Keras

Keras for Deep Learning Research

Feedback is greatly appreciated!

Unsupervised Learning

- Most of the world's data is unlabeled!
- New articles, movie reviews, user Netflix ratings, images on the internet, etc.
- What approaches can we take to make use of this type of unlabeled data?

Source: https://www.udemy.com/deeplearning/learn/v4/overview

Unsupervised Learning

- Two main approaches
  - Dimensionality Reduction
  - Clustering

Source: https://www.udemy.com/deeplearning/learn/v4/overview

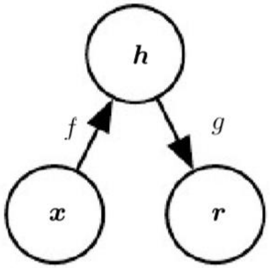
Autoencoder

- The autoencoder is actually a very simple neural network and will feel similar to a multi-layer perceptron model.
- It is designed to reproduce its input at the output layer.
- The key difference between an autoencoder and a typical MLP network is that the number of input neurons is equal to the number of output neurons.

Source: https://www.udemy.com/deeplearning/learn/v4/overview

Autoencoder

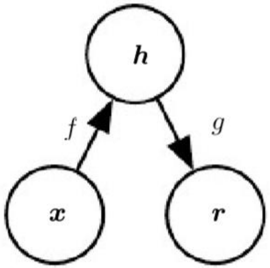
- An autoencoder is a neural network that has three layers: an input layer, a hidden (encoding) layer, and a decoding layer.
- The network is trained to reconstruct its inputs, which forces the hidden layer to try to learn good representations of the inputs.



Source: http://www.shogun-toolbox.org/static/notebook/current/autoencoders.html

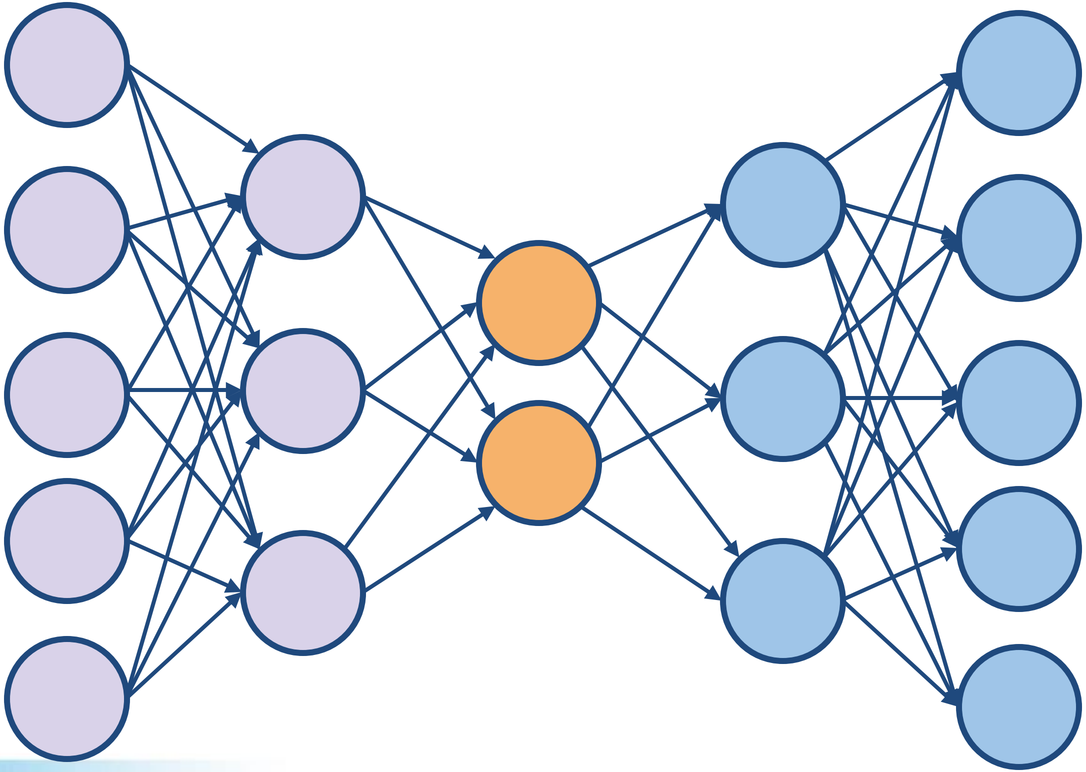
Autoencoder

- The general structure of an autoencoder, mapping an **input x** to an **output r** (called **reconstruction**) through an internal representation or **code h**.
- The autoencoder has two components: the encoder f (mapping x to h) and the decoder g (mapping h to r).



