Lecture 2 Deep Residual Neural Networks

Shiwei Lan[1]

[1]School of Mathematical and Statistical Sciences
Arizona State University

STP598 Advanced Deep Learning Models
Spring 2026

Table of Contents

Gradient Descent Optimization

- Most deep learning algorithms involve optimization of some sort.
- Training neural network often relies on minimizing some objective (cost/loss) function $f(x)$.
- To minimize f, we would like to find the direction u in which f decreases the fastest by using the directional derivative:

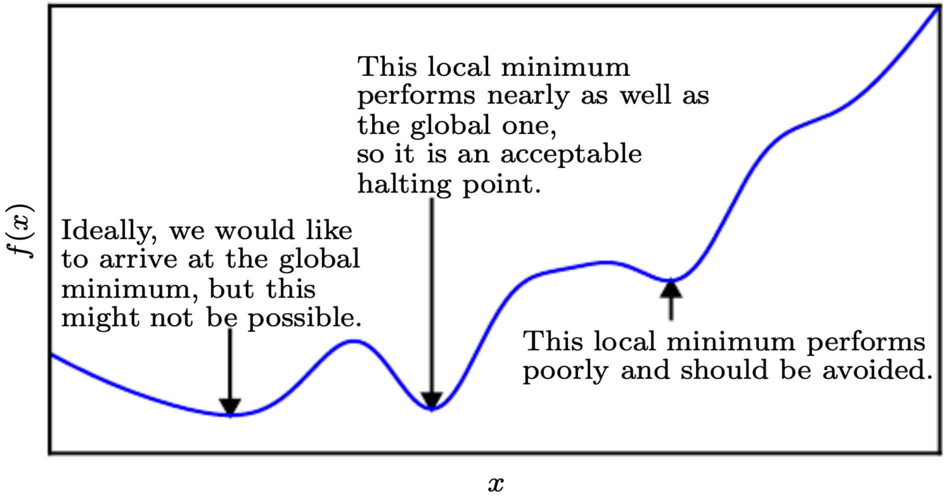
$$\min_{u,\|u\|=1} \left.\frac{\partial}{\partial\alpha}\right|_{\alpha=0} f(x+\alpha u) = \min_{u,\|u\|=1} u^T \nabla_x f(x) = \|\nabla_x f(x)\|_2 \min_{u,\|u\|=1} \cos\theta$$

- The minimal is achieved when $\theta = \pi$, i.e. the direction $u = -\nabla_x f(x)$ is the steepest descent or gradient descent.
- Then we update the state by

$$x' = x - \epsilon \nabla_x f(x)$$

where ϵ is called *learning rate*.

Challenge: Multimodalities



Other Challenges

There are other challenges like:

- overflow/underflow, e.g. softmax function.
- ill-conditioning: $f(x) = A^{-1}x$ where $A \in \mathbb{R}^{n\times n}$ with eigenvalues $\{\lambda_i\}$, then condition number is $\max_{i,j} |\lambda_i/\lambda_j|$.
- complex landscape, e.g. plateaus, saddle points, cliffs...
- expensive gradients: large data volume.

Training Deep Models

- Learning $\neq$ pure optimization.
- In most machine learning scenarios, we care about some performance performance measure P, defined with respect to test set.
- We reduce a different cost function $J(\theta)$ in the hope that doing so will improve P. This is in contrast to pure optimization with J as the goal.
- Typically, the cost function is defined as an expectation of some loss function $L(\cdot,\cdot)$, namely, risk,

$$J(\theta) = \mathrm{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y)$$

- In reality, we often minimize the an approximate version, empirical risk,

$$\tilde{J}(\theta) = \mathrm{E}_{(x,y)\sim \hat{p}_{data}} L(f(x;\theta), y) = \frac{1}{N}\sum_{i=1}^{N} L(f(x^{(i)};\theta), y^{(i)})$$

(Mini)-batch Algorithms

- Empirical risk minimization is prone to overfitting. In stead, we often consider a surrogate loss function, e.g. negative log-likelihood, i.e.

