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

- This work is released under a Creative Commons License with the following terms:
  - **Attribution**
    - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees must give the original authors credit.
  - **Non-Commercial**
    - The licensor permits others to copy, distribute, display, and perform the work. In return, licensees may not use the work for commercial purposes – unless they get the licensor’s permission.
  - **Share Alike**
    - The licensor permits others to distribute derivative works only under a license identical to the one that governs the licensor’s work.
- [Full Text of the License](#)
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


Remark about notation for this lecture

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

<table>
<thead>
<tr>
<th>Previous lectures</th>
<th>This lecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>data/sample</td>
<td>a</td>
</tr>
<tr>
<td>label</td>
<td>b</td>
</tr>
<tr>
<td>bias</td>
<td>μ</td>
</tr>
<tr>
<td>weight</td>
<td>x</td>
</tr>
</tbody>
</table>

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

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

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

Figure: Linearly separable versus non-linearly 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)

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

- **Parameters**: $\beta \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times d}$ (weights), $b \in \mathbb{R}^m$ (bias)
- **Activation function**: $\sigma : \mathbb{R} \rightarrow \mathbb{R}$
Why neural networks?: An approximation theoretic motivation

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

**Caveat**

**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 $C(I_d)$.

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

$$\sup_{x \in I_d} |g(x) - f(x)| \leq \epsilon$$
Why were NNs not popular before 2010?

- too big to optimize!
- did not have enough data
- could not find the optimum via algorithms
A natural generalization: Multilayer neural networks

Definition (2-hidden-layer network)

\[ f(x; \beta, W_1, b_1, W_2, b_2) = \beta^T \sigma(W_2 \sigma(W_1 x + b_1) + b_2) \]

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

\( k \)-layer networks are constructed analogously
Recall: Empirical risk minimization (ERM)

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

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

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

Some frequently used loss functions

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

$$f(x; w_1, w_2) = w_2 w_1 x$$

with $w_1, w_2 \in \mathbb{R}$. For a sample $(x_0, y_0) = (1, 1)$ the squared error is

$$(y_0 - f(x_0; w_1, w_2))^2 = (1 - w_2 w_1)^2$$

Show that it is neither convex nor concave.

- non-convexity even though activation function $\sigma$ is linear
What is the role of the activation function $\sigma$?

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

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

**Example**

- If $\sigma(x) = x$, the network computes an affine function.
- $W_2(W_1x + b_1) + b_2 = Wx + b$.
- Cannot approximate non-affine functions.

ReLU: Rectified Linear Unit

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

Figure: some activation functions for which universal approximation holds
Supervised learning: Multi-class classification

- So far: Only single output networks

**Goal**

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

\[
\min_{f \in \mathcal{F}} P(f(x) \neq y)
\]
Using networks for multi-class classification

**Definition (Multi-output network)**

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

$$f(x; W_1, b_1, B) := B\sigma(W_1x + b_1)$$

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 $c = 1$

**Definition (Score-based classifier)**

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

$$i_f(x) = \arg \max_{i \in \{1, \ldots, c\}} f_i(x)$$

**Example:**

$$f(x_0) = \begin{bmatrix} 0.1 \\ -0.8 \\ 1.4 \\ 1.1 \end{bmatrix} \implies i_f(x_0) = 3$$
Cross-entropy loss for multiclass classification

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

**Definition (Cross-entropy loss)**

Let \( x \in \mathbb{R}^d \) be a sample with label \( y \in \{1, \ldots, c\} \)

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

\[
f(x_0) = \begin{bmatrix} 0.1 \\ -0.8 \\ 1.4 \\ 1.1 \end{bmatrix} \quad \mathcal{L}(f(x_0), 2) = 2.95 \quad \mathcal{L}(f(x_0), 3) = 0.75
\]

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

\[ \nabla_\theta R_n(\theta) := \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(f(x_i; \theta), y_i) := \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}_i(\theta) \quad (2) \]