Source: http://www.deeplearningbook.org/contents/autoencoders.html

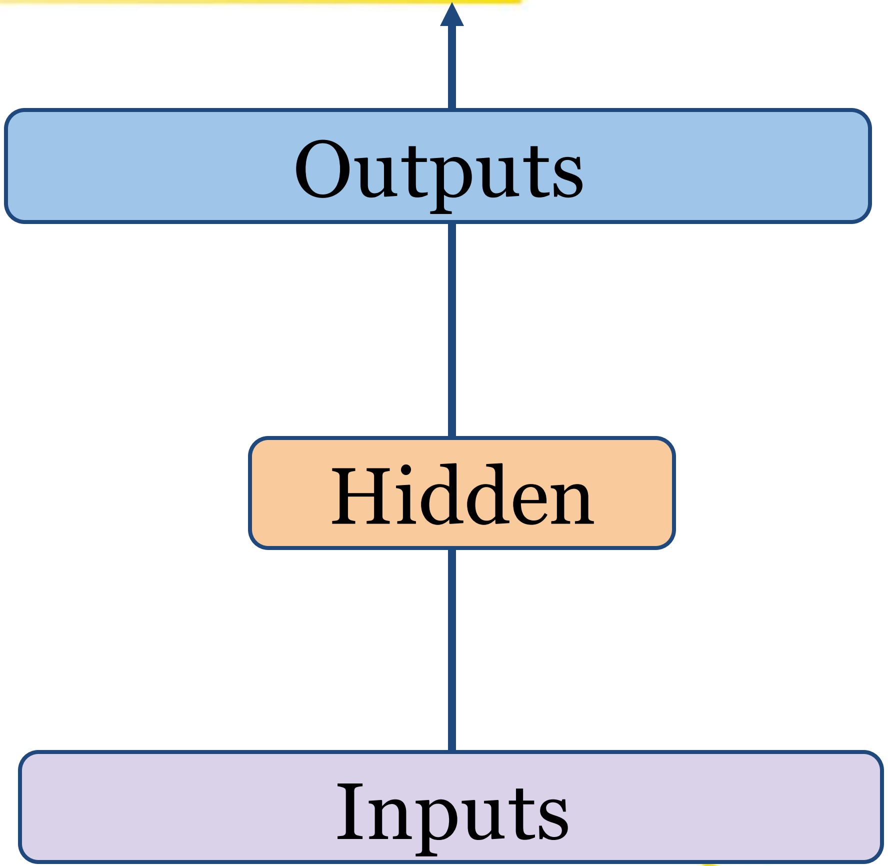
Example Autoencoder



Source: https://www.udemy.com/deeplearning/learn/v4/overview

Autoencoder

- Feed forward network trained to reproduce its input at the output layer.
- Output size is the same as the input layer.



