# Visualizing Networks, TensorBoard, Understanding data

11785 / Fall 2018/ Recitation 4

## Installation

- Installation
  - Comes pre-installed with TensorFlow
  - It’s recommended to use “conda” to manage a TF environment
    - `conda create -n <env_name> anaconda python=3.6`
    - `source activate <env_name>`
    - `conda install -c conda-forge tensorflow`
- Please follow the link to download today’s notebook

“https://github.com/cmudeeplearning11785/Fall2018-tutorials ”

## Why visualization is important?

- Helps you answer “What am I learning?”
- Looking at the weight matrices and their gradients change, you can
  - Remove extra layers upon seeing redundancy in the matrices.
  - Add extra layers to see if they learn something unique.
- Problem of vanishing gradients/exploding gradients comes attached with recurrent networks.
  - Adjust the Learning Rate (LR).
  - Understand how changing the LR affects your loss and accuracy.
- When to stop training?
  - Better to use tools than always logging either loss or prediction accuracy of dev sets.

## Why visualization is important?

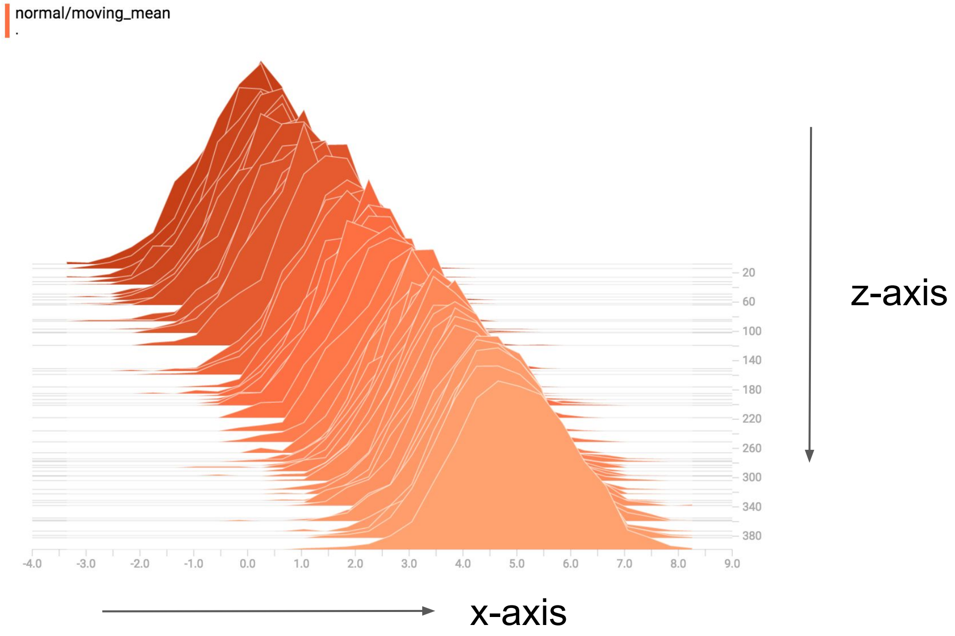
- Weight initialization
  - Not easy to understand which weight initialization performs better than the other using standard logging practices.
  - You get to see why initializing your weights/biases to zero is **not** preferred.
- Is your Softmax, Activation Functions performing?
- Is your Dropout rate is too high?
- In general, it helps you fine tune your network for optimal performance.

## TensorBoard

- A Visual Logger.
- Helps with understanding, debugging and optimizing
- Among many, tf.summary()
    - It's a public API available for use in multiple deep learning frameworks (tf.Summary() class for logging in other frameworks including pytorch)
    - Lets you log "operands" (think of it as nodes in your data flow graph) in a user defined directory
    - TB does the rest.
- TB can log "scalars", "histograms", "images", "text" and more.
- "scalars" tf.summary.scalar()
    - As the name suggest, plots 1-D operand on a 2-D surface where "x-axis" is time and "y-axis" is the value of the operand itself.
    - Keep track of "loss", "prediction_accuracy", "learning_rate", any scalar with time/number_of_epochs.

# TensorBoard

- "histograms" tf.summary.histogram()
  - Generally, all weights/biases are matrices, 2D. Represented through a histogram in 3D.
  - "z-axis" is time, "x-axis" is the value of operand, "y-axis" is the frequency/count.