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

\[ \frac{\partial \mathcal{L}_i}{\partial \beta} = -2(y_i - \beta^T \sigma(Wx_i))\sigma(Wx_i) \quad (3) \]

\[ \frac{\partial \mathcal{L}_i}{\partial W} = -2(y_i - \beta^T \sigma(Wx_i))\beta \odot \sigma'(Wx_i)x_i^T \quad (4) \]

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive derivative</td>
<td>$\mathcal{O}(k^2 H^2)$</td>
</tr>
<tr>
<td>Backpropagation</td>
<td>$\mathcal{O}(k H^2)$</td>
</tr>
</tbody>
</table>

Where $H$ is number of neurons per layer and $k$ is the number of layers.
*Forward pass*

<table>
<thead>
<tr>
<th>Forward pass scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> ( x^{(0)} = x, W^{(l)} ) and ( b^{(l)} ) for ( l = 1, \ldots, k ).</td>
</tr>
</tbody>
</table>

1. For \( l = 1, \ldots, k \)
   - Compute \( u^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)} \)
   - Compute \( x^{(l)} = \sigma(u^{(l)}) \)

![Diagram](image)

**Figure:** Computation of \( u^{(l)} \) and \( x^{(l)} \) starting from \( x^{(l-1)} \)
Backward pass

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

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

1. 
   \[
   u^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)} \Rightarrow \begin{cases} 
   \frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial u^{(l)}}(x^{(l-1)})^T \\
   \frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial u^{(l)}} 
   \end{cases}
   \] (chain rule)

2. 
   \[
   x^{(l)} = \sigma(u^{(l)}) \Rightarrow \frac{\partial L}{\partial u^{(l)}} = \frac{\partial L}{\partial x^{(l)}} \odot \sigma'(u^{(l)})
   \] (chain rule)

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

3. Finally we have
   \[
   u^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)} \Rightarrow \frac{\partial L}{\partial x^{(l-1)}} = (W^{(l)})^T \frac{\partial L}{\partial u^{(l)}}
   \] (chain rule)
Backward pass

**Backward pass scheme**

**Input:** Gradient of the loss w.r.t. the last layer values \( \frac{\partial L}{\partial x^{(k)}} \)

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

   - Compute \( \frac{\partial L}{\partial x^{(l-1)}} = (W^{(l)})^T \frac{\partial L}{\partial u^{(l)}} \)

---

**Figure:** Computation of \( \frac{\partial L}{\partial b^{(l)}} \), \( \frac{\partial L}{\partial W^{(l)}} \) and \( \frac{\partial L}{\partial x^{(l-1)}} \) starting from \( \frac{\partial L}{\partial x^{(l)}} \)
*Complexity of Backpropagation*

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

**Forward pass scheme**

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

Forward pass is $O(kH^2)$

**Backward pass scheme**

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

Backward pass is $O(kH^2)$
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)

<table>
<thead>
<tr>
<th>Vanilla Minibatch SGD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> learning rate ( { \gamma }_{t=0}^{N-1} )</td>
</tr>
<tr>
<td>1. initialize ( \theta_0 )</td>
</tr>
</tbody>
</table>
| 2. For \( t = 0, 1, \ldots, N-1 \):
  | obtain the minibatch gradient \( \hat{g}_t \)
  | update \( \theta_{t+1} \leftarrow \theta_t - \gamma_t \hat{g}_t \) |
Convergence of SGD in non-convex problems

Assumptions

1. Function $L$ is lower bounded: $\exists \theta^* \text{ s.t. } \forall \theta \in \Theta, L(\theta) \geq L(\theta^*)$

2. Function $L$ has Lipschitz continuous gradient:

$$
\|\nabla L(\theta_1) - \nabla L(\theta_2)\|_2 \leq L\|\theta_1 - \theta_2\|_2 \quad (5)
$$

