Mathematics of Data: From Theory to Computation

Prof. Volkan Cevher
volkan.cevher@epfl.ch

Lecture 7: Optimization for Deep Learning

Laboratory for Information and Inference Systems (LIONS)
École Polytechnique Fédérale de Lausanne (EPFL)

EE-556 (Fall 2019)

License Information for Mathematics of Data Slides

Outline

- This class
  - From convex to nonconvex optimization
  - Backpropagation
  - Convergence of SGD in nonconvex problems
  - Escaping saddle points
  - Overparametrization
  - Generative Adversarial Networks
  - Reinforcement Learning
- Next class
  - Composite convex minimization

Recommended reading material

- I. Goodfellow; Y. Bengio and A. Courville *Deep Learning*, Chapters 6 and 8. MIT Press. 2016.
- R. Ge; F. Huang; C. Jin and Y. Yuan *Escaping from saddle points: Online stochastic gradient for tensor decomposition* In Conference on Learning Theory. 2015.

Remark about notation for this lecture

For consistency with the deep learning literature, we use the following notation:

	Previous lectures	This lecture
data/sample	$\mathbf{a}$	$\mathbf{x}$
label	b	y
bias	μ	b
weight	$\mathbf{x}$	W, β, B

Parameters are usually named *weights* and *biases* and are denoted by W and b, respectively.

Power of linear classifiers–I

Problem (Recall: Logistic regression)

Given a sample vector $\mathbf{x}_i \in \mathbb{R}^d$ *and a binary class label* $y_i \in \{-1, +1\}$ $(i = 1, \ldots, n)$, *we define the conditional probability of* y_i *given* $\mathbf{x}_i$ *as:*

$$\mathbb{P}(y_i | \mathbf{x}_i, \beta) \propto 1/(1 + e^{-y_i(\langle \beta, \mathbf{x}_i \rangle)}),$$

where $\beta \in \mathbb{R}^d$ *is some weight vector.*

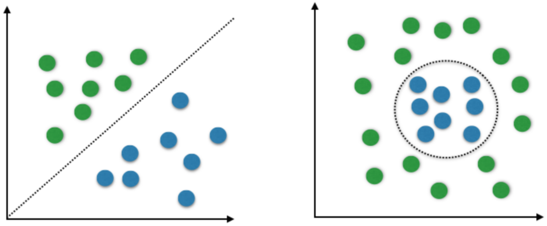
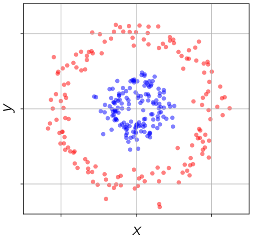


Figure: Linearly separable versus nonlinearly separable dataset

Power of linear classifiers–II

- Lifting dimensions to the rescue
  - Convex optimization objective
  - Might introduce the curse-of-dimensionality
  - Possible to avoid via kernel methods, such as SVM



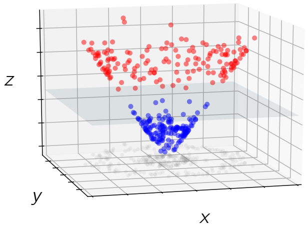
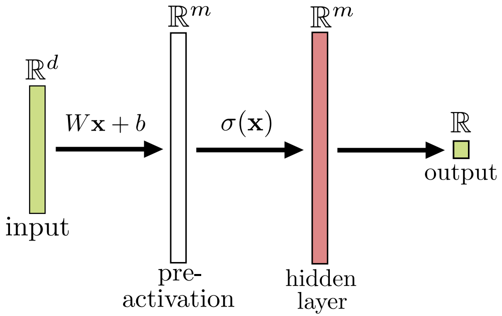


Figure: Non-linearly separable data (left). Linearly separable in $\mathbb{R}^3$ via $z = \sqrt{x^2 + y^2}$ (right).

An important alternative for non-linearly separable data

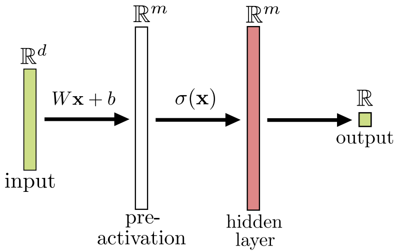


Definition (1-hidden-layer network with m neurons)

$$f(\mathbf{x}; \beta, W, b) = \beta^T \sigma(W\mathbf{x} + b)$$

- *Parameters:* $\beta \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times d}$ *(weights)*, $b \in \mathbb{R}^m$ *(bias)*
- *Activation function:* $\sigma : \mathbb{R} \to \mathbb{R}$

Why neural networks?: An approximation theoretic motivation



Caveat

The number of neurons m needed to approximate some function g can be exponentially large!

Theorem (Universal approximation (Cybenko, 1989) [2])

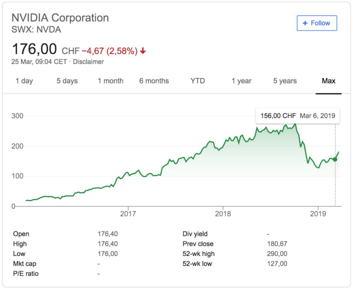
Let $\sigma(\cdot)$ be a nonconstant, bounded, and increasing continuous function. Let $I_d = [0,1]^d$. The space of continuous functions on I_d is denoted by $\mathcal{C}(I_d)$.

Given $\epsilon > 0$ and $g \in \mathcal{C}(I_d)$ there exists a 1-hidden-layer network f with m neurons such that f is an ϵ-approximation of g, i.e.,

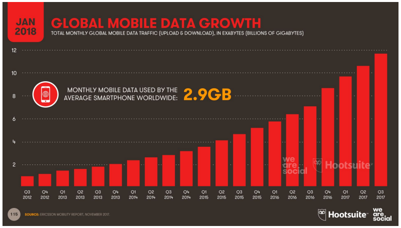
$$\sup_{\mathbf{x} \in I_d} |g(\mathbf{x}) - f(\mathbf{x})| \leq \epsilon$$

Why were NNs not popular before 2010?

