Analyses of Deep Learning

STATS385
Stanford University
D. Donoho, V. Papyan, Y. Zhong
Details about course

- Wed 3:00-4:20 PM in Bishop Auditorium
- Sept 25 - Dec 6 (10 Weeks)
- Website: http://stats385.github.io
- Instructors:
  - David Donoho: donoho@stanxxx.edu
  - Yiqiao Zhong: yiqiaoz@stanxxx.edu
  - Vardan Panyan: panyan@stanxxx.edu
Deep learning is a transformative technology that has delivered impressive improvements in image classification and speech recognition. Many researchers are trying to better understand how to improve prediction performance and also how to improve training methods. Some researchers use experimental techniques; others use theoretical approaches. In this course we will review both experimental and theoretical analyses of deep learning. We will have 8 guest lecturers as well as graded projects for those who take the course for credit.

Instructors and office hours:

- David Donoho
  Sequoia 128

- Vardan Papyan
  Sequoia 208

- Yiqiao Zhong
  Packard 239
Analyses of Deep Learning (STATS 385)

Wednesday 3:00 - 4:20 PM
at Bishop Auditorium
Stanford University Fall 2019

https://stats385.github.io/
List of speakers

October 2: Stefano Soatto

October 9: Tengyu Ma

October 16: Jeffrey Pennington

October 23: Song Mei

October 30: Arthur Jacot

November 6: Aleksander Madry

November 13: Nati Srebro

November 20: Andrew Saxe

December 6: Vardan Papyan
Why are we here?

- Unprecedented success of deepnets
- Tremendous media attention
- Dramatic investments in deepnet technology
- Purely empirical understanding
- Perception: massive stakes
- Please see slides 2017 Theories of Deep Learning
Standard Notations in Deep Learning
Linear regression from a deep learning perspective

Linear regression:
\[
\min_\beta \sum_i \|y_i - \beta^T x_i\|_2^2
\]

Logistic regression:
\[
\min_\beta \sum_i \rho(\beta^T x_i y_i), \quad \rho(t) = -\ln(1+e^{-t})
\]

Multinomial logistic regression (cross-entropy loss):
\[
\min_\beta \sum_i \rho(\beta^T x_i), \quad \rho(z) = -\sum_c y_c \log \left( \frac{\exp\{z_c\}}{\sum_{c'} \exp\{z_{c'}\}} \right)
\]

General:
\[
\min_\beta \text{Loss}(f(x_i; \beta), y_i), \quad f(x_i; \beta) = \beta^T x_i
\]
Linear regression from deep learning perspective

```python
import math
import torch
import torch.nn as nn
import torch.optim as optim

n = 100000
p = 512
X = torch.randn(n, p)
beta = torch.randn(p, 1) / math.sqrt(p)
y = torch.mm(X, beta) + 0.5 * torch.randn(n, 1)

class LinearReg(nn.Module):
    def __init__(self):
        super(LinearReg, self).__init__()
        self.linear = torch.nn.Linear(p, 1)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```
Stochastic gradient descent (SGD)

\[
\beta_{t+1} = \beta_t - \eta_t \sum_{i \in B} \nabla \beta \text{Loss}_i(f(x_i; \beta_t), y_i)
\]

- Batch size
- Epoch
- Learning rate = step size
- Learning rate scheduler
- Variations:
  - Weight decay = ridge regularization
  - Momentum
- Beyond SGD:
  - RMSProp, Adagrad, Adam
Optimizing linear regression using SGD

```python
from torch.optim.lr_scheduler import MultiStepLR
batch_sz = 128
epochs = 9
model = LinearReg()
optimizer = optim.SGD(model.parameters(), lr=0.01,
momentum=0.9, weight_decay=5e-4)
scheduler = MultiStepLR(optimizer, milestones=[3,6], gamma=0.2)
Loss = nn.MSELoss()

for epoch in range(epochs):
    scheduler.step()
    for idx in range(n // batch_sz):
        min_idx = batch_sz*idx
        max_idx = batch_sz*idx+batch_sz
        x_batch, y_batch = X[min_idx:max_idx,:,], y[min_idx:max_idx]