$$\theta_{ML} = \arg\max_{\theta} \sum_{i=1}^{N} \log p_{model}(x^{(i)}, y^{(i)}; \theta)$$

- To combat the issue of expensive gradients when N is large, a small batch of data size m is (randomly) chosen to approximate the gradient in gradient descent algorithms:

$$\theta' = \theta - \frac{N\epsilon}{m} \nabla_{\theta} \log p_{model}(x^{(i)}, y^{(i)}; \theta)$$

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) and its variants are probably the most used optimization algorithms for machine learning in general and for deep learning in particular.
- In practice, it is common to decay the learning rate ϵ linearly in the minibatch gradient descent until iteration τ:

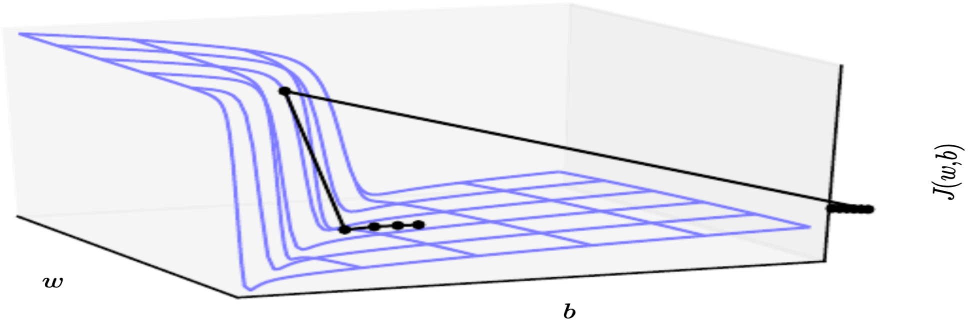
$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau, \quad \alpha = k/\tau.$$

such that the convergence condition, $\sum_{k=1}^\infty \epsilon_k = \infty$, $\sum_{k=1}^\infty \epsilon_k^2 < \infty$, is met.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$
end while

Challenge: cliff



- Like general gradient descent optimization, training neural network also faces the same challenges including cliffs, or exploding gradients.
- To alleviate such issue, gradient clipping is adopted when the norm of gradient $\|g\| > \mathrm{max\_norm}$ for some threshold $\mathrm{max\_norm}$:

$$g \leftarrow g \frac{\mathrm{max\_norm}}{\|g\|}$$

Challenge: compositions in deep models

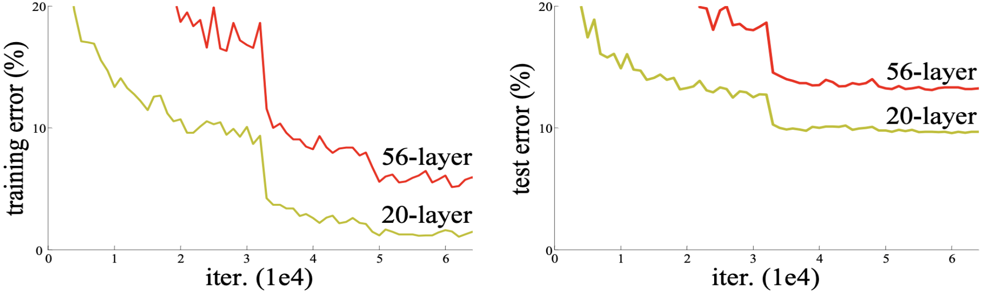
- Very deep models involve the composition of several functions or layers.
- In practice, when we update all of the layers simultaneously, unexpected results can happen because many functions composed together are changed simultaneously, e.g. $\hat{y} = x w_1 w_2 \cdots w_l$ where $h_i = h_{i-1} w_i$, then the gradient in back-propagation could be either too small or too large.
- To solve this issue, batch normalization is adopted.
- Given a minibatch of activations $\mathbf{H}$, we normalize $\mathbf{H}$ and replace it with

$$\mathbf{H}' = \frac{\mathbf{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}, \quad \boldsymbol{\mu} = \frac{1}{m}\sum_i \mathbf{H}_i, \quad \boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\mathbf{H} - \boldsymbol{\mu})_i^2}, \ \delta \approx 10^{-8}$$

- At test time, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ may be replaced by running averages that were collected during training time.

