Mathematics of Data: From Theory to Computation

Prof. Volkan Cevher
volkan.cevher@epfl.ch

Lecture 8: Non-convex optimization

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

EE-556 (Fall 2018)

lions@epfl

MARIE CURIE ACTIONS
FNSNF
FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION
erc
EPFL

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
  - Adaptive learning algorithms (Adam, Adagrad)
- 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

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.*

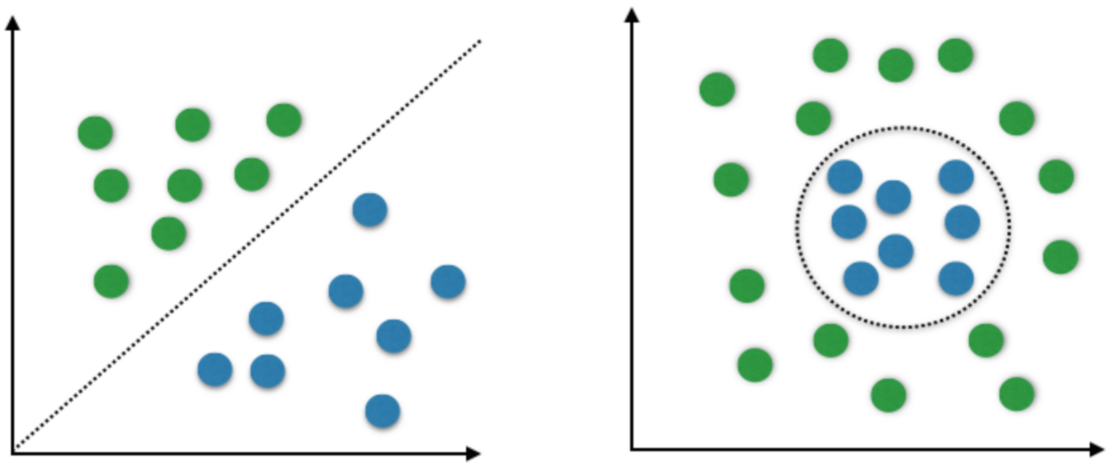


Figure: Linearly separable versus nonlinearly separable dataset

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

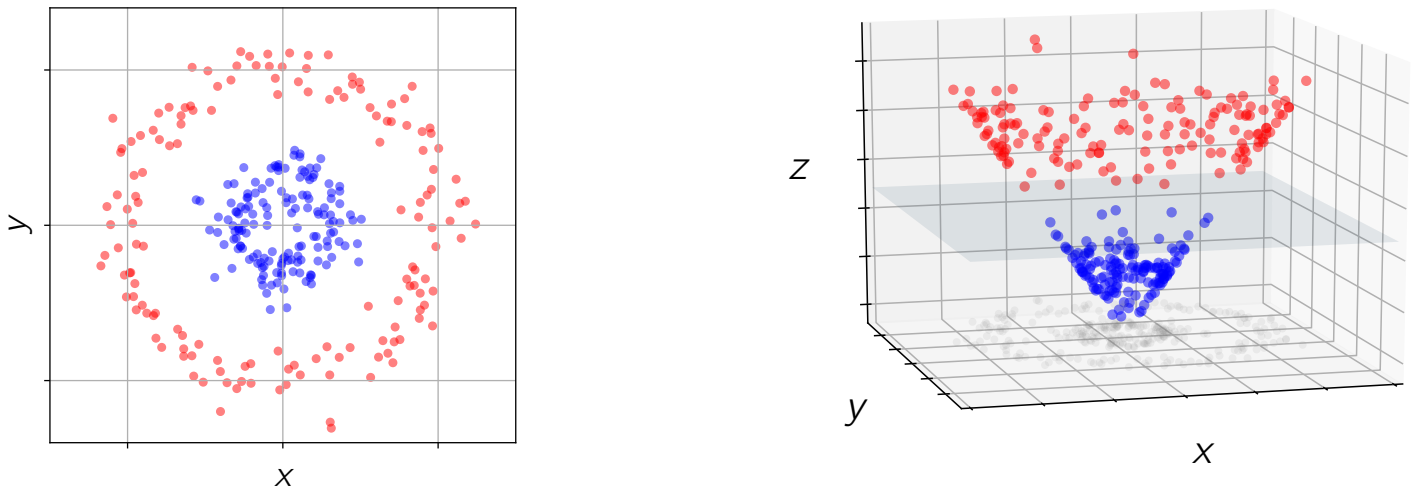
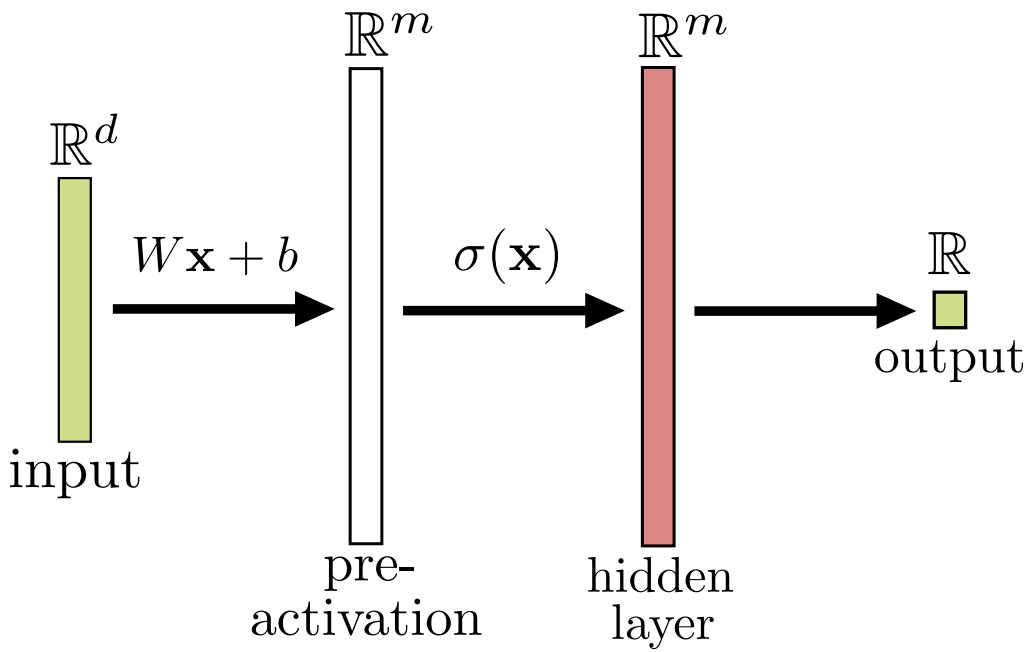


