CESI LINEACT

Introduction to AutoEncoders (AE)

By: BENATIA M. A.
mbenatia@cesi.fr

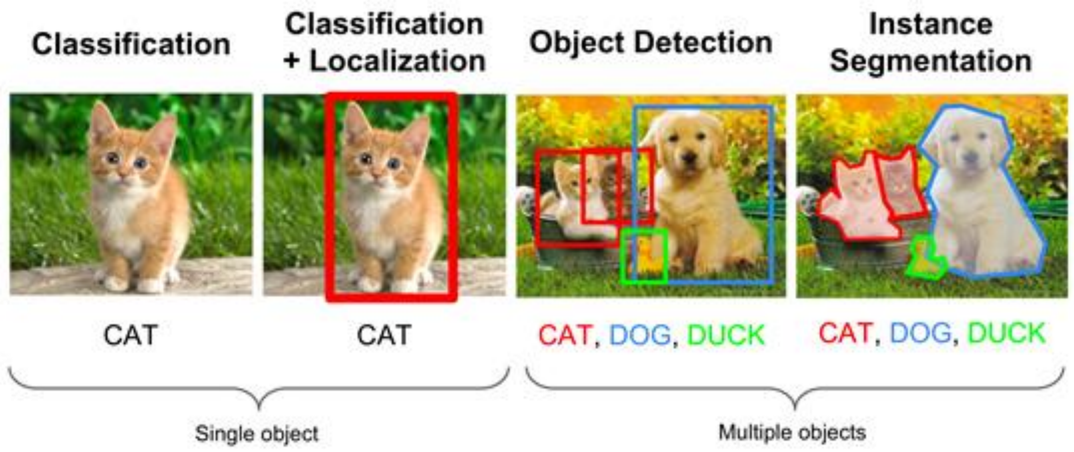
Outline

- Course Introduction
- Intro to ML
- Autoencoders (AE)
- Latent Variable Models (LVMs)
- Variational Autoencoders (VAE)
- Motivation for GANs

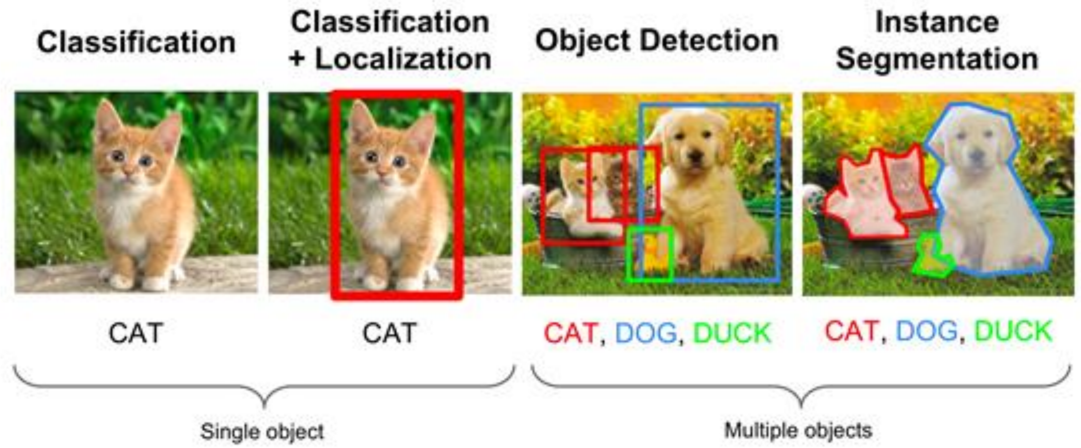
Course Introduction (Computer Vision and ML)

What is Computer Vision?

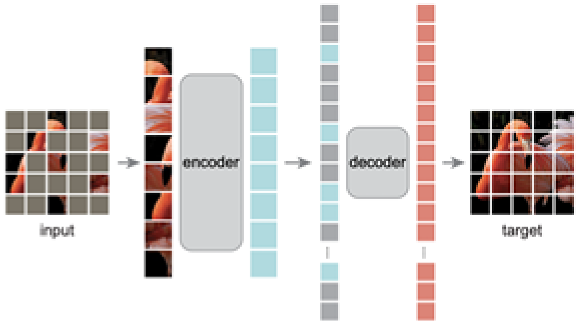
- Computer vision is broadly the subset of AI that deals with images
- It is used to clear checks, deliver mail, drive cars, and create art
- And more other stuffs!