        optimizer.zero_grad()
        Loss(model(x_batch), y_batch).backward()
        optimizer.step()
```

AUTOGRAD!
Optimizing linear regression using SGD

Learning rate initialized to 0.01 and decreased by factor of 0.2 at $\frac{1}{3}$ and $\frac{2}{3}$ of iterations
From linear regression to feedforward fully connected neural network

Regression:

\[ f(x_i; \beta) = \beta^T x_i \]

Fully connected feedforward neural network:

\[ f(x_i; \beta) = W_3 \sigma(W_2 \sigma(W_1 x_i)) \quad \sigma(x) = \max(x, 0) \]

A cascade of linear and non-linear operators.
Terminology (use these words & symbols)

- Weights: $\mathbf{W}$  
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\delta$
- Gradients
Terminology (use these words & symbols)

- **Weights**: $W$
- **Biases**: $b$
- **Features (post-activations)**: $h$
- **Pre-activations**: $z$
- ** Masks**: $D$
- **Logits**
- **Loss**: $\mathcal{L}(\theta)$
- **Backpropagated errors**: $\delta$
- **Gradients**

![Diagram of a neural network with labeled nodes and terms](image)
Terminology (use these words & symbols)

- Weights: $W$
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\delta$
- Gradients

$x_i \xrightarrow{W_1} \text{ReLU} \xrightarrow{h_1} W_2 \xrightarrow{\text{ReLU}} W_3 \xrightarrow{\text{logits}} \text{MSE Loss}$
Terminology (use these words & symbols)

- Weights: $W$
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\delta$
- Gradients

\[ x_i \xrightarrow{W_1} z_1 \xrightarrow{\text{ReLU}} z_2 \xrightarrow{W_2} \xrightarrow{\text{ReLU}} z_3 \xrightarrow{W_3} \text{logits} \xrightarrow{\text{MSE Loss}} y_i \]
Terminology (use these words & symbols)

- Weights: $\mathbf{W}$  
- Biases: $b$
- Features (post-activations): $\mathbf{h}$
- Pre-activations: $\mathbf{z}$
- Masks: $\mathbf{D}$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\mathbf{\delta}$
- Gradients

$x_i \rightarrow W_1 \rightarrow $ ReLU $\rightarrow W_2 \rightarrow $ ReLU $\rightarrow W_3 \stackrel{\text{logits}}{\rightarrow} $ MSE Loss $y_i$
Terminology (use these words)

- Weights: $W$
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\delta$
- Gradients

Diagram:

$x_i \xrightarrow{W_1} \text{ReLU} \xrightarrow{W_2} \text{ReLU} \xrightarrow{W_3} \text{logits} \xrightarrow{\text{MSE Loss}} y_i$
Terminology (use these words)

- Weights: $W$
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- **Loss:** $\mathcal{L}(\theta) = \min \| W_3 \sigma(W_2 \sigma(W_1 x_i)) - y_i \|^2_2$
- Backpropagated errors: $\delta$
- Gradients

$$x_i \xrightarrow{W_1 \text{ ReLU}} W_2 \xrightarrow{\text{ReLU}} W_3 \xrightarrow{\logits} \text{MSE Loss}$$
Terminology (use these words)

- Weights: \( \mathbf{W} \)
- Biases: \( \mathbf{b} \)
- Features (post-activations): \( \mathbf{h} \)
- Pre-activations: \( \mathbf{z} \)
- Masks: \( \mathbf{D} \)
- Logits
- Loss: \( \mathcal{L}(\theta) \)

- Backpropagated errors: \( \delta \)
- Gradients

\[
\begin{align*}
\mathcal{L} \text{ (logits)} &= \delta_3 = \text{logits} - y_i \\
\mathcal{L} \text{ (z_2)} &= \delta_2 = D_2 W_3^T \delta_3 \\
\mathcal{L} \text{ (z_1)} &= \delta_1 = D_1 W_2^T \delta_2
\end{align*}
\]

\[x_i \xrightarrow{W_1} \text{ReLU} \xrightarrow{W_2} \text{ReLU} \xrightarrow{W_3} \text{logits} \xrightarrow{\text{MSE Loss}} y_i\]
Terminology (use these words)

- Weights: $W$
- Biases: $b$
- Features (post-activations): $h$
- Pre-activations: $z$
- Masks: $D$
- Logits
- Loss: $\mathcal{L}(\theta)$
- Backpropagated errors: $\delta$
- Gradients

$$\frac{\mathcal{L}}{W_3} = \delta_3 h_2^T$$
$$\frac{\mathcal{L}}{W_2} = \delta_2 h_1^T$$
$$\frac{\mathcal{L}}{W_1} = \delta_1 x_i^T$$

$x_i$ $\rightarrow$ $W_1$ $\rightarrow$ ReLU $\rightarrow$ $h_1$ $\rightarrow$ $W_2$ $\rightarrow$ ReLU $\rightarrow$ $h_2$ $\rightarrow$ $W_3$ $\rightarrow$ logits $\rightarrow$ MSE Loss

$y_i$
Feedforward fully connected network in PyTorch

```python
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(p, p)
        self.fc2 = nn.Linear(p, p)
        self.fc3 = nn.Linear(p, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```
Linear regression trained with a neural net

Learning rate initialized to 0.01 and decreased by factor of 0.2 at $\frac{1}{3}$ and $\frac{2}{3}$ of iterations
LeNet: first success of CNNs