# TensorBoard

- “images” tf.summary.images()
    - It’s helpful to log training images and visualize which images are difficult for your model to understand by looking at loss function.
    - A single matrix is an image where batch size is 1, height/width corresponds to the rows/columns and number of channels is 1, ie. Any 2D matrix can be represented in 4 dimensions.
    - You can plot weight matrices, CNN filters/kernels and even a confusion matrix
    - *tf.summary.image* takes care of normalizing the matrix values between [0-255]

## I am training on AWS!

- TensorBoard is pre-installed in the environment
  - Activate using `source activate tensorflow_p36`
  - Activate using `source activate pytorch_p36`
- Local Port Forwarding (-L) when you “ssh” in a remote instance.
- Default local port for TensorBoard is 6006.
- `ssh -i key.pem -L your_machine_port:127.0.0.1:6006 ubuntu@ec2-xyz.amazonaws.com`
- Let’s shift to Jupyter Notebook and understand its usage.

## Data Visualization

## Dimension:

- What is dimension?
- Dimension is a unit that helps define some entity in space
- For e.g. a line can be represented by one coordinate, square by two coordinates, etc
- x, y, z axes each represent one dimension
- Humans are most used to perceive things in 4 dimensions

- The concept of dimension we have in our heads is the spacetime dimension. This differs significantly with the concept of dimension in mathematics.
- As per Wikipedia, a dimension is defined as the minimum number of coordinates required to specify any point within it.

## Data and Dimensions

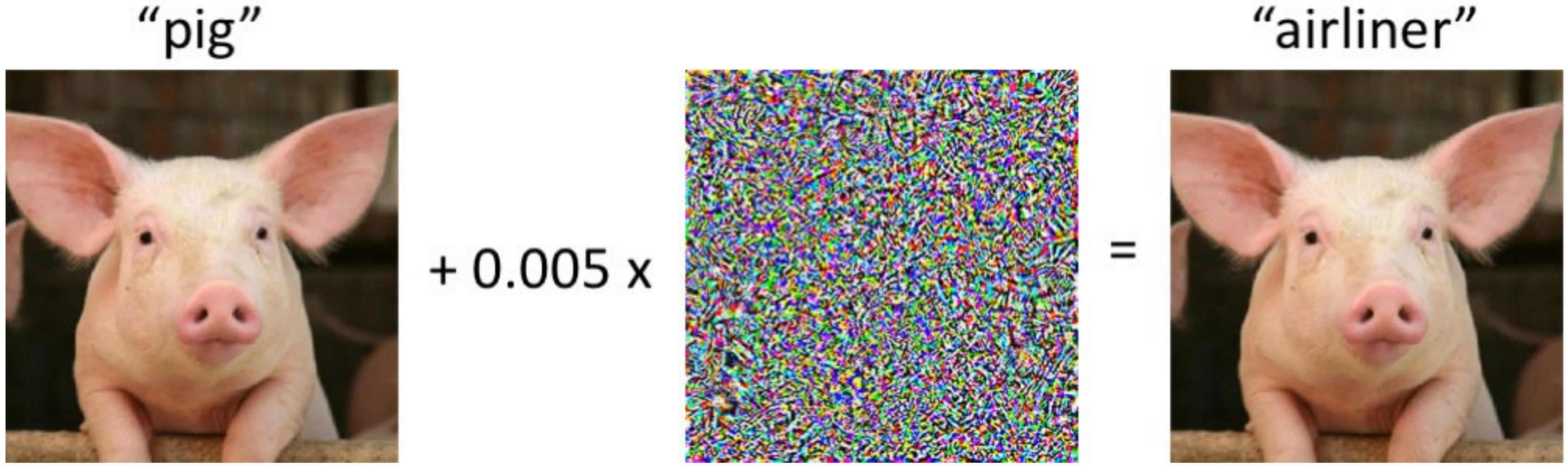
- Images:
  Most images we use these days have the number of pixels (each pixel is a dimension) in the range of thousands to millions.
- But do we really need all those dimensions to represent an image?

- Dimensionality Reduction

- How much change can wiggling a few pixels bring to the output of the neural network?

