Autoencoders

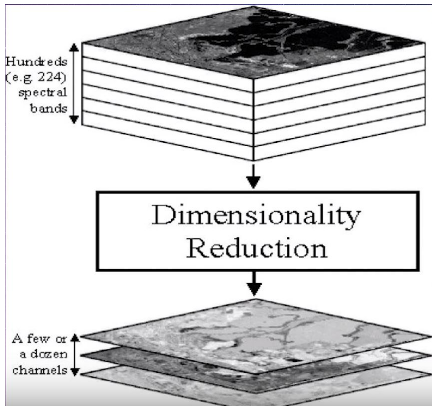
Daniel Marchena and Pedro Casas

What are autoencoders?

- Type of Artificial Neural Network
- Used to learn efficient data coding
    - Unsupervised (no labels)
    - Learn a **representation** (encoding) for a **set of data** typically for **dimensionality reduction**
    - Learning **generative models**
    - Could also be used for **image compression**

What is Dimensionality Reduction?

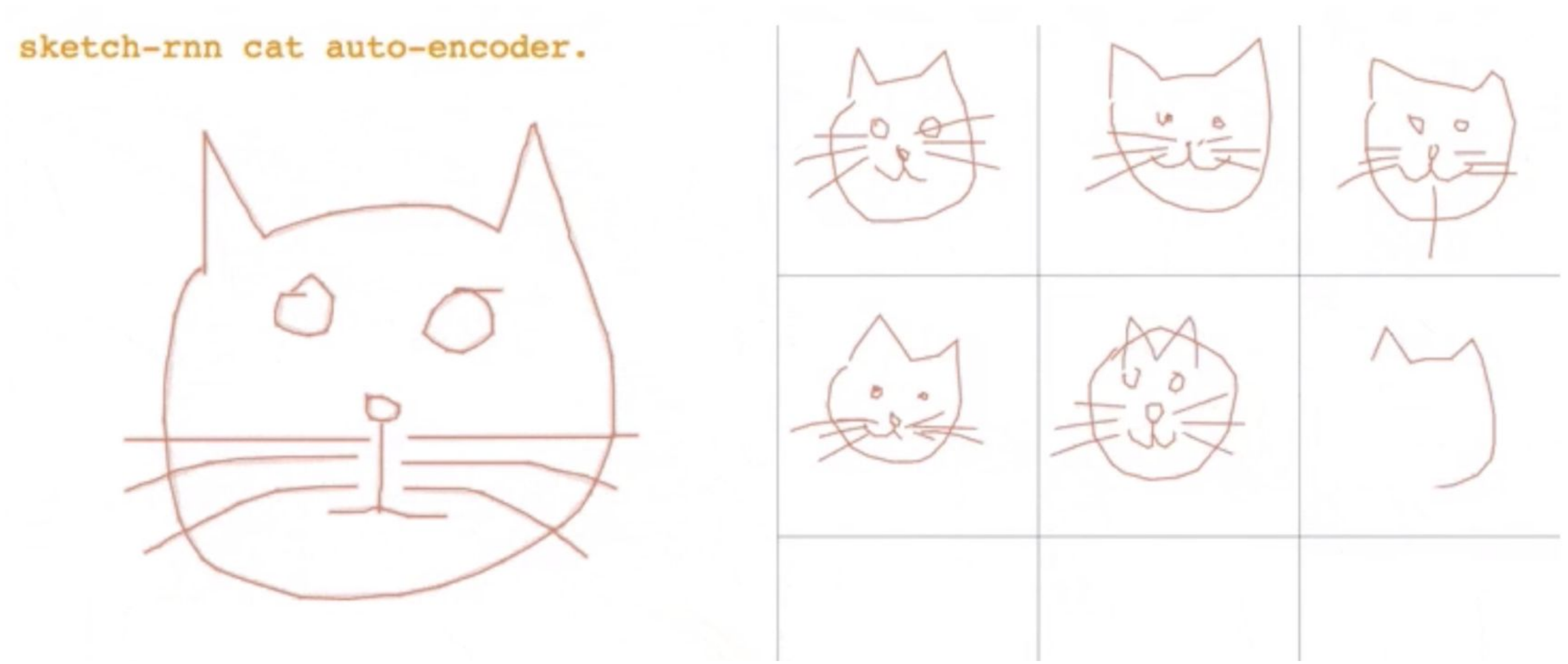
- The process of **reducing the number of random variables** under consideration by obtaining a set of principal variables
- Divided into feature selection and feature extraction



What are autoencoders used for?