- Fully connected layers are not enough for computer vision
- Backpropagation applied to handwritten ZIP code recognition (1989)
- Convolutional layers
- Pooling layers
ImageNet

- Created by Fei-Fei Li
- Crowdsourced annotations
- More than 20,000 classes
- Image size: variable-resolution, often $224 \times 224 \times 3$ after cropping
- ILSVRC competition: Subset of 1000 classes from ImageNet; training set contains 1.2 million images.
AlexNet

- First use of ReLU
- Dropout 0.5 (explained later)
- Batch size 128
- SGD Momentum 0.9
- Learning rate $1e^{-2}$, reduced by 10 manually when validation accuracy plateaus
- $5e^{-4}$ weight decay
Remedy to optimization/generalization problems

- Dropout
- Skip connections
- Batch normalization

*During training*, randomly zeros/removes a fraction of nodes for each iteration
Remedy to optimization/generalization problems

- Dropout
- **Skip connections**
- Batch normalization

General form: \( \tilde{z}_\ell = z_\ell + F(z_\ell, \theta_\ell) \)
Remedy to optimization/generalization problems

- Dropout
- Skip connections
- Batch normalization

\[ \text{BN}_{\gamma,\beta}(x) = \gamma \hat{x} + \beta, \text{ where } \hat{x} = \frac{x - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \]
ILSVRC competition
Intriguing phenomena in deep learning
First layer weights of AlexNet
Same filters obtained from sparse coding
Transfer learning

- Low layer features learned on one dataset can be transferred to another similar dataset.
Adversarial examples (Aleksander Madry, Nov. 6th)

- Adding small perturbations can change drastically prediction results
Interpolation regime

- Overparameterization can memorize & generalize.
Double descent curve (Song Mei, Oct. 23rd)

- Under overparameterization, new phenomenon beyond “bias-variance” tradeoff.
- Belkin et al., 2018.
- Hastie et al., 2019
Implicit bias (Srebro, Nov. 13th, Tengyu Ma Oct. 9th)

- Neyshabur et al., In search of the real inductive bias, ICLR, 2015.
Optimization landscape

Deepnet spectra (Vardan Papyan Dec. 6)

- **Weights**: Martin and Mahoney (2018)
- **Fisher information matrix**: Papyan (2019), Li et al. (2019)
- **Gradients**: Gur-Ari et al. (2018)
- **Features**: Verma et al. (2019)
- **Backpropagated errors**: Oymak et al. (2019)
Deepnet spectra: Hessian

LeCun et al. (1998)
MNIST downsampled to 10 × 10 pixels

Dauphin et al. (2014)

Sagun et al. (2017)
100 dimensional input, two hidden layers, 30 hidden units each

Papyan (2019)
VGG11, 28 million parameters
Towards understanding phenomena
Generalization (Srebro, Nov. 13th, Tengyu Ma Oct. 9th)

- Generalization error bounds based on different complexity measures.
- Uniform control over a function class, no generative model.
- Bartlett et al., 2017.
- Neyshabur et al., 2018.
- Arora et al., 2018.
- Wei and Ma, 2019.

\[ \mathcal{R}_S(\mathcal{F}) = \frac{1}{n} \mathbb{E}_{\xi \sim \{\pm 1\}^n} \left[ \sup_{f \in \mathcal{F}} \sum_{i=1}^{n} \xi_i f(x_i) \right] \]
Kernels (Arthur Jacot, Oct. 30, Jeffrey Pennington, Oct. 16)

- Under certain limits, training and inference is characterized by kernels.

First-order expansion:

\[ f(x; \theta) \approx f(x; \theta_0) + \langle \theta - \theta_0, \nabla_\theta f(x; \theta_0) \rangle \]

Induced Kernel:

\[ K(x, x') = \langle \nabla_\theta f(x; \theta_0), \nabla_\theta f(x'; \theta_0) \rangle \]
Mean-field perspective (Song Mei, Oct. 23rd)

- Understanding training dynamics (SGD trajectory)

\[
\hat{y}(x; \theta) = \frac{1}{N} \sum_{i=1}^{N} \sigma_*(x; \theta) \xrightarrow{N \to \infty} \int \sigma_*(x; \theta) \rho(d\theta)
\]

SGD Dynamics of $\theta$  
Gradient Flow of $\rho$
Summary

● Summary of basic concepts.
● Intriguing phenomena:
  ○ First layer filters.
  ○ Transfer learning.
  ○ Adversarial examples.
  ○ Interpolation regime.
  ○ Double descent curve.
  ○ Implicit bias.
  ○ Optimization landscape.
  ○ Structure in deepnet spectra.
● Attempts at understanding phenomena:
  ○ Generalization bounds.
  ○ Neural tangent kernels.
  ○ Mean-field approach.