Deep Learning

Bengio, Goodfellow, Courville

June 19, 2018

Contents

Chapter 1

Introduction

This is a summary of deep learning.

Chapter 2

Section I: Applied Math and Machine Learning Basics

2.1 Chapter 2 Linear Algebra

2.1.1 2.1 Scalars, Vectors, Matrices and Tensors

- Scalars - single numerical values
- Vectors - 1-D array of numbers
- Matrices - 2-D array of numbers
- Tensors - more than 2-D array of numbers
- Transpose - a reflection across the diagonal - $M_{i,j} \rightarrow M'_{j,i}$
- Main Diagonal - the entries with the same row and column offset, $M_{i,i}$
- Broadcasting - the implicit copying of a vector to be added to multiple rows in a matrix

2.1.2 2.2 Multiplying Matrices and Vectors

- Matrix product - **C**=**AB**, $C_{i,j} = \sum_k A_{i,k} B_{k,j}$
- element-wise product/ Hadamard product - element-wise multiplication, expressed as **C**=**A B**

2.1.3 2.3 Identity and Inverse Matrices

A matrix multiplied by its identity produces the same original matrix.

2.1.4 2.4 Linear Dependence and Span

A vector is linearly dependent on other vectors if it can be written as a linear combination of other vectors.

2.1.5 2.5 Norms

Norms are measures of length or magnitude of a matrix.

- L1-norm: $||M||_1 = \sum_i |M_i|$
- L2-norm/ euclidean norm: $||M||_2 = \sqrt{\sum_i M_i^2}$
- L∞-norm: $||M||_\infty = maxM_i$

2.1.6 2.6 Special Kinds of Matrices and Vectors

2.1.7 2.7 Eigendecomposition

Eigendecompositions are decompositions of a square matrix into the matrix product of three matrices. Two matrices are orthonormal and composed of the matrix's eigenvectors while the other is the diagonalized matrix of eigenvalues. The largest eigenvalue and corresponding eigenvectors correspond to the axis of greatest information in PCA.

2.1.8 2.8 Singular Value Decomposition

Singular value decompositions are decompositions of a matrix into the matrix product of three matrices. The first and third matrices are orthonormal and composed of the original matrix's eigenvectors. The second one is the nonzero singular values of the original matrix and the square roots of the eigenvalues of $M^*MandMM^*$. The largest eigenvalue and corresponding vectors correspond to the axis of greatest information in PCA. Note that if the original matrix is denoted by M, M^TM easily be simplified to an eigendecomposition.

2.1.9 2.9 The Moore-Penrose Pseudoinverse

2.1.10 2.10 The Trace Operator

2.1.11 2.11 The Determinant

2.1.12 2.12 Example: Principal Components Analysis

2.2 Chapter 3 Probability and Information Theory

2.2.1 3.1 Why Probability?

2.2.2 3.2 Random Variables

2.2.3 3.3 Probability Distributions

2.2.4 3.4 Marginal Probability

2.2.5 3.5 Conditional Probability

2.2.6 3.6 The Chain Rule of Conditional Probabilities

2.2.7 3.7 Independence and Conditional Independence

2.2.8 3.8 Expectation, Variance and Covariance

2.2.9 3.9 Common Probability Distributions

2.2.10 3.10 Useful Properties of Common Functions

2.2.11 3.11 Bayes' Rule

2.2.12 3.12 Technical Details of Continuous Variables

2.2.13 3.13 Information Theory

2.2.14 3.14 Structured Probabilistic Models

2.3 Chapter 4 Numerical Computation

2.3.1 4.1 Overflow and Underflow

2.3.2 4.2 Poor Conditioning

2.3.3 4.3 Gradient-Based Optimization

2.3.4 4.4 Constrained Optimization

2.3.5 4.5 Example: Linear Least Squares

2.4 Chapter 5 Machine Learning Basics

2.4.1 5.1 Learning Algorithms

2.4.2 5.2 Capacity, Underfitting, and Overfitting

2.4.3 5.3 Hyperparameters and Validation Sets

Hyperparameters are architectural and structural parameters that are set before the learner is run. Examples are margin size in SVM, network size in deep learning, learning rate in gradient descent, etc.

