Contents

1 Introduction and Recap **1**
1.1 Vision and Retina 1
1.2 n-dimensions and probability distribution 1

2 Supervised Learning **2**
2.1 Gradient Descent 2
2.2 Connection between Gradient Descent and Neural Networks 2
2.3 Overfitting 2

3 Unsupervised Learning **3**
3.1 Johnson-Lindenstrauss Lemma (JLL) 3
3.2 Unsupervised Learning Techniques 4
3.2.1 PCA 4
3.2.2 SVD 5
3.3 Sparse coding 5
3.3.1 Algorithms 5

1 Introduction and Recap

In this lecture [1], Professor Christos Papadimitriou starts with recapitulation of past lecture.

1.1 Vision and Retina

Retina is a highly complex neural feast. We discussed Darwin's doubts on relating eye with evolution, but then clarifications were made on pathways from which the eye evolved. Optic chiasms [3] were discussed and we moved onto Deep Net architectures. The layers in the visual system in the brain don't only process information but also compare and predict it. The Hubel-Wiesel conjecture discusses the organization of simple receptive fields. The alignment of the neurons is very interesting – how does the LGN know? The mechanism by which it is done is unclear. The Hubel-Wiesel conjecture was a crucial moment for neuroscience and computer science as it gave inspiration for the neocognitron model. "Unsupervised learning of weights" actually is supervised learning.

1.2 n-dimensions and probability distribution

We discussed how the volume of a unit ball in n-dimensional mappings shrinks with increasing values of n. This sprung up while discussing on approximations of differential equations. The idea related the complexity of data in higher dimensions. For a multi-dimensional data, the probabilities of hitting a local minima or a saddle point is tiny. This is why problem statements which may not be convex still work fine on neural network architectures.

2 Supervised Learning

2.1 Gradient Descent

To approximate the minimum of a differentiable function. The most important part was choosing the learning rate. The theorem about gradient descent *doesn't matter*. We can examine gradient descent in 3-d to get a better intuition about it. Higher dimensional data is difficult to visualize. As you inscribe a circle, sphere, etc. in a square, cube, hypercube, etc. the round object occupies exponentially less area/volume. This means that data will occupy exponentially less space as dimensionality increases.

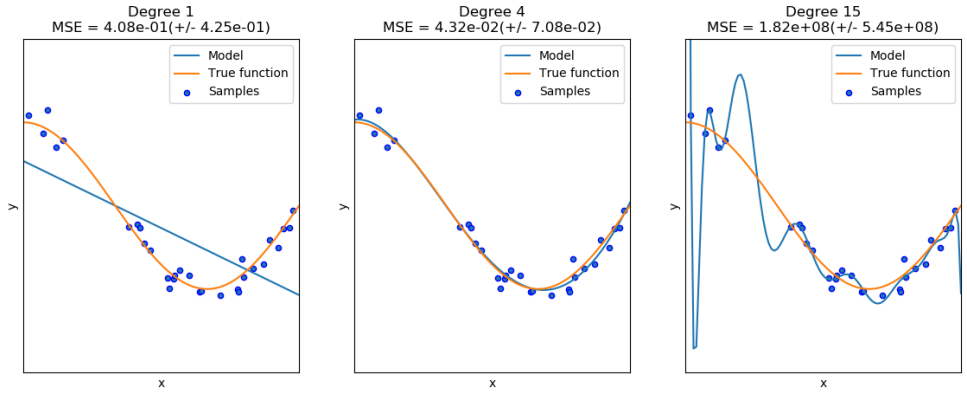
2.2 Connection between Gradient Descent and Neural Networks

You can train deep nets to answer questions such as "does this image contain a car?" which can be formalized as $f : \{\text{images}\} \to \{0, 1\}$. You approximate the minimum of a differential function which calculates loss by stochastic gradient descent. In order to do this, apply the chain rule on the parameters and spatial results. This is called back propogation. This is a supervised learning algorithm, i.e. the data is labelled.

Remark 2.1. Does backpropogation happen in the brain? Not really, but perhaps.

2.3 Overfitting

Why does gradient descent succeed in approximating the optimum? It is efficient at finding optima. But getting to the optimum is the easy part, but avoiding underfitting or overfitting is difficult.



How to avoid overfitting? Regularization helps prevent the model from matching the data too heavily. Dropout forces nodes to arbitrarily be ignored. But you may be able to just use gradient descent and avoid overfitting.

3 Unsupervised Learning

Starting with a quote from Yann LeCunn: "The Revolution Will Not be Supervised", we discuss unsupervised learning.

We start with the real curse of dimensionality. The real curse is that we have never been in those spaces and don't understand them well enough. Reducing the dimensionality of data is a valid way to proceed in such cases. As seen with the sphere-cube example, high dimensionality leads to a lack of geometric intuition and is in general unfamiliar. Is there such a thing as "redundantly high dimension?" e.g. 10^8 points in 10^5 dimensions. Yes – if $\sqrt{\ln \frac{\#\text{ points}}{\text{dimension}}} << 1$ we are wasting dimensions.

3.1 Johnson-Lindenstrauss Lemma (JLL)

For any set of n d-dimensional vectors $x_1, ..., x_n$ there is another set of vectors $x_1^{'}, ..., x_n^{'}$ with dimension $k = 8\frac{\ln n}{\varepsilon^2}$ such that for all $i, j : |x_i - x_j|^2 \approx |x_i^{'} - x_j^{'}|^2(1 \pm \varepsilon)$.

