Nonlinear Models
Numerical Methods for Deep Learning

Lars Ruthotto

Departments of Mathematics and Computer Science, Emory University
Course Overview
Course Overview

- Lecture 1: Linear Models
  1. Introduction and Applications
  2. Linear Models: Least-Squares and Logistic Regression

- Lecture 2: Neural Networks
  1. Introduction to Nonlinear Models
  2. Single Layer Neural Networks
  3. Training Algorithms for Single Layer Neural Networks
  4. Neural Networks and Residual Neural Networks (ResNets)

- Lecture 3: Neural Networks as Differential Equations
  1. Resnets as ODEs
  2. Residual CNN and their relation to PDEs
Introducción
Motivation: Nonlinear Models

In general, impossible to find a linear separator between classes

Goal/Trick
Embed the points in higher dimension and/or move the points to make them linearly separable
Example: Linear Fitting

Assume $\mathbf{C} \in \mathbb{R}^{nc \times n}$, $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ and $n \gg n_f$. Goal: Find $\mathbf{W} \in \mathbb{R}^{nc \times n_f}$ such that

$$\mathbf{C} = \mathbf{WY}$$

If $\text{rank}(\mathbf{Y}) < n$, may not be possible to fit the data.

Two options:

1. Regression: Solve $\min_{\mathbf{W}} \| \mathbf{WY} - \mathbf{C} \|_F^2 \sim$ always has solutions, but residual might be large

2. Nonlinear Model: Replace $\mathbf{Y}$ by $\sigma(\mathbf{K} \mathbf{Y})$, where $\sigma$ is element-wise function (aka activation) and $\mathbf{K} \in \mathbb{R}^{m \times n_f}$ where $m \gg n_f$
Illustrating Nonlinear Models

Remarks

- instead of $WY = C$ solve $\hat{W}_{\sigma(KY)} = C$
- solve bigger problem $\sim$ memory, computation, . . .
- what happens to $\text{rank}(\sigma(KY))$ when $\sigma(x) = x$?
Universal Approximation Theorem

Given the data $\mathbf{Y} \in \mathbb{R}^{n_f \times n}$ and $\mathbf{C} \in \mathbb{R}^{n_c \times n}$ with $n \gg n_f$ there is nonlinear function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$, a matrix $\mathbf{K} \in \mathbb{R}^{m \times n_f}$, and a bias $b \in \mathbb{R}$ such that

$$\text{rank}(\sigma(\mathbf{K}\mathbf{Y} + b)) = n.$$ 

Therefore, possible [2, 7] to find $\mathbf{W} \in \mathbb{R}^{n_c \times m}$

$$\mathbf{W}\sigma(\mathbf{K}\mathbf{Y} + b) = \mathbf{C}.$$
Choosing Nonlinear Model

\[ W\sigma(KY + b) = C \]

- **how to choose \( \sigma \)?
  - early days: motivated by neurons
  - popular choice: \( \sigma(x) = \tanh(x) \) (smooth, bounded, . . .)
  - nowadays: \( \sigma(x) = \max(x, 0) \) (aka ReLU, rectified linear unit, non-differentiable, not bounded, simple)

- **how to choose \( K \) and \( b \)?
  - pick randomly \( \sim \) branded as *extreme learning machines* [8]
  - train (optimize) \( \sim \) deep learning (when we have multiple layers)
First Experiment: Random Transformation

Select activation function and choose $K$ and $b$ randomly and solve the least-squares/classification problem

The Pros:

- universal approximation theorem: can interpolate any function
- very easy to program
- can serve as a benchmark to more sophisticated methods

Some concerns:

- may require very large $K$ (size of the data)
- may not generalize well
- large dense linear algebra

EELM_Peaks.m
Single Layer Neural Networks
Learning the Weights

Assume that the number of examples, $n$, is very large. Using random weights, $K$ might need to be very large to fit training data. Solution may not generalize well to test data.