Figure: Non-linearly separable data (left). Linearly separable in $\mathbb{R}^3$ via $z = \sqrt{x^2 + y^2}$ (right).

An important alternative for non-linearly separable data



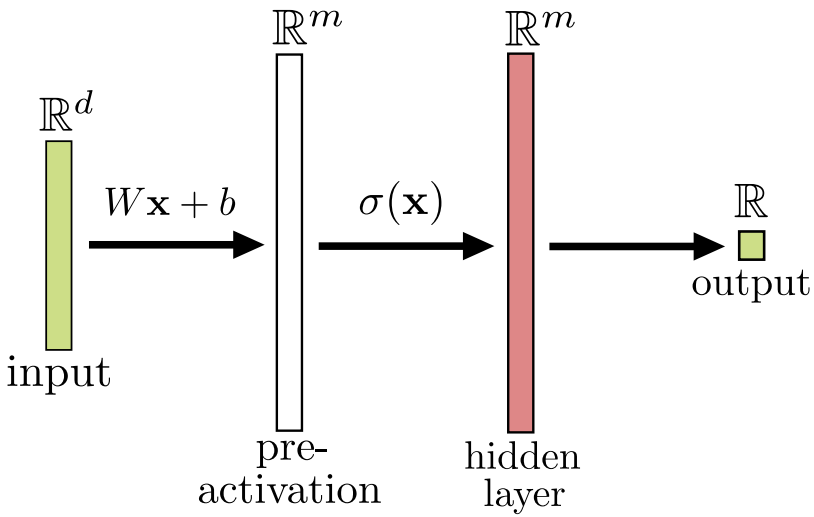
Definition (Non-linear functions: A key example)

A 1-hidden-layer network with m nodes is a function $f : \mathbb{R}^d \to \mathbb{R}$ of the form

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

where β and $b \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times n}$. $\sigma : \mathbb{R} \to \mathbb{R}$ is called the activation function. We use $\sigma(\mathbf{x}) = (\sigma(x_1), \ldots, \sigma(x_n))$. W and b are called the weight and the bias, respectively.

Why neural networks?: An approximation theoretic motivation



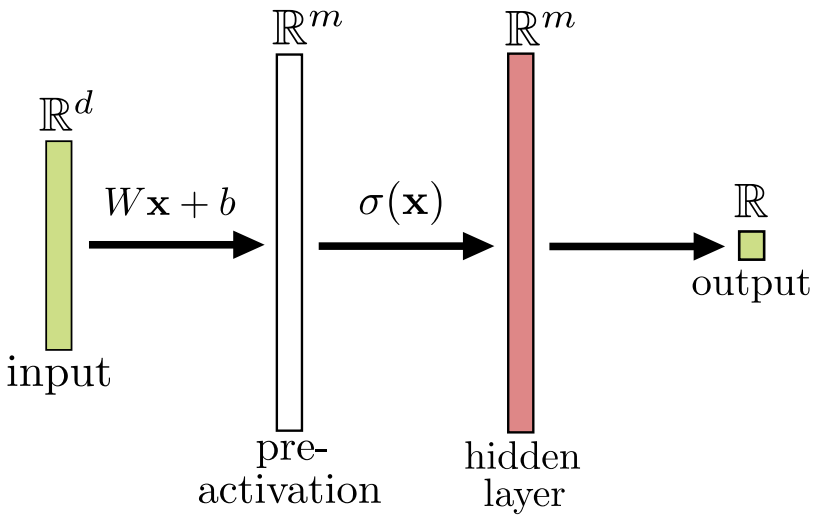
Theorem (Universal approximation (Cybenko, 1989) [2])

Let $\sigma(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_n denote the m-dimensional unit hypercube $[0,1]^n$. The space of continuous functions on I_n is denoted by $\mathcal{C}(I_n)$.

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

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

Why neural networks?: An approximation theoretic motivation



Caveat

The number of nodes m function f can be exponentially large!

