generator additionally to help the generator generate better fake data.
- Ex: thispersondoesnotexist.com

Convolutional Neural Networks (CNNs)

With a CNN, it borrows from computer vision:

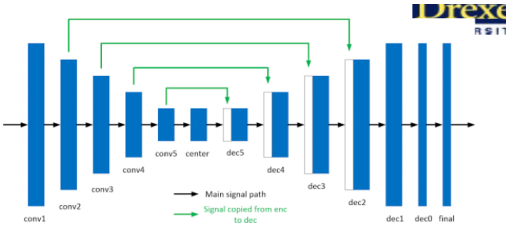
- The network is attempting to learn kernels s.t. when they're convoluted with an image, we get a response of what things are where (where things are located).





- CNNs have been used for superresolution, the process of taking a lower revolution image and generating a higher resolution image by determining intelligently what values should be where.

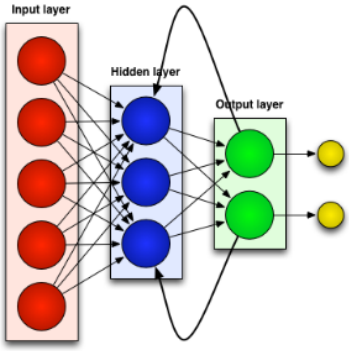
Adobe Super Resolution

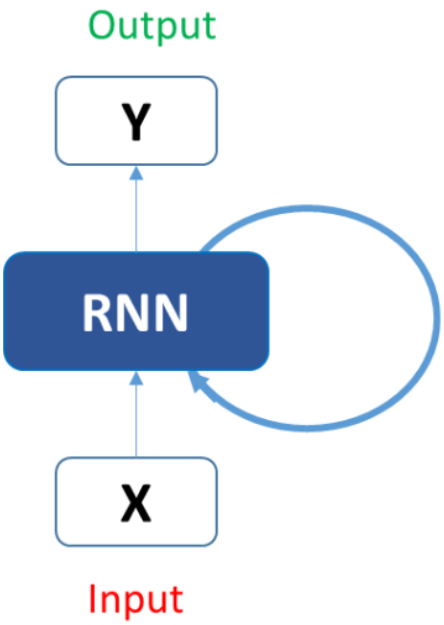


Ed Kim

Recurrent Neural Network (RNNs)

With a recurrent neural network (RNN), instead of having a feed-forward network, we get feedback:



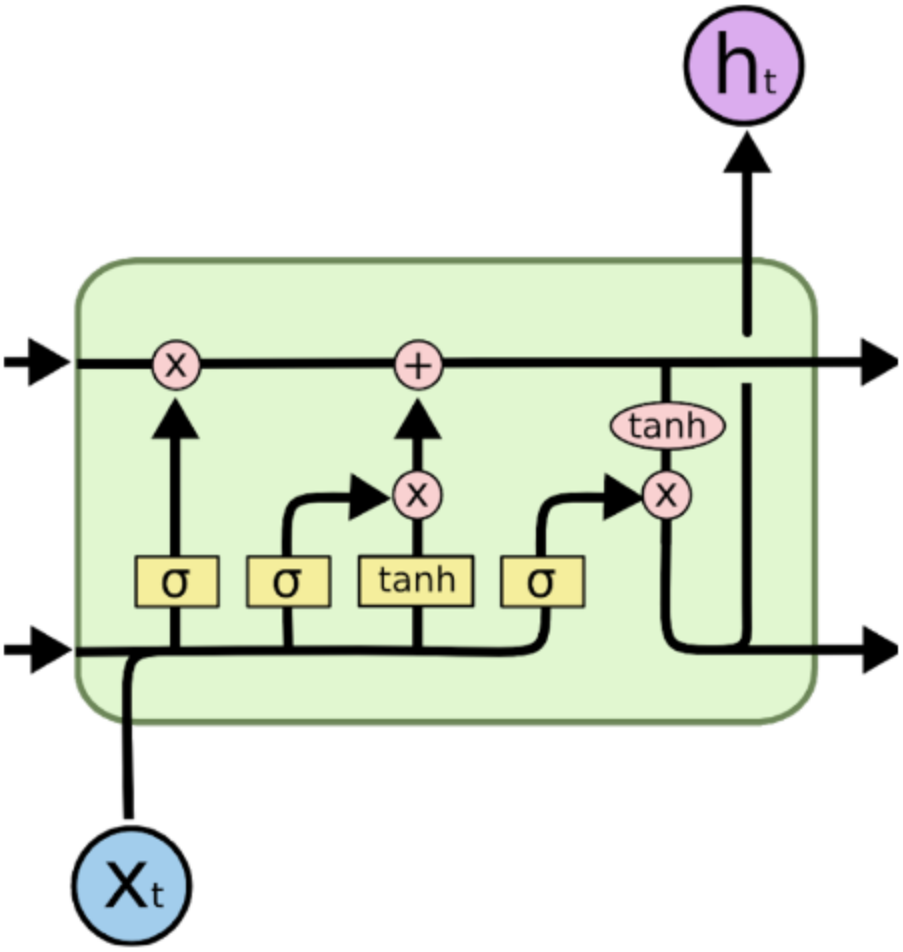


- In this network type, what we generate at time t is dependent on what was calculated at $t-1$.
  - Effectively, the past is relevant to the present.
- In this network, temporal feedback is used for several applications.
  - Ex: Watch a video, by watching it, we can possibly classify it.
    - Possibly for generative purposes → given code, a network could possibly be able to generate code from having seen previous examples of code
- Ex: Code generator
  - At 10 iterations, the network likes whitespace, the “function” and “end” keywords and variable names have “dY” in them since the code has many derivatives.
    - As more iterations are performed, the network learns more, learning to tab better and ending lines with semicolons.

Long Short-Term Memory (LSTM) Networks

Newer versions of **recurrent neural networks (RNNs)** are **long short-term memory networks (LSTMs)**, which attempt to overcome issue of generic RNNs.

- LSTM overcomes difficulties with “cell state” in determining what to remember and what to forget so it can hopefully remember stuff from the past to make a relevant decision for the now.



Summary

Overall, the general idea of deep learning is to try to have network layers come up with a useful representation of the data.