3. The unbiased stochastic gradient $\hat{g}_\theta$ has bounded variance.

$$
\mathbb{E}(\hat{g}) = g \quad (6)
$$

$$
\mathbb{E}(\|\hat{g} - g\|^2) \leq \sigma^2 \quad (7)
$$

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

$$
\mathbb{E}\left[\frac{1}{N} \sum_{t=0}^{N-1} \|\hat{g}_t\|^2\right] \sim O\left(\frac{1}{\sqrt{N}}\right) \quad (8)
$$

- Convergence is captured by the gradient norm
Proof

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

$$
L(\theta_{t+1}) - L(\theta_t) \leq (\theta_{t+1} - \theta_t)^T g_t + \frac{L}{2} \|\theta_{t+1} - \theta_t\|^2_2
$$

$$
= -\gamma_t \hat{g}_t^T g_t + \frac{\gamma_t^2 L}{2} \|\hat{g}_t\|^2_2
$$

(9)

Take the expectation and use the assumption 3

$$
\mathbb{E}[L(\theta_{t+1}) - L(\theta_t)] = -\gamma_t \|g_t\|^2_2 + \frac{\gamma_t^2 L}{2} (\|g_t\|^2_2 + \sigma^2)
$$

(10)

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

$$
\mathbb{E}[L(\theta_{t+1}) - L(\theta_t)] = -\frac{1}{L \sqrt{N}} \|g_t\|^2_2 + \frac{1}{2LN} (\|g_t\|^2_2 + \sigma^2)
$$

(11)

$$
\leq -\frac{1}{2L \sqrt{N}} \|g_t\|^2_2 + \frac{\sigma^2}{2LN}
$$
* Convergence of SGD

Proof (Cont’d).

Sum the inequality of $N$ steps together and use assumption 1

$$\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) \geq \mathcal{L}(\theta_0) - \mathbb{E}[\mathcal{L}(\theta_N)]$$

$$= \mathbb{E} \left[ \sum_{t=0}^{N-1} (\mathcal{L}(\theta_t) - \mathcal{L}(\theta_{t+1})) \right]$$

$$\geq \frac{1}{2L} \mathbb{E} \left[ \sum_{t=0}^{N-1} \left( \frac{\|g_t\|_2^2}{\sqrt{N}} - \frac{\sigma^2}{N} \right) \right] \quad (12)$$

Rearrange the inequality, we have the following

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

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

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Minibatch</th>
<th>Momentum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fast, unbiased, no extra memory</td>
<td>Help escape poor local minima</td>
</tr>
<tr>
<td></td>
<td>Help escape saddle points</td>
<td>Help smooth out variations</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Might get stuck in poor local minima</td>
<td>Might overshoot with high $\rho$ and $\gamma$</td>
</tr>
</tbody>
</table>

**Figure:** Stochasticity introduced by minibatch can help escape 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 \( \bar{x} \) be a stationary point. Let \( \{\lambda_i\}_{i=1}^n \) be the eigenvalues of the hessian \( \nabla^2 f(\bar{x}) \), then

- \( \lambda_i > 0 \) for all \( i \) \( \Rightarrow \) \( \bar{x} \) is a local minimum
- \( \lambda_i < 0 \) for all \( i \) \( \Rightarrow \) \( \bar{x} \) is a local maximum
- \( \lambda_i > 0, \lambda_j < 0 \) for some \( i, j \) and \( \lambda_i \neq 0 \) for all \( i \) \( \Rightarrow \) \( \bar{x} \) 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 $x$ at least one of the following is true

1. $\|\nabla f(x)\| \geq \epsilon$.
2. $\lambda_{\min}(\nabla^2 f(x)) \leq -\gamma$.
3. there is a local minimum $x^*$ such that $\|x - x^*\| \leq \delta$ and the function $f$ restricted to a $2\delta$ neighborhood of $x^*$ is $\alpha$ 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 $SG(x)$, initial point $x_0$, number of iterations $T$, step size $\eta$
| --- |
| 1. For $t = 0$ to $T - 1$:  
  sample noise $\xi$ uniformly from unit sphere  
  update $x_{t+1} \leftarrow x_t - \eta(SG(x_t) + \xi)$