CESI LINEACT

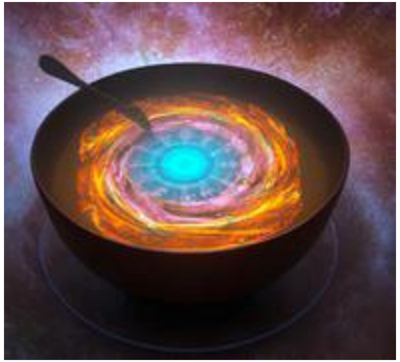


Basic computer vision tasks



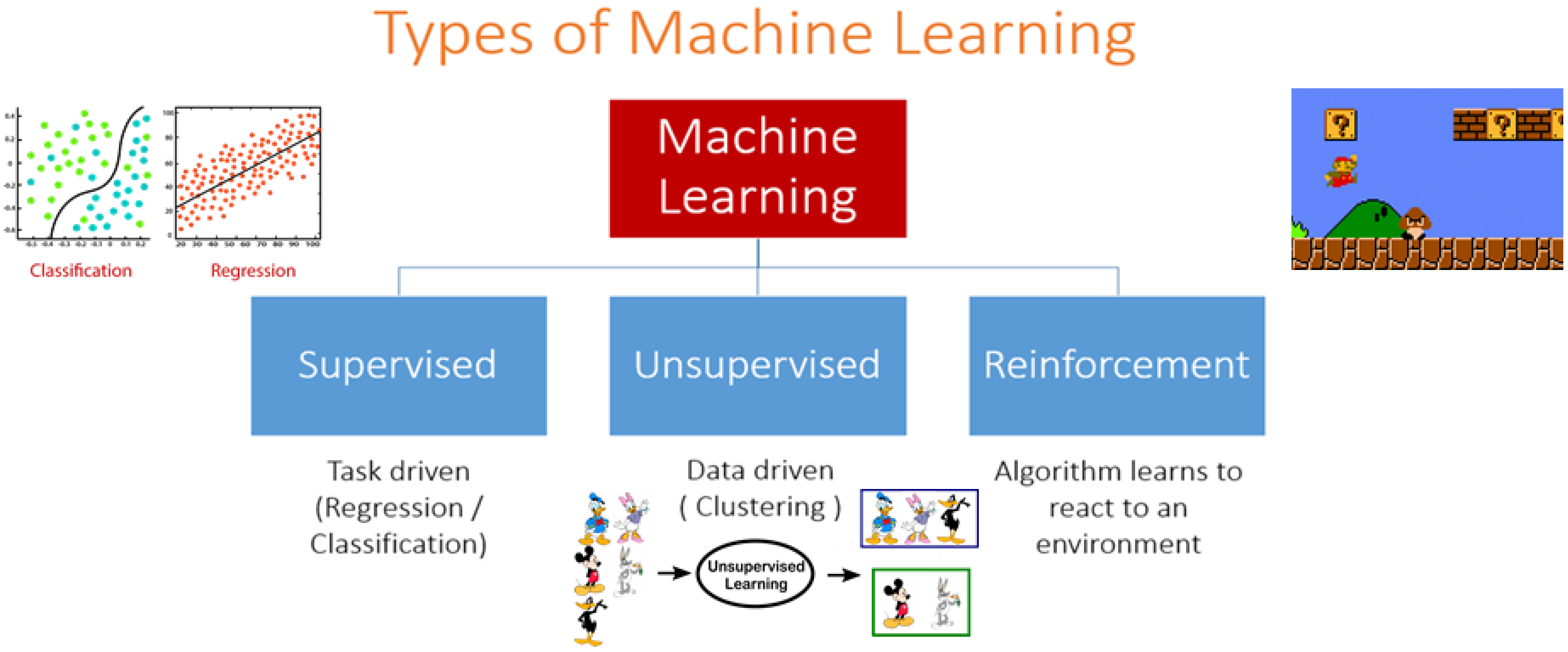
Large-scale unsupervised learning

3D Vision



Text-Based Image Generation

Types of Machine Learning



Used Vocab

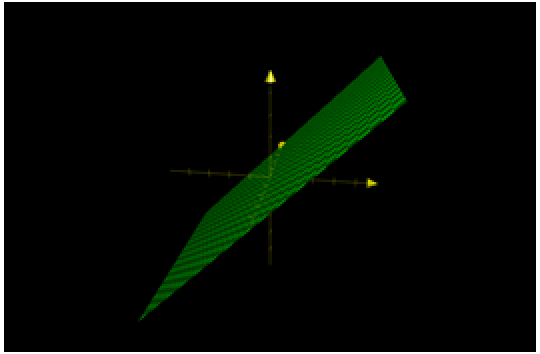
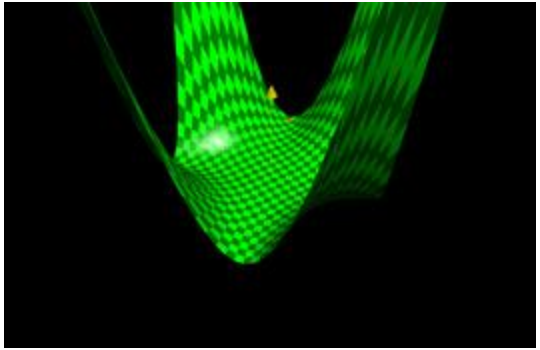
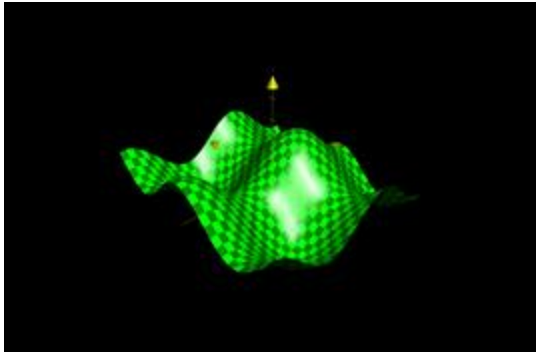
- Function / Model
  - These terms are used interchangeably
  - These refer to the function template we have chosen for our problem
- Parameters / Weights (and Biases)
  - Weights are terms used to denote the parameters in ML models that are learned from data
- Hyperparameter
  - This is some non-learnable parameter (like model size, model class, details about training procedure, etc) that further specifies our overall learnable function
  - Again, we choose these ourselves before we start learning the learnable parameters
- Loss Function / Cost Function / Risk Function
  - We haven't introduced these terms yet, but they will come up later; just note that they are the same (at least for our purposes)
- "Feature"
  - This can refer to bits of our data (either the inputs themselves or some representation of them)

Universal Function Approximator

CESI LINEACT

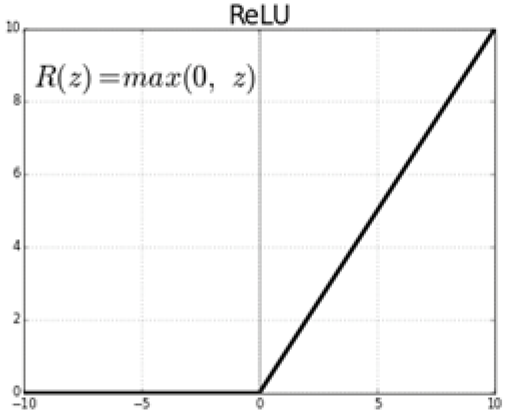
Motivation for Neural Networks

- Lots of problems that we can (or want to) solve — regression, classification, etc — frequently center around creating functions that are **super non-linear**
- Hard to figure out which class of models works the best for each task / dataset
  - Is there some model class that can do any one of these tasks almost straight out of the box?