Source: https://www.udemy.com/deeplearning/learn/v4/overview

UMKC

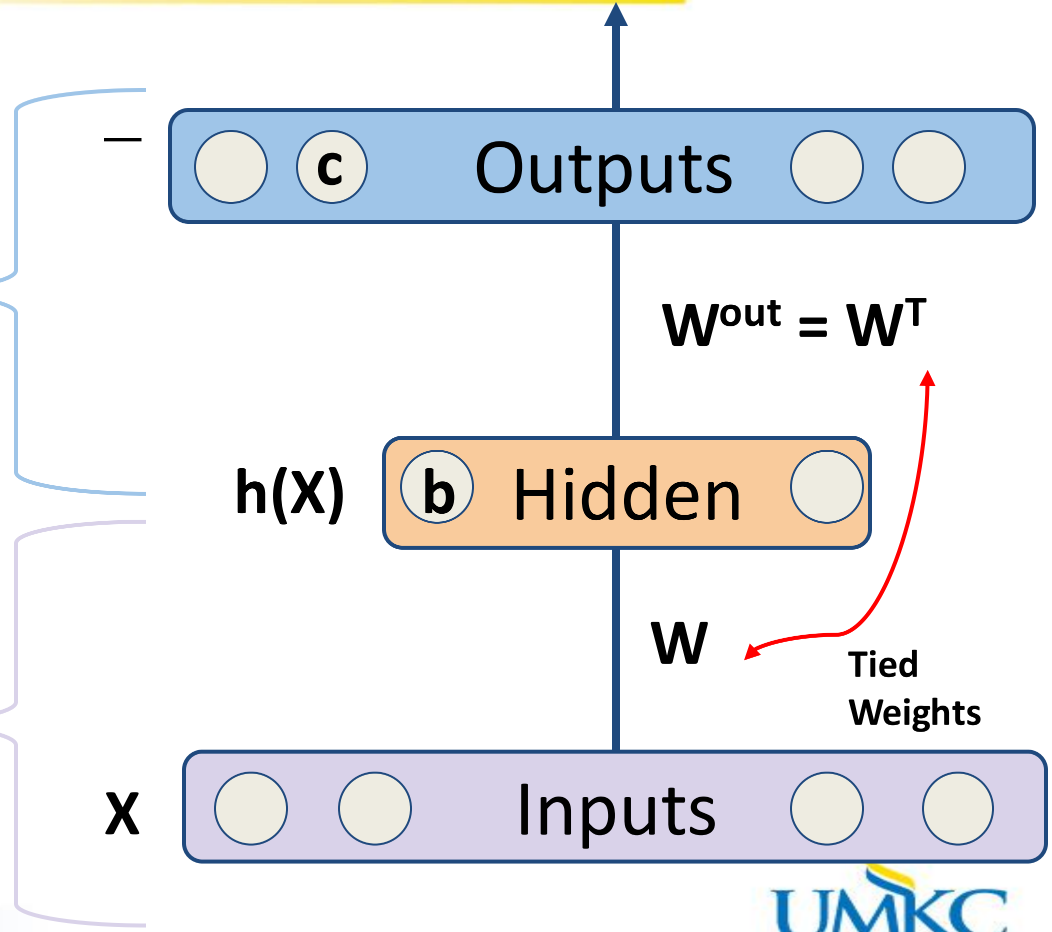
Autoencoder

Decoder

$= \mathrm{sigm}(c + W^{out}\, h(x))$

Encoder

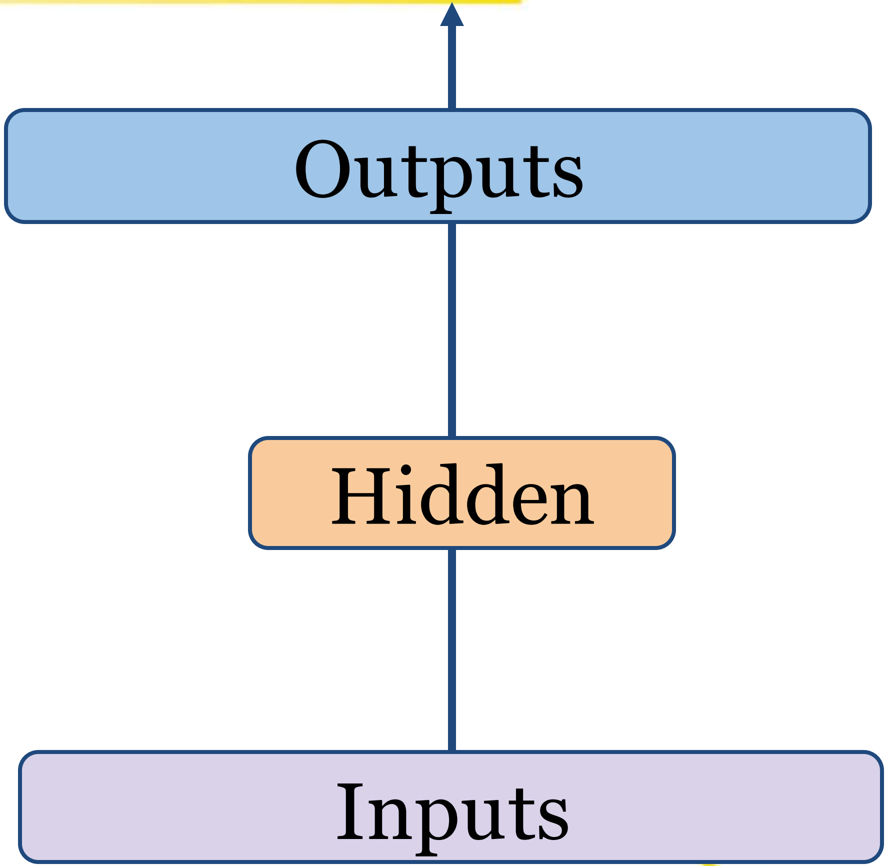
$h(x) = \mathrm{sigm}(b + Wx)$



Source: https://www.udemy.com/deeplearning/learn/v4/overview

Autoencoder

- The hidden/internal representation maintains all the information of the input.
- We can use the hidden layer to extract meaningful features.