---

**Minibatch SGD**

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

**Stochastic Gradient Langevin Dynamics [17]**

| **Input:** Stochastic Gradient Oracle $SG(x)$, initial point $x_0$, number of iterations $T$, step size $\eta$
| --- |
| 1. For $t = 0$ to $T - 1$:  
  sample noise $\xi$ standard Gaussian  
  update $x_{t+1} \leftarrow x_t - \eta SG(x_t) + \sqrt{2\eta}\xi$ |
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 $\beta$-smooth.
- Its Hessian is $\rho$-Lipschitz. i.e. \[
\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq \rho \|x - y\|.
\]

Then there exists a threshold $\eta_{\text{max}}$ such that by choosing

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

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

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])**

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

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

\[
R_n(\beta_t, W_t, b_t) \leq (1 - \eta)^t R_n(\beta_0, W_0, b_0)
\]  

(14)

Linear convergence of GD for overparametrized 1-hidden layer networks.
Figure: Loss landscape with few parameters (left) vs overparametrized regime (right).
The generalization error

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

What about performance on unseen data?

**Definition (Generalization error)**

Suppose $f$ is trained by ERM on a set $X_{\text{train}} = \{x_i, y_i\}_{i=1}^n$.

Generalization error := True error – Training error

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

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

$$
\mathbb{P}(i_f(x) \neq y) \simeq \frac{1}{n'} \sum_{i=1}^{n'} \mathbb{1}\{i_f(\hat{x}_i) \neq \hat{y}_i\} \quad \text{(test error)}
$$
The effect of overparametrization on generalization

Figure: MNIST (left) and CIFAR10 (right)
Adaptivity may lead to overfit

- 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]

Figure: Sharp Minima vs Flat Minima [10]
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).

---

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

$$\text{cost}(T) := E_X \|X - T(X)\|$$

(15)

Might not exist!
The Wasserstein distance

**Definition**
Let $\mu$ and $\nu$ be two probability measures on $\mathbb{R}^d$. Their set of couplings is defined as

$$\Gamma(\mu, \nu) := \{ \pi \text{ probability measure on } \mathbb{R}^d \times \mathbb{R}^d \text{ with marginals } \mu, \nu \}$$  \hspace{1cm} (16)

**Definition (Primal form of the Wasserstein distance)**

$$W(\mu, \nu) := \inf_{\pi \in \Gamma(\mu, \nu)} \mathbb{E}_{(x,y) \sim \pi} \| x - y \|$$  \hspace{1cm} (17)

**Theorem (Kantorovich-Rubinsteiin duality)**

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

The coupling $\pi$ is the **primal variable**, the function $f$ is the **dual variable**
Minimax formulation of GANs

Ingredients:

- fixed noise distribution $\theta$ (e.g., normal)
- target distribution $\nu$ (natural images)
- class of functions $\mathcal{G}$ (generators)
- class of functions $\mathcal{F}$ (dual variables)

GANs formulation

$$\min_{g \in \mathcal{G}} \max_{f \in \mathcal{F}} E_{x \sim \theta} f(g(x)) - E_{y \sim \nu} f(y)$$  \hspace{1cm} (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)

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

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

- Environment: Markov Decision Process (MDP) \( \mathcal{M} = (S, A, T, \gamma, P_0, R) \)
- Agent: Parameterized deterministic policy \( \mu_\theta : S \to A \), 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 \( S_t \in S \)
- agent chooses an action \( A_t = \mu_\theta(S_t) \in A \)
- 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): \[ \max_{\theta \in \Theta} J(\theta) := \mathbb{E} \left[ \sum_{t=1}^{\infty} \gamma^{t-1} R_t \mid \mu_\theta, \mathcal{M} \right] \]

