PCSE 595
Special Topics in Machine Learning

Dr. Sam Henry

samuel.henry@cnu.edu

Luter 325

Office Hours

- Please stop by!
- This is an **Open Question Answering Time**

Monday, Wednesday, Friday

11:00-12:00 , 1:00-1:30

Or by appointment

Office hours are in-person (just come by my office, LUTR 325)

Pizza My Mind

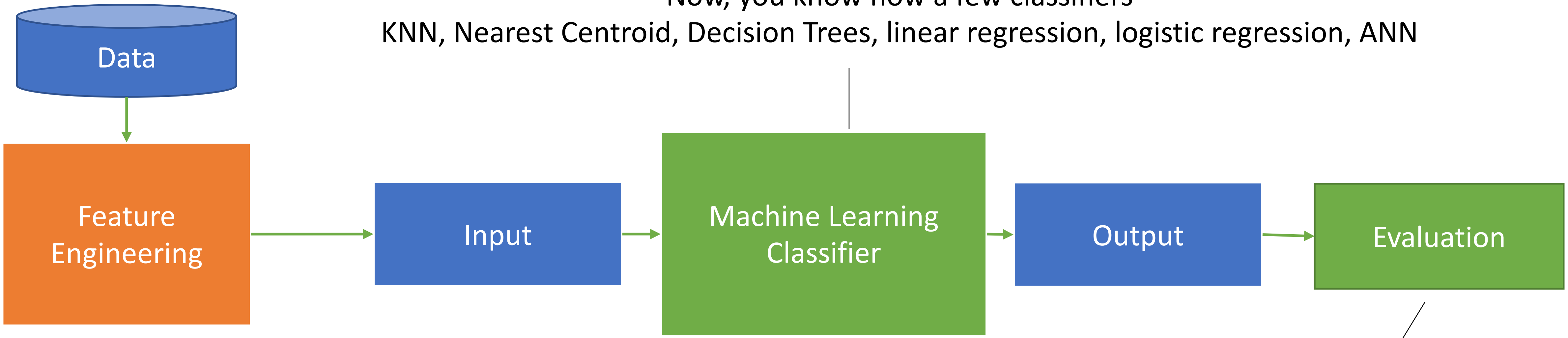
- You can get extra credit
  - Up to two extra points on your final grade for one PCSE course
- Attend them! They’re fun, informative, and employers present
  - Don’t wait until you need a job or internship, go now!
- Thursdays at 12:20

… and you get free pizza!!

Students in PCSE classes can get extra credit if they attend at least 10 events. 10-11 events: 1 extra point; 12-13 events: 2 extra points.

A simple machine learning pipeline

Now, you know how a few classifiers
KNN, Nearest Centroid, Decision Trees, linear regression, logistic regression, ANN

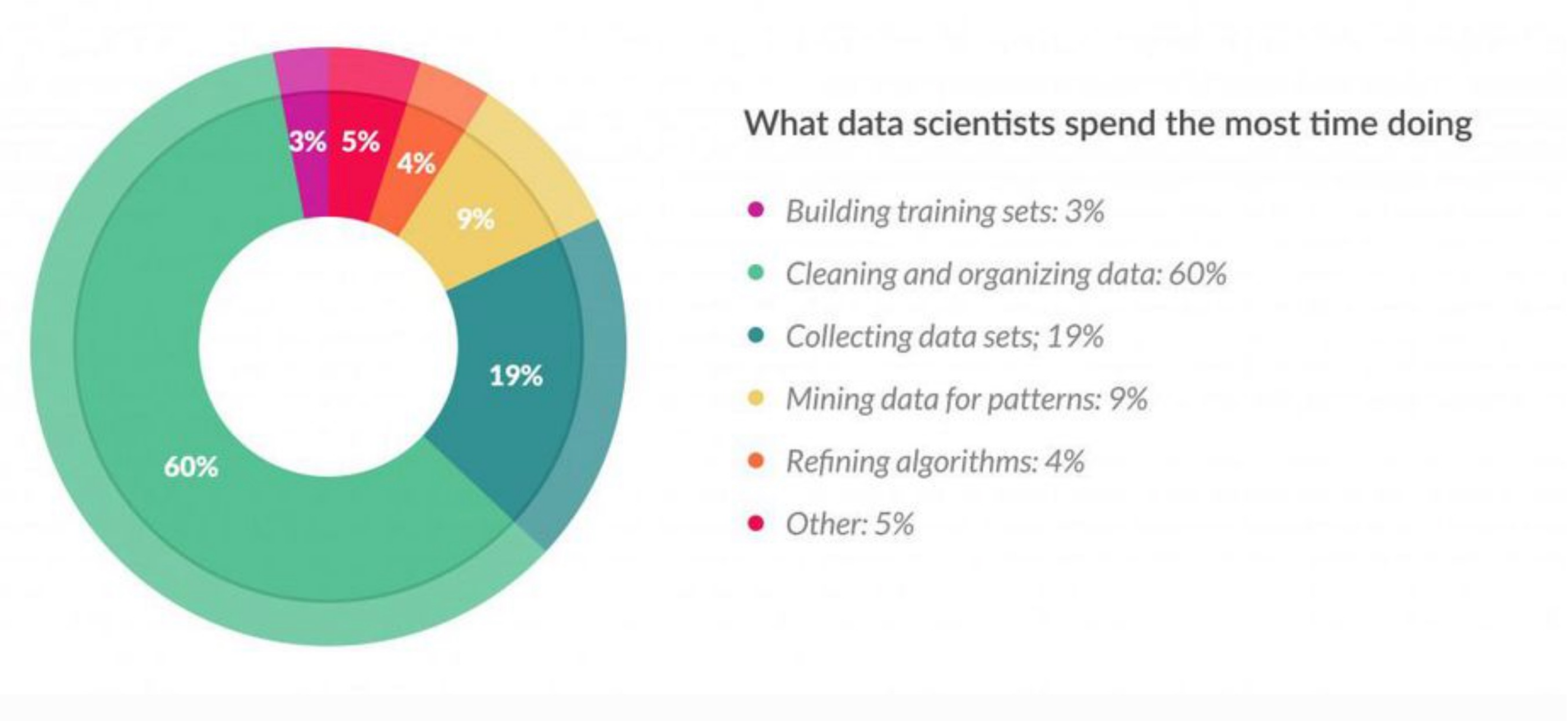


1. All supervised learning is focused on optimizing an objective function.
2. Optimizers all work kind of like gradient descent
3. Loss, regularization, over-fitting, under-fitting, hyper-parameter tuning, etc. are challenges of all classifiers

You know how to evaluate machine learning:

- Evaluation metrics
- Confusion matrices

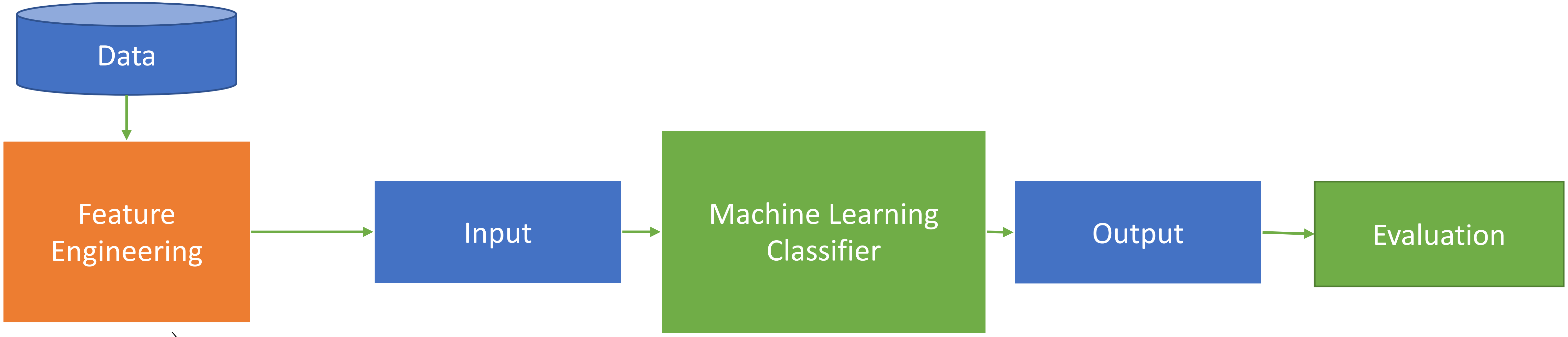
Feature Engineering



Collecting, Cleaning, and organizing data takes a lot of time

https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/?sh=630f381a6f63

A simple machine learning pipeline



Feature Engineering is hard.
It is a manual process that requires a lot of domain expertise and data analysis which is expensive in terms of time and therefore money too.

Why Don't I automatically learn features which serve as input into a machine learning classifier!?

Deep Learning

Deep Learning

- Deep learning accepts raw data as input and automatically learns features
- Deep learning does this using deep (3+ layer) neural networks
- The actual classifier used can be any that we have learned about, but differentiable classifiers are usually used because this allows back-propagation through the entire network
- Deep learning typically requires a massive amount of data (millions of samples) to effectively learn these features (**big data**)
- There are various configurations of the deep neural networks used for feature engineering – this is the variety of deep learning methods we will learn

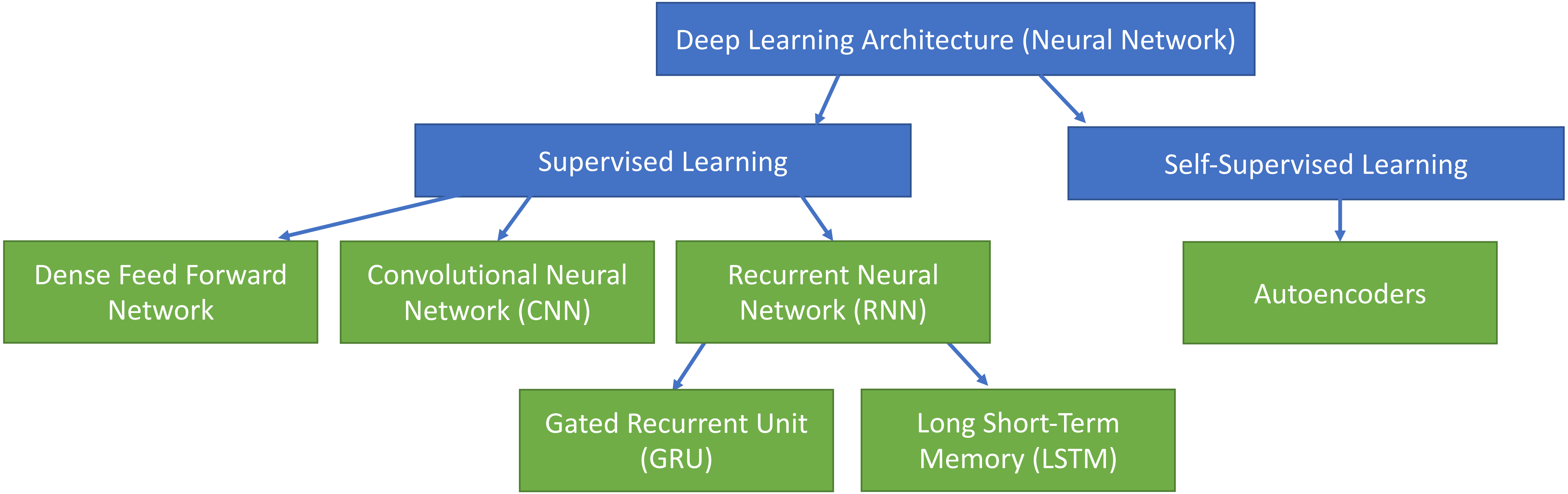
Deep Learning?

If I create a neural network that has 100 hidden layers am I doing deep learning?

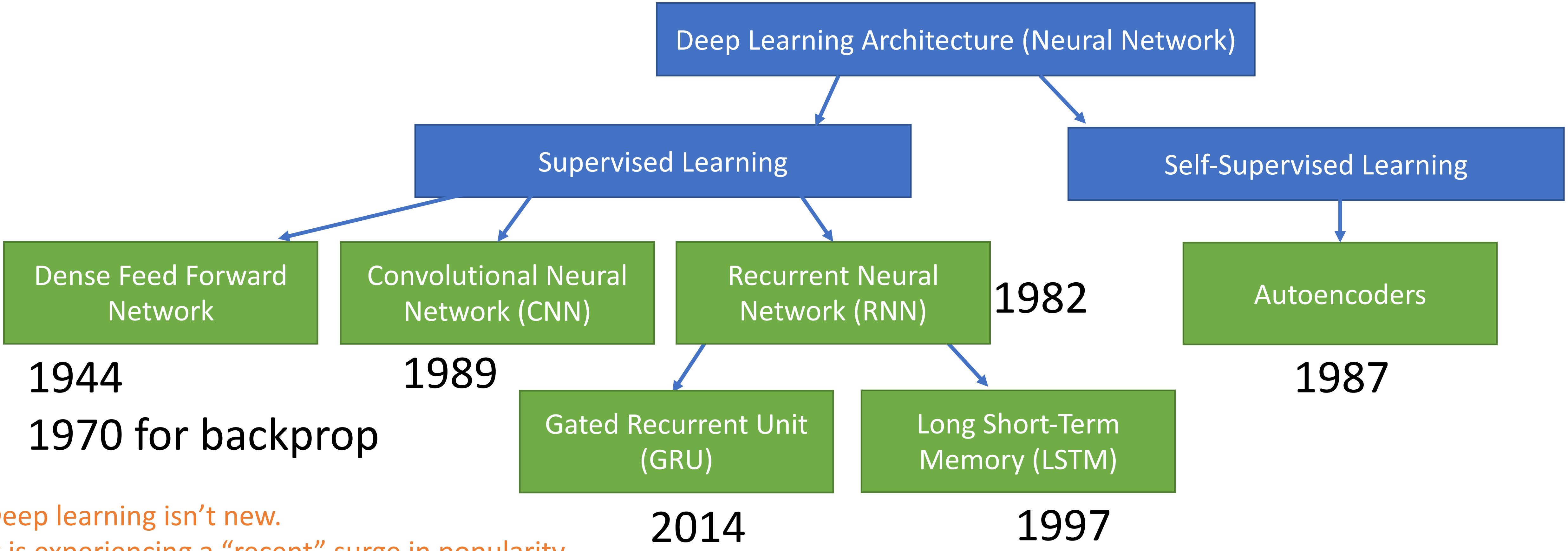
Technically, but really deep learning's goal is for feature engineering

Layers are typically connected in specific configurations to encourage the learning of features

Deep Learning Overview



Deep Learning Overview



Deep learning isn't new.
It is experiencing a "recent" surge in popularity.
Due in part to the combination of **Deep Learning and Big Data**