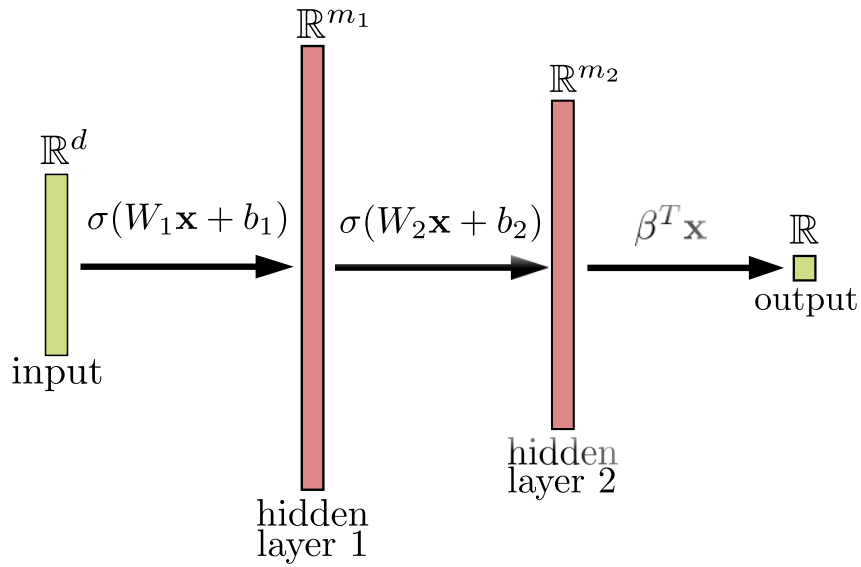
Theorem (Universal approximation (Cybenko, 1989) [2])

Let $\sigma(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_n denote the m-dimensional unit hypercube $[0,1]^n$. The space of continuous functions on I_n is denoted by $\mathcal{C}(I_n)$.

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

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

A natural generalization: Multilayer neural networks



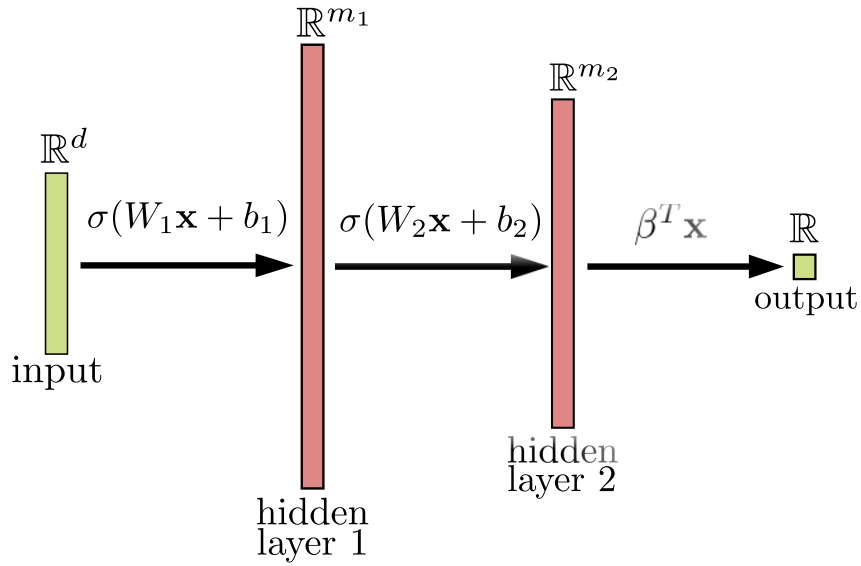
Definition (Multilayer neural networks)

A 2-hidden-layer network is a parametric function $f : \mathbb{R}^d \to \mathbb{R}$ *of the form*

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

Where W_1, W_2, b_1, b_2 *weight matrices/bias vectors of appropriate size and* $\sigma : \mathbb{R} \to \mathbb{R}$ *is the activation function. We define a k-hidden-layer network in an analogous way.*

but, why more layers?



Theorem (Benefits of depth [11])

Let $k \geq 1$ and $n \geq 1$. There exists a function f computed by a network with $O(k^3)$ layers and $O(k^3)$ nodes such that

$$\inf_{g \in \mathcal{G}} \int_{[0,1]^n} |f(\mathbf{x}) - g(\mathbf{x})| d\mathbf{x} \geq \frac{1}{64}$$

where the class $\mathcal{G}$ is composed of all networks with $\leq k$ layers and $\leq 2^k$ nodes.

What is the role of the activation function σ?

Theorem (Universal approximation (Leshno, 1993) [9])

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be continuous and locally bounded. Then, the set of 1-hidden-layer neural networks with activation σ is a universal approximator of $C(I_n)$ (continuous functions on $[0,1]^n$) if and only if σ is not a polynomial.

What is the role of the activation function σ?

Theorem (Universal approximation (Leshno, 1993) [9])

Let $\sigma : \mathbb{R} \to \mathbb{R}$ be continuous and locally bounded. Then, the set of 1-hidden-layer neural networks with activation σ is a universal approximator of $C(I_n)$ (continuous functions on $[0,1]^n$) if and only if σ is 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 with arbitrary precision.

What is the role of the activation function σ?

Theorem (Universal approximation (Leshno, 1993) [9])

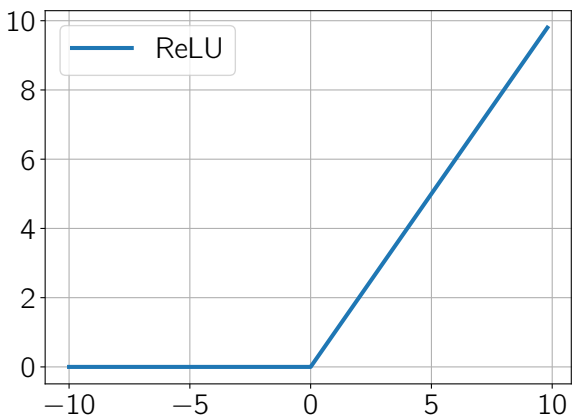
Let $\sigma : \mathbb{R} \to \mathbb{R}$ be continuous and locally bounded. Then, the set of 1-hidden-layer neural networks with activation σ is a universal approximator of $C(I_n)$ (continuous functions on $[0,1]^n$) if and only if σ is not a polynomial.