2.4.4 5.4 Estimators, Bias and Variance

Bias variance tradeoff refers to the tradeoff process between high bias (underfitted) and high variance (overfitted) models. High bias models have lower representation ability, but are more likely to generalize. High variance models have greater representation ability but have a possibility to overfit the training data.

2.4.5 5.5 Maximum Likelihood Estimation

Maximum likelihood estimation is applied in generative models

2.4.6 5.6 Bayesian Statistics

2.4.7 5.7 Supervised Learning Algorithms

2.4.8 5.8 Unsupervised Learning Algorithms

2.4.9 5.9 Stochastic Gradient Descent

2.4.10 5.10 Building a Machine Learning Algorithm

2.4.11 5.11 Challenges Motivating Deep Learning

Chapter 3

Section II: Modern Practical Deep Networks

3.1 Chapter 6 Deep Feedforward Networks

3.1.1 6.1 Example: Learning XOR

3.1.2 6.2 Gradient-Based Learning

3.1.3 6.3 Hidden Units

Hidden units are units between input and output units.

3.1.4 6.4 Architecture Design

Neural networks are able to represent any possible function. A single layer neural network theoretically able to represent any function. However, increasing the width and representation ability leads to more quickly scaling computational costs. Instead, increasing network depth can also increase representation ability with lower computational cost.

3.1.5 6.5 Back-Propagation and Other Differentiation Algorithms

Back propagation begins with the cost gradient and uses the reverse chain rule to determine the gradient for each units output. Gradient descent uses these gradients to optimize the final output.

3.1.6 6.6 Historical Notes

3.2 Chapter 7 Regularization for Deep Learning

3.2.1 7.1 Parameter Norm Penalties

One can apply norms as penalties on the weights to control constraints for the parameter. For example, L2 norm favors smaller, similarly scaled weight values while L1 norms indicates a preference for sparse parameters.

- L2 normalization - $J^*(\Theta; x, y) = J(\Theta; x, y) + \lambda||w||$ - One can understand it by understanding the function of the Euclidean norm - the closer the weights lie to 0, the lesser the penalty.
- L1 normalization - $J^*(\Theta; x, y) = J(\Theta; x, y) + \lambda||w||_1$ - Since the graph of the absolute value norm allows most variation along the axes, L1 norm favors sparse parameterization values.

3.2.2 7.2 Norm Penalties as Constrained Optimization

3.2.3 7.3 Regularization and Under-Constrained Problems

3.2.4 7.4 Dataset Augmentation

Dataset augmentation applies transformations to allow networks to learn . In the example of object recognition, an image can be reflected, translated, or randomly cropped to allow the learner to recognize mutations of an object.

3.2.5 7.5 Noise Robustness

3.2.6 7.6 Semi-Supervised Learning

3.2.7 7.7 Multitask Learning

3.2.8 7.8 Early Stopping

Early stopping is often used to test for hyperparameter optimization, specifically the number of training steps to take. The algorithm stores the optimal set of parameters and returns to the saved optimal parameterization after every iteration. If validation error does not decrease after a specified number of iterations, then the algorithm returns the parameters and training steps that have been saved.

3.2.9 7.9 Parameter Tying and Parameter Sharing

Parameter tying or **parameter sharing** is used to apply the same set of parameters multiple times across a network. One of the biggest examples is in convolutional neural networks - the same convolution kernel, say in edge detection, is applied to every pixel in the unit.

3.2.10 7.10 Sparse Representations

Sparse parameterizations have mostly been studied - weights are mostly 0 - but in sparse representation, elements of the representation (input) are mostly 0. This is achieved by adding an L1 regularization term based on **h**, the hidden unit outputs.

$$J * (\Theta; X, y) = J(\Theta; X, y) + \alpha\Omega(h)$$

3.2.11 7.11 Bagging and Other Ensemble Methods

Bagging (bootstrap aggregating) models reduce generalization by training multiple models and having them vote on the output. The simplest way is through **model averaging** that uses the outputs of all the learners. By doing this, the goal is that the errors will be different for each model and cancel out.