Idea: Learn $K$ and $b$ from the data (in addition to $W$)

$$
\min_{K,W,b} E(W\sigma(KY + b), C^{obs}) + \lambda R(W, K, b)
$$

About this optimization problem:

- more unknowns $K \in \mathbb{R}^{m \times n_f}, W \in \mathbb{R}^{n_c \times m}, b \in \mathbb{R}$
- non-convex problem $\sim$ local minima, careful initialization
- need to compute derivatives w.r.t. $K, b$
Non-Convexity

The optimization problem is non-convex. Simple illustration of cross-entropy along two random directions $dK$ and $dW$

(see ESingleLayer_PlotObjective)

Expect worse when number of layers grows!
Optimization for Single Layer Neural Networks
Gauss-Newton Method

Goal: Use curvature information for fast convergence

\[ \nabla_K E(K, b, W) = (J_K Z)^\top \nabla_Z E(W \sigma(KY + b), C), \]

where \( J_K Z = \nabla_K \sigma(KY + b)^\top \). This means that Hessian is

\[ \nabla^2_K E(K) = (J_K Z)^\top \nabla^2_Z E(C, Z, W) J_K Z \]

\[ + \sum_{i=1}^n \sum_{j=1}^m \nabla^2_K \sigma(KY + b)_{ij} \nabla_Z E(C, Z, W)_{ij} \]

First term is spsd and we can compute it. We neglect second term since

- can be indefinite and difficult to compute
- small if transformation is roughly linear or close to solution (easy to see for least-squares)

do the same for \( b \) and use full Hessian for \( W \sim \) ignore coupling!
Variable Projection - 1

Idea: Treat learning problem as coupled optimization problem with blocks \( \theta = (K, b) \) and \( W \).

Simple illustration for coupled least-squares problem [4, 3, 10]

\[
\min_{\theta, w} \phi(\theta, w) = \frac{1}{2} \| A(\theta)w - c \|^2 + \frac{\lambda}{2} \| Lw \|^2 + \frac{\beta}{2} \| M\theta \|^2
\]

Note that for given \( \theta \) the problem becomes a standard least-squares problem. Define:

\[
w(\theta) = (A(\theta)^T A(\theta) + \lambda L^T L)^{-1} A(\theta)^T c
\]

This gives optimization problem in \( \theta \) only (aka reduced/projected problem)

\[
\min_{\theta} \tilde{\phi}(\theta) = \frac{1}{2} \| A(\theta)w(\theta) - c \|^2 + \frac{\lambda}{2} \| Lw(\theta) \|^2 + \frac{\beta}{2} \| M\theta \|^2
\]
Variable Projection - 2

$$\min_{\theta} \tilde{\phi}(\theta) = \frac{1}{2} \| A(\theta)w(\theta) - c \|^2 + \frac{\lambda}{2} \| Lw(\theta) \|^2 + \frac{\beta}{2} \| M\theta \|^2$$

Optimality condition:

$$\nabla \tilde{\phi}(\theta) = \nabla_{\theta} \phi(\theta, w) + \nabla_{\theta} w(\theta) \nabla_{w} \phi(\theta, w) = 0.$$  

Less complicated than it seems since

$$\nabla_{w} \phi(\theta, w(\theta)) = A(\theta)^{\top} (A(\theta)w(\theta) - c) + \lambda L^{\top} Lw(\theta) = 0$$

Conclusion:

- ignore second term in gradient computation
- apply steepest descent or Gauss-Newton to minimize $\tilde{\phi}$
- need to solve least-squares problem in each evaluation of objective
- gradient is only correct if LS problem is solved exactly
Variable Projection for Single Layer

\[
\min_{\theta, W} E(W\sigma(K(\theta)Y), C) + \lambda R(\theta, W)
\]

Assume that the regularizer is separable, i.e.,

\[
R(\theta, W) = R_1(\theta) + R_2(W)
\]

and that \(R_2\) is convex and smooth. Hence, the projection requires solving the regularized classification problem

\[
W(\theta) = \arg \min_{W} E(W\sigma(K(\theta)Y), C) + \lambda R_2(W)
\]

practical considerations:

- solve for \(W(\theta)\) using Newton (need accuracy)
- need good solver to approximate gradient w.r.t. \(\theta\) well
- use Gauss-Newton or steepest descent to solve for \(\theta\)
Stochastic Optimization

Assume that each \( y_i, c_i \) pair is drawn from some (unknown probability distribution).
Then, we can interpret the learning problem as minimizing the expected value of the cross entropy, e.g., in linear regression

\[
E(W) = \mathbb{E} \left( \frac{1}{2} \| y^\top W - c \|^2 \right)
\]

This is a stochastic optimization problem [1]. One idea: **Stochastic Approximation:** Design iteration \( W_k \rightarrow W^* \) so that expected value decreases. 
Example: Stochastic Gradient Descent, ADAM, . . .
Pro: sample can be small (*mini batch*)
Con: how to monitor objective, linesearch, descent, . . .
Stochastic Average Approximation

Alternative way to solve stochastic optimization problem

\[ E(W) = \mathbb{E} \left( \frac{1}{2} \| y^T W - c \|^2 \right) \]

Pick relatively large sample \( S \subset \{1, \ldots, n\} \) and use deterministic optimization method to solve

\[ \min_W \frac{1}{2|S|} \sum_{s \in S} \| y_s W - c_s^T \|^2. \]

Pro: use your favorite solver, linesearch, stopping...
Con: large batches needed

Note: Sample stays fixed during iteration.
Experiment: Peaks

Compare the three approaches for training a single layer neural network

- ESingleLayer_PeaksSGD.m - stochastic gradient descent
- ESingleLayer_PeaksNewtonCG.m - Newton CG with block-diagonal Hessian approximation
- ESingleLayer_PeaksVarPro.m - Fully coupled solver. Eliminate $\theta$ and use steepest descent/Newton CG for reduced problem.
Deep Neural Networks
Why Deep Networks?

- Universal approximation theorem of NN suggests that we can approximate any function by two layers.
- But - The width of the layer can be very large $O(n \cdot n_f)$
- Deeper architectures can lead to more efficient descriptions of the problem.
- No real proof but lots of practical experience.
Deep Neural Networks

How deep is deep?
We will answer this question later ...

Until recently, the standard architecture was

\[ \begin{align*}
Y_1 &= \sigma(K_0 Y_0 + b_0) \\
\vdots &= \vdots \\
Y_N &= \sigma(K_{N-1} Y_{N-1} + b_{N-1})
\end{align*} \]

And use \( Y_N \) to classify. This leads to the optimization problem

\[
\min_{K_0, \ldots, K_{N-1}, b_0, \ldots, b_{N-1}, W} E \left( W Y_N(K_1, \ldots, K_{N-1}, b_1, \ldots, b_{N-1}), C_{\text{obs}} \right)
\]
Deep Neural Networks in Practice

(Some) challenges:
- Computational costs (architecture have millions or billions of parameters)
- difficult to design
- difficult to train (exploding/vanishing gradients)
- unpredictable performance

In 2015, He et al. [5, 6] came with a new architecture that solves many of the problems
Residual Neural Networks
Simplified Residual Neural Network

Residual Network

\[
Y_1 = Y_0 + \sigma(K_0 Y_0 + b_0)
\]

\[
\vdots = \vdots
\]

\[
Y_N = Y_{N-1} + \sigma(K_{N-1} Y_{N-1} + b_{N-1})
\]

And use \(Y_N\) to classify. This leads to the optimization problem

\[
\min_{K_0,\ldots,N-1,b_0,\ldots,N-1,w} E \left( W Y_N(K_1, \ldots, K_{N-1}, b_1, \ldots, b_{N-1}), C^{obs} \right)
\]

Leads to smoother objective function [9].


References