Source: https://www.udemy.com/deeplearning/learn/v4/overview

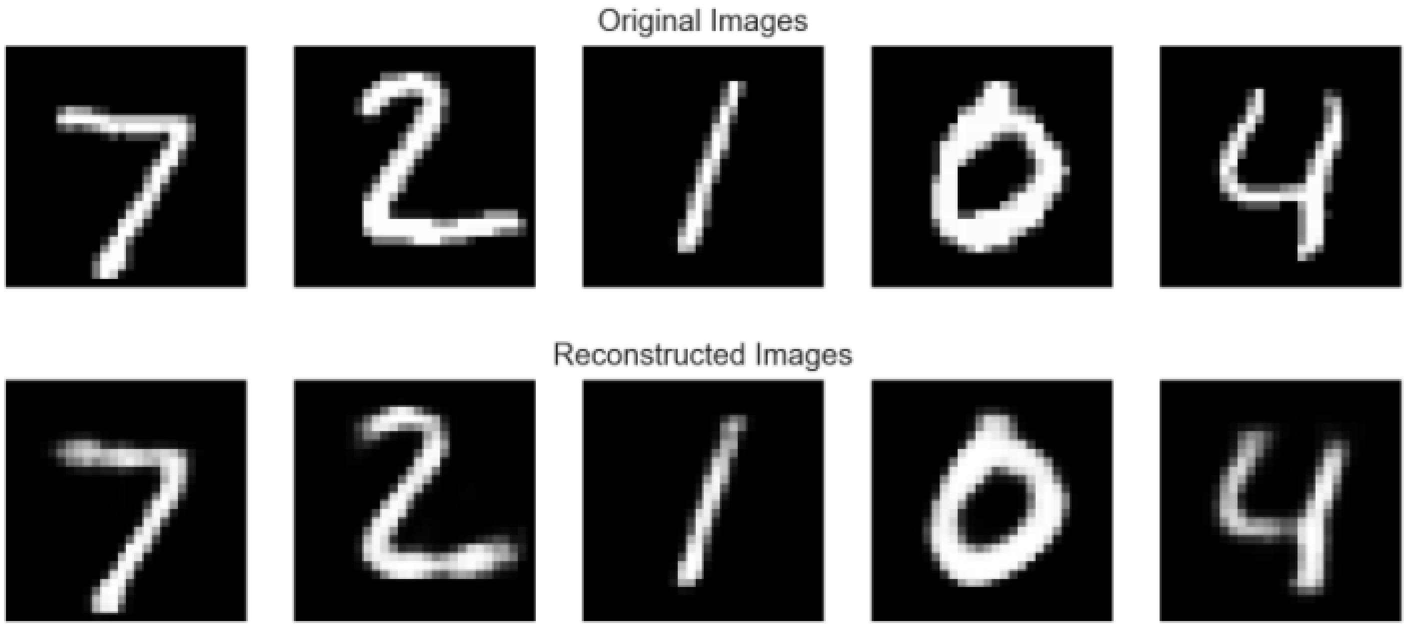
Applications of Autoencoder

- Dimensionality Reduction
- Data Denoising
- Image Reconstruction
- Information Retrieval

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

UMKC

Dimensionality Reduction



Source: https://github.com/

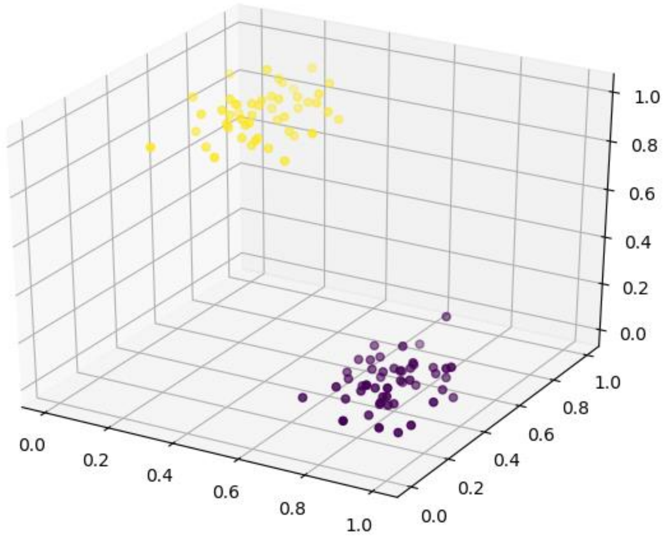
Dimensionality Reduction

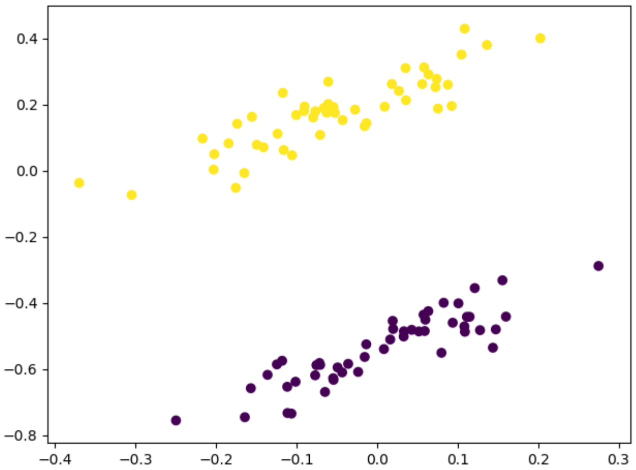
- Dimensionality reduction allows us to get a lower dimension representation of our data.
- The encoder creates new (fewer) features from the input features.
- For example we can input a 3 dimensional data set and output a 2 dimensional representation of it.