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 with arbitrary precision.

ReLU: Rectified Linear Unit

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



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

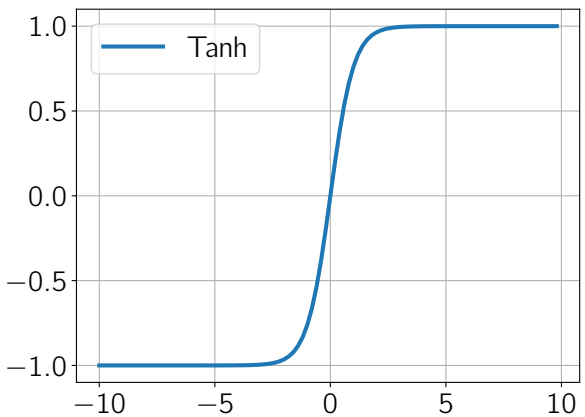


Figure: Examples of activation functions for which universal approximation holds

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.

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)

Obvious issue: Non-convexity

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 $(b = 0)$

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

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.

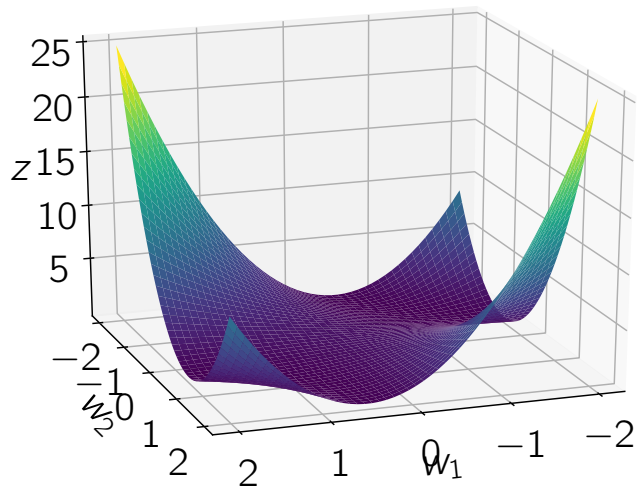


Figure: Loss surface $(1 - w_2 w_1)^2$

Obvious issue: Non-convexity

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 $(b = 0)$

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

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.

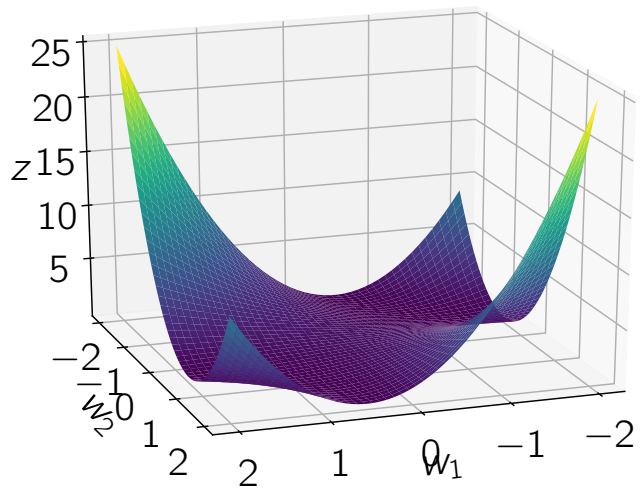


Figure: Loss surface $(1 - w_2 w_1)^2$

- The non-convexity even though activation function σ is linear

Towards multiple output networks: Multi-class classification

- So far: Only single output networks

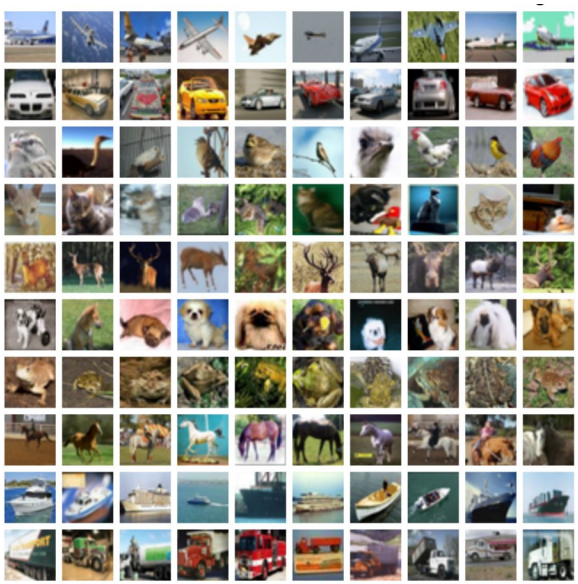


Figure: CIFAR10 dataset: 60000 32x32 color images (3 channels) from 10 classes

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

Goal

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

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

Using multilayer networks for multi-class classification

Definition (Multi-output network)

A 1-layer multi-output network $f : \mathbb{R}^d \to \mathbb{R}^c$ *is defined as*