Why the Surge in Popularity

- Deep learning works really well
- It allows us to solve problems that we can do naturally
  - e.g. understanding vision, language, sounds
- Deep learning algorithms require a lot of data, but most (all?) provably perform better with more data
- The rise of **Big Data** – storage is cheap, and we can record and save everything
- Efficient backpropagation, **GPUs and Tensor cores**
  - You can parallelize backpropagation
  - GPUs have had a rapid increase in power
  - Tensor cores are specialized cores for matrix multiplication

Why the Surge in Popularity

- Today a Nvidia RTX 3090 has 35 Tflops of single precision (32bit) performance which is the same as the fastest supercomputer in the world in 2002

=

Implementation: TensorFlow and Keras

- **Keras** is a Python package for deep learning
  - Allows you to program deep neural networks at a high level

- **TensorFlow** is the first (and most common) backend of Keras
- Theano is an alternative backend

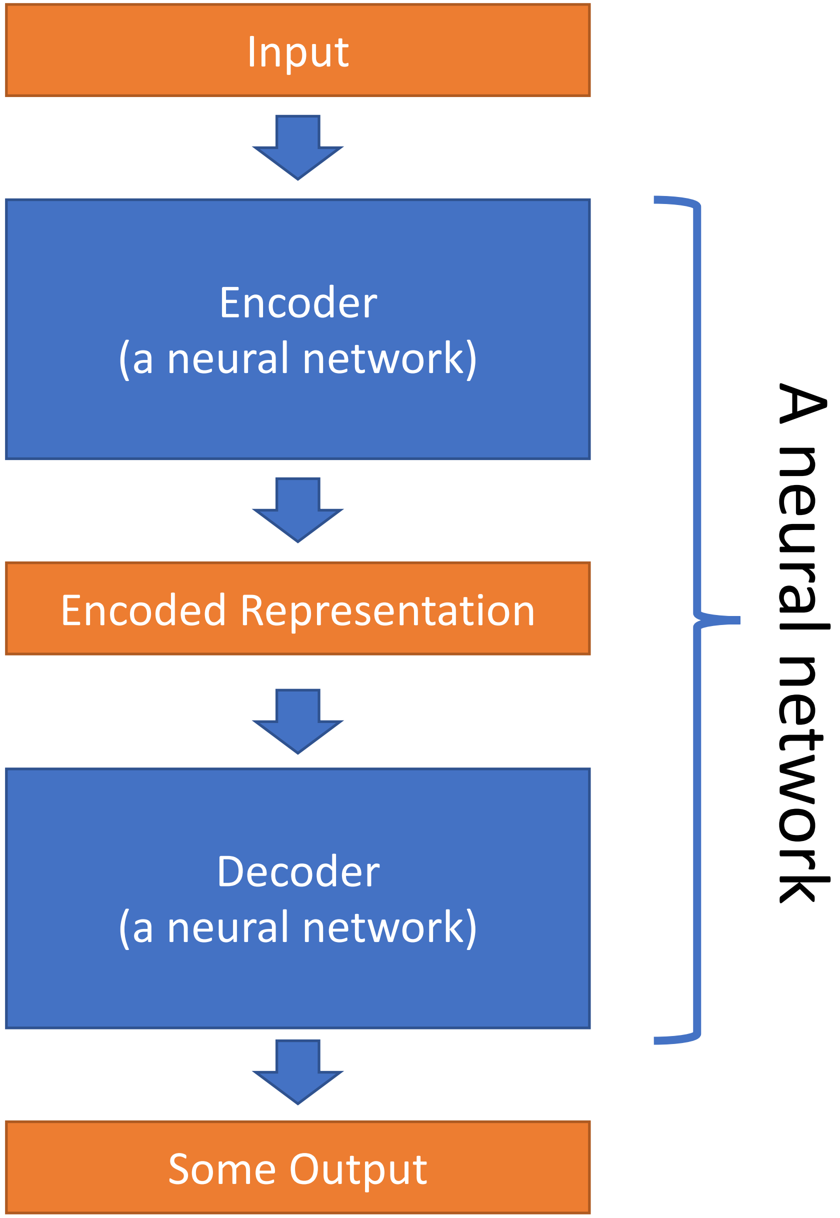
- **Pytorch** is an alternative to Keras

Encoder-Decoder Architecture

Encoder-Decoder

Encoder-Decoders consist of two components

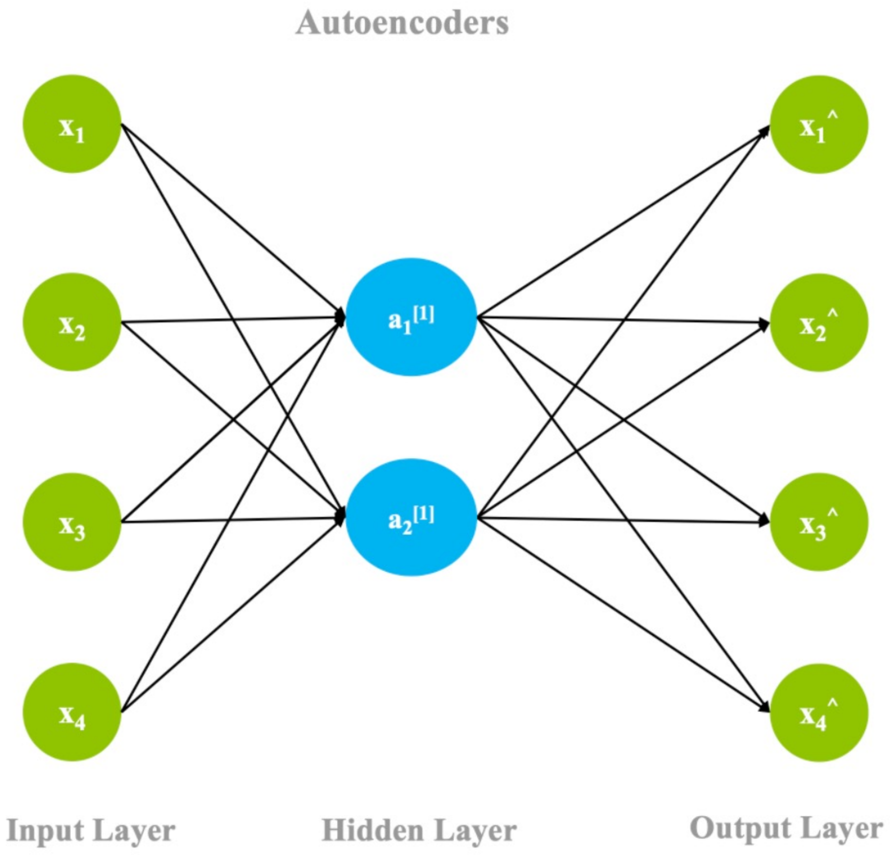
1. Encoder – transforms the input into a reduced dimensionality encoded representation
2. Decoder – uses the encoded representation as input and produces some output
3. The whole system is set up as a single neural network and the encoder learns efficient encodings for the task via back-propagation
4. The encoder can be removed and used for other tasks (transfer learning)