Motivation for Neural Networks

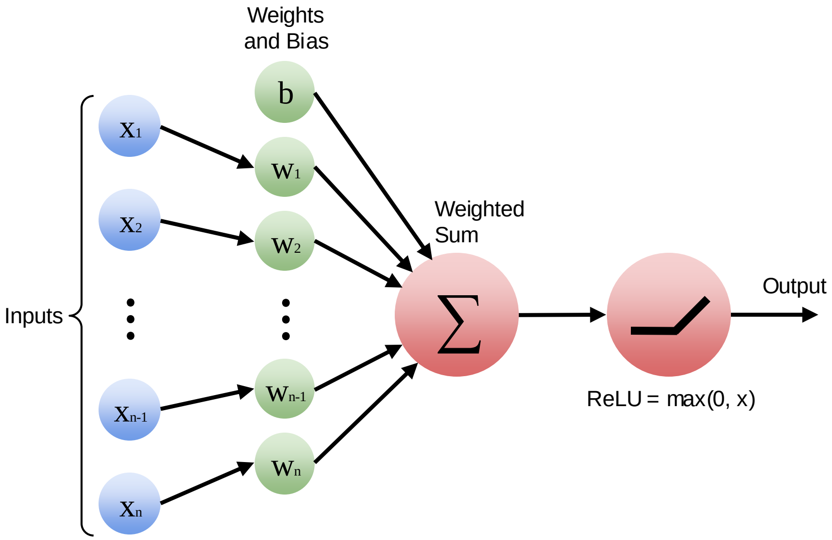
- Our brain is able to do lots of tasks with the same machinery
  - What if we tried to create a model of the brain?
- Our brain has neurons
  - Neurons take in signal from surrounding neurons
  - Neurons output a signal based on the amount of signal taken in
  - Output(inputs) = ReLU(weighted sum of inputs)
  - ReLU(x) = max(0, x) = x if x > 0 else 0
- **Fair warning: deep learning isn't the same as cognitive science**



Neural Networks (graphical view)

Perceptron

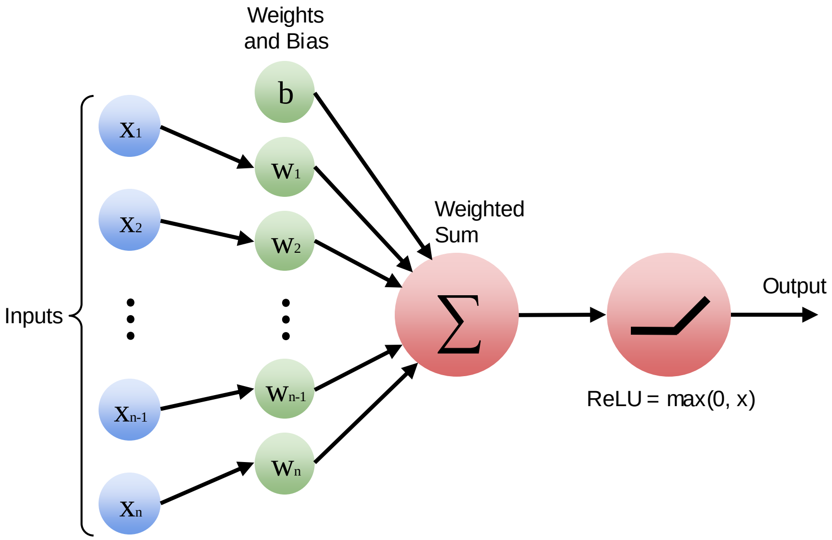
- Rough model of a neuron
- Weight each component of the input by some amount, then "activate" on the sum
  - Note: **b** is just a scalar being added to the weighted sum and is independent of the input — we call this term a "bias" value
- The function we use for "activating" is a ReLU



$$f(x_1, x_2, \ldots, x_n, w_1, w_2, \ldots, w_n, b) = f(\bar{x}, \bar{w}, b) = \mathrm{ReLU}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b)$$

Perceptron

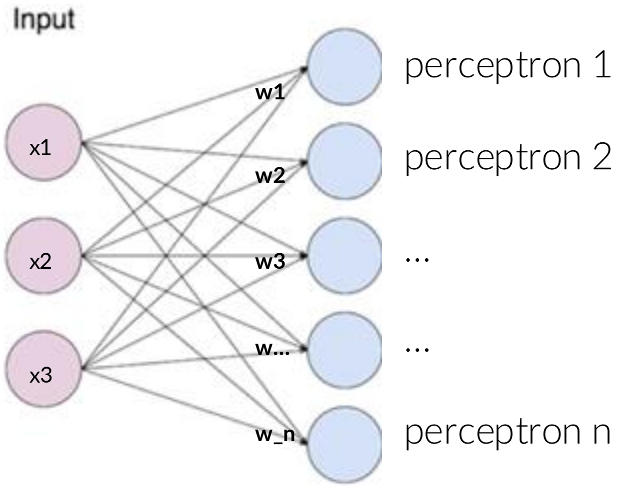
- Let's simplify the notation a bit
- The sum in the previous slide can be rewritten as a dot product between a weight vector and an input vector plus a bias scalar



$$f(\bar{x}, \bar{w}, b) = \mathrm{ReLU}(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b) = \mathrm{ReLU}\left(\bar{w}^\top \bar{x} + b\right)$$

Perceptron Layer

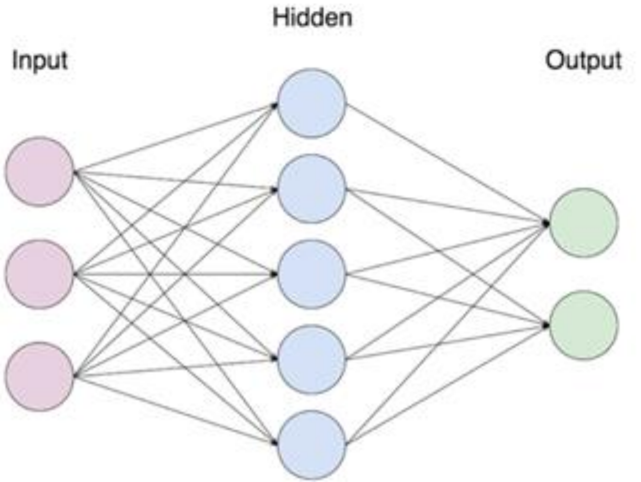
- What if we have multiple perceptrons that share the same input?
  - Neurons in a brain form all kinds of connections
  - Maybe different perceptrons can be used to extract different kinds of signals out of the same input
- **Each perceptron (with its unique weights and biases) can be stacked into a “layer”**



Neural Network

Note: We don't activate with ReLU on the output layer

- Each node (except for the column of input nodes) is just a perceptron
- Each perceptron layer is also called a neural network layer
  - We can choose to stack ***arbitrary*** numbers of perceptrons at each layer or cascade with an ***arbitrary*** number of layers
  - Each layer can have a different number of perceptrons
  - The middle layers are often called "hidden layers" since they aren't immediately interpretable
- We will still choose to use the ReLU function for each layer for now



This model is also called a "Multi-Layer Perceptron" or MLP for obvious reasons

What are we trying to learn?