3.2.12 7.12 Dropout

Dropout is based off of natural neural networks, which applies a variety of network structures to data. It works by probabilistically removing a unit and all associated connection from a neural network (usually 0.8 chance to keep output units and 0.5 chance to keep hidden units). The greater the probability of dropping a unit, the greater the effect of regularization. The data is then trained on the subnetwork. Every iteration of algorithm uses a different subnet of the network.
At test time, the full network is used.

3.2.13 Understanding Dropout

According to Andrew Ng -
One of the most common forms of dropout is called **inverse dropout**, in which the remaining weights of a layer is divided by the keep-probability to keep the expected value of the output the same.

An effect of dropout is that the cost function J is not well defined due to the stochastic nature of the network.
Some intuition about why dropout works as a form of regularization:

- The simpler subnets have less representational power
- Any unit an't rely on any one feature, so have to spread out weights - as a result, units can't put too much weight on a single input resulting in an effect similar to L2 norm.

3.2.14 7.13 Adversarial Training

Adversarial training involves training images on specifically designed data meant to trick the algorithm (e.g. applying a local mutation to an image in an object recognition system)

3.2.15 7.14 Tangent Distance, Tangent Prop and Manifold Tangent Classifier

3.3 Chapter 8 Optimization for Training Deep Models

3.3.1 8.1 How Learning Differs from Pure Optimization

Sometimes the ideal loss function (e.g. 0-1 loss for classification) is not efficiently optimizable. Instead a surrogate loss function (e.g. negative log loss) can be used instead. In some cases, the optimized loss descent can yield a result better than the original. Depending on the loss algorithm, one can simply optimize a simpler term or component, which in turn optimizes the entire function. (e.g. $argmax \sum log(P_{model}(x^i, y^i, ; Theta)) = argmax \sum P_{model}(x^i, y^i, ; Theta))$
Minibatch or stochastic gradient methods use randomly chosen samples for descent. Ideally the minibatch size is large enough to fully utilize multicore capabilities of the hardware while not using too much due to diminishing returns. Solely gradient-based methods require smaller batch size while those that use Hessians require a much larger batch size. Different algorithms use minibatches differently - e.g. stochastic gradient descent.
When the stochastic algorithm doesn't reuse examples, it is essentially equivalent to gradient descent. However, if it does reuse examples (e.g. multiple passes through the training data), only the first epoch is an unbiased estimate that reduces generalization. Later passes are biased, but still decrease training error enough to offset differences in training and true error.

3.3.2 8.2 Challenges in Neural Network Optimization

3.3.3 Local Minima

Empirically, it has been observed that in general, higher dimension data has fewer problems with getting stuck in local minima. In addition, it has also been observed that local minima encountered by neural networks tend to have low enough costs that are close enough to the global optimum to be acceptable.

3.3.4 Saddle Points

Sadde points are more common than minima in higher dimensions. Though gradient-based methods shouldn't work in practice, they tend to do so.

3.3.5 Plateaus

In flat regions, it can be difficult and computationally expensive to apply gradient descent.

3.3.6 Cliffs/Exploding Gradients

A common problem with neural networks is that transformations repeatedly applied across multiple layers leads to the **vanishing and exploding gradient** problem. This is very common in recurrent networks that apply the same weight multiple times. If the gradient descent algorithm encounters an exploding gradient, there exists a possibility that the algorithm will overstep completely. This

can be addressed by adjusting the step size.
See: gradient clipping heuristics

3.3.7 8.3 Basic Algorithms

3.3.8 8.4 Parameter Initialization Strategies

Depending on intialization, that may determine whether or not the algorithm converges on a minimum. One rule for initialization is that symmetries must be broken - that is, if two units have the exact same input and output connections, the initial values must differ, otherwise there is no way to differentiate them.
Random initialization usually draws values from a Gaussian or uniform distribution. The scale of the initial distribution usually doesnt have an effect on the outcome or generalization ability. However, larger scales allow for less redundancy and more symmetry breaking. It also prevents signal loss from propagation and backprop.