Table of Contents

Performance Degradation



- Even with the above remedy measures, training deep models may still suffer from the performance *degradation* issue as the model goes deeper.
- A natural question arises:
  Is learning better networks as easy as stacking more layers?

Deep Residual Neural Networks

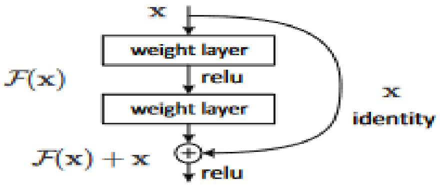
- *Deep Residual Learning for Image Recognition* by He et al (CVPR, 2016) provides a simple yet an effective solution.
- Instead of hoping each few stacked layers directly fit a desired underlying mapping, they let these layers fit a residual mapping and demonstrate that it is easier to optimize.
- It makes training very **deep** (100, 1000 layers or more) neural networks successful.
- It won the First place on ILSVRC 2015 classification task with ResNet152 and achieved 3.57% error on ImageNet test set.
- It won the First place on the task of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation in ILSVRC COCO 2015 competitions.

Deep ResNet Structure

- Consider the desired underlying mapping $\mathcal{H}(x)$ to be fit by a few stacked layers.
- Instead of stacking deep layers to directly train $\mathcal{H}(x)$, it lets the stacked nonlinear layers fit residual mapping $\mathcal{F}(x) = \mathcal{H}(x) - x$.
- Equivalently, ResNet learns the mapping of the following form

$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

- Such formulation can be realized by feedforward neural networks with "shortcut connects".



Deep ResNet Structure

- "If the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counter-part."
- "The degradation problem suggests that the solvers might have difficulties in approximating identity mappings by multiple nonlinear layers."
- Consider the building block

$$y = \mathcal{F}(x, \{W_i\}) + x$$

  where $\mathcal{F}(x, \{W_i\})$ represents residual mapping, e.g. $\mathcal{F} = W_2\sigma(W_1 x)$ with σ as ReLU activation, or CNN layers.
- When there is discrepancy between input and output dimensions, consider a projection matrix W_s to match the dimensions

$$y = \mathcal{F}(x, \{W_i\}) + W_s x$$

How does ResNet structure help?

- How does the residual block $y = \mathcal{F}(x) + x$ help with the gradient?
- Consider the gradient of some loss function L in the back-propagation:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\left(\frac{\partial \mathcal{F}}{\partial x} + \mathbf{I}\right)$$

- Why it works?

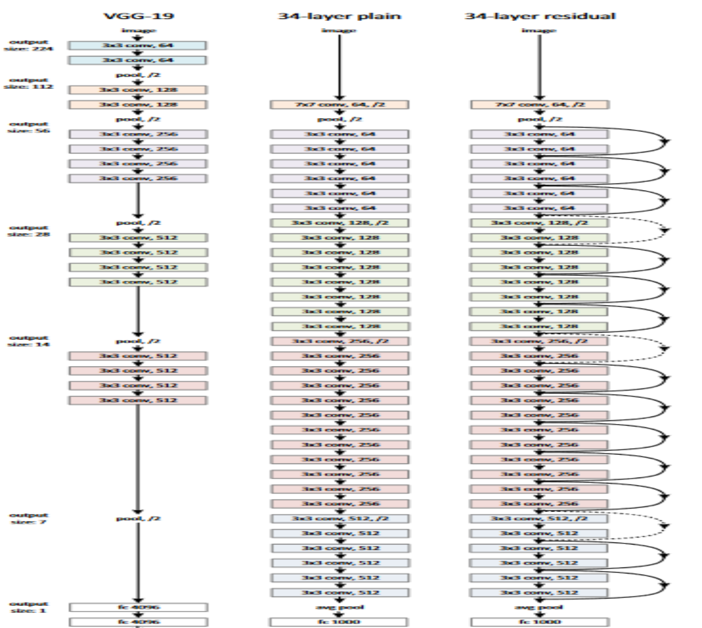
How does ResNet structure help?