Source: https://www.udemy.com/deeplearning/learn/v4/overview

UMKC

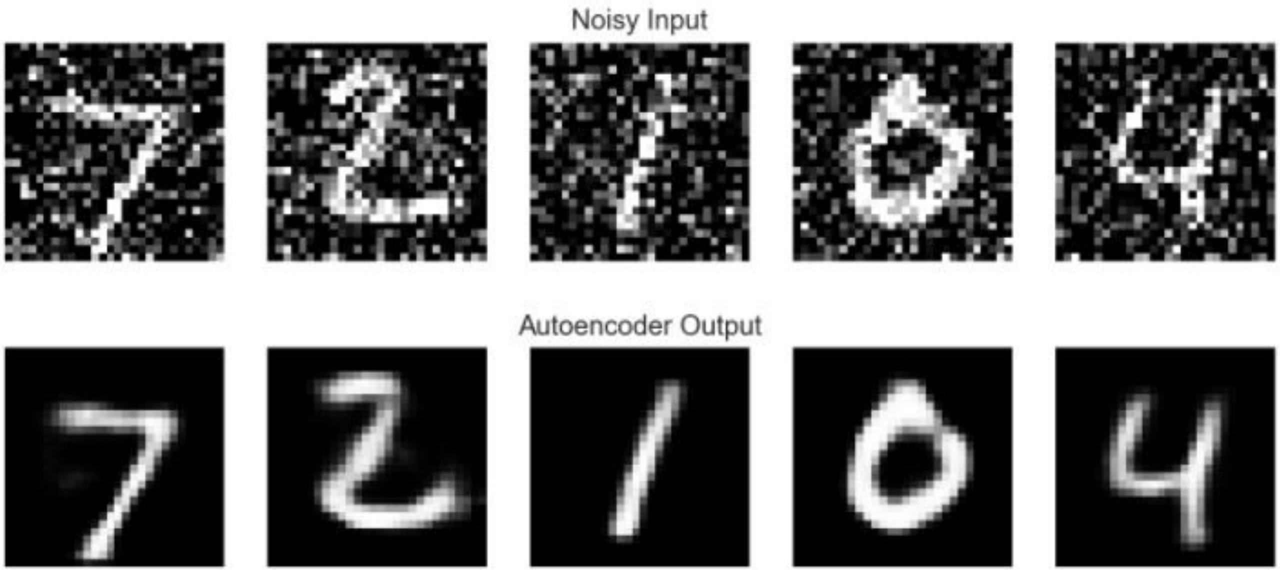
Dimensionality Reduction





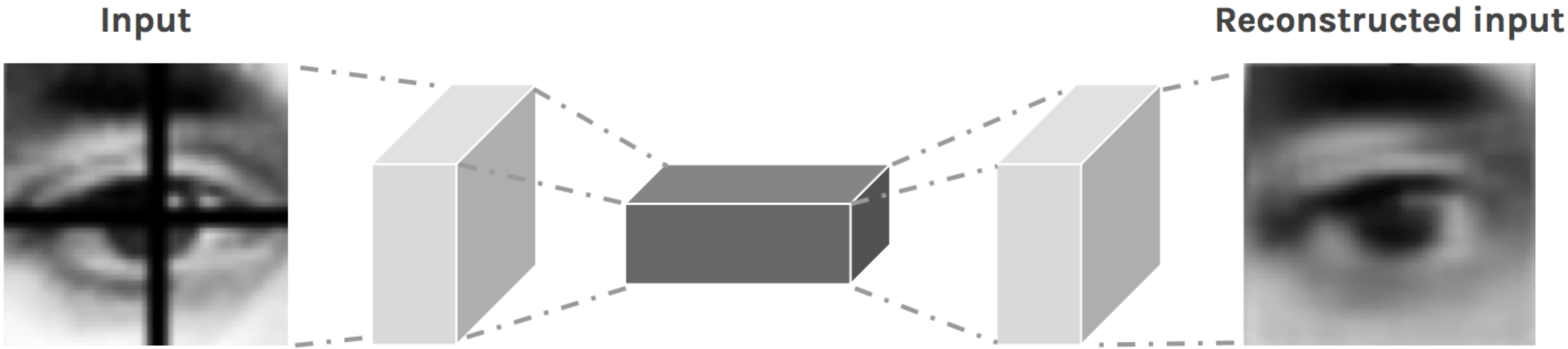
Source: https://github.com/

Data Denoising



Source: https://github.com/

Image Reconstruction



Removes the dark cross-bar in the image

Source: https://github.com/

Information Retrieval

- This is related to text domain
- We can use an autoencoder to find low dimensional codes for documents that allow fast and accurate retrieval of similar documents from a large set.