What is a "feature"

- Consider the classical machine learning problem:
  - You have an input **x** and some function **Φ(x)** that returns any relevant features from **x**
    - For example, if **x** is a house and **y** is it's selling price, then **Φ(x)** could be a vector with information like the *house age, the number of rooms, whether it has a basement or not, etc.*
  - You pass the features into some model $\mathbf{f_\theta(\Phi(x))}$, parameterized by **θ**, to predict the label **y**
  - **The machine learning problem is to "learn" θ from a training dataset of (x, y) pairs**

House	House Age	Number of Bathrooms	Number of Bedrooms	Size (sq. feet)	Floors	Basement?	Garage?	Backyard?	Pool?
1	12 months	4	4	2500	2	Yes	Yes	No	No
2	30 months	2	3	2000	1	No	Yes	Yes	Yes

What is a “feature”

- Consider the classical machine learning problem:
  - You have an input **x** and some function **Φ(x)** that returns any relevant features from **x**
    - For example, if **x** is a house and **y** is it’s selling price, then **Φ(x)** could be a vector with information like the house age, the number of rooms, whether it has a basement or not, etc.
  - You pass the features into some model **f_θ(Φ(x))**, parameterized by **θ**, to predict the label **y**
  - **The machine learning problem is to “learn” θ from a training dataset of (x, y) pairs**

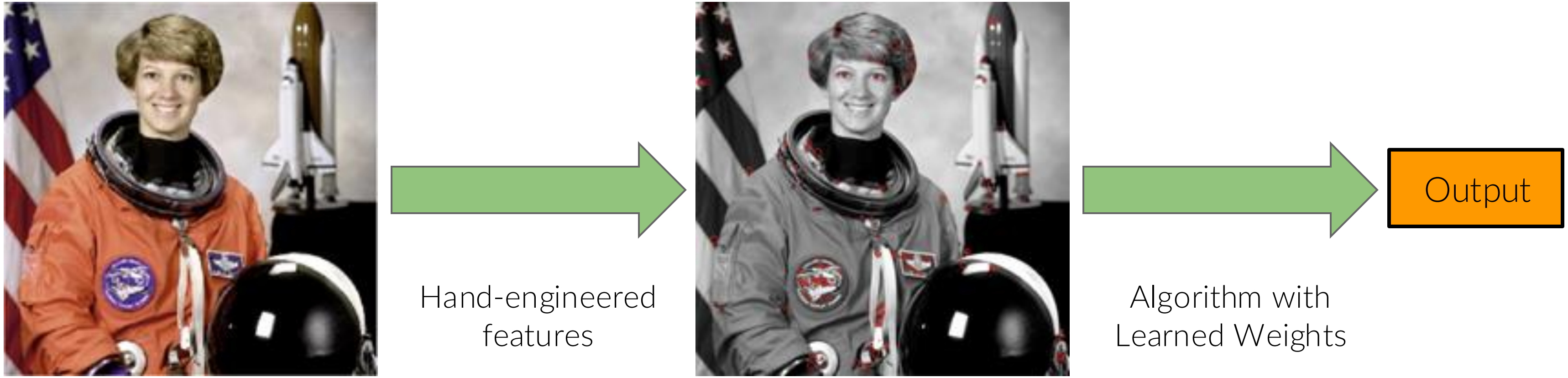
Input → Features → Output

Feature Extraction
x → Φ(x)

ML Algorithm
Φ(x) → y
f_θ

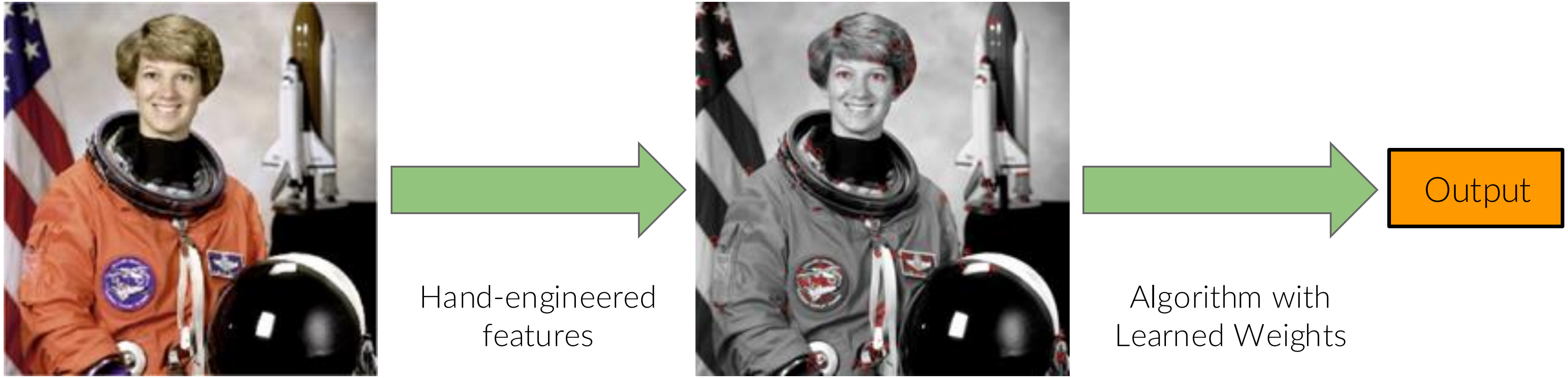
An alternate view of deep learning

- In the early days, feature extractors were programmed by hand, i.e., ML practitioners would manually select what features they would feed into the model of their choice.
  - If the input is an image, this could include edge and corner information, presence of certain shapes, patterns or colors, etc — these are all different *representations* of the same image