- Adversarial inputs to trick the network

## Adversarial Example:



## 1. Dimensionality Reduction:

- We have two types:

i) Linear Dimensionality Reduction and

ii) Non-Linear Dimensionality Reduction

- Data in real world have low intrinsic dimensionality
- Most of the content we encounter in daily life exists in three physical dimensions i.e, roll, pitch and yaw
- This low dimensional space is embedded into the high dimensional space which can only be recovered via specific mathematical methods.
- There is also a complete branch of machine learning which is called ‘Unsupervised Learning’
- We will be discussing a few mathematical methods to do this: t-SNE, PCA and ZCA

# T-distributed Stochastic Neighbor Embedding

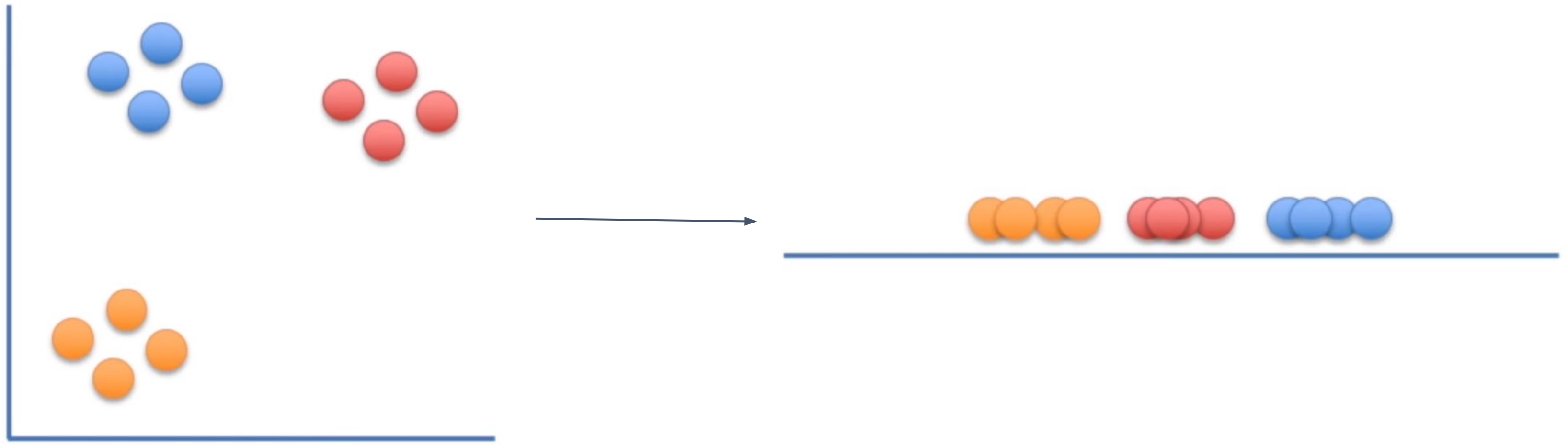
Overview:

Given a collection of N high-dimensional objects [x1,..xn], how can we get an understanding of how these objects are arranged in the high dimensional space?

Since our brain works in the spacetime dimension, we cannot interpret higher dimensions. We need to be able map the high dimensional data into 1,2 or 3 dimensions to be able to make any sense of it.

t-SNE does exactly that!

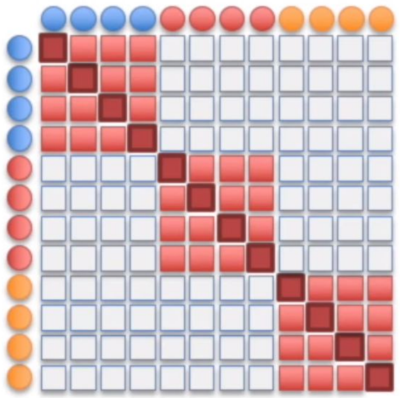
## Explained:



**Images taken from StatQuest channel on YouTube*

## Similarity Scores:

Tells us the similarity scores for each sample with respect to another.