- Widely used for learning **generative models** of data.
    - **New images, new text, etc...**
    - Given an observable variable X and a target variable Y, a generative model is a statistical model of the joint probability distribution on X × Y, P(X,Y)
        - that gives the probability that each of X, Y, ... falls in any particular range or discrete set of values specified for that variable
- Some of the most powerful AI in the 2010s have involved sparse autoencoders stacked inside of deep neural networks.

What are autoencoders used for?



What are autoencoders used for?

- An autoencoder learns to **compress data** from the input layer into a **short code**, and then **uncompress that code** into something that **closely matches the original data**.

- This forces the autoencoder to engage in **dimensionality reduction**, for example by learning how to **ignore noise**.

Example

- Some architectures use **stacked sparse autoencoder layers** for image recognition.

- The **first** autoencoder might **learn to encode easy features** like corners, the **second** to **analyze the first layer's output** and then encode less local features like the **tip of a nose**, the third might encode a whole nose, etc., until the final autoencoder encodes the whole image into a code that matches (for example) the concept of "cat".

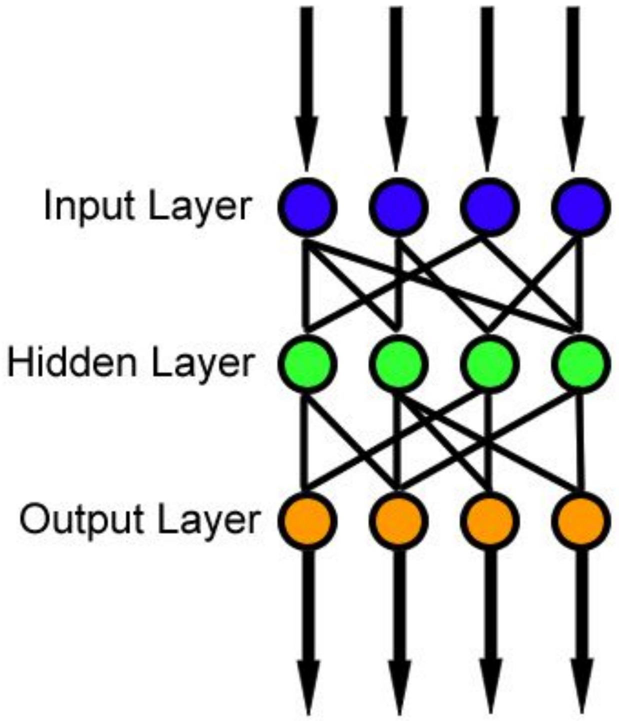
Another example

- An alternative use is as a **generative model**: for example, if a system is **manually fed the codes it has learned** for "cat" and "flying", it may attempt to **generate an image** of a flying cat, even if it has never seen a flying cat before.