Autoencoders

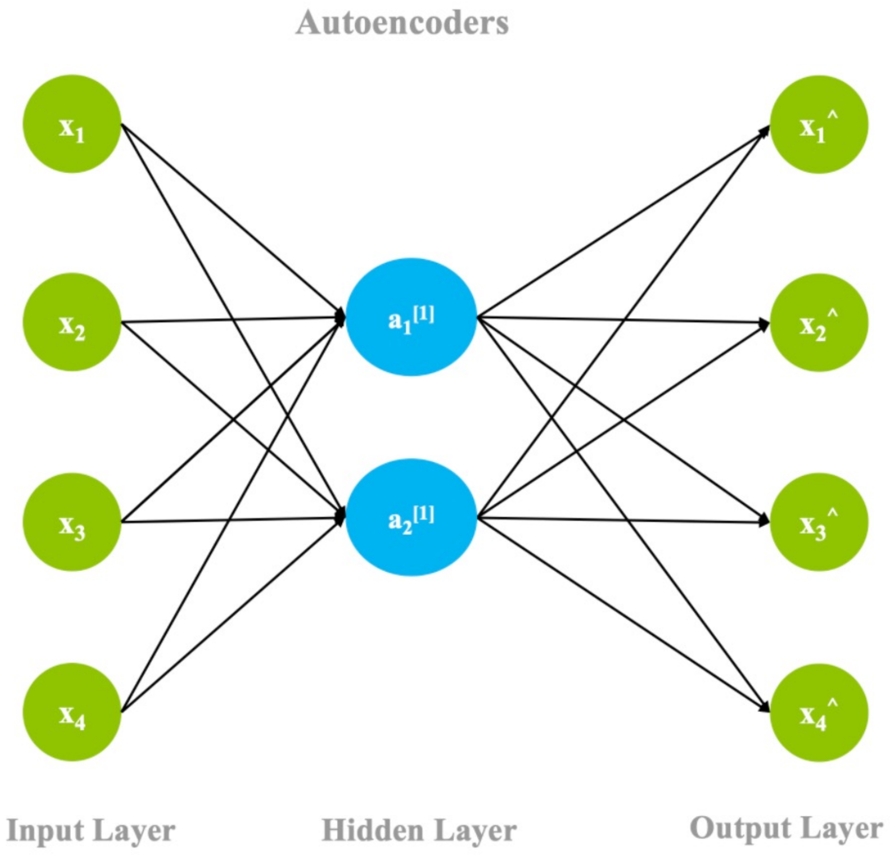
One example of an Encoder-Decoder Architecture are Autoencoders

- The first known usage of autoencoders was LeCun in 1987.
- ANN composed of 3 layers: input, hidden, and output
- The goal is to find a compressed data representation that minimizes data loss.
    - These are essentially compression algorithms



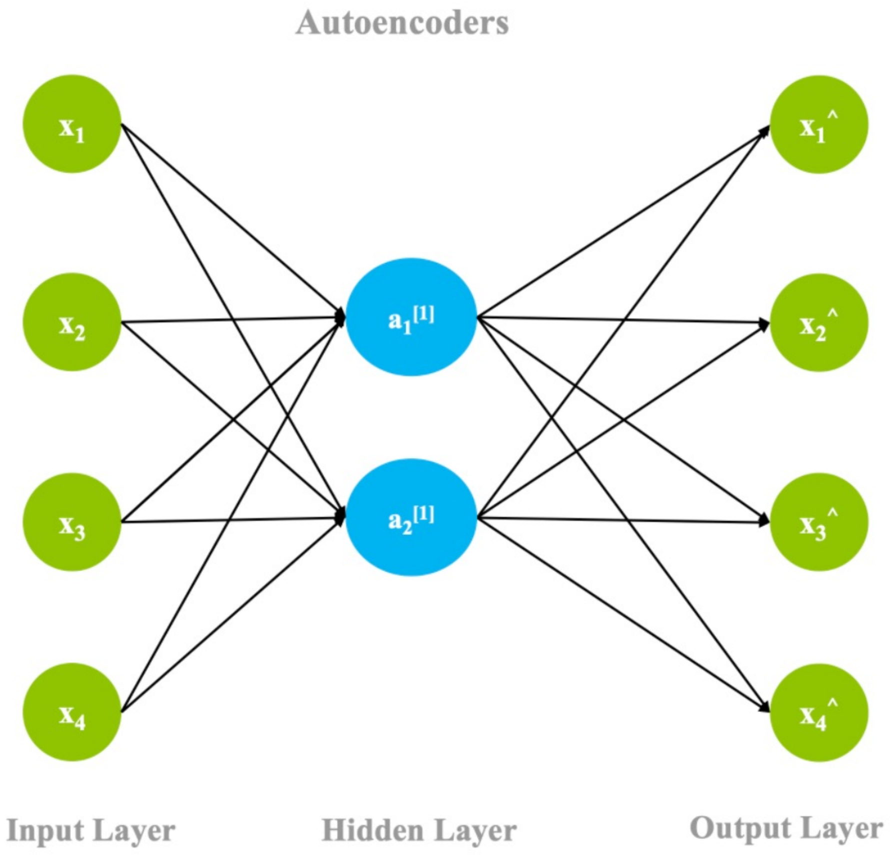
Autoencoders

1. The input layer is encoded into the hidden layer
2. The hidden layer contains the compressed representation of the original input
   - The number of nodes in the hidden layer should be much less than the number of nodes in the input layer
3. The output layer aims to reconstruct the input layer



Autoencoders

- During the training phase:
  - The difference between the input and the output layer is calculated using an error function
  - The weights are adjusted to minimize the error
  - Thereby minimizing the difference between the input and output functions



Autoencoders

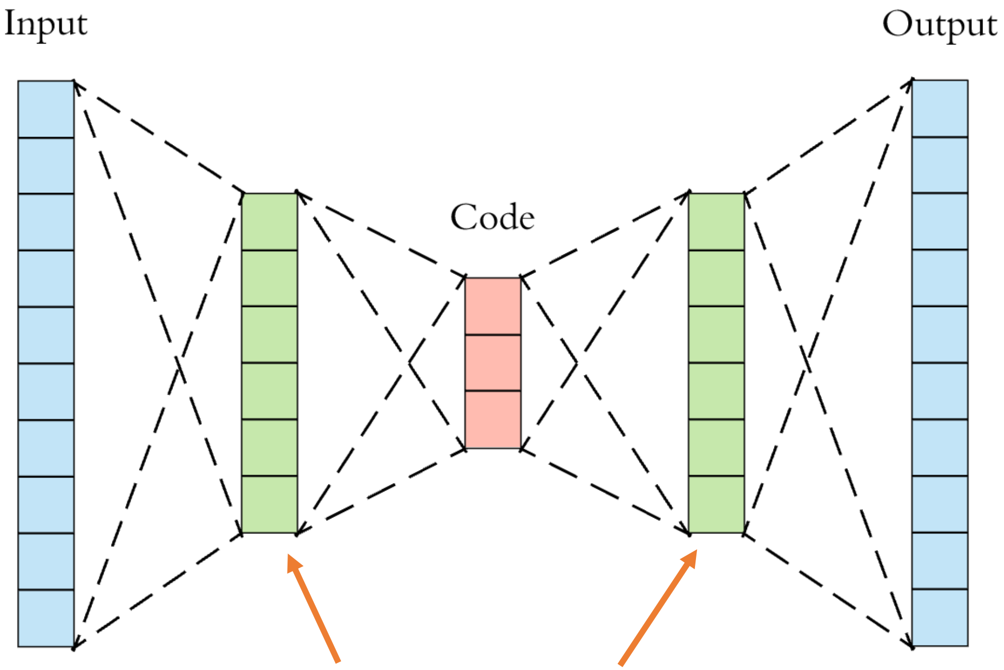
- Autoencoders are mainly compression algorithms
- Example applications:
  - Data Compression - for transmitting or storing data
  - Data Interpolation – if a dataset or image has some missing values, the auto-encoder will likely fill in those values
  - Dimensionality reduction – for feature reduction, or even visualizing data