Source: https://github.com/

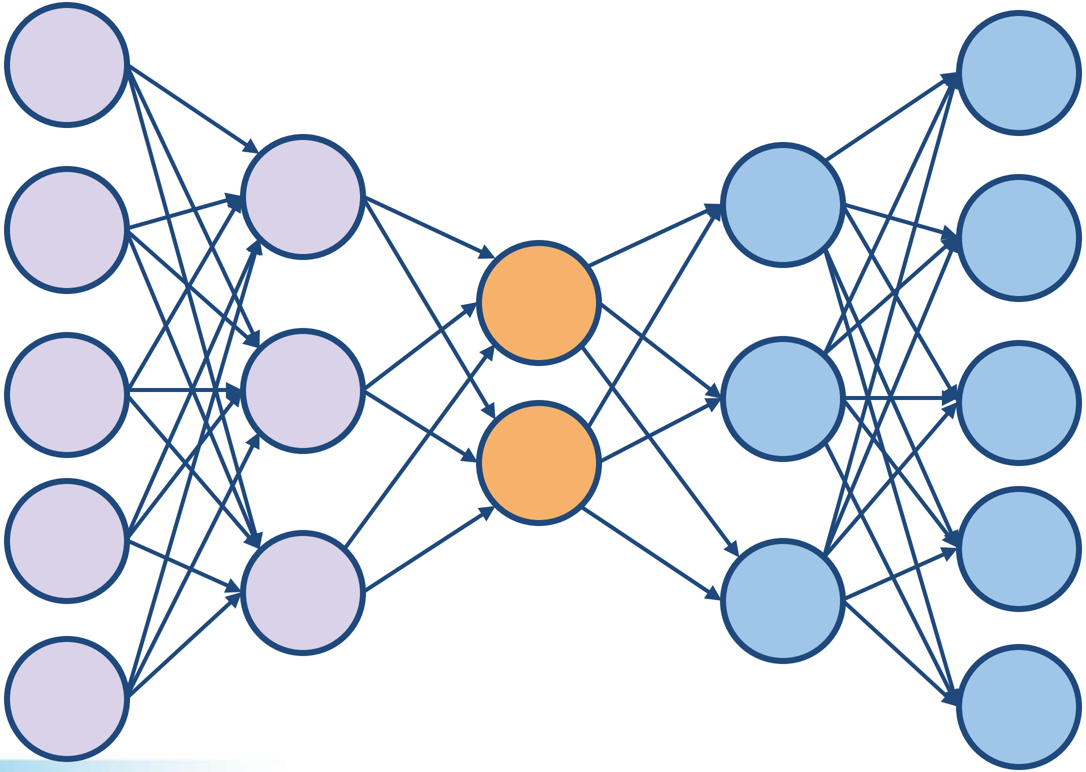
Autoencoder vs PCA

- PCA is restricted to a linear map, while auto encoders can have nonlinear encoder/decoders.
- A single layer auto encoder with linear transfer function is nearly equivalent to PCA, where nearly means that the **W** found by AE and PCA won't be the same--but the subspace spanned by the respective **W's** will.

Source: https://www.cs.toronto.edu/~urtasun/courses/CSC411/14_pca.pdf

UMKC

Stacked Autoencoder



Source: https://www.udemy.com/deeplearning/learn/v4/overview

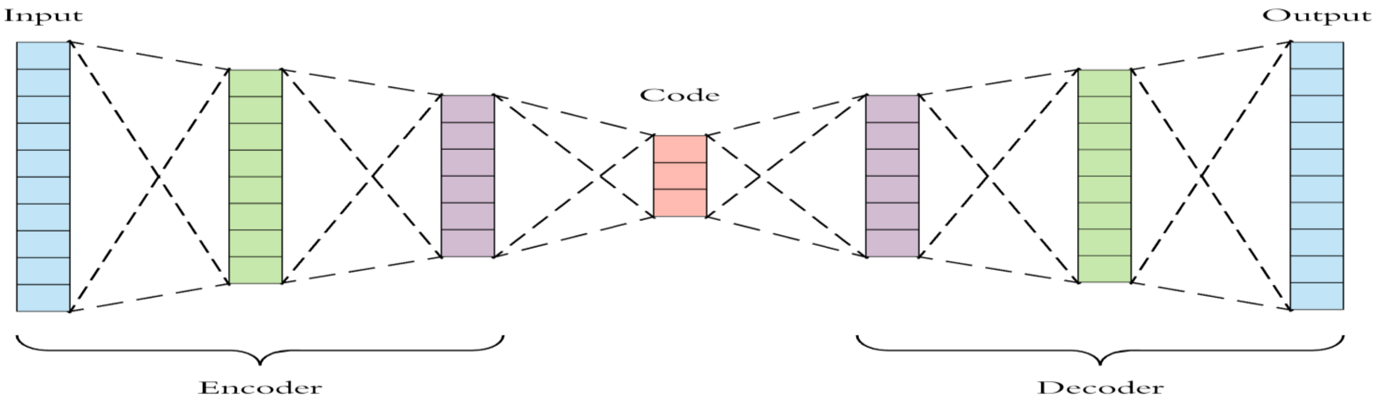
UMKC

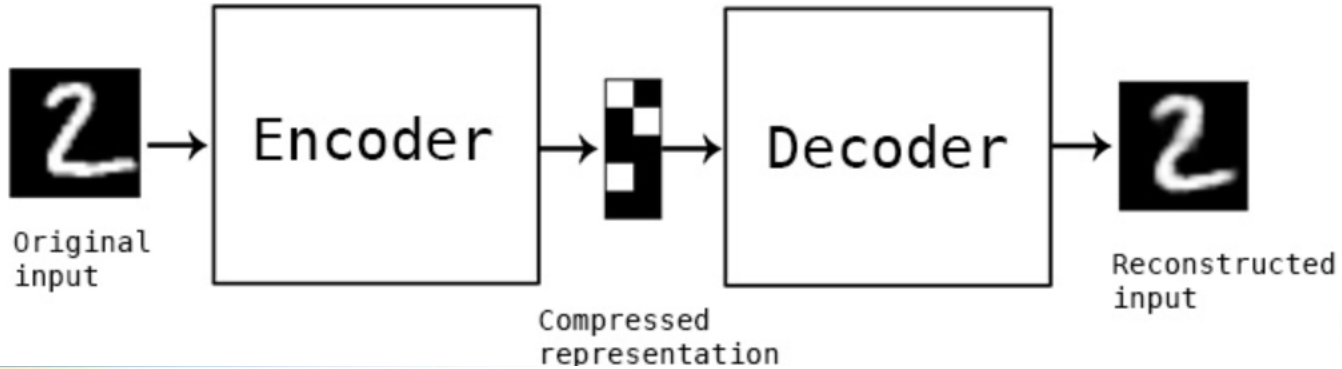
Stacked Autoencoder

- A stacked autoencoder is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer is wired to the inputs of the successive layer.
- Then the encoding step for the stacked autoencoder is given by running the encoding step of each layer in forward order: $z^{(l+1)} = W^{(l,1)} a^{l} + b^{(l,1)}$
- The decoding step is given by running the decoding stack of each autoencoder in reverse order: $z^{(n+l+1)} = W^{(n-l,2)} a^{(n+l)} + b^{(n-l,2)}$

Source: http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Architecture of Autoencoder





UMKC

Architecture of Autoencoders

- Both the encoder and decoder are fully-connected feedforward neural networks
- Code is a single layer of an ANN with the dimensionality of our choice.
- The number of nodes in the code layer (**code size**) is a hyperparameter that we set before training the autoencoder.
- Note that the decoder architecture is the mirror image of the encoder.
- The only requirement is the dimensionality of the input and output needs to be the same. Anything in the middle can be played with.