Structure

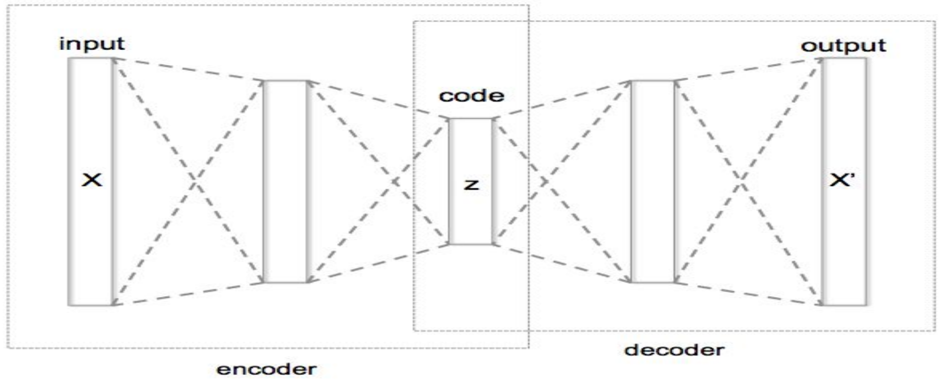
- Architecturally, the simplest form of an autoencoder is a **feedforward**, **non-recurrent neural network**
    - A **feedforward** neural network is an artificial neural network wherein connections between the nodes **do not form a cycle**.
        - You provide an input, and they provide an output. **No storage capability**.
        - **Recurrent neural networks have state**. This means that they **retain some of the activation from the previous input** and feed it in to the current calculations.
            - better suited for dealing with sequences, time series, etc

Structure



Structure - Multilayer perceptron

- An MLP consists of, at least, three layers of nodes: an **input layer**, one or more **hidden layers** and an **output layer**
- The **output layer** having the **same number of nodes as the input layer**, and with the **purpose of reconstructing its own inputs** (instead of predicting the target value Y given inputs X)



Structure - Multilayer perceptron

- Autoencoders are **unsupervised learning** models.
  - Learns from test data that has **not been labeled, classified or categorized**
  - Identifies **commonalities in the data** and **reacts** based on the **presence or absence of such commonalities** in each new piece of data

Structure

- An autoencoder always consists of **two parts**, the **encoder** and the **decoder**, which can be defined as transitions ϕ and ψ such that:

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

Structure

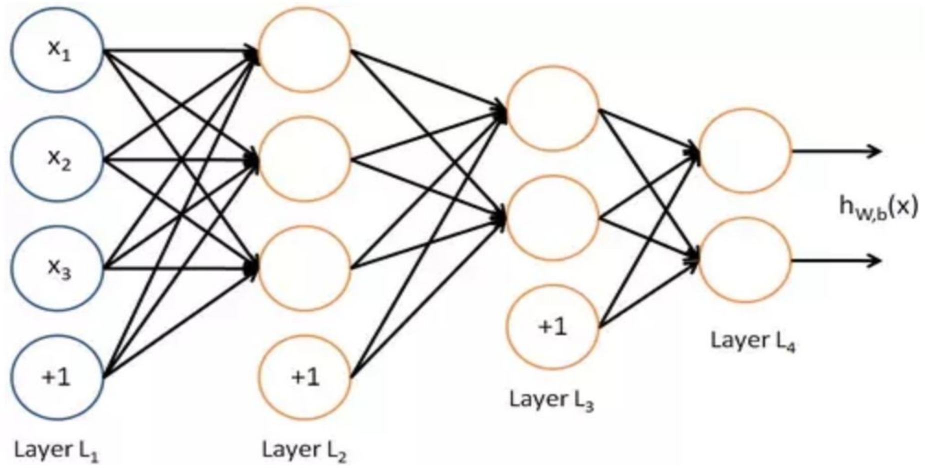
- In the simplest case, where there is one hidden layer.
- The **encoder stage** of an autoencoder **takes the input** $\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$ and **maps it to** $\mathbf{z} \in \mathbb{R}^p = \mathcal{F}$

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- This image $\mathbf{z}$ is usually referred to as **code**, **latent variables**, or **latent representation**
- Here, σ is an element-wise **activation function** such as a **sigmoid function** or a **rectified linear unit**.
- $\mathbf{W}$ is a **weight matrix** and $\mathbf{b}$ is a **bias vector**