What happens if you project a **specific** d-dim vector on a **random** k-dim space? This is not an easy mathematical problem because you have to define a random k-dimensional space. Instead, ask equivalent question: What happens if you project a random d-dim vector on a specific k-dim space?

Lemma: $\text{prob}(|\text{length} - \frac{k}{d}| > \varepsilon\frac{k}{d}) < e^{-\frac{k}{4}\varepsilon^2}$.

To prove JLL: $k = \frac{8 \ln n}{\varepsilon^2}$. $\text{prob} \rightarrow e^{-2\log n} = \frac{1}{n^2}$

We are projecting the vectors from a higher dimension into a lower dimension, not throwing away any information. Let us ask two questions:

- What happens on projecting a **specific** d-dimensional vector on a **random** k-dimensional space?
- Or, what happens if we project a **random** d-dimensional vector on a **specific** k-dimensional space?

We are aiming at conserving the inter-distance information, a d-dimensional vector of expected length 1 unit will be morphed into a k-dimensional vector of expected length of $\frac{k}{d}$ units. If k is large enough, it is very concentrated around k. So, the chance that any of the n^2 vectors (distance of vectors: $\binom{n}{2}$) is deformed is $\frac{1}{n^2}$

So, the probability that at least one vector is deformed more than epsilon is less than 1. Thus, there exists a dimension where no vector is deformed.

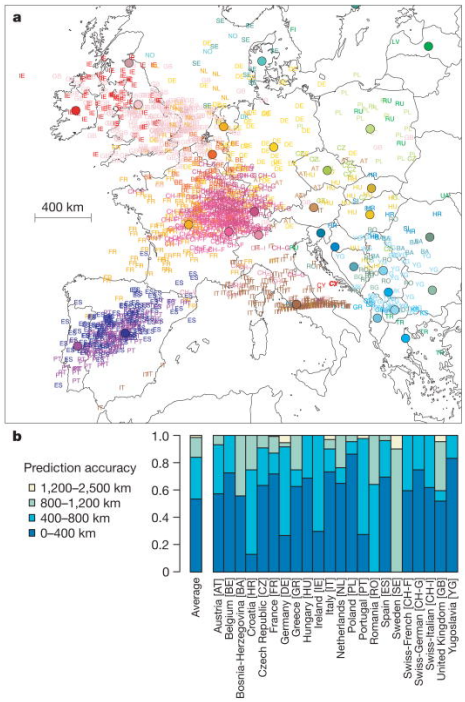
Remark 3.1. An application to complexity theory: The d-dimensional traveling salesman problem can be approximated when d is constant, but when dimension $\approx$ cities, it is NP-complete to approximate. What happens when dimension $\approx \sqrt{\text{cities}}$? It is still NP-complete because you can project so the distances are essentially the same.

3.2 Unsupervised Learning Techniques

So, what if the dimensions were not super high? What if we claim a dataset is "special". You can use principal component analysis (PCA) or singular value decomposition (SVD) or LVM.

3.2.1 PCA

Suppose you can approximate five 10-d vectors $x_1, ..., x_5$ as $x_i \approx x_\mu + a_i v + b_i u$ for two appropriate vectors u, v . Example of PCA on DNA of Europeans: $n = 3000, d = 500000$



Consider latent semantic indexing of documents. Every book is a 10,000 dimensional vector, where every dimension is a word. Preprocess using stop words, word forms, log of # of appearances. The Library of Congress is a collection of 15,000,000 vectors of dimension 10,000. What are the principal components? How many?

How do you do PCA?

Pick the top k eigenvectors corresponding to the top k eigenvalues, which can be done by well-optimized algorithms such as the iteration method shown in class.

Preprocessing: Curate data, subtract to zero mean, normalize coordinates. Then do some linear algebra and geometry. Similar to linear regression, but linear regression believes there is one preferential dimension, while PCA is agnostic.

Equations from board: $\min_{v_1,v_2,v_k} \sum_{\text{data}} \text{dist}^2(x_1, S_{v_1 v_k})$. $\min \sum_{\text{data}} \text{dist}^2 = ||X||^2 - < x_1, v_1 > k = 1$. (Scribe note: I wasn't able to follow math on board quickly enough).

Detailed explanations on PCA [2]

3.2.2 SVD

Decompose matrix $A = UDV^*$. There were illustrations on the board.

Detailed explanations on SVD [2]

3.3 Sparse coding

As in the Olshausen-Fields paper. You are given data vectors $y_1, ..., y_n \in R^d$. Find a mXd coding matrix B with column vectors $b_1, ..., b_m \in R^d$ (expect $m > d$) and sparse vectors $x_1, ..., x_n \in R^m$ such that $\min_{A,x} = \sum_i |y_i - Bx_i|^2 + \sum_i S(x_i)$ (a sparsity penalty function).

3.3.1 Algorithms

The algorithms include radient descent, maximum likelihood assuming a generative model and noise, alternating minimization, and performance guarantees.

References

[1] Christos Papadimitriou. Coms 6998-06: Computation and the brain. `https://computationandbrain.github.io/slides/Lecture3.pptx`, Sep 2018.

[2] Gregory Valiant Tim Roughgarden. The modern algorithmic toolbox: Lecture 8 - how pca works. `https://web.stanford.edu/class/cs168/l/l8.pdf`, Apr 2018.

[3] Wikipedia contributors. Optic chiasm — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Optic_chiasm`, 2018. [Online; accessed 26-September-2018].