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

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

- Single output networks correspond to $B = \beta^T$

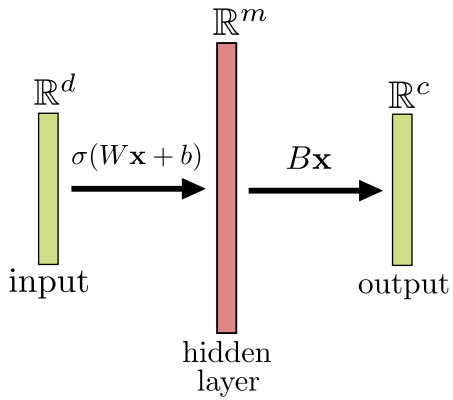


Figure: Multi-output 1-layer network

Using multilayer networks for multi-class classification

Definition (Multi-output network)

A 1-layer multi-output network $f : \mathbb{R}^d \to \mathbb{R}^c$ *is defined as*

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

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

- Single output networks correspond to $B = \beta^T$

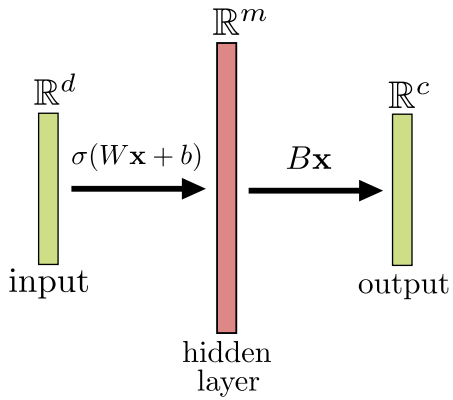


Figure: Multi-output 1-layer network

Definition (Score-based classifier)

Let $f : \mathbb{R}^d \to \mathbb{R}^c$ *be a function. Define* $i_f : \mathbb{R}^d \to \{1, \dots, c\}$ *as*

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

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 score-based classification

- Let $(\mathbf{x}, y)$ be an image-label sample and $f : \mathbb{R}^d \to \mathbb{R}^c$.
- Goal: define $\mathcal{L}$ s.t. $\mathcal{L}$ is small (large) if $f(\mathbf{x})_y$ is large (small).

Cross-entropy loss for score-based classification

- Let $(\mathbf{x}, y)$ be an image-label sample and $f : \mathbb{R}^d \to \mathbb{R}^c$.
- Goal: define $\mathcal{L}$ s.t. $\mathcal{L}$ is small (large) if $f(\mathbf{x})_y$ is large (small).

Definition (Soft-max)

Let $\mathbf{z} \in \mathbb{R}^c$. The vector

$$\mathbf{q}(\mathbf{z})_i := \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{c} \exp(\mathbf{z}_j)}$$

is a probability vector i.e., $\|\mathbf{q}(\mathbf{z})\|_1 = 1$ and $\mathbf{q}(\mathbf{z}) \geq 0$.

Definition (Relative entropy)

Let $\mathbf{p}, \mathbf{q} \in \mathbb{R}^c$ be two probability vectors.

$$KL(\mathbf{p}||\mathbf{q}) := -\sum_{j=1}^{c} \mathbf{p}_j \log\left(\frac{\mathbf{q}_j}{\mathbf{p}_j}\right)$$

Cross-entropy loss for score-based classification

- Let $(\mathbf{x}, y)$ be an image-label sample and $f : \mathbb{R}^d \to \mathbb{R}^c$.
- Goal: define $\mathcal{L}$ s.t. $\mathcal{L}$ is small (large) if $f(\mathbf{x})_y$ is large (small).

Definition (Soft-max)

Let $\mathbf{z} \in \mathbb{R}^c$. The vector

$$\mathbf{q}(\mathbf{z})_i := \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{c} \exp(\mathbf{z}_j)}$$

is a probability vector i.e., $\|\mathbf{q}(\mathbf{z})\|_1 = 1$ and $\mathbf{q}(\mathbf{z}) \geq 0$.

Definition (Relative entropy)

Let $\mathbf{p}, \mathbf{q} \in \mathbb{R}^c$ be two probability vectors.

$$KL(\mathbf{p}||\mathbf{q}) := -\sum_{j=1}^{c} \mathbf{p}_j \log\left(\frac{\mathbf{q}_j}{\mathbf{p}_j}\right)$$

Definition (Cross-entropy loss)

Let $(\mathbf{x}, y) \in \mathbb{R}^d \times \{1, \dots, c\}$

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

$\mathbf{e}_i \in \mathbb{R}^c$ denotes the i-th canonical vector.

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

Cross-entropy loss for score-based classification

- Let $(\mathbf{x}, y)$ be an image-label sample and $f : \mathbb{R}^d \to \mathbb{R}^c$.
- Goal: define $\mathcal{L}$ s.t. $\mathcal{L}$ is small (large) if $f(\mathbf{x})_y$ is large (small).

Definition (Soft-max)

Let $\mathbf{z} \in \mathbb{R}^c$. The vector

$$\mathbf{q}(\mathbf{z})_i := \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{c} \exp(\mathbf{z}_j)}$$

is a probability vector i.e., $\|\mathbf{q}(\mathbf{z})\|_1 = 1$ and $\mathbf{q}(\mathbf{z}) \geq 0$.

Definition (Relative entropy)

Let $\mathbf{p}, \mathbf{q} \in \mathbb{R}^c$ be two probability vectors.

$$KL(\mathbf{p}||\mathbf{q}) := -\sum_{j=1}^{c} \mathbf{p}_j \log\left(\frac{\mathbf{q}_j}{\mathbf{p}_j}\right)$$

Definition (Cross-entropy loss)

Let $(\mathbf{x}, y) \in \mathbb{R}^d \times \{1, \dots, c\}$

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

$\mathbf{e}_i \in \mathbb{R}^c$ denotes the i-th canonical vector.