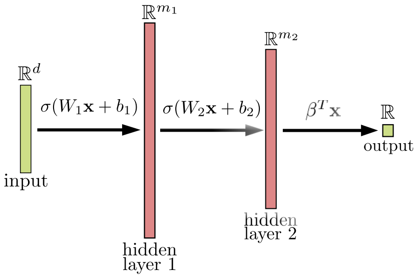
- too big to optimize!
- did not have enough data
- could not find the optimum via algorithms



Open	176,40	Div yield	-
High	176,40	Prev close	180,67
Low	176,00	52-wk high	290,00
Mkt cap	-	52-wk low	127,00
P/E ratio	-		



A natural generalization: Multilayer neural networks



Definition (2-hidden-layer network)

$$f(\mathbf{x};\beta,W_1,b_1,W_2,b_2)=\beta^T\sigma(W_2\sigma(W_1\mathbf{x}+b_1)+b_2)$$

- *Parameters:* β, W_1, W_2 *(weights)* b_1, b_2 *(biases) of appropriate size.*

k-layer networks are constructed analogously

The Landscape of ERM with multilayer networks

Recall: Empirical risk minimization (ERM)

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a multilayer network and let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be a sample with $y_i \in \{-1, 1\}$ and $\mathbf{x}_i \in \mathbb{R}^n$. The *empirical risk minimization* (ERM) is defined as

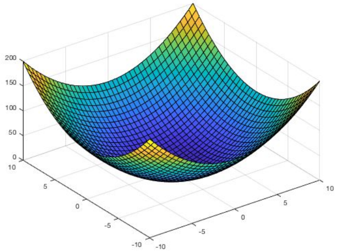
$$\min_{\theta} \left\{ R_n(\theta) := \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(\mathbf{x}_i; \theta), y_i) \right\} \tag{1}$$

where $\mathcal{L}(f(\mathbf{x}_i; \theta), y_i)$ is the value of a loss function on the sample $(\mathbf{x}_i, y_i)$ and θ are the parameters of a network f.

Some frequently used loss functions

- $\mathcal{L}(f(\mathbf{x}), y) = \log(1 + \exp(-yf(\mathbf{x})))$ (logistic loss)
- $\mathcal{L}(f(\mathbf{x}), y) = (y - f(\mathbf{x}))^2$ (squared error)
- $\mathcal{L}(f(\mathbf{x}), y) = \max(0, 1 - yf(\mathbf{x}))$ (hinge loss)

The Landscape of ERM with multilayer networks



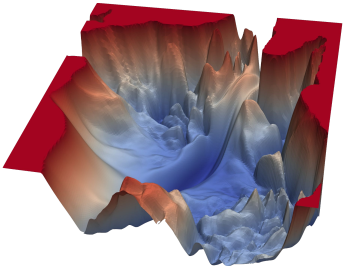


Figure: convex (left) vs non-convex (right) optimization landscape

Conventional wisdom in ML until 2010:
Simple models + simple errors

Why is the loss non-convex?

Example

Consider a 1-hidden-layer network $f : \mathbb{R} \to \mathbb{R}$ with activation function $\sigma(x) = x$, one hidden node and no bias

$$f(\mathbf{x}; w_1, w_2) = w_2 w_1 \mathbf{x}$$

with $w_1, w_2 \in \mathbb{R}$. For a sample $(\mathbf{x}_0, y_0) = (1, 1)$ the squared error is

$$(y_0 - f(\mathbf{x}_0; w_1, w_2))^2 = (1 - w_2 w_1)^2$$

Show that it is neither convex nor concave.

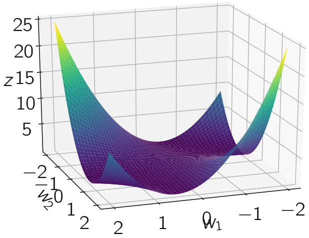


Figure: Loss surface $(1 - w_2 w_1)^2$

- non-convexity even though activation function σ is linear

What is the role of the activation function σ?

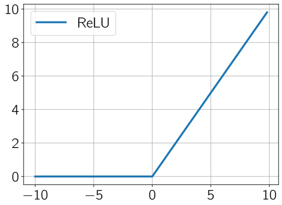
Theorem (Universal approximation (Leshno, 1993) [12])

1-hidden-layer networks have the universal approximation property if and only if σ is continuous and not a polynomial.

Example

- If $\sigma(x) = x$, the network computes an affine function.
- $W_2(W_1 x + b_1) + b_2 = Wx + b$.
- Cannot approximate non-affine functions.

ReLU: Rectified Linear Unit
$\sigma(x) = \max(0, x)$



$\sigma(x) = \text{Tanh}(x)$

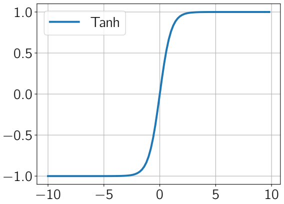


Figure: some activation functions for which universal approximation holds

Supervised learning: Multi-class classification

- So far: Only single output networks

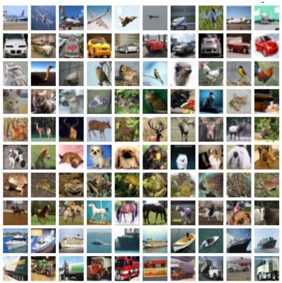


Figure: CIFAR10 dataset: 60000 32x32 color images (3 channels) from 10 classes

Figure: MNIST dataset: 60000 28x28 grayscale images (1 channel) from 10 classes

Goal

Image-label pairs $(\mathbf{x}, y) \subseteq \mathbb{R}^d \times \{1, \ldots, c\}$ follow an unknown distibution $\mathbb{P}$. Find $f : \mathbb{R}^d \rightarrow \{1, \ldots, c\}$ with minimum *misclassification probability*

$$\min_{f \in \mathcal{F}} \mathbb{P}(f(\mathbf{x}) \neq y)$$

Using networks for multi-class classification

Definition (Multi-output network)

A 1-hidden-layer network with c outputs $f : \mathbb{R}^d \to \mathbb{R}^c$ is defined as

$$f(\mathbf{x}; W_1, b_1, B) := B\sigma(W_1\mathbf{x} + b_1)$$

with $b_1 \in \mathbb{R}^d$, $W_1 \in \mathbb{R}^{m\times d}$ and $B \in \mathbb{R}^{c\times m}$.