- How does the residual block $y = \mathcal{F}(x) + x$ help with the gradient?
- Consider the gradient of some loss function L in the back-propagation:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\left(\frac{\partial \mathcal{F}}{\partial x} + \mathbf{I}\right)$$

- Why it works?
- Even if $\frac{\partial \mathcal{F}}{\partial x} \to 0$, $\frac{\partial L}{\partial x} \to \frac{\partial L}{\partial y}\mathbf{I}$!
- Gradients can flow directly backward, avoiding vanishing gradients!

Deep ResNet Structure Illustration



Deep ResNet Building Blocks

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of block

Deep ResNet Numerical Results

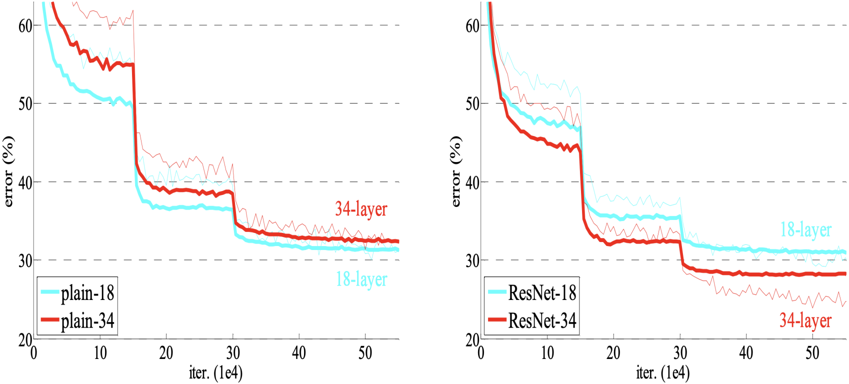


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Deep ResNet Numerical Results

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Deep ResNet Numerical Results

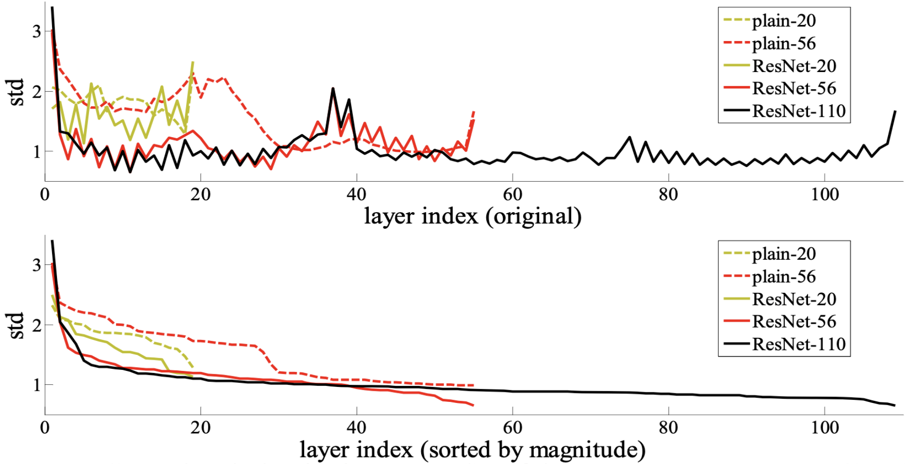


Table of Contents

Build ResNet

- We will build a ResNet and run it on `sol`.
- Check out some pre-trained ResNet https://colab.research.google.com/github/pytorch/pytorch.github.io/blob/master/assets/hub/pytorch_vision_resnet.ipynb.
- *How Deep Are Deep Gaussian Processes?* (Dunlop et al, 2018) shows that the effectiveness of DGP diminishes as the depth increases. Could residual structure save?
- This has been explored in ICLR 2015! – https://openreview.net/forum?id=JWtrk7mprJ.