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

Generalizes logistic loss to multi-class problems

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?

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 $\mathbf{X}_{train} = \{\mathbf{x}_i, y_i\}_{i=1}^n$.

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

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

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

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

Neural network training

- Challenge: Non-convex objective with no guarantees of global optimality
- Tons of heuristics, very little theory
  - First order stochastic methods
  - Early stopping
  - ...

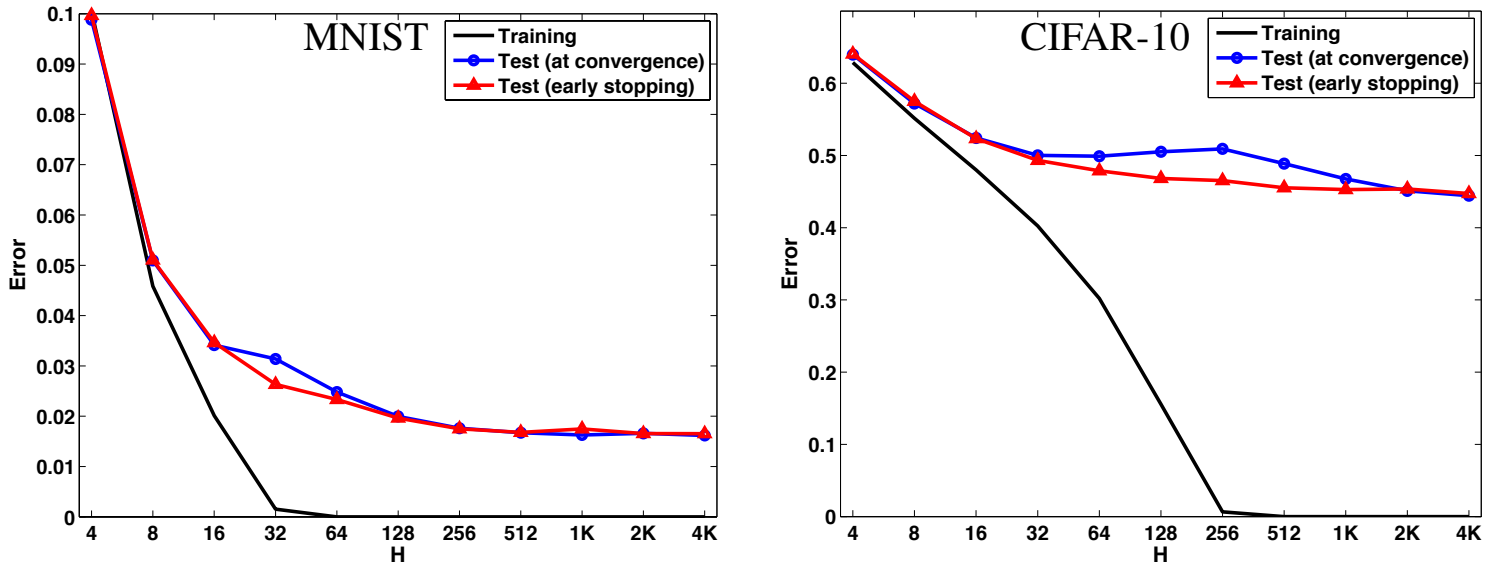


Figure: SGD performance on MNIST (left). CIFAR10 (right). Source: [10].

The effect of overparametrization on generalization

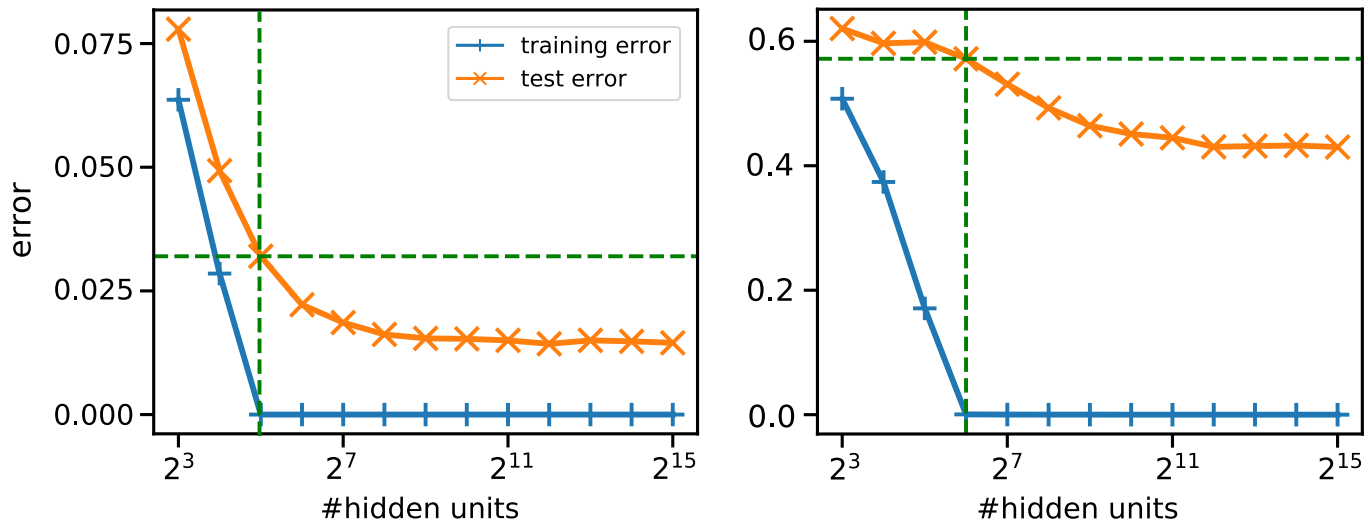


Figure: MNIST (left) and CIFAR10 (right)

Back to reality: Computing the gradient 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 a k-layer network.