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

Hyperparameters of Autoencoder

- Code size
- Number of layers
- Number of nodes per layer
- Loss function

UMKC

Types of Autoencoder

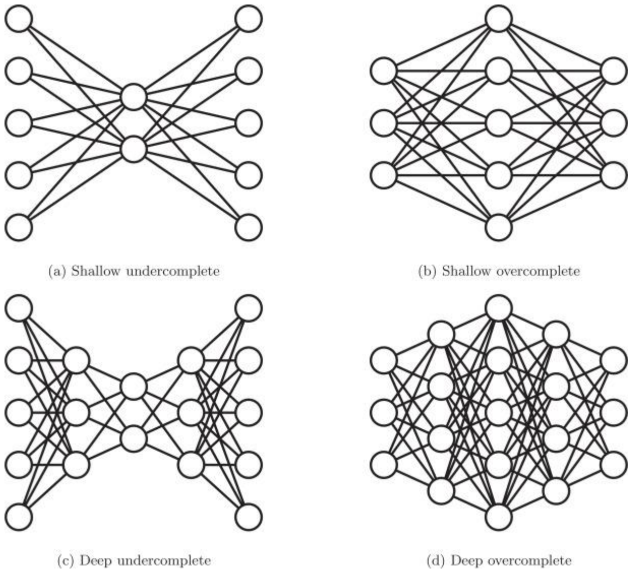
- **Undercomplete Autoencoder:** An autoencoder whose **code** dimension is **less than** the input dimension is called undercomplete. Learning an undercomplete representation forces the autoencoder to capture the most salient features of the training data.

- **Overcomplete Autoencoder:** An autoencoder whose **code** dimension is **greater than** the input dimension is called overcomplete.

- **Regularized Autoencoders:** Rather than limiting the size of the code dimension for the sake of feature learning, we can add a loss function to prevent it memorizing the task and the training data.

Source: http://machinelearninguru.com/deep_learning/tensorflow/neural_networks/autoencoder/autoencoder.html

Undercomplete and Overcomplete AE



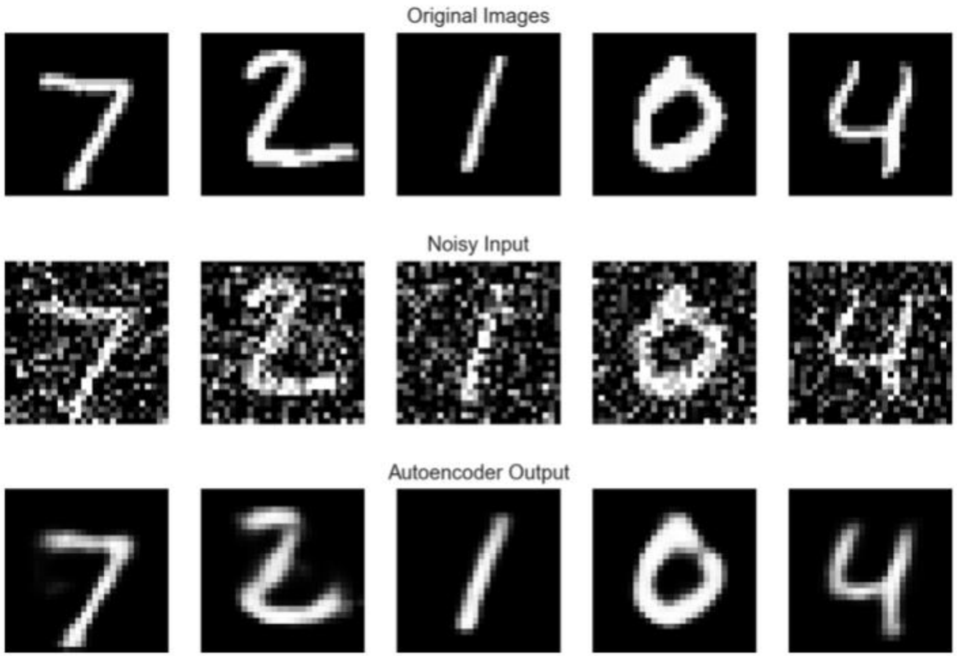
Source: http://machinelearninguru.com/deep_learning/tensorflow/neural_networks/autoencoder/autoencoder.html
https://www.sciencedirect.com/science/article/pii/S1566253517307844

Types of Regularized Autoencoders

- Denoising Autoencoder (DAE)
- Sparse Autoencoder (SAE)
- Variational Autoencoder (VAE)
- Contractive Autoencoder (CAE)