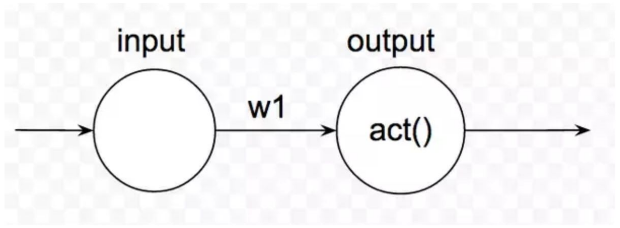
Bias unit

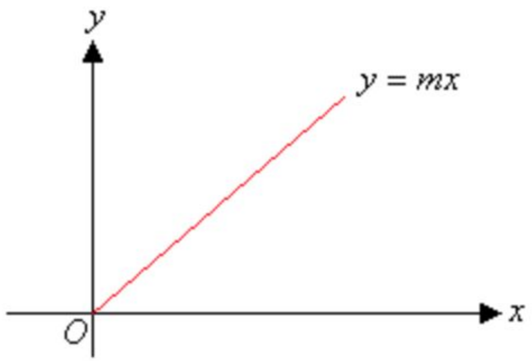
- A **bias unit** is an **"extra" neuron added to each pre-output layer** that stores the value of 1. **Bias units aren't connected to any previous layer** and in this sense don't represent a true "activity". They can contribute to the output of the ANN

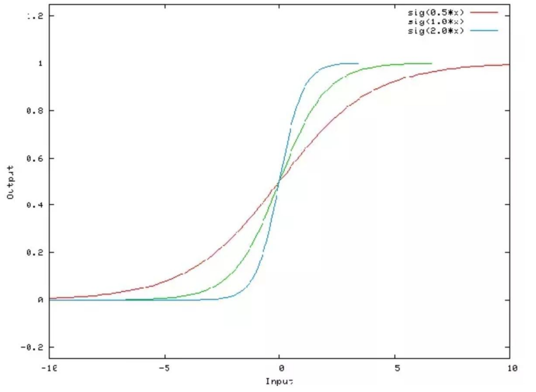


Bias unit

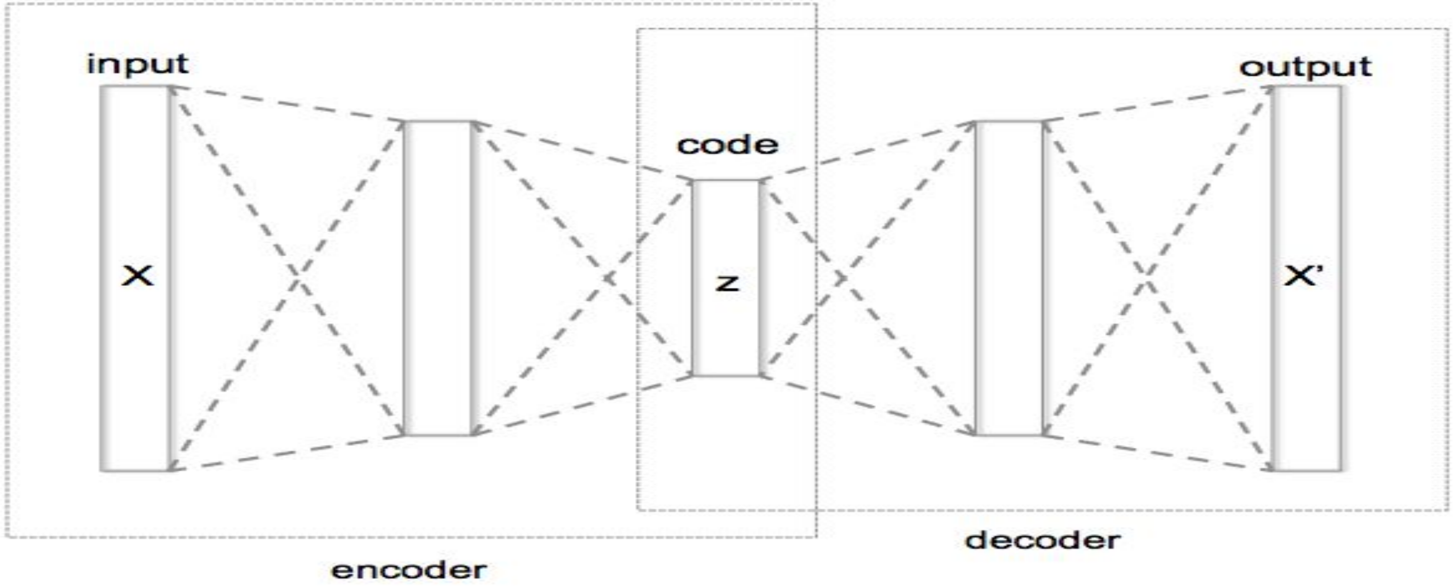
- When we **change our weight w1**, we will change the gradient of the function to make it **steeper or flatter**. But what about **shifting the function vertically?** In other words, what about setting the **y-intercept**. This is crucial for many modelling problems! Our optimal models **may not pass through the origin**.