  - 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
  - $T: \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$: state transition dynamics
  - $\gamma \in (0, 1)$: discounting factor
  - $P_0: \mathcal{S} \rightarrow [0, 1]$: initial state distribution
  - $R: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$: reward function

- **Agent’s (deterministic) policy:** $\mu: \mathcal{S} \rightarrow \mathcal{A}$

---

**Reinforcement Learning Game**

```latex
\begin{align*}
& \text{for } t = 1, 2, \ldots \text{ do:} \\
& \quad \text{agent observes the environment’s state } S_t \in \mathcal{S} \\
& \quad \text{agent chooses an action } A_t = \mu(S_t) \in \mathcal{A} \\
& \quad \text{agent receives a reward } R_{t+1} = R(S_t, A_t), \text{ and finds itself in a new state } S_{t+1}
\end{align*}
```
Standard Reinforcement Learning

- Discounted return:
  \[ Z = \sum_{t=1}^{\infty} \gamma^{t-1} R_t \]

- State and state-action value functions:
  \[ V^\mu(s) := \mathbb{E}[Z | S_1 = s ; \mu, \mathcal{M}] \]
  \[ Q^\mu(s, a) := \mathbb{E}[Z | S_1 = s, A_1 = a ; \mu, \mathcal{M}] \]

- Performance objective:
  \[ \max_{\mu} J(\mu) := \mathbb{E}_{s \sim \mathcal{D}}[V^\mu(s)] = \mathbb{E}_{s \sim \mathcal{D}}[Q^\mu(s, \mu(s))] \]
*Deterministic Policy Gradient*

- Deterministic policy parametrization:
  \[ \{ \mu_\theta : \theta \in \Theta \} \]

- The off-policy performance objective:
  \[
  \max_{\theta \in \Theta} J(\theta) := J(\mu_\theta) = \mathbb{E}_{s \sim D} [Q^{\mu_\theta}(s, \mu_\theta(s))]
  \]

- The off-policy gradient:
  \[
  \nabla_\theta J(\theta) \approx \mathbb{E}_{s \sim D} \left[ \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s, a) \big| a = \mu_\theta(s) \right] \\
  \approx \frac{1}{N} \sum \nabla_a Q^{\phi}(s, a) \nabla_\theta \mu_\theta(s)
  \]
  - biased gradient estimate
  - function approximation \( Q^{\phi} \) for critic
An optimization interpretation

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

- Exploitation: Progress in the gradient direction

\[
\theta_{t+1} \leftarrow \theta_t + \eta_t \nabla_{\theta} J(\theta_t)
\]

- 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]**

<table>
<thead>
<tr>
<th>AdaGrad</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> global learning rate ${\gamma}_{t=0}^{N-1}$, damping coefficient $\delta$</td>
</tr>
<tr>
<td><strong>1.</strong> initialize $\theta_0$, $r \leftarrow 0$</td>
</tr>
<tr>
<td><strong>2.</strong> For $t = 0, 1, \ldots, N-1$:</td>
</tr>
<tr>
<td>- obtain the minibatch gradient $\hat{g}_t$</td>
</tr>
<tr>
<td>- update $r \leftarrow r + \hat{g}_t \odot \hat{g}_t$</td>
</tr>
<tr>
<td>- update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{r}} \hat{g}_t$</td>
</tr>
</tbody>
</table>

- Adaptive learning rate.
- ‘Effective learning rate’ decreases monotonically if $\gamma_t$ 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 $\tau$ can be used to maintain an online average estimate.
- Recent gradients have more weight when calculating the average.