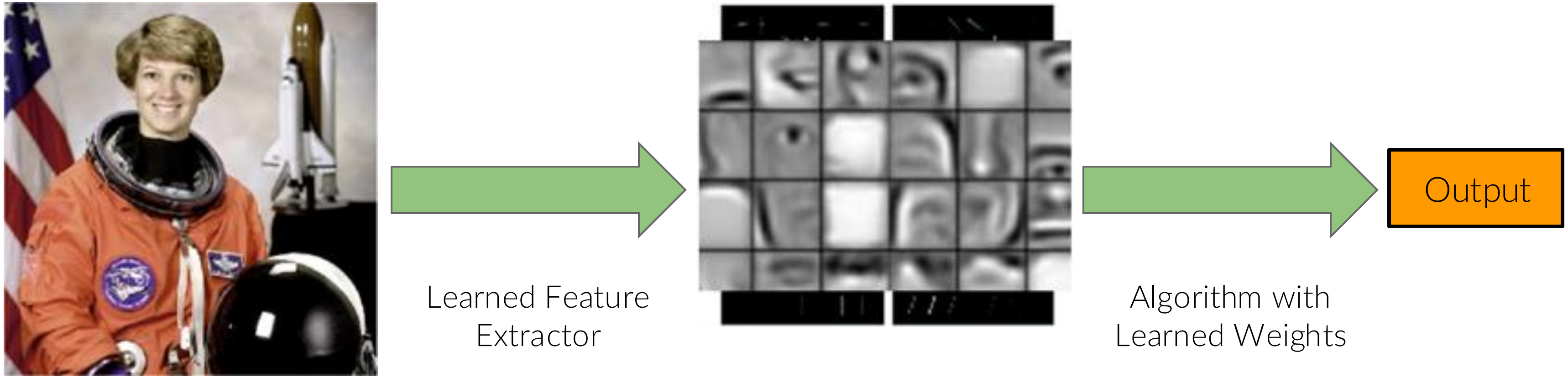
An alternate view of deep learning

- However, this is a compromise! The model is learning its parameters from data yet we are still hand-programming the feature extractors ourselves. Can we make the entire process learned from *end-to-end*?
  - Yes! This is where deep learning comes in!
  - Learn the feature extractor as well — learn EVERYTHING in the entire pipeline!



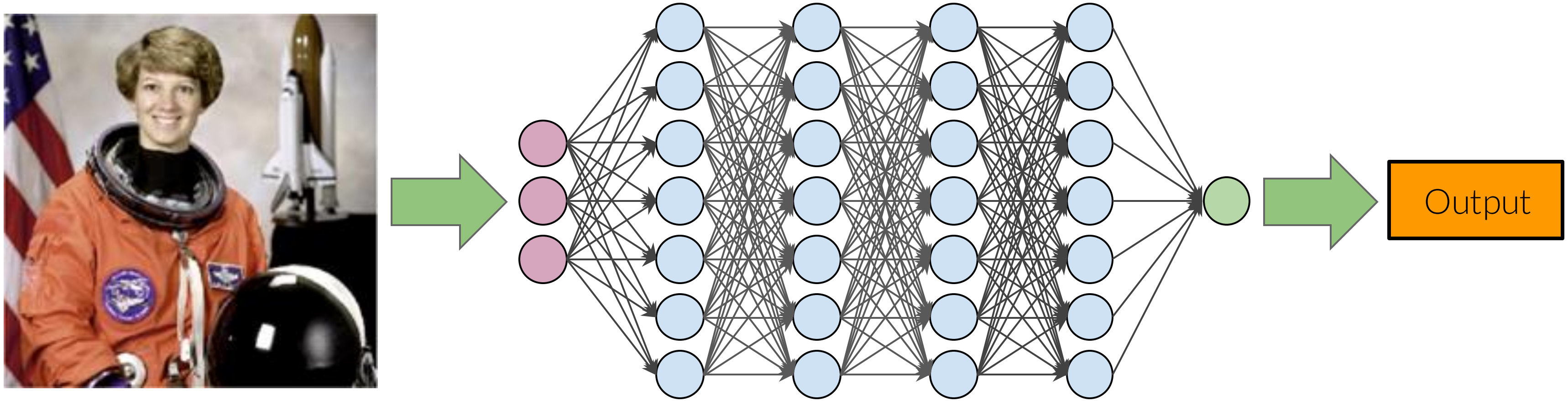
An alternate view of deep learning

- However, this is a compromise! The model is learning its parameters from data yet we are still hand-programming the feature extractors ourselves. Can we make the entire process learned from *end-to-end*?
  - Yes! This is where deep learning comes in!
  - Learn the feature extractor as well — learn EVERYTHING in the entire pipeline!