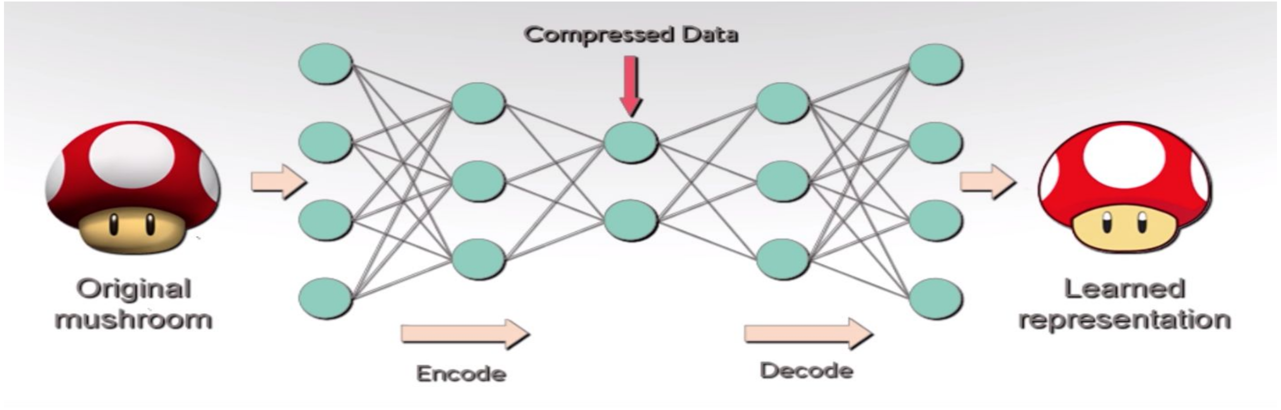
Structure



Structure

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

- Where σ', $\mathbf{W}'$, and $\mathbf{b}'$ for the decoder may differ in general from the corresponding σ, $\mathbf{W}$, and $\mathbf{b}$ for the encoder, depending on the design of the autoencoder.



Variations

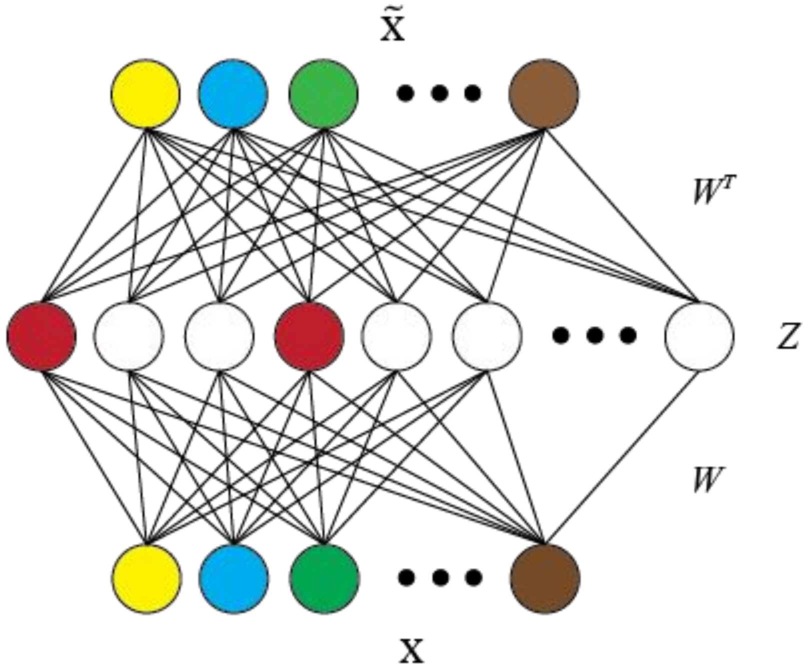
- Various techniques exist to prevent autoencoders from learning the identity function and to improve their ability to **capture important information and learn richer representations**