<table>
<thead>
<tr>
<th>AdaGrad</th>
<th>RMSProp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> global learning rate ${\gamma}_{t=0}^{N-1}$, damping coefficient $\delta$</td>
<td><strong>Input:</strong> global learning rate ${\gamma}_{t=0}^{N-1}$, damping coefficient $\delta$, decaying coefficient $\tau$</td>
</tr>
<tr>
<td>1. initialize $\theta_0$, $r \leftarrow 0$</td>
<td>1. initialize $\theta_0$, $r \leftarrow 0$</td>
</tr>
<tr>
<td>2. For $t = 0, 1, \ldots, N-1$:</td>
<td>2. For $t = 0, 1, \ldots, N-1$:</td>
</tr>
<tr>
<td>obtain the minibatch gradient $\hat{g}_t$</td>
<td>obtain the minibatch gradient $\hat{g}_t$</td>
</tr>
<tr>
<td>update $r \leftarrow r + \hat{g}_t \odot \hat{g}_t$</td>
<td>update $r \leftarrow \tau r + (1 - \tau) \hat{g}_t \odot \hat{g}_t$</td>
</tr>
<tr>
<td>update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\gamma_t + \sqrt{r}} \hat{g}_t$</td>
<td>update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{r}} \hat{g}_t$</td>
</tr>
</tbody>
</table>
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 [11]**

<table>
<thead>
<tr>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> global learning rate ${\gamma}_{t=0}^{N-1}$, damping coefficient $\delta$, first order decaying parameter $\beta_1$, second order decaying parameter $\beta_2$</td>
</tr>
<tr>
<td>1. initialize $\theta_0, \mathbf{m}_1 \leftarrow 0, \mathbf{m}_2 \leftarrow 0$</td>
</tr>
</tbody>
</table>
| 2. For $t = 0, 1, \ldots, N-1$:
  - 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**

\[
\theta_{t+1} = \theta_t - \alpha_t H_t^{-1} \hat{g}_t + \beta_t H_t^{-1} H_{t-1} (\theta_t - \theta_{t-1}) \\
G_t = H_t \circ H_t, D_t = \hat{g}_t \circ \hat{g}_t
\]

<table>
<thead>
<tr>
<th></th>
<th>SGD</th>
<th>HB</th>
<th>AdaGrad</th>
<th>RMSProp</th>
<th>Adam</th>
</tr>
</thead>
<tbody>
<tr>
<td>(G_t)</td>
<td>I</td>
<td>I</td>
<td>(G_{t-1} + D_t)</td>
<td>(\tau G_{t-1} + (1 - \tau) D_t)</td>
<td>(\frac{\beta_2}{1 - \beta_2} G_{t-1} + \frac{1 - \beta_2}{1 - \beta_2} D_t)</td>
</tr>
<tr>
<td>(\alpha_t)</td>
<td>(\gamma_t)</td>
<td>(\gamma_t)</td>
<td>(\gamma_t)</td>
<td>(\gamma_t)</td>
<td>(\gamma_t \frac{1 - \beta_1}{1 - \beta_1})</td>
</tr>
<tr>
<td>(\beta_t)</td>
<td>0</td>
<td>(\rho)</td>
<td>0</td>
<td>0</td>
<td>(\beta_1 (1 - \beta_1^{t-1}) \frac{1}{1 - \beta_1})</td>
</tr>
</tbody>
</table>

- 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 \(\hat{g}_t\).
References I

Signsgd: compressed optimisation for non-convex problems.

Approximation by superpositions of a sigmoidal function.

Sharp minima can generalize for deep nets.

Gradient descent provably optimizes over-parameterized neural networks.

Adaptive subgradient methods for online learning and stochastic optimization.

[6] Rong Ge, Furong Huang, Chi Jin, and Yang Yuan.
Escaping from saddle points—online stochastic gradient for tensor decomposition.
Generative adversarial nets.

Global optimality in neural network training.

Progressive growing of GANs for improved quality, stability, and variation.

On large-batch training for deep learning: Generalization gap and sharp minima.

Adam: A method for stochastic optimization.
References III


Bayesian learning via stochastic gradient langevin dynamics.

[18] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht.
The marginal value of adaptive gradient methods in machine learning.

Understanding deep learning requires rethinking generalization.