3.3.9 8.5 Algorithms with Adaptive Learning Rates

Adaptive learning rates allow algorithms to get more precise as they approach the optimum to prevent overshooting

3.3.10 8.6 Approximate Second-Order Methods

Higher order methods are used to reduce the propagation of

3.3.11 8.7 Optimization Strategies and Meta-Algorithms

Batch Normalization

Similar to input normalization, batch normalization is used to determine transform activation outputs from hidden units to the desired mean and variance.

$$h' = \frac{1}{m}\sum h, \sigma = \sqrt{\epsilon + \frac{1}{m}\sum (h-\mu)^2}$$

$$h^* = \gamma h'^{(l)} + \beta$$

where γ, β are learned parameters and ϵ is small (less than 10^{-8}) to avoid divide by zero error

Batch normalization also has a regularization effect since it cancels out higher level interactions. At test time, there are no batches so you can't normalize the data using μ, σ^2. However, you can keep track of an exponentially weighed average of μ, σ^2 across all minibatches during training.

Polyak Averaging

The idea of Polyak averageing is to average the previous trajectory of an optimization algorithm. So for t iterations of a gradient descent that has used $\Theta^1, ..., \Theta^t$, The average is $\overline{\Theta}^t = \frac{1}{t}\sum_i \Theta^i$. Polyak averaging results in strong convergence when added to gradient descent on a strongly convex problem.
On nonconvex problems, it is more typical to use an exponentially decaying average.

$$\overline{\Theta}^t = \alpha\overline{\Theta}^{t-1} + (1-\alpha)\Theta^t$$

3.4 Chapter 9 Convolutional Networks

Convolutional networks are most often applied to plotted (2-D) inputs such as time series data and image processing.

3.4.1 9.1 The Convolution Operation

The **convolution** of two functions of two functions is denoted as follows:

$$f * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da = \int_{-\infty}^{\infty} f(x-a)g(a)da$$

When considering discrete convolutions across matrices, one must flip the matrix kernel across both axes and apply the dot product.

$$F * K(x) = \sum_i F_i K_{x-i} = \sum_i F_{x-i} K_i$$

Another way of understanding convolutions is that it is equivalent to multiplying the input by a **Toeplitz Matrix**. (Note: In a Toeplitz matrix, every row is a single shift of the previous row). Higher dimensional convolutions are equivalent to multiplications by a **doubly block circulant matrix**.

One convolution variant is the **cross correlation** operation. It is essentially the same as a convolution without flipping the kernel.

3.4.2 9.2 Motivation

The idea of a convolutional neural network is that it models **sparse interactions** (or **sparse connectivity/weights**), meaning that outputs liekly won't actually interact with every input, so eliminating connections saves time and computation. In a convolutional net, a unit isn't connected to every node in the previous layer. The **receptive field** consists of all units that eventually feed into a node as input.
Convolutional neural networks utilize **parameter sharing**, meaning that the same set of parameters are used for multiple functions. In traditional neural nets, parameters are unique per node and only applied once. In parameter sharing, the weights are shared (e.g. the edge detection kernel is applied to every pixel in the image). This approach also saves memory and runtime ($O(kn)$ where k is the maximum number of connections per unit, rather than the size of the input layer).

3.4.3 9.3 Pooling

There are three stages in the "convolution layer" - although referred to in singular, this usually involves multiple layers and convolutions.