Training

To train an autoencoder we must minimize the construction error:

min||x−x˜||^2

Typically done through back-propagation



Pre-training

Autoencoders suffer from all the same problems as neural networks:

- Training can lead to poor local solutions
- Very slow to train
- Vanishing gradient: As the error is propagated backwards it gets very small

Hinton proposed a new way of pre-training the networks

Pre-training

Deep belief networks

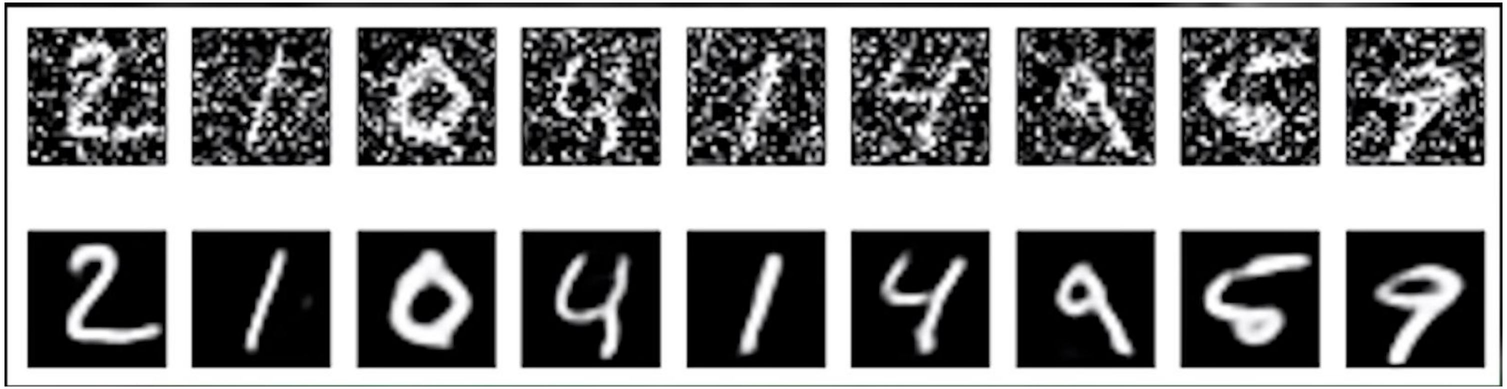
Treat each neighbouring set of 2 layers as a **restricted Boltzmann machine** to train.

Then apply backpropagation

A restricted Boltzmann machine is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

Denoising Autoencoder

- Take a partially corrupted input whilst training to recover the original undistorted input.
- This technique has been introduced with a specific approach to **good representation**



Denoising Autoencoder

- A **good representation** is one that **can be obtained robustly from a corrupted input** and that will be useful for recovering the corresponding clean input
  - The higher level representations are relatively stable and robust to the corruption of the input;
  - It is necessary to extract features that are useful for representation of the input distribution.
- To train an autoencoder to denoise data, it is necessary to **perform preliminary stochastic mapping** $\mathbf{x} \rightarrow \tilde{\mathbf{x}}$ in order to corrupt the data and use $\tilde{\mathbf{x}}$ as input for a normal autoencoder

Sparse Autoencoder

In sparse autoencoders there are more hidden units than inputs.

We introduce a new concept called the **sparsity constraint.**

This is used to lower the activations of the hidden layers.