- Single output networks correspond to $c = 1$

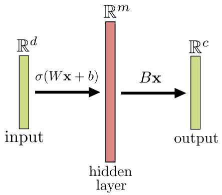


Figure: Multi-output 1-layer network

Definition (Score-based classifier)

For a network f define $i_f : \mathbb{R}^d \to \{1, \ldots, c\}$ as

$$i_f(\mathbf{x}) = \underset{i\in\{1,\ldots,c\}}{\arg\max}\; f_i(\mathbf{x})$$

Example:

$$f(\mathbf{x}_0) = \begin{bmatrix} 0.1 \\ -0.8 \\ \mathbf{1.4} \\ 1.1 \end{bmatrix} \implies i_f(\mathbf{x}_0) = 3$$

Cross-entropy loss for multiclass classification

- Goal: define a differentiable loss that correlates with misclassification error

Definition (Cross-entropy loss)

Let $\mathbf{x} \in \mathbb{R}^d$ *be a sample with label* $y \in \{1, \dots, c\}$

$$\mathcal{L}(f(\mathbf{x}), y) = -\log\left(\frac{\exp(f(\mathbf{x})_y)}{\sum_{j=1}^{c} \exp(f(\mathbf{x})_j)}\right)$$

$$f(\mathbf{x}_0) = \begin{bmatrix} 0.1 \\ \mathbf{-0.8} \\ \mathbf{1.4} \\ 1.1 \end{bmatrix} \quad \begin{array}{l} \mathcal{L}(f(\mathbf{x}_0), 2) = 2.95 \\ \\ \mathcal{L}(f(\mathbf{x}_0), 3) = 0.75 \end{array}$$

Generalizes logistic loss to multi-class problems

Minimization of the loss function

In order to use first order methods, we need to derive the gradient

$$\nabla_\theta R_n(\theta) := \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \mathcal{L}(f(\mathbf{x}_i;\theta), y_i) := \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \mathcal{L}_i(\theta) \tag{2}$$

where $\theta = [W_1, b_1, \ldots, W_k, b_k, \beta]$ are the weights and biases of the network.

Example (Naive computation of the gradient)

Let $f(\mathbf{x}; W, \beta) = \beta^T\sigma(W\mathbf{x})$, and $\mathcal{L}_i(W,\beta) = (y_i - \beta^T\sigma(W\mathbf{x}_i))^2$ be the loss on a sample, then

$$\frac{\partial \mathcal{L}_i}{\partial \beta} = -2(y_i - \beta^T\sigma(W\mathbf{x}_i))\sigma(W\mathbf{x}_i) \tag{3}$$

$$\frac{\partial \mathcal{L}_i}{\partial W} = -2(y_i - \beta^T\sigma(W\mathbf{x}_i))\beta \odot \sigma'(W\mathbf{x}_i)\mathbf{x}_i^T \tag{4}$$

where $\odot$ denotes element-wise product of vectors.

Many similar terms in both derivatives $\Rightarrow$ Inefficient to compute them independently

Backpropagation

- Recursive computation of the derivative $\nabla_\theta \mathcal{L}_i(\theta)$
  1. **Forward pass:** Compute all pre-activation and hidden layer values
  2. **Backward pass:** Compute the derivative of $\mathcal{L}_i$ with respect to the weights and biases, from last to first layer.

Complexity of computing $\nabla_\theta \mathcal{L}_i(\theta)$

Method	Complexity
Naive derivative	$\mathcal{O}(k^2 H^2)$
Backpropagation	$\mathcal{O}(k H^2)$

Where H is number of neurons per layer and k is the number of layers.

⋆Forward pass

Forward pass scheme
Input: $\mathbf{x}^{(\mathbf{0})} = \mathbf{x}$, $W^{(l)}$ and $b^{(l)}$ for $l = 1, \ldots, k$.
1. For $l = 1, \ldots, k$ Compute $\mathbf{u}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)}$ Compute $\mathbf{x}^{(l)} = \sigma(\mathbf{u}^{(l)})$

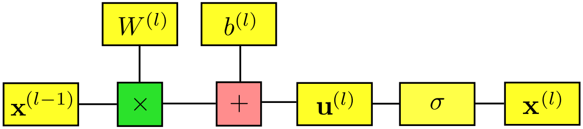


Figure: Computation of $\mathbf{u}^{(l)}$ and $\mathbf{x}^{(l)}$ starting from $\mathbf{x}^{(l-1)}$

⋆Backward pass

Suppose $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$ is given, as well as all pre-activation and hidden layer values.

- **Goal**: obtain $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$.

1.
$$\mathbf{u}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)} \Rightarrow \begin{cases} \frac{\partial \mathcal{L}}{\partial W^{(l)}} & = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}}(\mathbf{x}^{(l-1)})^T \\ \frac{\partial \mathcal{L}}{\partial b^{(l)}} & = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} \end{cases} \qquad \text{(chain rule)}$$

2.
$$\mathbf{x}^{(l)} = \sigma(\mathbf{u}^{(l)}) \Rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \odot \sigma'(\mathbf{u}^{(l)}) \qquad \text{(chain rule)}$$

   Where $\odot$ is the Hadamard product (element-wise product).

3. Finally we have

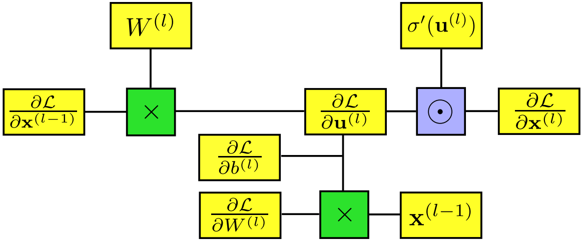
$$\mathbf{u}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)} \Rightarrow \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = (W^{(l)})^T \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} \qquad \text{(chain rule)}$$

⋆Backward pass

Backward pass scheme

Input: Gradient of the loss w.r.t. the last layer values $\partial \mathcal{L} / \partial \mathbf{x}^{(k)}$

```
1. For $l = k, \dots, 1$
       Compute $\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \odot \sigma'(\mathbf{u}^{(l)})$
       Compute $\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} (\mathbf{x}^{(l-1)})^T$, $\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}}$
       Compute $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = (W^{(l)})^T \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}}$
```



$W^{(l)}$
$\sigma'(\mathbf{u}^{(l)})$
$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$
$\times$
$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}}$
$\odot$
$\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$
$\frac{\partial \mathcal{L}}{\partial b^{(l)}}$
$\frac{\partial \mathcal{L}}{\partial W^{(l)}}$
$\times$
$\mathbf{x}^{(l-1)}$

Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

⋆Complexity of Backpropagation

The size of each layer (including input) is $\mathcal{O}(H)$, and the number of layers is $\mathcal{O}(k)$.