Self Supervised Learning

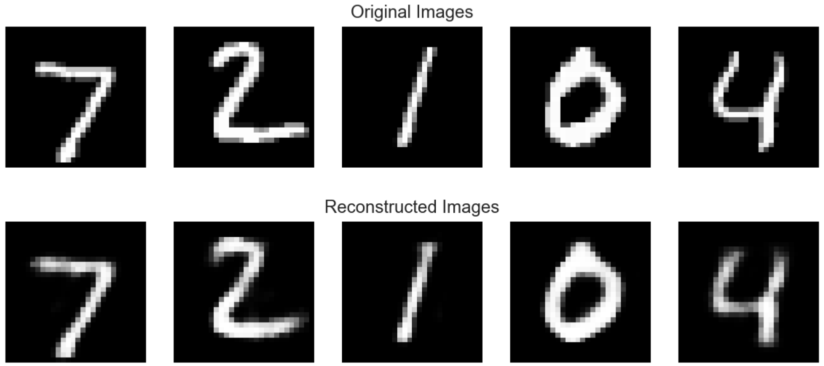
- Auto-encoders self-supervised algorithms
- Self-supervised algorithms fall somewhere between unsupervised and supervised learning
    - Unsupervised learning – learns patterns in data without labels (input is just the data, X)
    - Supervised learning – learns to assign labels to data with labels (input is data, X and labels, Y)
- **Self-supervised algorithms** automatically generate labels (Y) from the input data (X), then learn using supervised learning methods.
    - Autoencoders learn using backward propagation.

Implementation: Keras

```
input_size = 784
hidden_size = 128
code_size = 32
input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu’)(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)
autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy’)
autoencoder.fit(x_train, x_train, epochs=5)
```

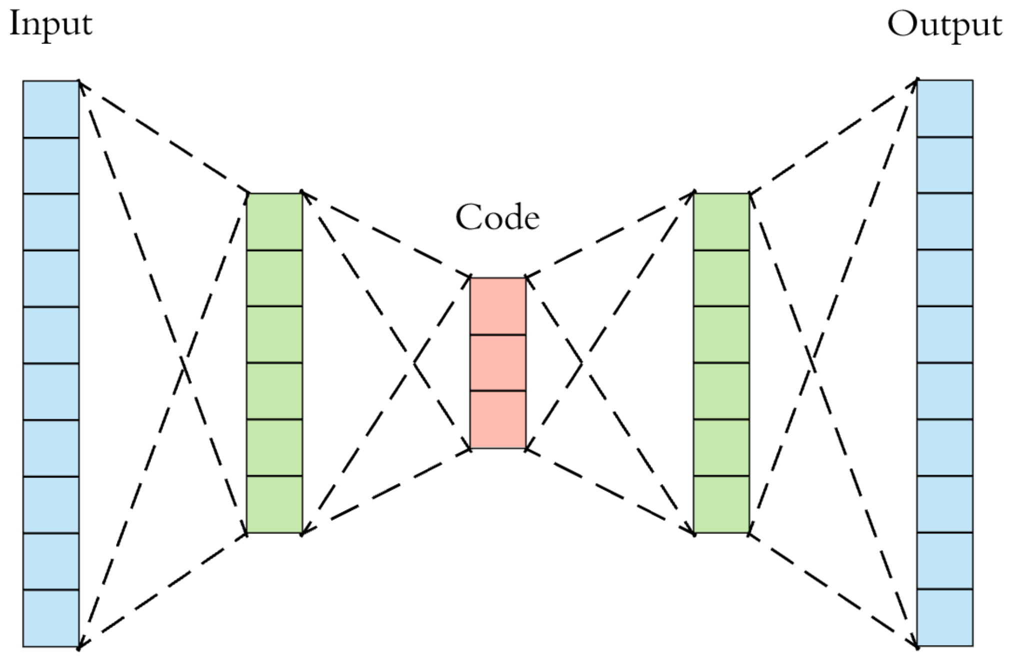


Auto-encoders generally have 1 or more layers between input and encoded layer and output and encoded layer



From: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Implementation: Keras

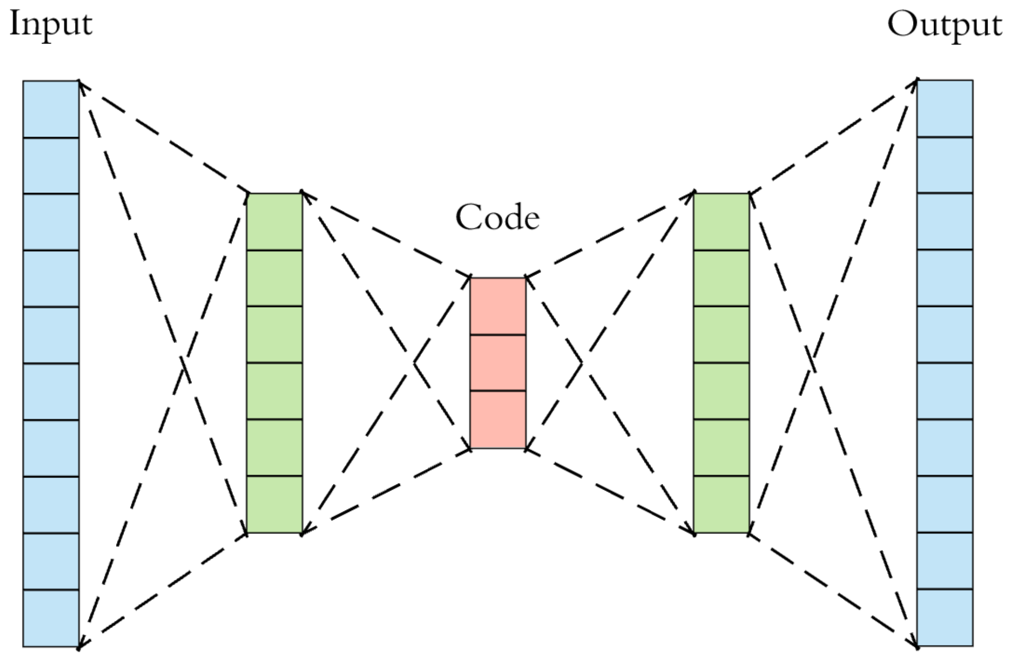



```
input_size = 784
hidden_size = 128
code_size = 32
input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)
autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
autoencoder.fit(x_train, x_train, epochs=5)
```

They are constructing the network layer by layer.
They create a new layer, then specify which layer it is connected to in parenthesis afterwards
… this is an older syntax, but I included it so that you would be able to read in when searching online

From: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Implementation: Keras




```
input_size = 784
hidden_size = 128
code_size = 32
input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu’)(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)
autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy’)
autoencoder.fit(x_train, x_train, epochs=5)
```

Create the network, specifying the input and output layers (connections are already specified)
Then, compile and Train it

From: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Implementation: Keras

```
input_size = 784
hidden_size = 128
code_size = 32
input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu’)(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)
autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy’)
autoencoder.fit(x_train, x_train, epochs=5)
```

Code from the internet, but then I saw this

Does anyone know what’s weird with this?