An alternate view of deep learning

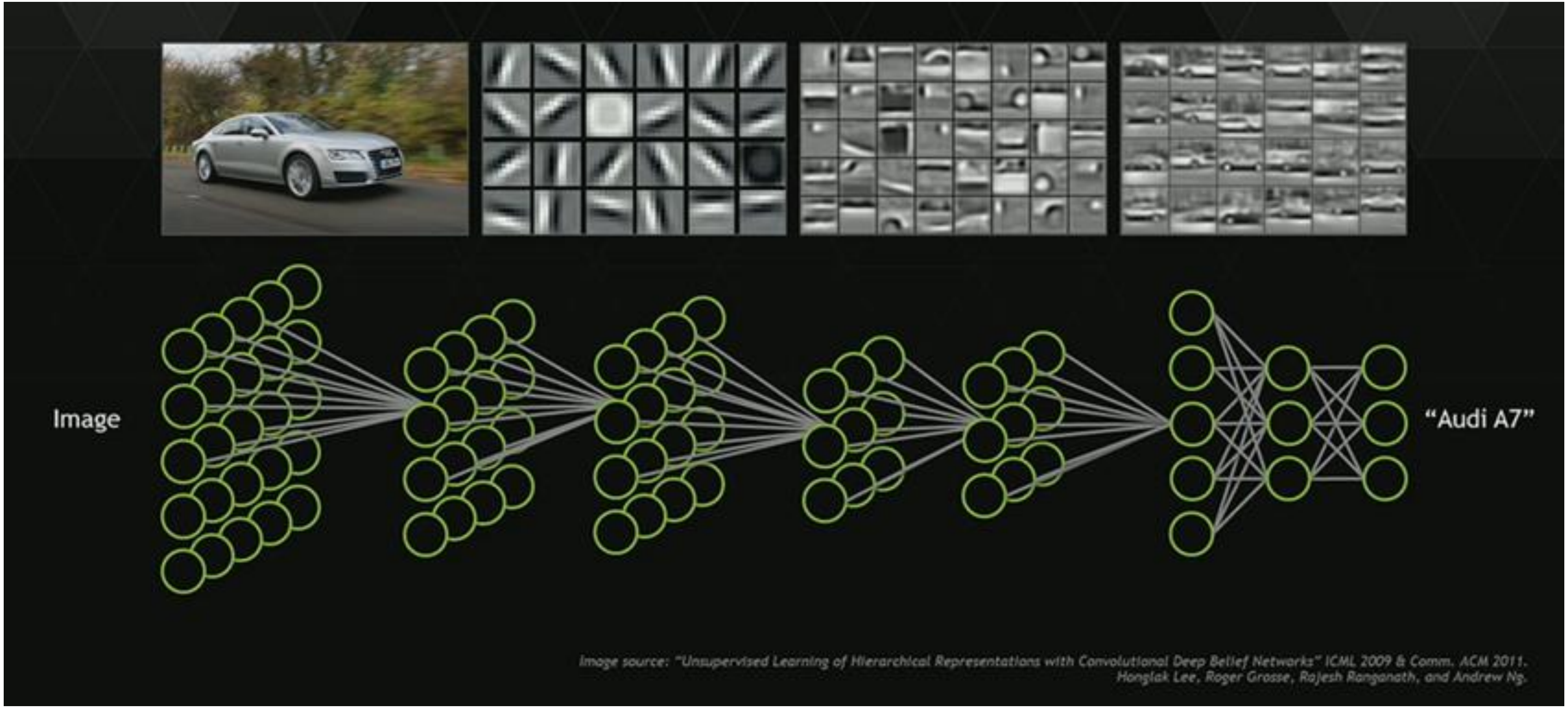
- However, this is a compromise! The model is learning its parameters from data yet we are still hand-programming the feature extractors ourselves. Can we make the entire process learned from *end-to-end*?
  - Yes! This is where deep learning comes in!
  - Learn the feature extractor as well — learn EVERYTHING in the entire pipeline!



Deep learning *is* Representation learning

- Deep learning allows a model to learn ***"good" representations*** directly from data!
  - The main idea is to ***relinquish all control to the model*** and let it learn whatever ***it*** feels is important to solve the task at hand
  - Features are synonymous with representations in ML
- The output of each layer in a neural network is a ***learned representation*** so deep learning can be viewed as the process of learning stacked representations
  - We call these representations ***hierarchical***, i.e., later representations depend on, and are more abstract than earlier ones — **depth refines representations**

Deep learning *is* Representation learning



Autoencoders

Let’s talk about JPEG

https://www.lifewire.com/the-effect-of-compression-on-photographs-493726

Why compress data?

- saves space
- send things over a network
- doesn’t have to be perfect! (lossy)

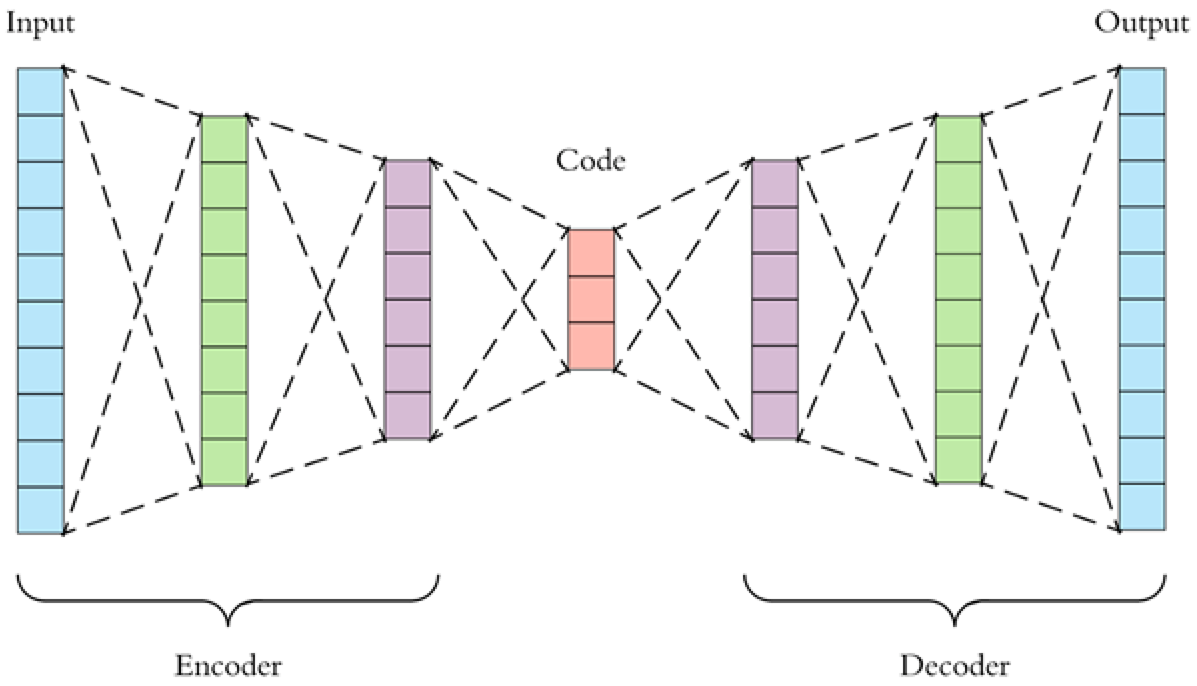
But why does JPEG work?

- nearby pixels are related
- very fine features are usually not important
- Fourier transforms!

What if we had a compression algorithm only for cats?