Forward pass scheme

```
1. For $l = 1, \dots, k$
    ▶ $\mathbf{u}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)} \Rightarrow \mathcal{O}(H^2)$
    ▶ $\mathbf{x}^{(l)} = \sigma(\mathbf{u}^{(l)}) \Rightarrow \mathcal{O}(H)$
```

Forward pass is $\mathcal{O}(kH^2)$

Backward pass scheme

```
1. For $l = k, \dots, 1$
    ▶ $\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}} \odot \sigma'(\mathbf{u}^{(l)}) \Rightarrow \mathcal{O}(H)$
    ▶ $\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} (\mathbf{x}^{(l-1)})^T \Rightarrow \mathcal{O}(H^2)$
    ▶ $\frac{\partial \mathcal{L}}{\partial b^{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} \Rightarrow \mathcal{O}(1)$
    ▶ $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}} = (W^{(l)})^T \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(l)}} \Rightarrow \mathcal{O}(H^2)$
```

Backward pass is $\mathcal{O}(kH^2)$

Towards training with neural networks

- What do we have at hand?
  1. Loss function $\mathcal{L}(\theta)$ from multi-layer, multi-class, etc.
  2. First-order gradient via backpropagation $g = \nabla\mathcal{L}(\theta)$
- Barriers to training of neural networks:
  1. Curse-of-dimensionality $\rightarrow$ first-order methods
  2. Non-convexity $\rightarrow$ stochasticity and momentum
  3. Ill-conditioning $\rightarrow$ adaptive gradient methods

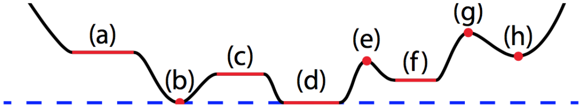


Figure: A non-convex function. (a) and (c) are plateaus, (b) and (d) are global minima, (f) and (h) are local minima, (e) and (g) are local maxima. [8]

Recall: Stochastic Gradient Descent (SGD)

Vanilla Minibatch SGD

Input: learning rate $\{\gamma\}_{t=0}^{N-1}$

```
1. initialize $\theta_0$
2. For t = 0, 1,... ,N-1:
       obtain the minibatch gradient $\hat{\mathbf{g}}_t$
       update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \hat{\mathbf{g}}_t$
```

Convergence of SGD in non-convex problems

Assumptions

1. Function $\mathcal{L}$ is lower bounded: $\exists \theta^*$ s.t. $\forall \theta \in \Theta, \mathcal{L}(\theta) \geq \mathcal{L}(\theta^*)$
2. Function $\mathcal{L}$ has Lipschitz continuous gradient:
$$\|\nabla \mathcal{L}(\theta_1) - \nabla \mathcal{L}(\theta_2)\|_2 \leq L\|\theta_1 - \theta_2\|_2 \tag{5}$$
3. The unbiased stochastic gradient $\hat{\mathbf{g}_\theta}$ has bounded variance.
$$\mathbb{E}(\hat{\mathbf{g}}) = \mathbf{g} \tag{6}$$
$$\mathbb{E}(\|\hat{\mathbf{g}} - \mathbf{g}\|_2^2) \leq \sigma^2 \tag{7}$$

Theorem (Convergence of SGD in non-convex problems [1])

For SGD with assumptions above, N iterations and stepsize $\gamma_t = \frac{1}{L\sqrt{N}}$ we have
$$\mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N-1}\|\hat{\mathbf{g}}_t\|_2^2\right] \sim \mathcal{O}\left(\frac{1}{\sqrt{N}}\right) \tag{8}$$

- Convergence is captured by the gradient norm

*Convergence of SGD

Proof

Take the assumption 2 and algorithmic update policy $\theta_{t+1} = \theta_t - \gamma \hat{\mathbf{g}}_t$

$$\begin{aligned}\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t) &\leq (\theta_{t+1} - \theta_t)^T \mathbf{g}_t + \frac{L}{2}\|\theta_{t+1} - \theta_t\|_2^2 \\ &= -\gamma_t \hat{\mathbf{g}_t}^T \mathbf{g}_t + \frac{\gamma_t^2 L}{2}\|\hat{\mathbf{g}_t}\|_2^2\end{aligned} \tag{9}$$

Take the expectation and use the assumption 3

$$\mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)] = -\gamma_t \|\mathbf{g}_t\|_2^2 + \frac{\gamma_t^2 L}{2}(\|\mathbf{g}_t\|_2^2 + \sigma^2) \tag{10}$$

Set the learning rate $\gamma_t = \frac{1}{L\sqrt{N}}$

$$\begin{aligned}\mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)] &= -\frac{1}{L\sqrt{N}}\|\mathbf{g}_t\|_2^2 + \frac{1}{2LN}(\|\mathbf{g}_t\|_2^2 + \sigma^2) \\ &\leq -\frac{1}{2L\sqrt{N}}\|\mathbf{g}_t\|_2^2 + \frac{\sigma^2}{2LN}\end{aligned} \tag{11}$$

⋆ Convergence of SGD

Proof (Cont'd).

Sum the inequality of N steps together and use assumption 1

$$\begin{aligned}\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) &\geq \mathcal{L}(\theta_0) - \mathbb{E}[\mathcal{L}(\theta_N)] \\ &= \mathbb{E}\left[\sum_{t=0}^{N-1} (\mathcal{L}(\theta_t) - \mathcal{L}(\theta_{t+1}))\right] \\ &\geq \frac{1}{2L}\mathbb{E}\left[\sum_{t=0}^{N-1} (\frac{\|\mathbf{g}_t\|_2^2}{\sqrt{N}} - \frac{\sigma^2}{N})\right]\end{aligned} \tag{12}$$

Rearrange the inequality, we have the following

$$\mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N-1} \|\mathbf{g}_t\|_2^2\right] \leq \frac{1}{\sqrt{N}}[2L(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) + \sigma^2)] \tag{13}$$

The right hand side vanishes as $N \to \infty$, so $\mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N-1} \|\mathbf{g}_t\|_2^2\right]$ vanishes also. This indicates the model converges to a critical point. □

Minibatch and momentum

	Minibatch	Momentum
Advantages	Fast, unbiased, no extra memory Help scape saddle points	Help escape poor local minima Help smooth out variations
Disadvantages	Might get stuck in poor local minimas	Might overshoot with high ρ and γ

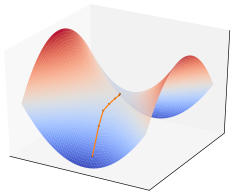
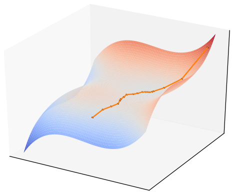



Figure: Stochasticity introduced by minibatch can help scape saddle points (Left). Momentum can help escape local minima (Right).

