# 1

# Introduction to Deep Learning

## 1.1 Motivation

**Example: is AI like Alchemy?** A core of deep learning is to optimize a loss function with respect to a collection of model parameters. In NIPS 2017 talk, Rahimi gives a typical example for choosing parameters for a 2-layer neural network.

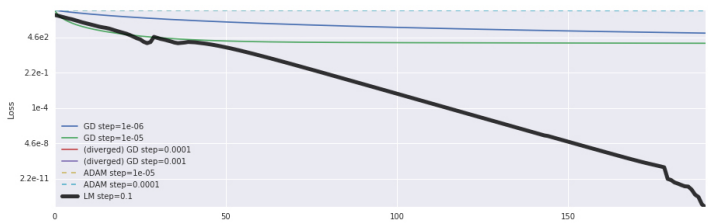


**Figure 1.1:** Numerical Results for solving $\min_{W_1,W_2} \hat{\mathbb{E}}\|W_1W_2x - Ax\|^2$

The gradient descent for solving this problem gets stuck at the objective value (error) around 400. In particular,

- It does not converge to a stationary point
- This phenomenon is not due to the statistical noise floor of the

dataset. Computing the expectation of the loss and directly minimize it leads to the same result.

Rahimi reveals that gradient descent cannot usually solve a 2-layer linear network. However, we expect it to solve a 1000-layer neural-net with 1 million data. It seems we don't understand neural-net optimization at all.

LeCun has a different pespective. He thinks that having theory for deep learning is important, but another goal is to invent new methods, new tricks, etc. Perhaps we don't have or don't need theory for neural networks.

### Comment from Ruoyu Sun

1. Do we need to study deep learning?

   For application researchers, there is no doubt. For applications, Neural-nets (not necessarily current deep learning) will probably become fundamental tool in many fields. For theoretical researchers such as optimizers, mathematicians, statisticians, this is still a big question. Deep learning is possible to become a theoretical field.

2. Do we need theory for deep learning?

   Yes, since we need to *udnerstand* why deep learning works. There are tons of hard deep learning problems, and we need to solve them. Moreover, theory can help deep learning work for other fields such as reinforcement learning, physics, biology, etc.

3. This course tries to offer answers to these questions, but it cannot answer them completely, not even a 20% answer. We will analysis deep learning from the perspective of optimization, and understand why some heuristics can help.

## 1.2 Syllabus

### Pre-requisite

- Pre-requisites: Calculus, Linear Algebra, Probability, Optimization, and Basics of Deep Learning.
- Related Subjects (helpful but not required): Basic knowledge from compute vision and NLP, the trend of Deep Learning and Artificial Intelligence.

**Grading Policy** 3-4 assignements and a simple course project.

**Course Objective and Audience** This course will talk about optimization theory of deep learning. This is not intended for practitioners with little interest in theory, and want to know how to tune parameters. Intended audience:

- Theorists (e.g., optimizers, machine learners, statisticians, mathematicians, theoretical computer scientists) who want to work on or interested in theory of deep learning;
- Practitioners interested in theory;
- Those who curious about frontiers of optimization in deep learning.

Also note that the theory discussed in this course does not necessarily solve real problems in deep learning, but it offers the logic of thinking that is the most helpful.

## 1.3 Neural Network Basis

Firstly, we will give mathematical descriptions of fully-connected neural networks. The Figure (1.2) gives a demostration of 3-layer fully-connected neural network. Neural networks give a function $f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}$ parameterized with $\theta \in \mathbb{R}^{d_\theta}$:

- Input: $x \in \mathbb{R}^{d_x}$;
- Output: $y \in \mathbb{R}^{d_y}$;
- A $L$-layer fully connected neural network consists of $L - 1$ hidden layers.

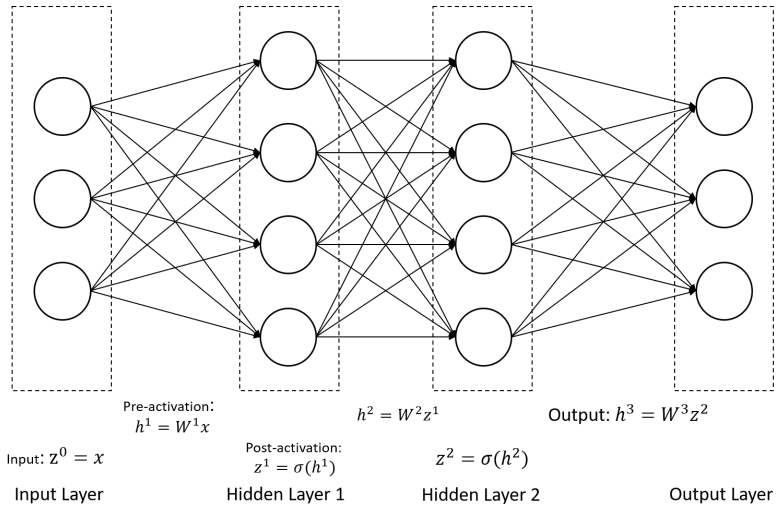


**Figure 1.2:** Example of a 3-layer fully-connected neural network.

- The values of the next hidden layer are a linear transformation of previous values, and then followed by a non-linear function:

$$\begin{aligned} &\text{pre-activiation}: \quad h^\ell = W^\ell z^{\ell-1}, \quad \ell = 1, \dots, L \\ &\text{post-activation}: \quad z^\ell = \phi(h^\ell), \quad \ell = 1, \dots, L \end{aligned}$$

  Here $W^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ denotes the weight matrix, and $\phi$ denotes the nonlinear function.

Using the notions above, the function $f_\theta$ can be written as

$$f_\theta(x) = W^L \phi(W^{L-1} \phi(\cdots \phi(Wx))) \tag{1.1}$$

**Remark 1.1.** The bias of the neural network is skipped. In general, it should be $z^\ell = \phi(W^\ell z^{\ell-1} + b^\ell)$.

**Remark 1.2.** There are other kinds of neural network, such as CNN, RNN, and ResNet.

**Why & When & How do we need neural-nets?** Consider the image classificaiton scenario. Imagine there is an orcle telling you the label

of the input image, say $f^*$. The motivation for using deep learning is that it can approximate such a function very well. The classical way for applying deep learning is the supervised learning:

> Given data $(x_i, y_i)$ for $i = 1, \ldots, n$, we need to find a model $f_\theta$ from a set of model candiates $\mathcal{F}$ such that $f_\theta(x_i) \approx y_i$, $i = 1, \ldots, n$.

This problem is often formulated as a finite-sum optimization problem:

$$\min_\theta \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i),$$

where $\ell(\cdot, \cdot)$ denotes the loss function.

1. When the representation model $f_\theta(x) = Wx$, and the loss is quadratic, the problems becomes a least-square linear regression problem

2. When the representation is a 2-layer neural network, say $f_\theta(x) = W^2W^1x$ and the loss is quadratic, the problem becomes

   $$\min_{W^2, W^1} \sum_i \|y_i - W^2W^1x_i\|^2 = \|Y - W^2W^1X\|_F^2$$

   When $X = I$, this problem reduces to the matrix factorization problem:

   $$\min_{U,V} \|Y - UV\|_F^2 \tag{1.2}$$

   For matrix $Y \in \mathbb{R}^{m \times n}$ and decision variables $U \in \mathbb{R}^{m \times p}, V \in \mathbb{R}^{p \times n}$, if $p < \text{rank}(Y)$, then the optimal solution

   $$U^* = \begin{pmatrix} \sqrt{\sigma_1}u_1 & \cdots & \sqrt{\sigma_p}u_p \end{pmatrix}, \quad V^* = \begin{pmatrix} \sqrt{\sigma_1}v_1 & \cdots & \sqrt{\sigma_p}v_p \end{pmatrix}^{\mathrm{T}}$$

   where $\{u_i, v_i, \sigma_i\}_{1:p}$ are from the first $p$ terms of the SVD decomposition of $Y$.

## 1.4 Gradient Explosion/Vanishing

Consider minimizing a quadratic loss for multi-layer neural network with scalar input:

$$\min_{w_1, \ldots, w_L} F(w) \triangleq \frac{1}{2}(1 - w_1 \cdots w_L)^2$$

**Solving by Classical Gradient Descent** If applying gradient descent, the step size should be bounded by 1 over the Lipschitz constant $\beta$ of the objective function. Here we compute this constant:

$$\frac{\partial F}{\partial w_1} = -(1 - w_1 \cdots w_L) w_2 \cdots w_L$$
$$\frac{\partial^2 F}{\partial w_1^2} = (w_2 \cdots w_L)^2$$

Note that $L \approx \lambda_{\max}(\nabla_w^2 F)$. Here we simply use $\frac{\partial^2 F}{\partial w_1^2}$ to loosely study how large (small)$\beta$ is.

- When $w_2 = \cdots = w_L = 2$, then $\beta = 2^{\mathcal{O}(L)}$
- When $w_2 = \cdots = w_L = \frac{1}{2}$, then $\beta = 2^{-\mathcal{O}(L)}$

Take $L = 100$ as an example. For the first initialization, the step-size is too big; for the second initialization, the step size is too small. This issue is called *Lipschitz constant explosion/vanishing*.