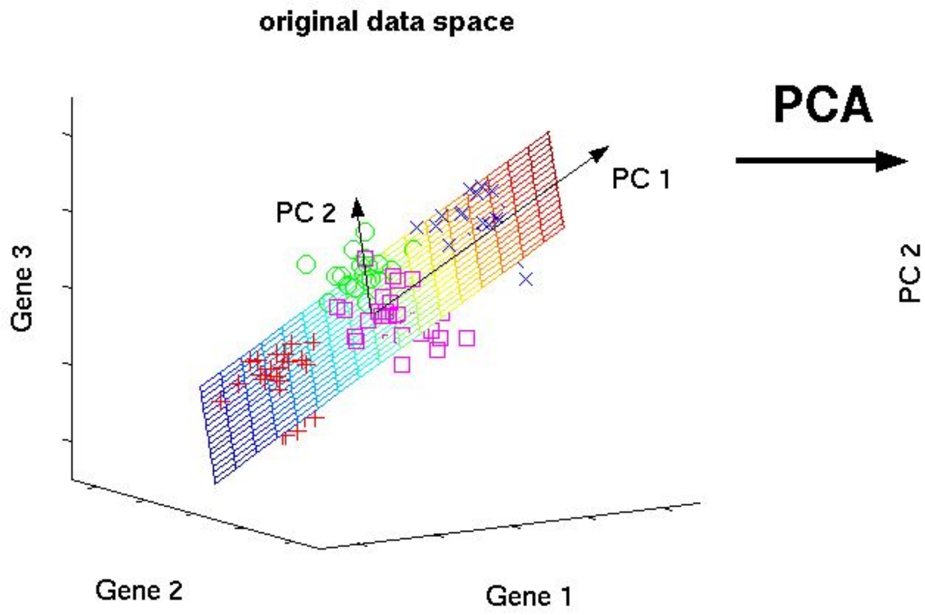
**Images taken from StatQuest channel on YouTube*

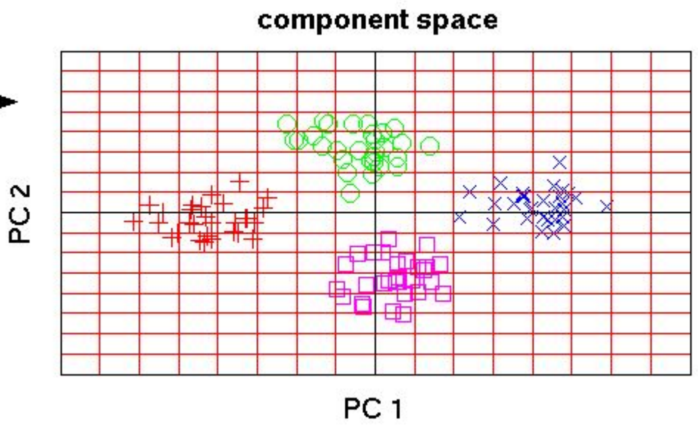
## t-SNE with python

- We will be using the MNIST database to implement t-SNE.
- MNIST dataset has images of size 28x28 which amounts to 784 dimensions. Do you think we need 784 variables to distinguish between digits?

## Principal  Component Analysis

- As we discussed earlier, most data that we expect the neural network to learn is redundant.
- This redundancy slows down the learning process and also consumes a lot of resources in terms of processing.
- PCA allows us to deal with this by reducing dimensions and at the same time retaining the most useful information.


original data space
PC 1
PC 2
Gene 3
Gene 2
Gene 1
PCA


component space
PC 2
PC 1

## How?

- PCA uses the correlation information between dimensions and minimizes the number of variables that are required to represent those dimensions (high correlation means reducible dimensions).
- This way, we are maximizing the variation in the available data which in turn retains all the useful information about how the original data is distributed.
- We will demonstrate how this works using MNIST dataset.
- Go to Jupyter Notebook

## ZCA

- This is a closely related preprocessing step also called as whitening.
- So when we are training on images, the raw input is redundant since adjacent pixels are typically correlated.
- The goal of whitening is to make the input less redundant.
- This can be intuitively understood as providing only useful features for the network to learn.
- The difference here is that we are not reducing the number of dimensions. We are just enhancing the dimensions which we believe to contain useful features and suppressing the dimensions which do not play an important role.

## References:

1) StatQuest: t-SNE, Clearly explained
https://www.youtube.com/watch?v=NEaUSP4YerM
2) StatQuest: PCA
3) Medium:
https://medium.com/@luckylwk/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b