1. Parallel convolutions applied to the input for linear activation
2. In the detection stage, the output is then run through a non-linear activation (e.g. sigmoid. ReLU)
3. Pooling functions are used to further modify output

Pooling refers to methods that summarize the data in the local neighborhood of a point. Some examples of common pooling operations in the context of computer vision and image processing:

- max pooling - returns the largest scalar value in the local neighborhood
- average pooling - takes the average of the scalar values in the local neighborhood
- weighed pooling - take a weighed average of scalar values based on distance

3.4.4 9.4 Convolution and Pooling as an Infinitely Strong Prior

One way to understand a convolutional neural network with only connections to local neighbors is that it is in fact a fully connected neural network with an infinitely strong prior indicating that only local neighbors have a nonzero influence. Another prior that is applied is that the weights are exactly the same for units in the same space (parameter sharing).

3.4.5 9.5 Variants of the Basic Convolution Function

Zero padding is necessary to apply convolutions to matrices. When kernel is of dimension $k \times k$:

- Valid convolution - no zero padding is applied. The convolution begins at offset $(\frac{k}{2}, \frac{k}{2})$ from the matrix so every unit in the output is based only on values in the original input. This results in fewer outputs than inputs.
- Same convolution - enough zero padding ($\frac{k}{2}$ in each direction) is applied to allow convolution to begin at offset $(0, 0)$ so units at the border of the output is also based on values not in the initial input. In additional border input pixels have less influence on the output. This results in same-sized input and output.
- Full convolution - enough zero padding (k in each direction) is applied to allow convolution to begin at $(-\frac{k}{2}, -\frac{k}{2})$ so units at the border of the output is also bsaed on values not in the initial input. This allows every unit in the input to have the same amount of influence on the output. This increases the size of the output.

Usually the best results come from somewhere between valid and same convolutions.

3.4.6 9.6 Structured Outputs

3.4.7 9.7 Data Types

Data is represented not as a 2-D matrix of unit connections, but often as a 3-D tensor reflecting image values across multiple channels. Since training involves multiple examples, this then becomes a 4-D tensor. For example, for kernel entry $K_{i,j,k,l}$, i, j is the respective output and input channels and k, l is the respective row and column offset.

3.4.8 9.8 Efficient Convolution Algorithms

3.4.9 9.9 Random or Unsupervised Features

3.4.10 9.10 The Neuroscientific Basis for Convolutional Networks

3.4.11 9.11 Convolutional Networks and the History of Deep Learning

3.5 Chapter 10 Sequence Modeling: Recurrent and Recursive Nets

Recurrent neural networks are most commonly applied to sequential input, whether temporally or spatially sequential.

3.5.1 10.1 Unfolding Computational Graphs

Usually computational graphs can be expressed either with a black box to show recurrence or as an unfolded computational graph, where each iteration's pathway is completely expressed.
Batch or deterministic gradient methods use the entire training set for descent.

3.5.2 10.2 Recurrent Neural Networks

3.5.3 10.3 Bidirectional RNNs

3.5.4 10.4 Encoder-Decoder Sequence-to-Sequence Architectures

3.5.5 10.5 Deep Recurrent Networks

3.5.6 10.6 Recursive Neural Networks

3.5.7 10.7 The Challenge of Long-Term Dependencies

3.5.8 10.8 Echo State Networks

3.5.9 10.9 Leaky Units and Other Strategies for Multiple Time Scales

3.5.10 10.10 The Long Short-Term Memory and Other Gated RNNs

3.5.11 10.11 Optimization for Long-Term Dependencies

3.5.12 10.12 Explicit Memory

3.6 Chapter 11 Practical Methodology

3.6.1 11.1 Performance Metrics

3.6.2 11.2 Default Baseline Models

3.6.3 11.3 Determining Whether to Gather More Data

3.6.4 11.4 Selecting Hyperparameters

3.6.5 11.5 Debugging Strategies

3.6.6 11.6 Example: Multi-Digit Number Recognition

3.7 Chapter 12 Applications

3.7.1 12.1 Large-Scale Deep Learning

3.7.2 12.2 Default Baseline Models

3.7.3 12.3 Determining Whether to Gather More Data

3.7.4 12.4 Selecting Hyperparameters

3.7.5 12.5 Debugging Strategies

3.7.6 12.6 Example: Multi-Digit Number Recognitions

Chapter 4

Part III: Deep Learning Research