Back to reality: Computing the gradient 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 a k-layer network.

Example (Naive computation of the gradient)

suppose $f(\mathbf{x}; W, \beta) = \beta^T \sigma(W\mathbf{x})$ is a 1-hidden-layer network with no bias. Let $(\mathbf{x}_i, y_i)$ be a sample and $\mathcal{L}_i(W, \beta) = (y_i - \beta^T \sigma(W\mathbf{x}_i))^2$ be the loss function, 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 $\mathcal{L}_i$ wrt the output of the network.

Complexity of naive derivative vs Backpropagation

Forward pass exploits the nested structure of the multilayer network by starting from the first layer and then by going towards the last. The backward pass starts with the last layer and computes $\partial \mathcal{L}/\partial W^{(l)}$, $\partial \mathcal{L}/\partial b^{(l)}$ re-using previous computations.

Suppose the size of each layer is bounded by H, and the number of layers is k. For a sample $(\mathbf{x}_i, y_i)$ the complexity of computing the derivative of $\nabla_\theta \mathcal{L}_i(\theta)$ is given by

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

Forward pass* (self-study)

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

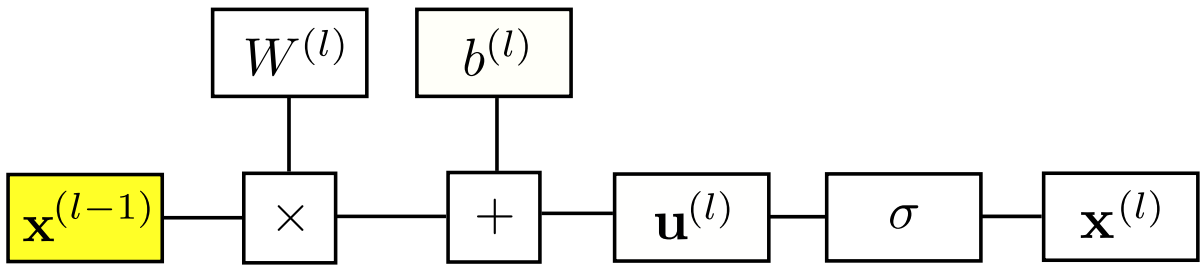


Figure: Computation of $\mathbf{u}^{(l)}$ and $\mathbf{x}^{(l)}$ starting from $\mathbf{x}^{(l-1)}$

Forward pass* (self-study)

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

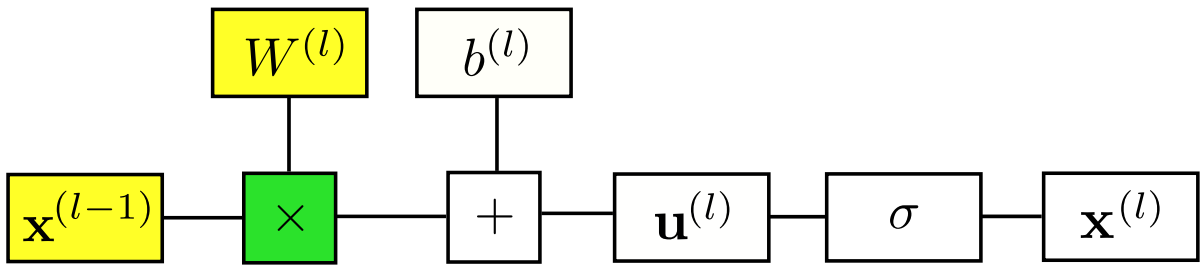


Figure: Computation of $\mathbf{u}^{(l)}$ and $\mathbf{x}^{(l)}$ starting from $\mathbf{x}^{(l-1)}$

Forward pass* (self-study)

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

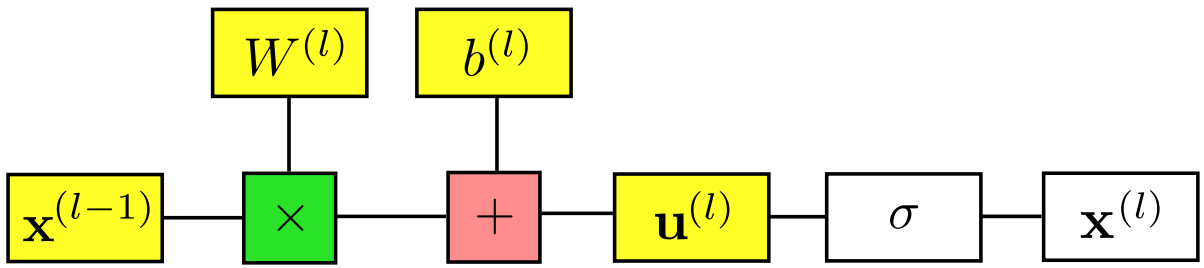


Figure: Computation of $\mathbf{u}^{(l)}$ and $\mathbf{x}^{(l)}$ starting from $\mathbf{x}^{(l-1)}$

