KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Avoiding local minima in Deep Learning: a nonlinear optimal control approach

Jan Scheers

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken

Promotor:
Prof. dr. ir. Panos Patrinos

Academiejaar 2019 – 2020

Preface

I would like to thank everybody who kept me busy the last year, especially my promoter and my assistants. I would also like to thank the jury for reading the text. My sincere gratitude also goes to my friends and my family.

Jan Scheers

Contents

Abstract

Chapter 1

Introduction

1.1 Neural Networks

Neural networks are a very popular machine learning model these days. They are known to be very expressive, leading to low bias. They are especially useful for learning from very large data sets. A deep neural net is shown in figure 1.1. It is made by connecting layers of 'neurons'. Each neuron is a nonlinear function transforming the weighted sum of its inputs:

$$y = \sigma(w_1 x_1 + w_2 x_2 + ... + w_n x_n)$$

This is the McCulloch-Pitts neuron model. The most commonly used activation function σ is the Rectified Linear Unit (ReLU):

$$\sigma(x) = x^+ = \max(0, x)$$

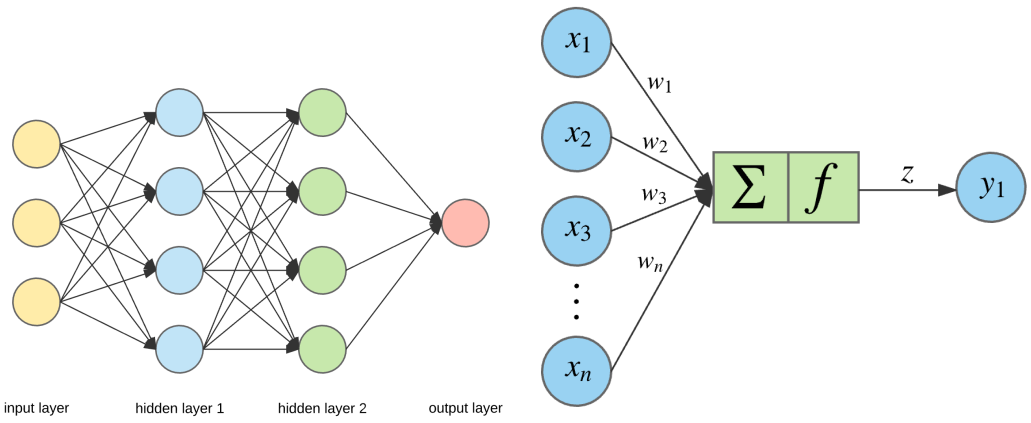

FIGURE 1.1: Feedforward Deep Neural Network and Single Neuron - McCulloch-Pitts model. (Retrieved from https://towardsdatascience.com)

1.2 Neural Networks as Dynamical Systems

A neural network can also be expressed as a function in matrix notation:

$$f(W, x) = W_H \sigma(W_{H-1} \sigma(...W_1 \sigma(W_0 x)...))$$

where W_n are the matrixes of the connection weights.

Training of this network is then done by minimizing the loss function:

$$\underset{W}{\text{minimize}} \quad L(W) = \sum_{j=0}^{N} ||f(W, x^j) - y^j||^2$$

The usual algorithm is called backpropagation. This algorithm has two steps: first the output of the network is calculated using the current weights. Then the error is propagated backward and the gradient is calculated. Then any gradient descent method can be used to find the step update.

A neural network can also be interpreted as a dynamical system where every layer is a state.

$$\begin{aligned} x_0 &= x \\ x_{k+1} &= \sigma(W_k x_k), \quad k = 0, ..., H-1 \\ y &= W_H x_H \end{aligned}$$

In this way optimization methods from Control Theory can be applied. In particular, training a neural network with ReLU activation function is equivalent to solving the following Optimal Control Problem (OCP)

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \sum_{j=0}^{N} ||W_H x_H^j - y^j||^2 \\ \text{subject to} \quad & x_{k+1}^j = \max(W_k x_k^j, 0), \quad k = 0, \ldots, H-1, j = 1, \ldots, N \end{aligned}$$

There are two approaches to solving an OCP. First is the sequential approach, where the states are eliminated using the dynamics. This is equivalent to the backpropagation algorithm which is the current standard method.

The other approach is the simultaneous approach, where the states are kept as variables and the dynamics are kept as constraints. In control theory this approach often works better for highly nonlinear problems, which is certainly the case for training neural networks. The simultaneous approach is novel to neural networks and will be the topic of this thesis.

The disadvantage of this method is the number of variables that need to be optimized is much larger. For a fully connected neural network of width W, each layer will contain W^2 weights. Combined with a depth D, that gives approximately $W^2 D$ weight variables to be optimized for both the backpropagation and simultaneous approach. Adding the states as variables however adds another WDN variables, where N is the number of samples in a training batch. The advantage of this method is that relaxing the states makes the problem more smooth, and will hopefully allow the optimization to converge more often to a good solution and not land in a bad local minimum.

Chapter 2

Initial exploration

In this chapter an initial exploration of the problem is done. This was done in MATLAB with the neural network toolbox for the backpropagation algorithm, and YALMIP with fmincon for the simultaneous approach.

2.1 First experiment

The first experiment conducted was to apply both algorithms to a test problem. A small neural network is constructed with 2 hidden layers with each layer containing 3 nodes with tansig activation function. This network is then trained to approximate a piece of a sine function.

The backpropagation algorithm converges every time for this problem, taking anywhere between 10 and 1000 iterations. The stopping criteria is that the validation error increases for 6 consecutive iterations.