Escaping from saddle points

Recall (Classification of stationary points)

Let $f : \mathbb{R}^d \to \mathbb{R}$ be twice differentiable and let $\bar{\mathbf{x}}$ be a stationary point. Let $\{\lambda_i\}_{i=1}^n$ be the eigenvalues of the hessian $\nabla^2 f(\bar{\mathbf{x}})$, then

- $\lambda_i > 0$ for all $i \Rightarrow \bar{\mathbf{x}}$ is a local minimum
- $\lambda_i < 0$ for all $i \Rightarrow \bar{\mathbf{x}}$ is a local maximum
- $\lambda_i > 0$, $\lambda_j < 0$ for some i, j and $\lambda_i \neq 0$ for all $i \Rightarrow \bar{\mathbf{x}}$ is a saddle point
- Other case $\Rightarrow$ inconclusive

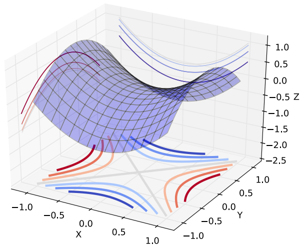


Figure: Minmax saddle ($\lambda_i \neq 0$ for all i)

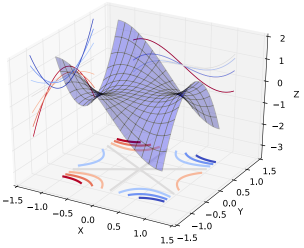


Figure: Monkey saddle ($\lambda_i = 0$ for some i)

The strict saddle property

Definition (Strict saddle)

A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $(\alpha, \gamma, \epsilon, \delta)$-strict saddle if for any point $\mathbf{x}$ at least one of the following is true

1. $\|\nabla f(\mathbf{x})\| \geq \epsilon$.
2. $\lambda_{\min}\left(\nabla^2 f(\mathbf{x})\right) \leq -\gamma$.
3. *there is a local minimum $\mathbf{x}^*$ such that $\|\mathbf{x} - \mathbf{x}^*\| \leq \delta$ and the function f restricted to a 2δ neighborhood of $\mathbf{x}^*$ is α strongly convex.*

(Informal)

For any point whose gradient is small, it is either close to a local minimum, or is a saddle point (or local maximum) with a significant negative eigenvalue.

Perturbed SGD algorithm

```
Perturbed Stochastic Gradient Descent [6]
Input: Stochastic Gradient Oracle $SG(\mathbf{x})$, initial point $\mathbf{x}_0$, number of iterations $T$, step size $\eta$
1. For t = 0 to T - 1:
       sample noise $\xi$ uniformly from unit sphere
       update $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta(SG(\mathbf{x}_t) + \xi)$
```

Minibatch SGD

If the noise from the stochastic gradient oracle already has nonnegligible variance in every direction then the additional noise ξ is not needed.

```
Stochastic Gradient Langevin Dynamics [17]
Input: Stochastic Gradient Oracle $SG(\mathbf{x})$, initial point $\mathbf{x}_0$, number of iterations $T$, step size $\eta$
1. For t = 0 to T - 1:
       sample noise $\xi$ standard Gaussian
       update $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta SG(\mathbf{x}_t) + \sqrt{2\eta}\xi$
```

Perturbed SGD escapes saddle points

Theorem (Convergence of PSGD [6])

Suppose that f has the following properties

- *f is an $(\alpha, \gamma, \epsilon, \delta)$-strict saddle,*
- *f is β-smooth.*
- *its Hessian is ρ-Lipschitz. i.e. $\left\|\nabla^2 f(x) - \nabla^2 f(y)\right\| \leq \rho \|x - y\|$.*

Then there exists a threshold η_{max} such that by choosing

- $\eta \leq \eta_{max} / \max\{1, \log(1/\zeta)\}$
- $T = O(\eta^{-2} \log(1/\zeta))$.

the algorithm ***Perturbed SGD*** *outputs with probability at least $1 - \zeta$ a point $\mathbf{x}_T$ that is $O(\sqrt{\eta \log(1/\eta\zeta)})$ close to some local minimum $\mathbf{x}^*$.*

Overparametrization

A few phenomena about neural networks:

- Deep neural networks can fit random labels. [19]
- In practice, simple first-order methods can find global minimizers.

Overparametrization can explain these mysteries!

Overparametrization

Number of parameters $\gg$ number of training data.

GD finds global minimizers of overparametrized networks

Theorem (Convergence Rate of Gradient Descent [4])

- *$f(\mathbf{x};\beta,W,b)$: 1-hidden-layer network with width m.*
- $m = \Omega(\frac{n^6}{\delta^3})$ *where n =number of samples.*
- *W_0 is initialized with a normal distribution, $\beta_0 \sim$ Unif$[-1,1]^m$.*
- *Stepsize $\eta = O(n^{-2})$.*

With probability at least $1-\delta$, for the empirical risk R_n we have

$$R_n(\beta_t, W_t, b_t) \leq (1-\eta)^t R_n(\beta_0, W_0, b_0) \tag{14}$$

Linear convergence of GD for overparametrized 1-hidden layer networks.

Optimization landscape of overparametrized neural networks

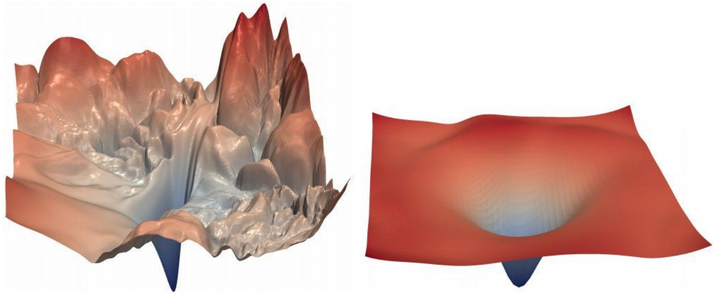


Figure: Loss landscape with few parameters (left) vs overparametrized regime (right).

The generalization error

- Goal: Minimize the misclassification error $\mathbb{P}(f(\mathbf{x}) \neq y)$
- ERM with cross-entropy loss $\Rightarrow$ few errors on the training set $\{\mathbf{x}_i, y_i\}_{i=1}^n$.

What about performance on unseen data?

Definition (Generalization error)

Suppose f is trained by ERM on a set $\mathbf{X}_{\textit{train}} = \{\mathbf{x}_i, y_i\}_{i=1}^n$.

$$\textit{Generalization error} := \textit{True error} - \textit{Training error}$$

$$:= \mathbb{P}(i_f(x) \neq y) - \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{i_f(\mathbf{x}_i) \neq y_i\}}$$

We use a sample $\mathbf{X}_{\textit{test}} = \{\hat{\mathbf{x}}_i, \hat{y}_i\}_{i=1}^{n'}$ to estimate

$$\mathbb{P}(i_f(x) \neq y) \simeq \frac{1}{n'}\sum_{i=1}^{n'} \mathbb{1}_{\{i_f(\hat{\mathbf{x}}_i) \neq \hat{y}_i\}} \qquad \text{(test error)}$$