Eigenfaces (1987)
(https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184)

Autoencoders



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

Autoencoders (Recap)

- layer sizes shrink towards a bottleneck
  - bottleneck size should be small compared to # sample pts, # dimensions, etc.
  - enforces that it learns a small representation
- train using reconstruction loss
- variants:
  - sparse autoencoders
  - denoising autoencoders

What do we want from a code?

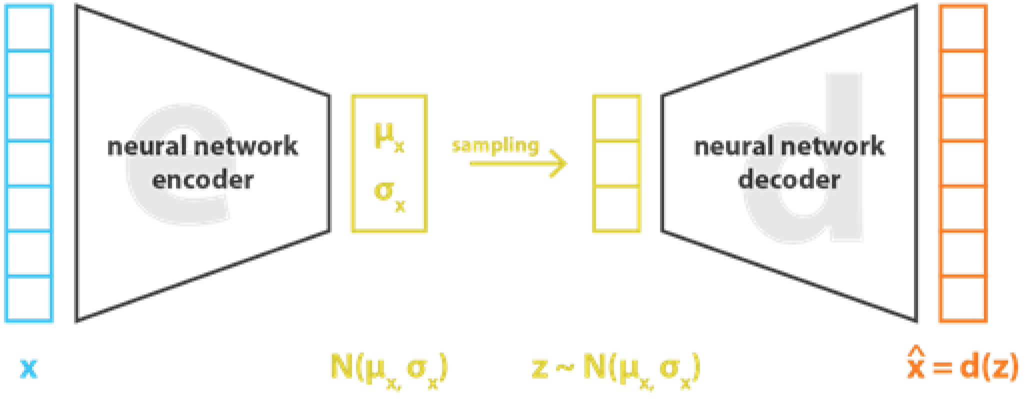
Some desirable properties…

- interpretable codewords
  - should be able to interpolate between codes
- ability to generate new images

Variational Autoencoders (VAE)

VAEs

- combine autoencoders with latent variable models
- you allow different data points to be associated with different priors over latent space (in a principled way)



$$\text{loss} = \| x - \hat{x} \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I)\,] = \| x - d(z) \|^2 + \mathrm{KL}[\, N(\mu_x, \sigma_x), N(0, I)\,]$$

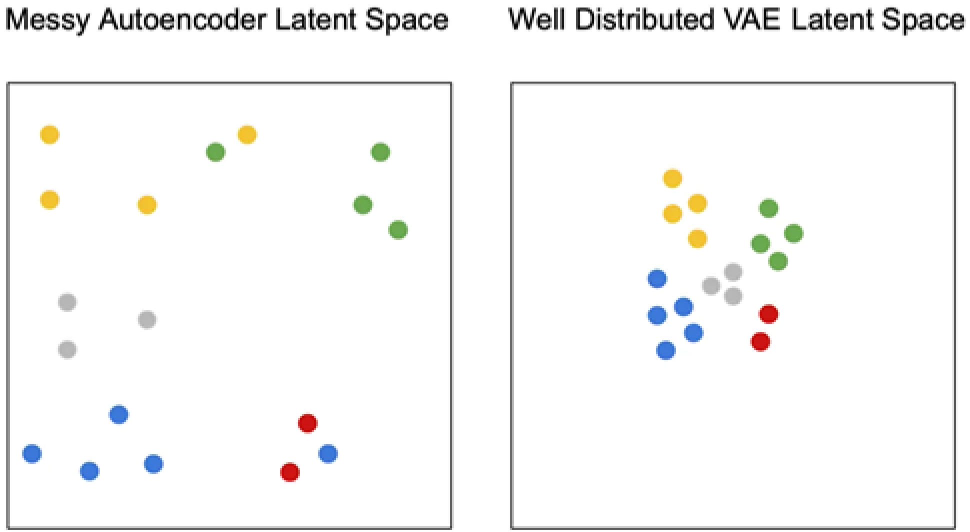
https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE Recap

- instead of outputting a single latent codeword from our encoder, we output a Gaussian
- train with a reconstruction loss and a regularizing term forcing the Gaussian to be close to standard

Structured Latents

- for regular AE, only the outputs of the encoder can be decoded
  - decoding a random latent might give garbage
- using probabilistic encoding enforces structured latents



https://mlberkeley.substack.com/p/vq-vae

How do we generate images with a VAE?

- need to choose a latent z and decode
- regularization term in loss makes z ~ N(0, I) a pretty good choice
- *advantage*: is that latents are interpretable
  - future lectures will focus on modeling the data distribution directly

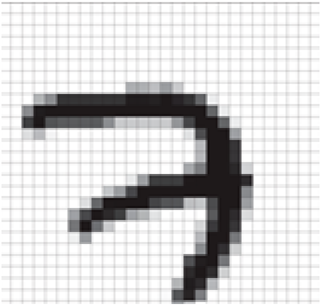
Motivating GANs

Fundamental Problem

- How can we generate realistic-looking data?

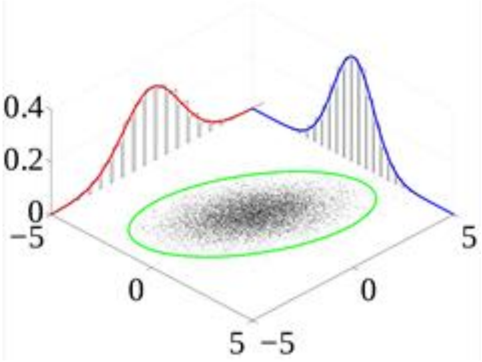
Given:

Generate:



Fundamental Problem

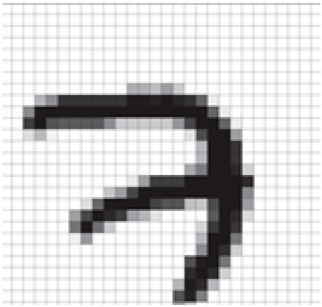
- More formally: How can we sample from a very complex distribution, when our only source of noise comes from “simple” distributions?
    - ie Gaussian, uniform, bernoulli, etc