From: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

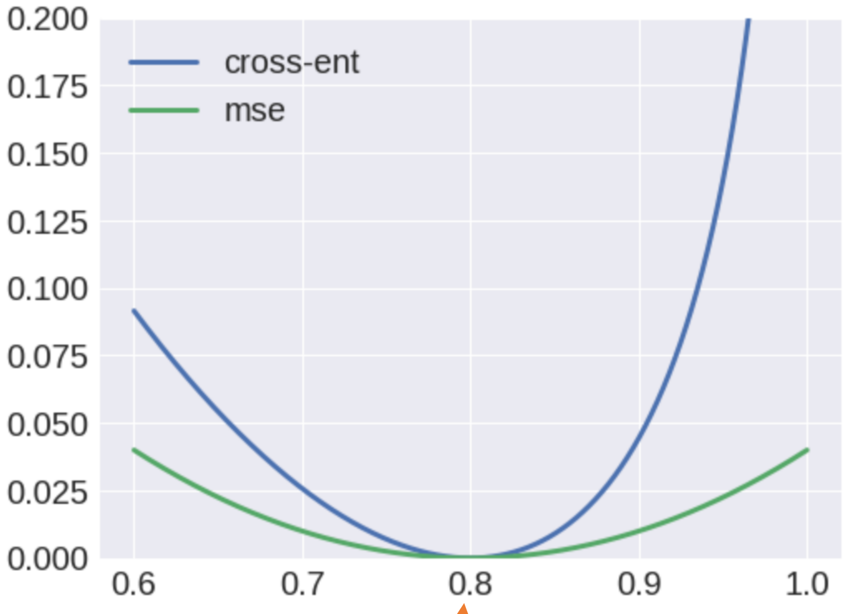
Potential Problems with this code

- Goal of the network is to minimize the error between the input and output layers
- Applying a sigmoid means that the output will be between 0 and 1
  - This implies that the input must be scaled between 0 and 1 too
    - In the article they briefly mention that they do scale their input between 0 and 1.
    - But, this is a pretty weird thing to do. Why scale pixel values between 0 and 1?

…more on next slide

Problems with this code

- Using BCE as a loss function isn't really appropriate either
  - It may work, but again, its weird.
  - You are minimizing the error between the input and output pixels
  - Do we expect the pixels to have values **either 0 and 1**?
  - Do the pixel values indicate any kind of **probability** of belonging to a class?
- This is really a regression tasks, so use Mean Squared Error (MSE) or something similar

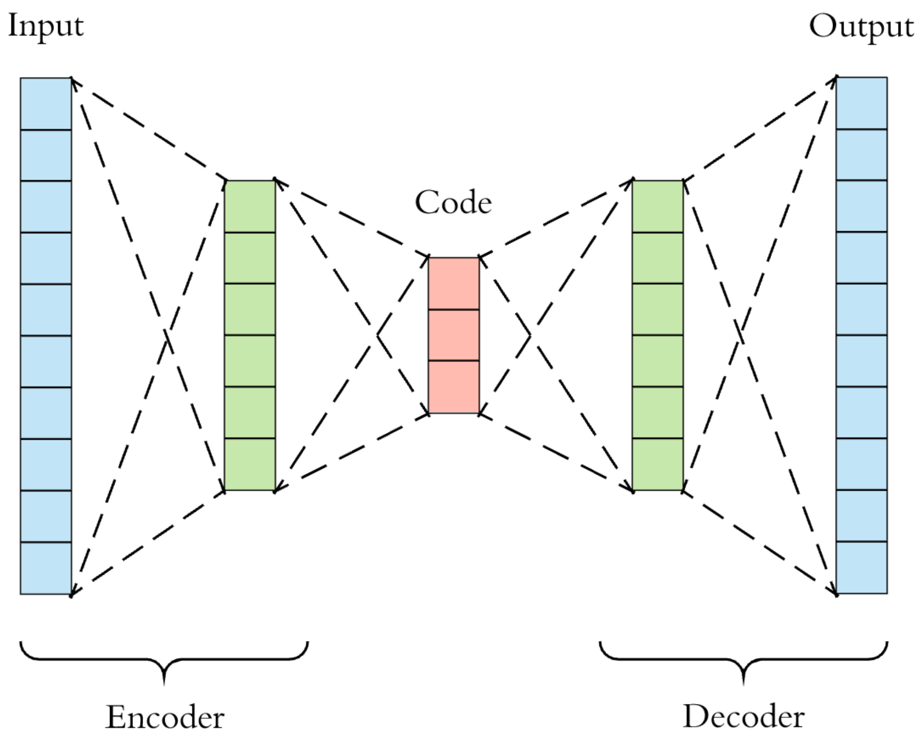


Target Pixel value is 0.8
Why penalize more for 0.9 than 0.7?

Image from: https://stats.stackexchange.com/questions/245448/loss-function-for-autoencoders