The effect of overparametrization on generalization

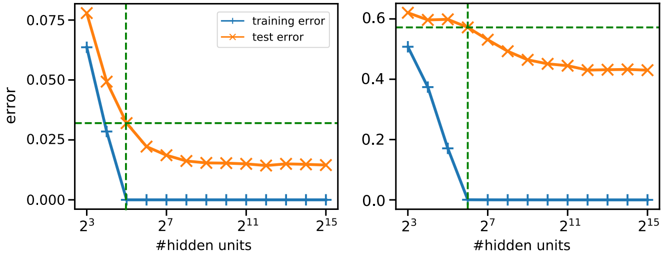


Figure: MNIST (left) and CIFAR10 (right)

Adaptivity may lead to overfit

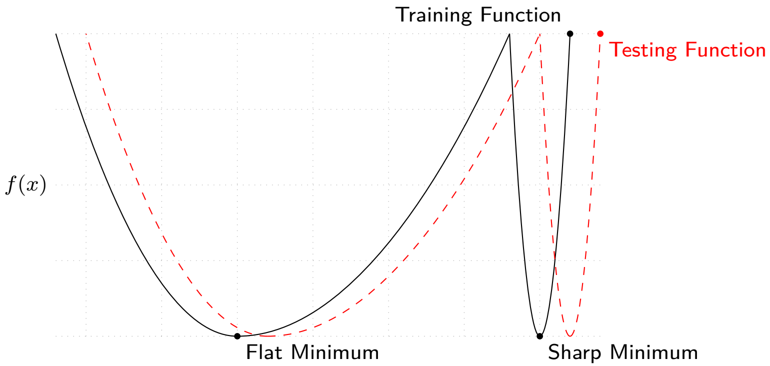
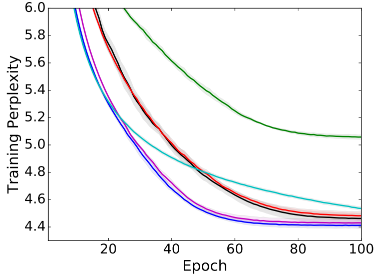


Figure: Sharp Minima vs Flat Minima [10]

- Intuition suggests flat minima has better generalization property than sharp minima
- Empirically, adaptive methods finds sharper minima than ones found by SGD
- The relationship between sharpness of minima and their generalization is open [3]

Adaptivity may lead to overfit

- Adaptive learning methods may converge fast but generalize worse



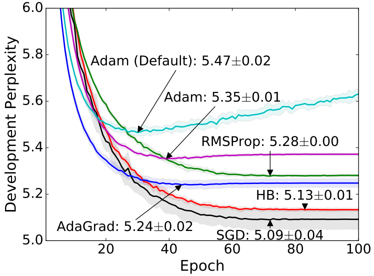


Figure: Performance of different optimizers in training and development set of a language modeling problem. The training and test perplexity are the exponential values of training and test losses.[18]

Neural Network Architectures

- Deeper and more complicated models correlates with better performance
- No universal optimizers other than slow and steady SGD
- A long way to go (makes it exciting)...

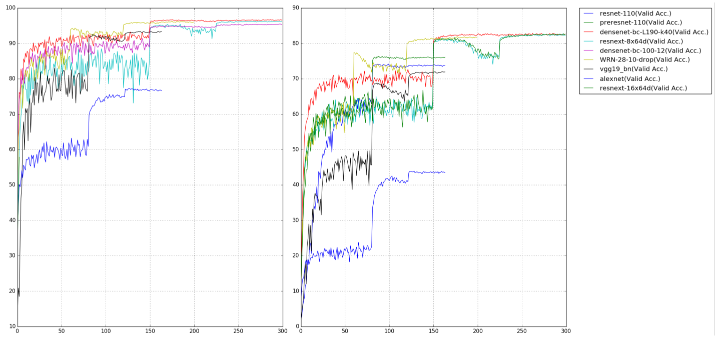


Figure: Performance of popular architectures on test set in CIFAR10 (left) and CIFAR100 (right). [2]

[1] Credit to: https://github.com/bearpaw/pytorch-classification

[2] Credit to: https://github.com/bearpaw/pytorch-classification

Beyond supervised learning: Generative Adversarial Networks

- Goal: Learn to generate samples from a distribution given a dataset.

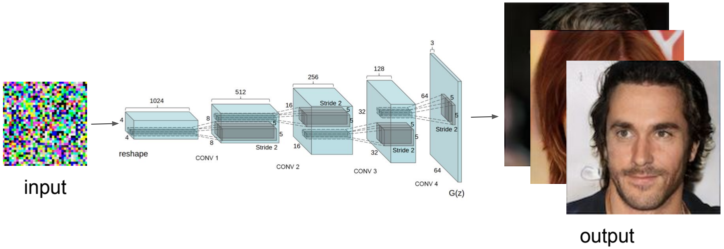


Figure: Schematic of a generative model [7, 9]

How to frame this as an optimization problem?

A notion of distance between distributions

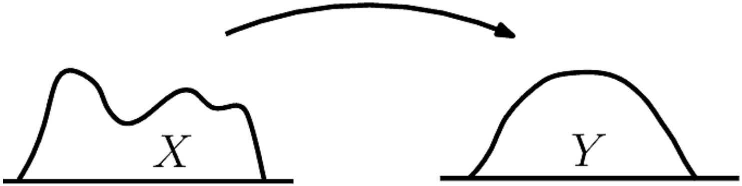


Figure: The Earth Mover's distance

Minimum cost transportation problem (Monge's problem)

Find a *transport map* $T : \mathbb{R}^d \to \mathbb{R}^d$ such that $T(X) \sim Y$, minimizing the cost

$$\text{cost}(T) := \boldsymbol{E}_X \|X - T(X)\| \tag{15}$$

Might not exist!

The Wasserstein distance

Definition

Let μ and ν be two probability measures on $\mathbb{R}^d$. Their set of couplings is defined as

$$\Gamma(\mu,\nu) := \{\pi \text{ probability measure on } \mathbb{R}^d \times \mathbb{R}^d \text{ with marginals } \mu,\nu\} \tag{16}$$

Definition (Primal form of the Wasserstein distance)

$$W(\mu,\nu) := \inf_{\pi\in\Gamma(\mu,\nu)} \boldsymbol{E}_{(x,y)\sim\pi} \|x-y\| \tag{17}$$

Theorem (Kantorovich-Rubinstein duality)

$$W(\mu,\nu) = \sup\{\boldsymbol{E}_{x\sim\mu} f(x) - \boldsymbol{E}_{y\sim\nu} f(y) : f \text{ is 1-Lipschitz}\} \tag{18}$$

The coupling π is the *primal variable*, the function f is the *dual variable*

Minimax formulation of GANs

Ingredients:

•: fixed *noise* distribution θ (e.g., normal)

•: target distribution ν (natural images)

•: class of functions $\mathcal{G}$ (generators)

•: class of functions $\mathcal{F}$ (dual variables)