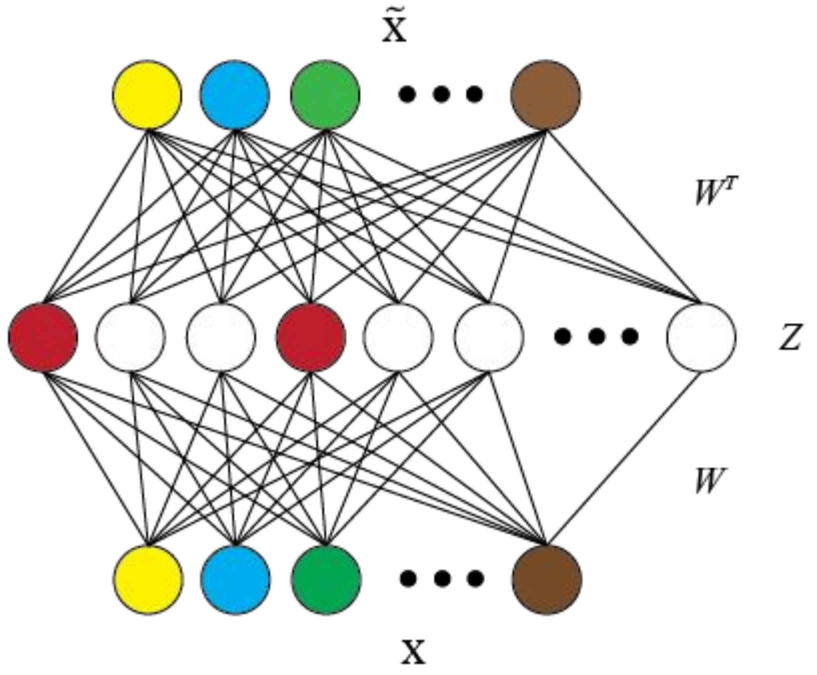
We want to lower how many nodes in the hidden layers get activated because then they can specialize to detect certain features.

Sparse Autoencoder

k-Sparse Autoencoders (Makhzani et al., 2013)

This implementation picks the k-highest activated hidden nodes and 0s out the rest.

The error is then only back-propagated through the active nodes



Sparse Autoencoder

Throughout training, k is set to a lower value each time so the appropriate neuron for each representation can be learned.

This increases the representation space for the possible inputs.

The features learned will also be more global with lower values of k.

For higher values of k the features learned become more specific.

Can be thought as introducing an information bottleneck

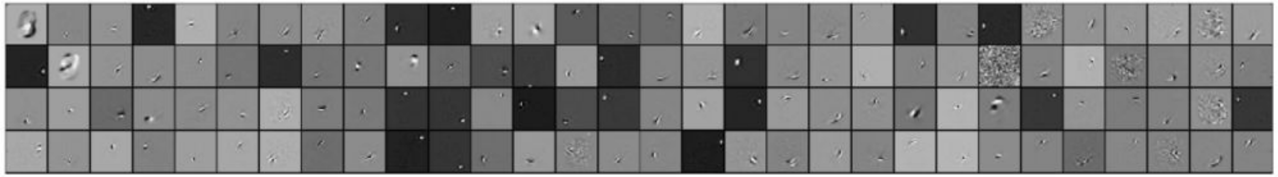
Example

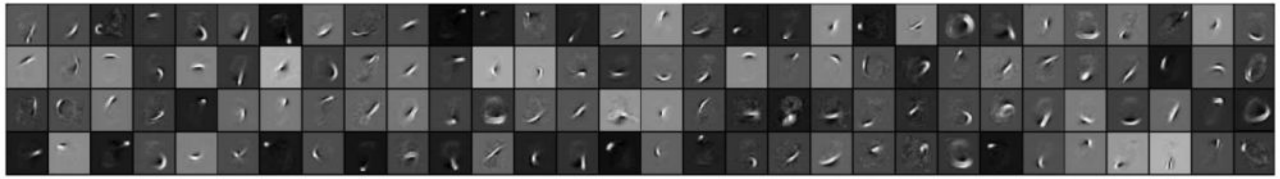
From (Makhzani et al., 2013)