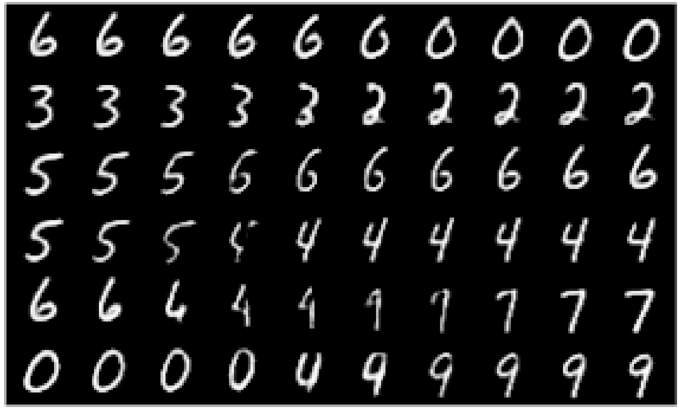


Latent Variables

- Sample data distribution is complex. How does our guess for pixel (0, 20) change if we know (31, 4)? What about if we know (0, 21)?
- We can’t just sample each pixel independently - leads to complete noise
- **Idea:** Hypothesize the existence of latent variables, and learn a mapping from latents to observation space

R^n ->

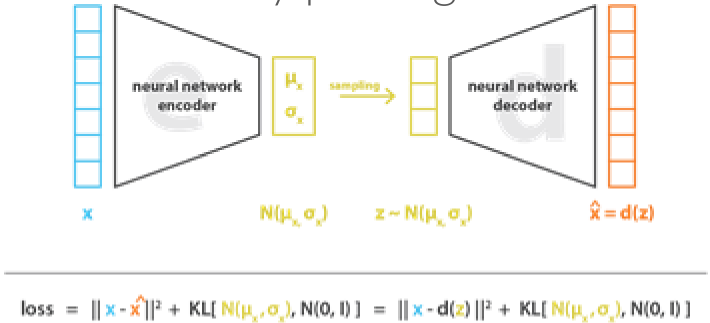




Interpolating over learned latents in MNIST

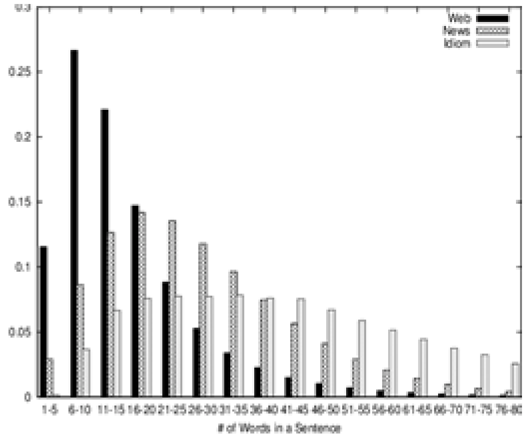
Latent Variables Review

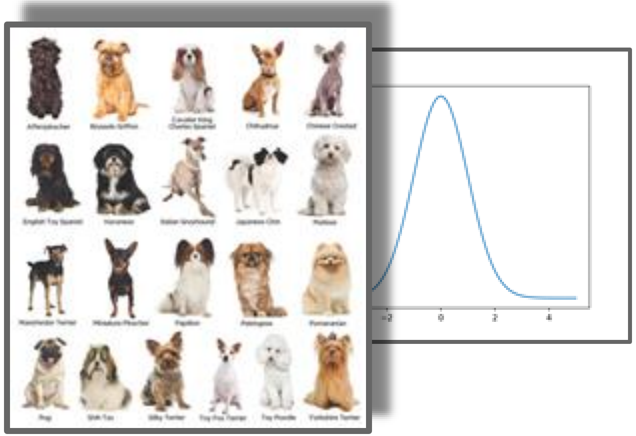
- A latent (aka a "code") space is a vector space that we sample a random vector from that is then passed into a generative model to generate some output
- Latent variable models are trying to learn a way to encode and decode data to and from this latent space
    - The latent vector can be uninterpretable (as in the case of a vanilla autoencoder)
    - We can encourage latents to have structure (ex. distributed like a Gaussian, as in the case of VAEs)
- We can also generate new data by passing unseen latents through a decoder



What we really want from our autoencoders

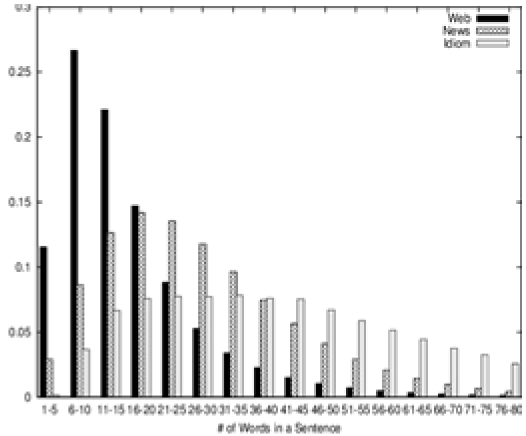
- We want an output that appears as if it was sampled from our data distribution
- In layman’s terms, this means that it should look like it could belong to our dataset
- How do we judge this directly?

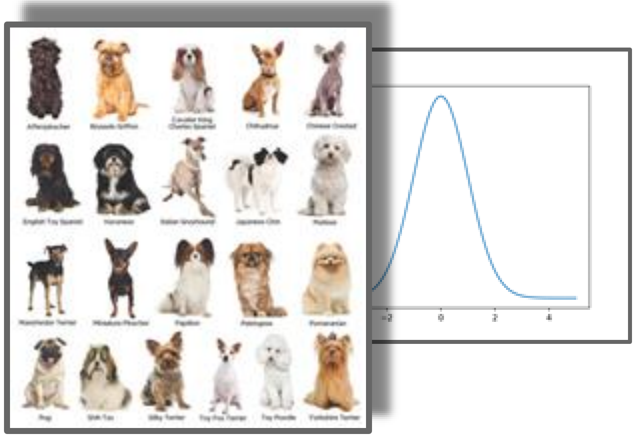




What we really want from our autoencoders

- When our data is something high dimensional like a sentence or an image, this can be really hard… how does one design a loss function that judges how similar a generated output is to the training dataset?
    - We often don’t have enough data to estimate the actual data distribution
    - We often don’t have enough compute to check if a sample belongs to a high-dim distribution





Solution

What if we just have another neural network try to judge how realistic our outputs are for us?