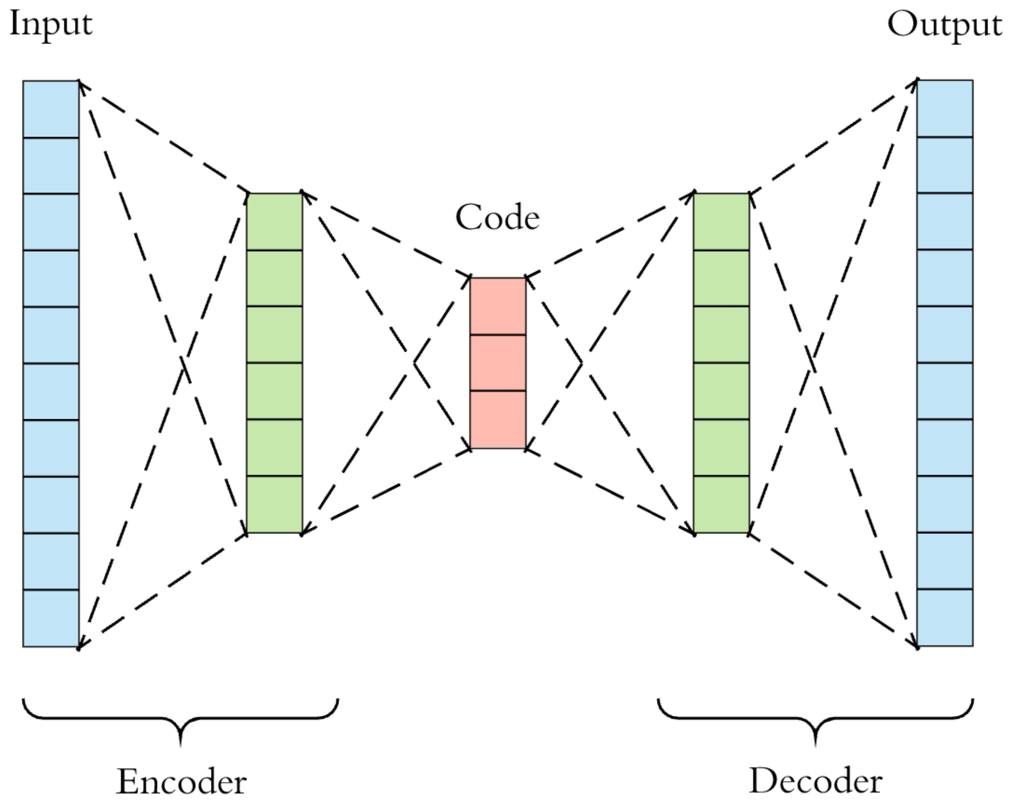
Encoder-Decoders

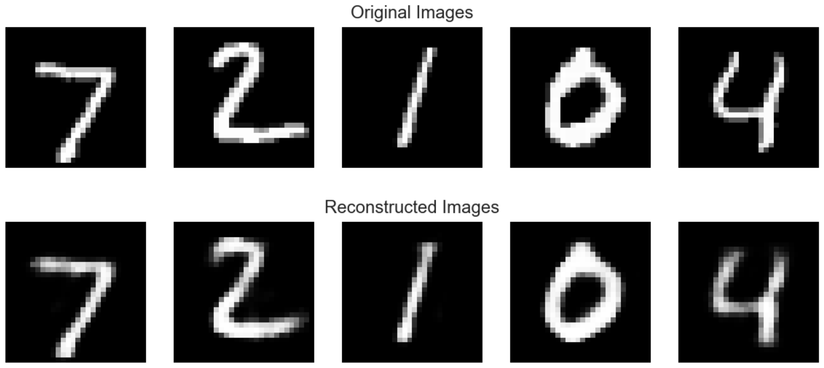
- More Generally, auto-encoders have an Encoder-Decoder architecture
- Specifically, they are a special case of encoder-decoders, where the input and output are the same
- Encoder-Decoders have two parts
  1. Encoder – represent input in some encoded (typically compressed format)
  2. Decoder – use the encoded representation as input for some task



Encoder

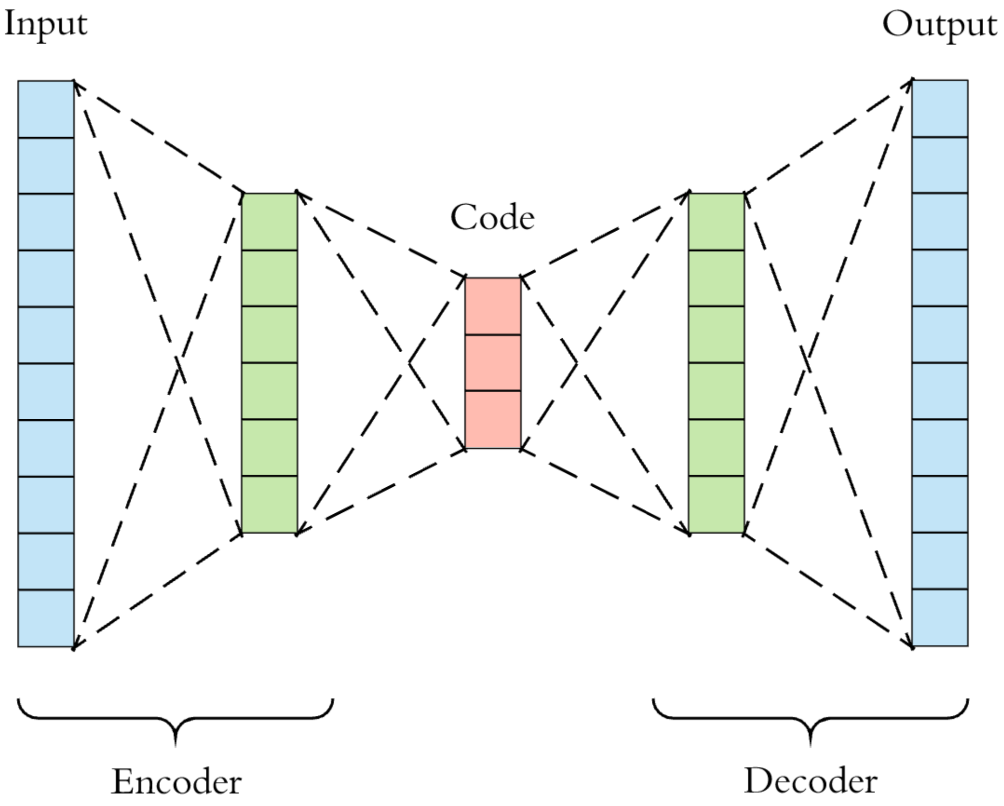
- The encoder encodes data by finding weights that map it to a compressed format
- The idea is that the compression removes variation in the input to a more general representation
- The removed variation can be thought of as noise

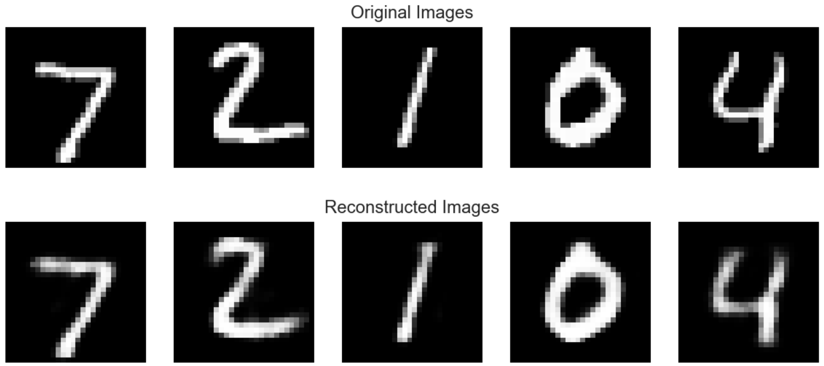




Decoder

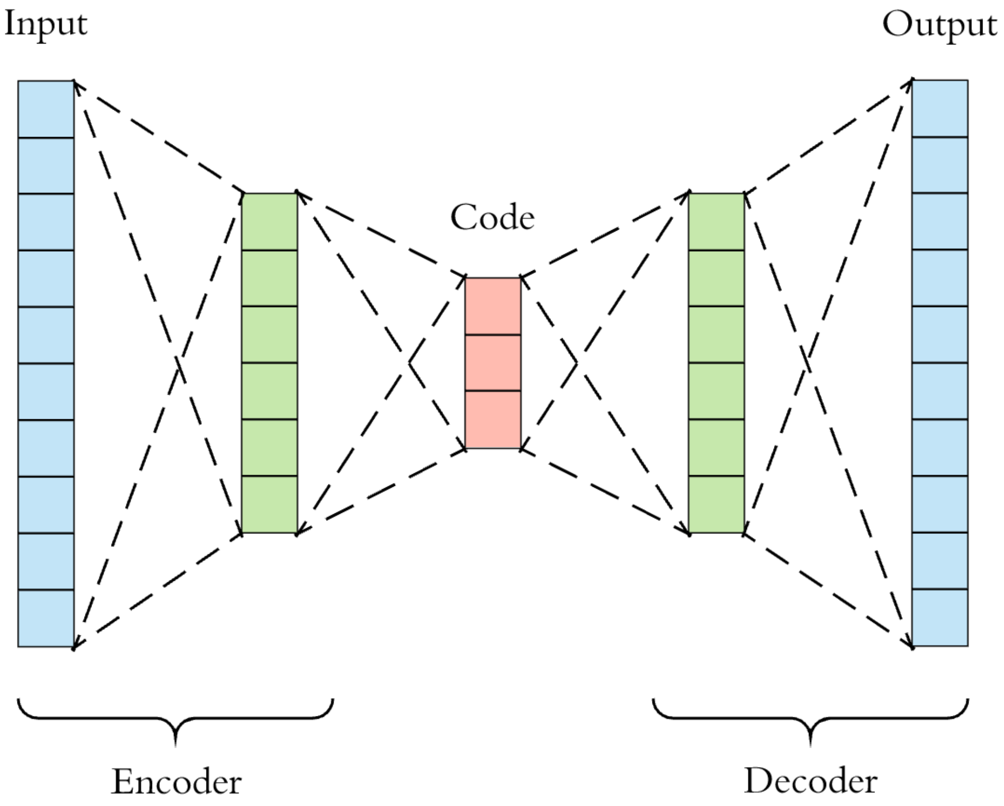
- The decoder decodes the compressed format by finding weights that minimize loss.
- Since the encoded representation reduces noise, ideally it makes the task of the decoder easier