The simultaneous approach also converges if the intial conditions are set correctly. The weight variables are randomly initialized and the state variables are initialized by simulating the network once using the input vector. Letting fmincon run for 200 iterations gives a good solution every time, but the algorithm runs much slower. Its not known if this is due to difference in optimization and the fact that the backpropagation algorithm is running on my GPU instead of the CPU.

2.2 Second experiment

For this experiment the same test problem is considered but with a different network. Here a shallow network 1 layer deep and 10 neurons wide is considered, with the ReLU activation function.

The backpropagation function will sometimes get stuck in a bad solution while training this network, but usually finds a good approximation.

For the simultaneous approach the constraints will first have to be adapted a bit for the ReLU function. The ReLU function can be transformed into smooth constraints as follows:

$$
\begin{gathered}
x_{k+1}^j = \max(W_k x_k^j, 0) \\
\Updownarrow \\
x_{k+1}^j = -\min(-W_k x_k^j, 0) \\
\Updownarrow \\
\min(x_{k+1}^j - W_k x_k^j) = 0 \\
\Updownarrow \\
(x_{k+1}^j - W_k x_k^j)^\top x_{k+1}^j = 0, \\
x_{k+1}^j \geq 0, x_{k+1}^j - W_k x_k^j \geq 0
\end{gathered}
$$

Even using these smooth constraints, the algorithm very often gets stuck in a bad local minimum.

2.3 Further exploration

There are many options that can be explored to further compare these algorithms.

- Activation function: ReLU is the most popular activation function in the field. Others are tansig, sigmoid, SoftPlus, leaky ReLU, etc.
- Network size: Networks can be up to thousands of neurons wide and hundreds of layers deep.
- Network architecture: There is a huge variety of possible network architectures. Convolutional neural networks for example are very popular for image recognition.
- Test problems: Neural networks have many applications. Some applications could benefit more from this training method than others.
- Optimization algorithm: fmincon is quite a general method, a more specific method might perform better
- Stopping criteria and initial conditions

Appendices

Appendix A

Source code

This appendix contains the source code of the experiments.

A.1 First experiment source

```
N = 21;
xin = linspace(0,1,21);
y = -sin(.8*pi*x);

w1 = sdpvar(3,1);
b1 = sdpvar(3,1);
x1 = sdpvar(3,N);
w2 = sdpvar(3,3,'full');
b2 = sdpvar(3,1);
x2 = sdpvar(3,N);
w3 = sdpvar(3,1);
b3 = sdpvar(1,1);


assign(w1,2*rand(3,1)-1);
assign(b1,2*rand(3,1)-1);
assign(x1,tansig(value(w1*xin+repmat(b1,1,N))));
assign(w2,2*rand(3,3)-1);
assign(b2,2*rand(3,1)-1);
assign(x2,tansig(value(w2*x1 +repmat(b2,1,N))));
assign(w3,2*rand(3,1)-1);
assign(b3,2*rand(1,1)-1);

res = w3'*x2 + b3 - y;
obj = res*res';
%
f1 = (x1-(w1*xin+repmat(b1,1,N)));
```

```
f2 = (x2-(w2*x1 +repmat(b2,1,N)));
% con = [f1 >= 0; x1 >= 0; f1.*x1 <= 0;
%        f2 >= 0; x2 >= 0; f2.*x2 <= 0];
% con = [x1 == max(w1*xin+repmat(b1,1,N),0);
%        x2 == max(w2*x1 +repmat(b2,1,N),0)];
con = [x1 == tansig(w1*xin+repmat(b1,1,N));
       x2 == tansig(w2*x1 +repmat(b2,1,N))];
ops = sdpsettings('usex0',1);
ops.fmincon.MaxFunEvals = 20000;
ops.fmincon.MaxIter = 200;

optimize(con,obj,ops);

%%
x1s = tansig(value(w1)*xin+value(b1));
x2s = tansig(value(w2)*x1s+value(b2));
ys = value(w3)'*x2s+value(b3);

hold off
plot(xin,y,'Linewidth',3);
hold on
plot(xin,ys,'g--','Linewidth',4);
```

A.1.1 Second experiment source

```
clear
N = 21;W = 10;
xin = linspace(0,1,21);
y = -sin(.8*pi*xin);

w1 = sdpvar(W,1);
b1 = sdpvar(W,1);
x1 = sdpvar(W,N);
w3 = sdpvar(W,1);
b3 = sdpvar(1,1);


assign(w1,2*rand(W,1)-1);
assign(b1,2*rand(W,1)-1);
assign(x1,poslin(value(w1*xin+repmat(b1,1,N))));
assign(w3,2*rand(W,1)-1);
assign(b3,2*rand(1,1)-1);

res = w3'*x1 + b3 - y;
obj = res*res';
```

```
%
f1 = (x1-(w1*xin+repmat(b1,1,N)));
%f2 = (x2-(w2*x1 +repmat(b2,1,N)));
con = [f1 >= 0; x1 >= 0; f1.*x1 <= 0;];
%        f2 >= 0; x2 >= 0; f2.*x2 <= 0];
% con = [x1 == max(w1*xin+repmat(b1,1,N),0);
%        x2 == max(w2*x1 +repmat(b2,1,N),0)];
% con = [x1 == tansig(w1*xin+repmat(b1,1,N));
%        x2 == tansig(w2*x1 +repmat(b2,1,N))];
ops = sdpsettings('usex0',1);
ops.fmincon.MaxFunEvals = 20000;
ops.fmincon.MaxIter = 200;

optimize(con,obj,ops);

%%
x1s = poslin(value(w1)*xin+value(b1));
%x2s = tansig(value(w2)*x1s+value(b2));
ys = value(w3)'*x1s+value(b3);

hold off
plot(xin,y,'Linewidth',3);
hold on
plot(xin,ys,'g--','Linewidth',4);
```