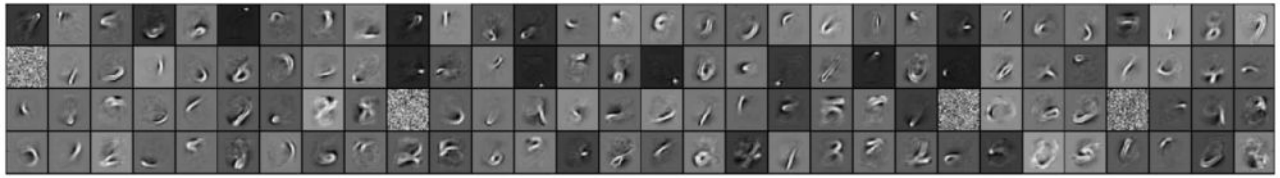
Each box corresponds to a learned feature

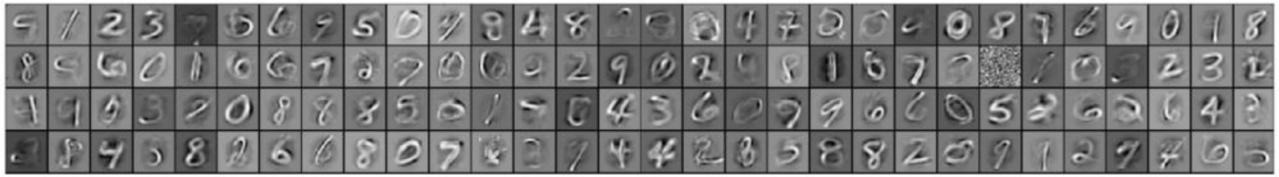
For higher values of k: more specific

Best results with k=40


(a) $k = 70$


(b) $k = 40$


(c) $k = 25$


(d) $k = 10$

Sparse Autoencoder L1 Regularization

Another way of introducing the sparsity constraint:

Apply an L1 normalization term to the loss function for activations

L1 norm: |a| = sqrt(a1^2 + a2^2 ... + an^2)

This prevents overfitting by penalizing very large values of individual as and favoring very small values of a, therefore introducing sparsity.

Contractive Autoencoder

Aim is to make the encoder robust to small changes in the input for representation.

Done by adding a regularizing term to the cost function for the encoder.

This is the Frobenius norm of the Jacobian matrix for the encoder activation sequence, with respect to the input

Contractive Autoencoder

Frobenius Norm - Vector norm, L2 norm $\|A\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}$

Jacobian matrix - matrix of all first-order partial derivatives of a vector-valued function.

Regularizing term: $||J_h(x)||_F^2 = \sum_{ij}\left(\frac{\delta h_j(x)}{\delta x_i}\right)$

Similarities

This will penalize large increases in the activation values with respect to the input.

Similar to sparse autoencoders (L1 norm) as it favours low values for activations.

Similar to denoising autoencoders as it promotes robustness. Difference is that denoisers favour the reconstruction and contractive autoencoders favour the encoder function.

Important to keep in mind for use case. (eg. Good for classification since it relies on the encoder.)

Relationship with PCA

PCA - converting a set of possibly correlated variables into uncorrelated values called principal components.

Autoencoders can be used to approximate these principle components.

This can be done by using linear activations and only one hidden layer in the autoencoder.

Relationship with PCA

The weights of the new Autoencoder with one layer can be used with **Singular Value Decomposition** to extract the principal components.

Singular Value Decomposition - Factorization of a real or complex matrix

From Principal Subspaces to Principal Components with Linear Autoencoders:

https://arxiv.org/abs/1804.10253

Advantages over PCA

PCA only is restricted to a linear map where Autoencoders can learn can learn non-linearities if more hidden layers are used.

Therefore, if you are using an autoencoder with one hidden layer, then it is probably better to use PCA to avoid training a NN and all of the other disadvantages that come with NNs.