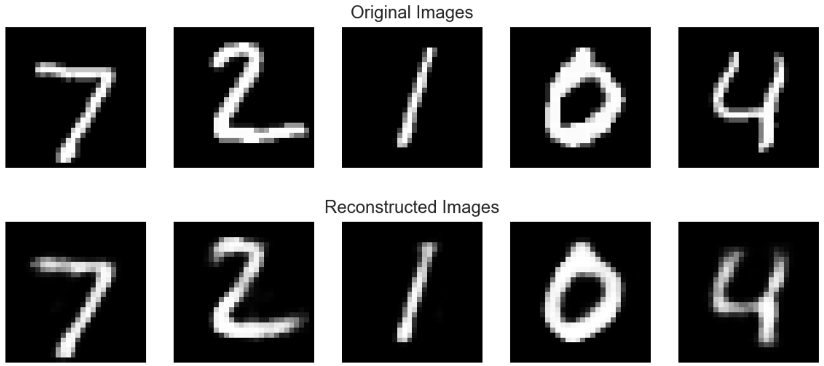




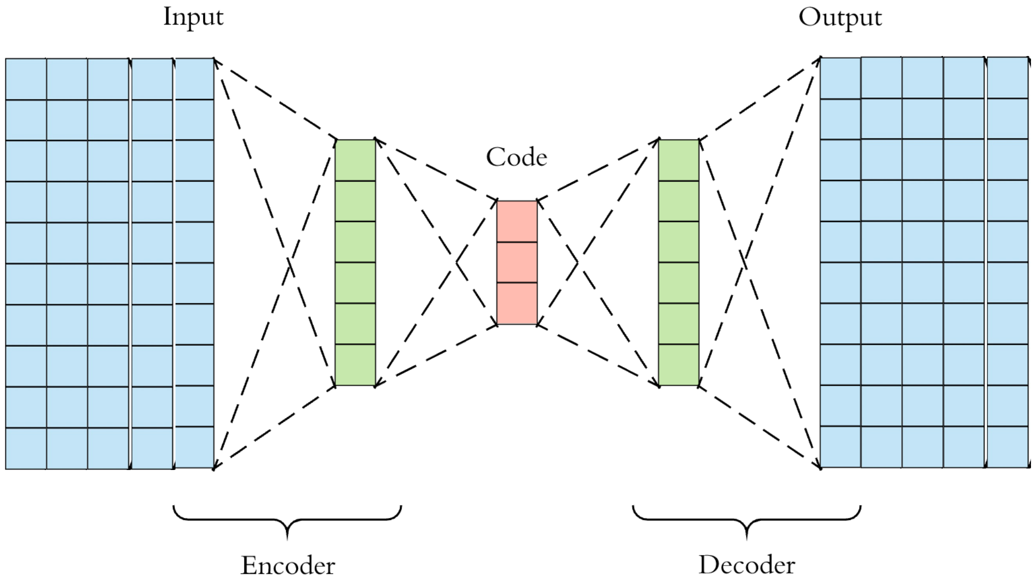
Transfer Learning

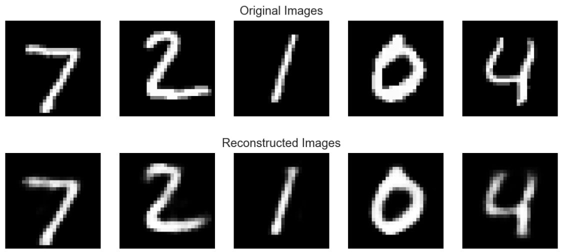
- Encoder-Decoder architectures are often used to facilitate transfer learning
- **Transfer learning** – using the weights of a model trained for one task as the starting point of the weights of a network for another task





Transfer Learning with Encoder-Decoders



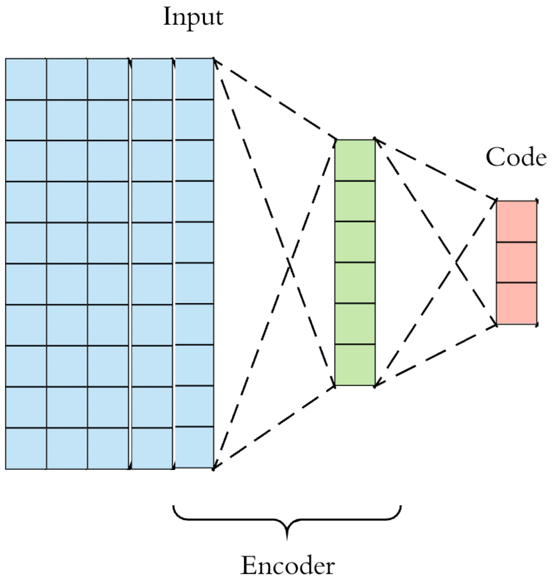


The encoder gets really good at compressing the images into a meaningful representation that removes noise and that is good at generalizing to new samples

Input and output are an image
(so I added more vectors)

What if we remove the decoder part of the architecture and put a neural network for a different task on?

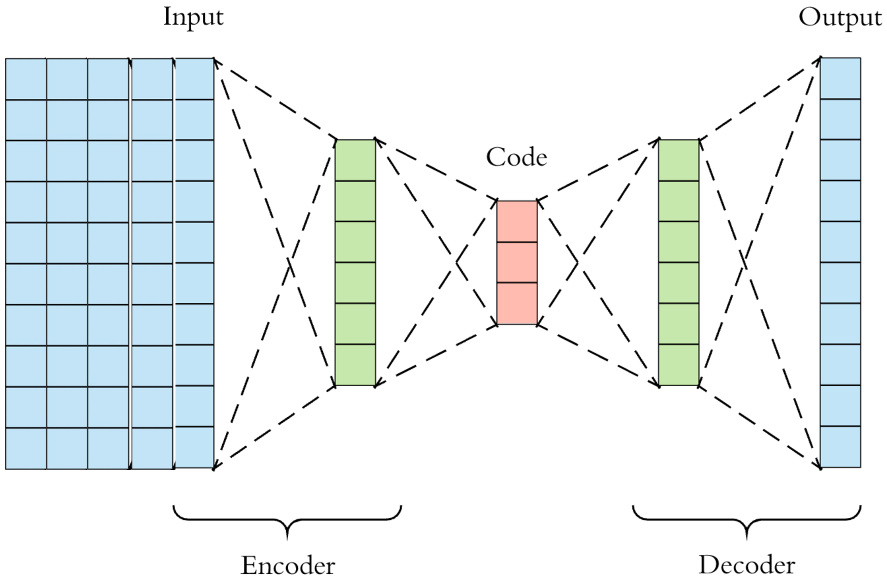
Transfer Learning with Encoder-Decoders



Now we have a network that is really good at encoding things

This is a great compressed representation of the input image

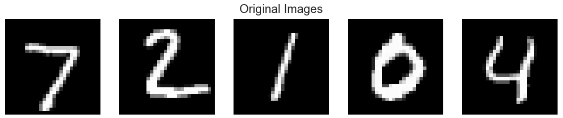
Transfer Learning with Encoder-Decoders



Original encoder network

New Decoder Network

1. Remove the old decoder
2. Replace it with a new network
3. Train the network for the new task



0000000100
0010000000
0100000000
1000000000
0000100000

New task is to predict which number is written