Source: http://machinelearninguru.com/deep_learning/tensorflow/neural_networks/autoencoder/autoencoder.html
https://en.wikipedia.org/wiki/Autoencoder#Variations

Denoising Autoencoder



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

Denoising Autoencoder

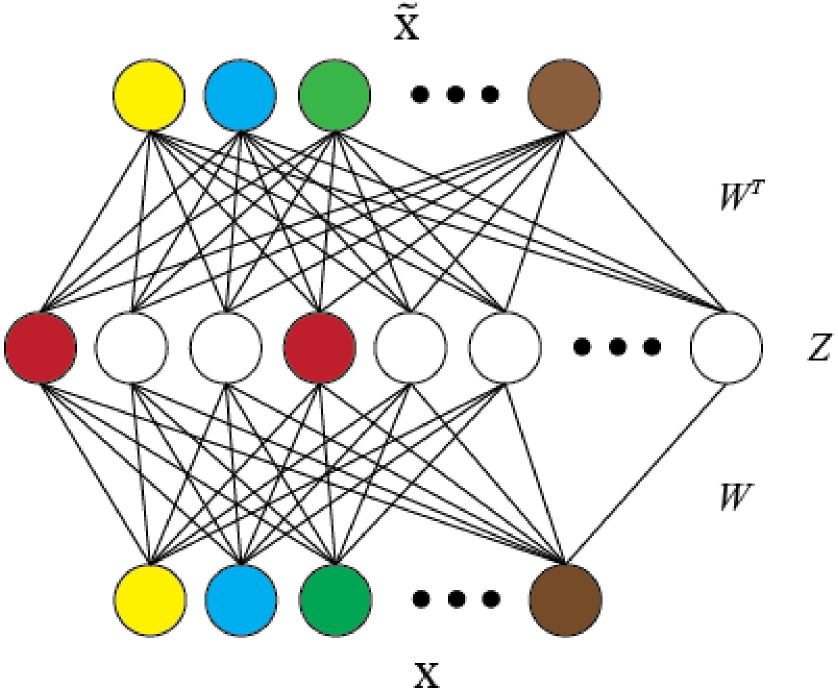
- There is another way to force the autoencoder to learn useful features, which is adding random noise to its inputs and making it recover the original noise-free data.
- This way the autoencoder can’t simply copy the input to its output because the input also contains random noise.
- We are asking it to subtract the noise and produce the underlying meaningful data.
- This is called a denoising autoencoder.

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

Different kinds of noises

- Salt and Pepper Noise
- Gaussian Noise
- Periodic Noise
- Speckle Noise

Sparse Autoencoder



Source: http://www.ericlwilkinson.com/blog/2014/11/19/deep-learning-sparse-autoencoders

UMKC

Sparse Autoencoder

- An autoencoder which has a sparsity penalty in the training loss in addition to the reconstruction error.
- We can regularize the autoencoder by using a sparsity constraint such that only a fraction of the nodes would have nonzero values, called active nodes.
- In particular, we add a penalty term to the loss function such that only a fraction of the nodes become active.
- This method works even if the code size is large, since only a small subset of the nodes will be active at any time.

Source: http://machinelearninguru.com/deep_learning/tensorflow/neural_networks/autoencoder/autoencoder.html

Usage of callbacks

- A callback is a set of functions to be applied at given stages of the training procedure
- You can use callbacks to get a view on internal states and statistics of the model during training

Usage of callbacks

- **TerminateOnNaN:** Callback that terminates training when a NaN loss is encountered.
- **History:** Callback that records events into a History object
- **ModelCheckpoint:** Save the model after every epoch.
- **EarlyStopping:** Stop training when a monitored quantity has stopped improving.
- **LearningRateScheduler:** Learning rate scheduler.
- **ReduceLROnPlateau:** Reduce learning rate when a metric has stopped improving.

Use Case: Fashion_MNIST data set

Creating model

```
encoding_dim = 32

# this is our input placeholder
input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Reading data set

- First, we'll configure our model to use a per-pixel binary crossentropy loss, and the Adadelta optimizer:

```
from keras.datasets import fashion_ mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
```

Fitting model

- We will normalize all values between 0 and 1 and we will flatten the 28x28 images into vectors of size 784.
- ```
  x_train = x_train.astype('float32') / 255.
  x_test = x_test.astype('float32') / 255.
  x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
  x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

  autoencoder.fit(x_train, x_train,
                  epochs=5,
                  batch_size=256,
                  shuffle=True,
                  validation_data=(x_test, x_test))


  prediction = autoencoder.predict(X_test[0])
  ```
  ```

Use case2: Denosing Autoencoder

- The only difference is that we feed a noisy data to the model:
- Noise factor controls the noisiness of images
- clip the values to make sure that the elements of feature vector representing image are between 0 and 1

```
#introducing noise
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

Visualizing the train data

```
from matplotlib import pyplot as plt
plt.imshow(x_train[1].reshape(28,28))
plt.show()
```

```
from matplotlib import pyplot as plt
plt.imshow(prediction.reshape(28,28))
plt.show()
```

References

- http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/
- https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd

UMKC