GANs formulation

$$\min_{g\in\mathcal{G}} \max_{f\in\mathcal{F}} \boldsymbol{E}_{x\sim\theta} f(g(x)) - \boldsymbol{E}_{y\sim\nu} f(y) \tag{19}$$

$\mathcal{F}$ is the class of 1-Lipschitz functions $\Rightarrow$ minimization of Wasserstein distance

General diagram of GANs

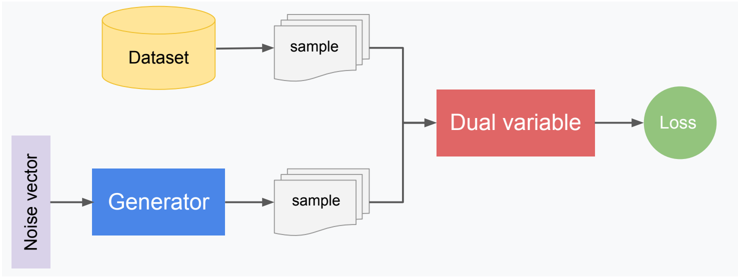
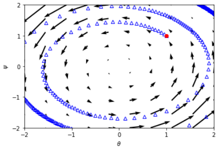


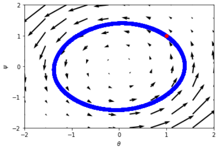
Figure: Generator/dual variable/dataset relation in GANs

Challenges in GANs training

- Tons of heuristics
- No guarantees of convergence (except some simple settings)
- Difficult to enforce 1-Lipschitz constraint
- Privacy concerns (memorization)



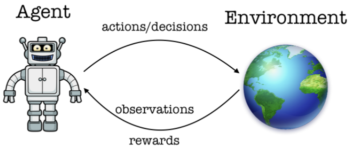
(a) SimGD



(b) AltGD

Figure: Mode collapse (left). Simultaneous vs alternating generator/discriminator updates (right).

*Reinforcement Learning Game



◦ Environment: Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \gamma, P_0, R)$

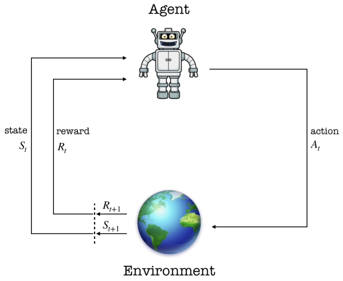
◦ Agent: Parameterized deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$, where $\theta \in \Theta$

Beyond supervised learning: Reinforcement Learning

```
At time step $t = 0$: $S_0 \sim P_0(\cdot)$
for $t = 1, 2, \ldots$ do:
    agent observes the environment's state $S_t \in \mathcal{S}$
    agent chooses an action $A_t = \mu_\theta(S_t) \in \mathcal{A}$
    agent receives a reward $R_{t+1} = R(S_t, A_t)$
    agent finds itself in a new state $S_{t+1} \sim T(\cdot \mid S_t, A_t)$
```

⋆Exploration vs. Exploitation in RL

◦ Challenge: Exploration vs. exploitation!



◦ Objective (non-concave): $\max_{\theta\in\Theta} J(\theta) := \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \,\middle|\, \mu_\theta, \mathcal{M}\right]$

▷ The environment only reveals the rewards after actions

▷ Exploitation: Maximize objective by choosing the appropriate action

▷ Exploration: Gather information on other actions

*Standard Reinforcement Learning

- Markov Decision Process (MDP): $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, \gamma, P_0, R)$
  - ▷ $\mathcal{S}$: state space
  - ▷ $\mathcal{A}$: action space
  - ▷ $T : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$: state transition dynamics
  - ▷ $\gamma \in (0, 1)$: discounting factor
  - ▷ $P_0 : \mathcal{S} \to [0, 1]$: initial state distribution
  - ▷ $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$: reward function
- Agent's (deterministic) policy: $\mu : \mathcal{S} \to \mathcal{A}$

Reinforcement Learning Game

```
for $t = 1, 2, \dots$ do:
    agent observes the environment's state $S_t \in \mathcal{S}$
    agent chooses an action $A_t = \mu(S_t) \in \mathcal{A}$
    agent receives a reward $R_{t+1} = R(S_t, A_t)$, and finds itself in a new state $S_{t+1}$
```

*Standard Reinforcement Learning

- Discounted return:

$$Z = \sum_{t=1}^{\infty} \gamma^{t-1} R_t$$

- State and state-action value functions:

$$\begin{aligned} V^\mu(s) &:= \mathbb{E}\left[Z \mid S_1 = s; \mu, \mathcal{M}\right] \\ Q^\mu(s,a) &:= \mathbb{E}\left[Z \mid S_1 = s, A_1 = a; \mu, \mathcal{M}\right] \end{aligned}$$

- Performance objective:

$$\max_\mu J(\mu) := \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[V^\mu(s)\right] = \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\left[Q^\mu(s,\mu(s))\right]$$

*Deterministic Policy Gradient

∘ Deterministic policy parametrization:

$$\{\mu_\theta : \theta \in \Theta\}$$

∘ The off-policy performance objective:

$$\max_{\theta \in \Theta} J(\theta) \;:=\; J(\mu_\theta) \;=\; \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[Q^{\mu_\theta}(s, \mu_\theta(s))\right]$$

∘ The off-policy gradient: [15]

$$\begin{aligned}\nabla_\theta J(\theta) \;&\approx\; \mathbb{E}_{s\sim\mathcal{D}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\right] \\ &\approx\; \frac{1}{N}\sum \nabla_a Q^{\phi}(s,a)\nabla_\theta \mu_\theta(s)\end{aligned}$$

▷ biased gradient estimate

▷ function approximation Q^ϕ for critic

*An optimization interpretation

- Objective (non-concave): $\max_{\theta\in\Theta} \; J(\theta) \;:=\; \mathbb{E}\left[\sum_{t=1}^{\infty}\gamma^{t-1}R_t \;\middle|\; \mu_\theta, \mathcal{M}\right]$

- Exploitation: Progress in the gradient direction

$$\theta_{t+1} \;\leftarrow\; \theta_t + \eta_t \widehat{\nabla_\theta J(\theta_t)}$$

- Exploration: Add stochasticity while collecting the episodes

  ▷ noise injection in the action space [15, 13]

$$a = \mu_\theta(s) + \mathcal{N}(0, \sigma^2 I)$$

  ▷ noise injection in the parameter space [14]

$$\tilde{\theta} = \theta + \mathcal{N}(0, \sigma^2 I)$$

*Deep learning optimizers: Ill-Conditioned Curvature

When optimizing a high-dimensional function, it is possible that the gradients in some dimensions are much larger than some others.

Using the same learning rate for all dimensions might cause either overshoot in dimensions of large gradients or slow convergence in ones of small gradients.

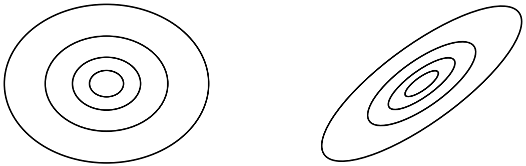


Figure: Examples of well-conditioned curvature (left) and ill-conditioned curvature (right).

*AdaGrad [5]

```
AdaGrad
Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient $\delta$
1. initialize $\theta_0$, $\mathbf{r} \leftarrow \mathbf{0}$
2. For t = 0, 1,... ,N-1:
       obtain the minibatch gradient $\hat{\mathbf{g}}_t$
       update $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$
       update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$
```

- Adaptive learning rate.
- 'Effective learning rate' decreases monotonically if γ_t is a constant (early stop).
- Suitable for the sparse gradient cases.

*RMSProp [16]

- To solve the early stop, average can be used in place of accumulation.
- A decaying coefficient τ can be used to maintain an online average estimate.
- Recent gradients have more weight when calculating the average.

AdaGrad

Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ

```
1. initialize $\theta_0$, $\mathbf{r} \leftarrow \mathbf{0}$
2. For t = 0, 1,... ,N-1:
       obtain the minibatch gradient $\hat{\mathbf{g}}_t$
       update $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$
       update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$
```

RMSProp

Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, decaying coefficient τ

```
1. initialize $\theta_0$, $\mathbf{r} \leftarrow \mathbf{0}$
2. For t = 0, 1,... ,N-1:
       obtain the minibatch gradient $\hat{\mathbf{g}}_t$
       update $\mathbf{r} \leftarrow \tau \mathbf{r} + (1 - \tau) \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$
       update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$
```

*Adam [11]

- Now there are two ways to accelerate training:
  - Momentum e.g. HB SGD
  - Adaptive learning rate e.g. AdaGrad, RMSProp
- How about mixing them together?

Adam

Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, first order decaying parameter β_1, second order decaying parameter β_2

```
1. initialize $\theta_0$, $\mathbf{m}_1 \leftarrow \mathbf{0}$, $\mathbf{m}_2 \leftarrow \mathbf{0}$
2. For t = 0, 1,... ,N-1:
       obtain the minibatch gradient $\hat{\mathbf{g}}_t$
       update $\mathbf{m}_1 \leftarrow \beta_1 \mathbf{m}_1 + (1-\beta_1)\hat{\mathbf{g}}_t$
       update $\mathbf{m}_2 \leftarrow \beta_2 \mathbf{m}_2 + (1-\beta_2)\hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$
       correct bias $\hat{\mathbf{m}}_1 \leftarrow \frac{\mathbf{m}_1}{1-\beta_1^{t+1}}$ $\hat{\mathbf{m}}_2 \leftarrow \frac{\mathbf{m}_2}{1-\beta_2^{t+1}}$
       update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \frac{\hat{\mathbf{m}}_1}{\delta + \sqrt{\hat{\mathbf{m}}_2}}$
```

*Summary: A Uniform Framework

$$\theta_{t+1} = \theta_t - \alpha_t H_t^{-1}\hat{\mathbf{g}}_t + \beta_t H_t^{-1} H_{t-1}(\theta_t - \theta_{t-1})$$
$$G_t = H_t \odot H_t, D_t = \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$$

	SGD	HB	AdaGrad	RMSProp	Adam
G_t	I	I	$G_{t-1} + D_t$	$\tau G_{t-1} + (1-\tau)D_t$	$\frac{\beta_2}{1-\beta_2^t}G_{t-1} + \frac{1-\beta_2}{1-\beta_2^t}D_t$
α_t	γ_t	γ_t	γ_t	γ_t	$\gamma_t \frac{1-\beta_1}{1-\beta_1^t}$
β_t	0	ρ	0	0	$\frac{\beta_1(1-\beta_1^{t-1})}{1-\beta_1^t}$

- Two ways to accelerate SGD: momentum and adaptive learning rate.
- The 'effective learning rate' of the algorithm is $\alpha_t H_t^{-1}$.
- The 'effective momentum' of the algorithm is $\beta_t H_t^{-1} H_{t-1}$.
- H_t is the preconditioner of the stochastic gradient $\hat{\mathbf{g}}_t$.

References I

[1] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar.
signsgd: compressed optimisation for non-convex problems.
arXiv preprint arXiv:1802.04434, 2018.

[2] George Cybenko.
Approximation by superpositions of a sigmoidal function.
Mathematics of control, signals and systems, 2(4):303–314, 1989.

[3] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio.
Sharp minima can generalize for deep nets.
arXiv preprint arXiv:1703.04933, 2017.

[4] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh.
Gradient descent provably optimizes over-parameterized neural networks.
arXiv preprint arXiv:1810.02054, 2018.

[5] John Duchi, Elad Hazan, and Yoram Singer.
Adaptive subgradient methods for online learning and stochastic optimization.
Journal of Machine Learning Research, 12(Jul):2121–2159, 2011.

[6] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan.
Escaping from saddle points—online stochastic gradient for tensor decomposition.
In *Conference on Learning Theory*, pages 797–842, 2015.

References II

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[8] Benjamin D Haeffele and René Vidal.
Global optimality in neural network training.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7331–7339, 2017.

[9] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen.
Progressive growing of GANs for improved quality, stability, and variation.
In *International Conference on Learning Representations*, 2018.

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang.
On large-batch training for deep learning: Generalization gap and sharp minima.
arXiv preprint arXiv:1609.04836, 2016.

[11] Diederik Kingma and Jimmy Ba.
Adam: A method for stochastic optimization.
arXiv preprint arXiv:1412.6980, 2014.

References III

[12] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken.
Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.
Neural Networks, 6(6):861 – 867, 1993.

[13] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra.
Continuous control with deep reinforcement learning.
arXiv preprint arXiv:1509.02971, 2015.

[14] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz.
Parameter space noise for exploration.
arXiv preprint arXiv:1706.01905, 2017.

[15] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller.
Deterministic policy gradient algorithms.
In *ICML*, 2014.

[16] Tijmen Tieleman and Geoffrey Hinton.
Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.
COURSERA: Neural networks for machine learning, 4(2):26–31, 2012.

References IV

[17] Max Welling and Yee W Teh.
Bayesian learning via stochastic gradient langevin dynamics.
In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[18] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht.
The marginal value of adaptive gradient methods in machine learning.
In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[19] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals.
Understanding deep learning requires rethinking generalization.
arXiv preprint arXiv:1611.03530, 2016.