Forward pass* (self-study)

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

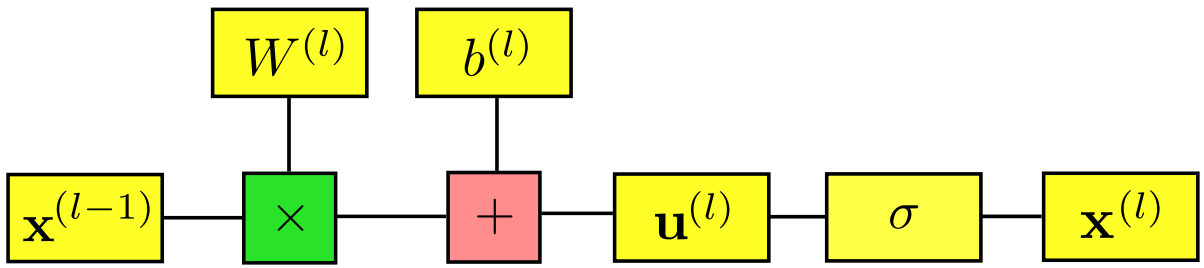


Figure: Computation of $\mathbf{u}^{(l)}$ and $\mathbf{x}^{(l)}$ starting from $\mathbf{x}^{(l-1)}$

Backward pass* (self-study)

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

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

Backward pass* (self-study)

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.

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

Backward pass* (self-study)

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

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

1.

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

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).

Backward pass* (self-study)

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

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

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

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* (self-study)

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

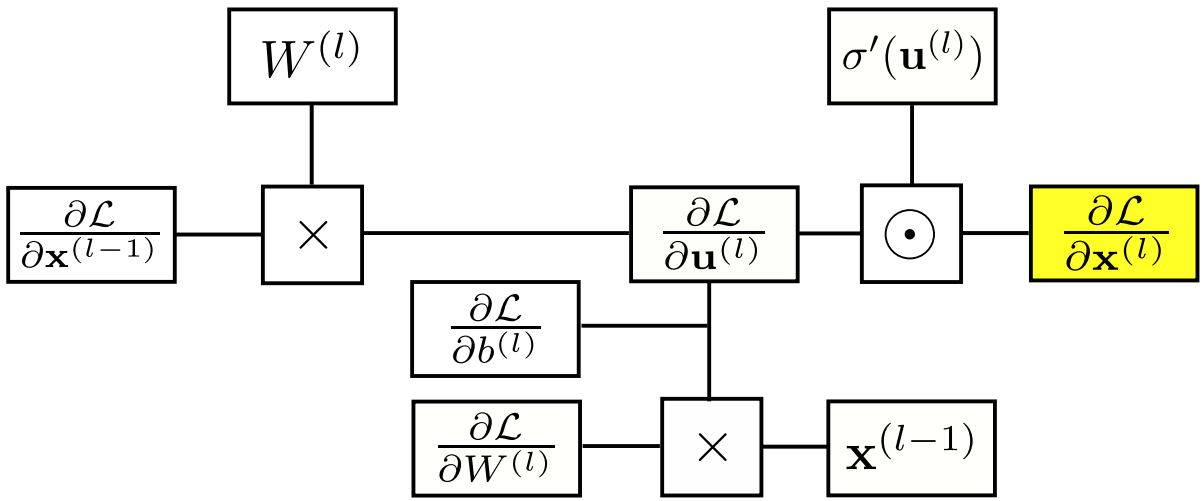


Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

Backward pass* (self-study)

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

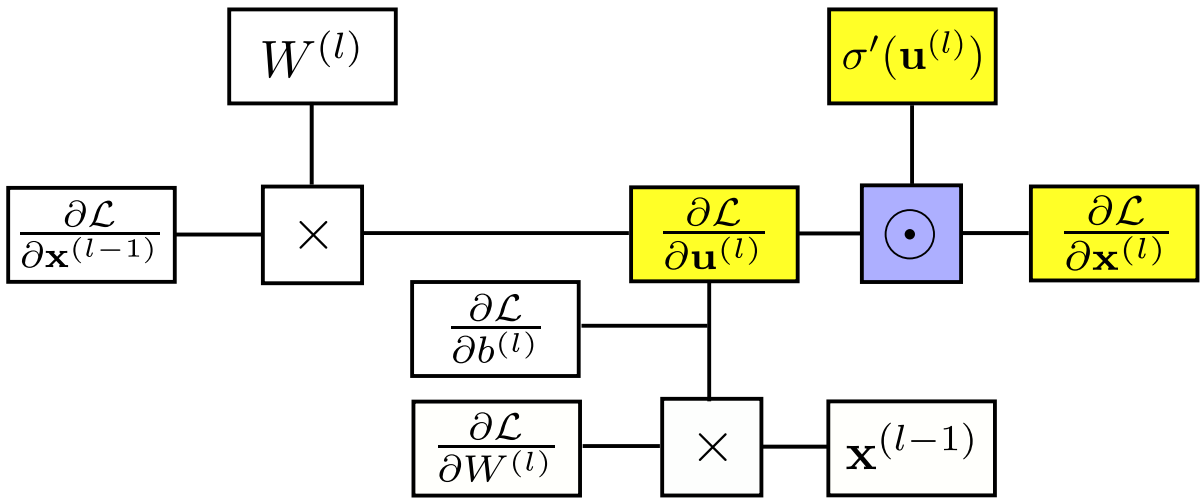


Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

Backward pass* (self-study)

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

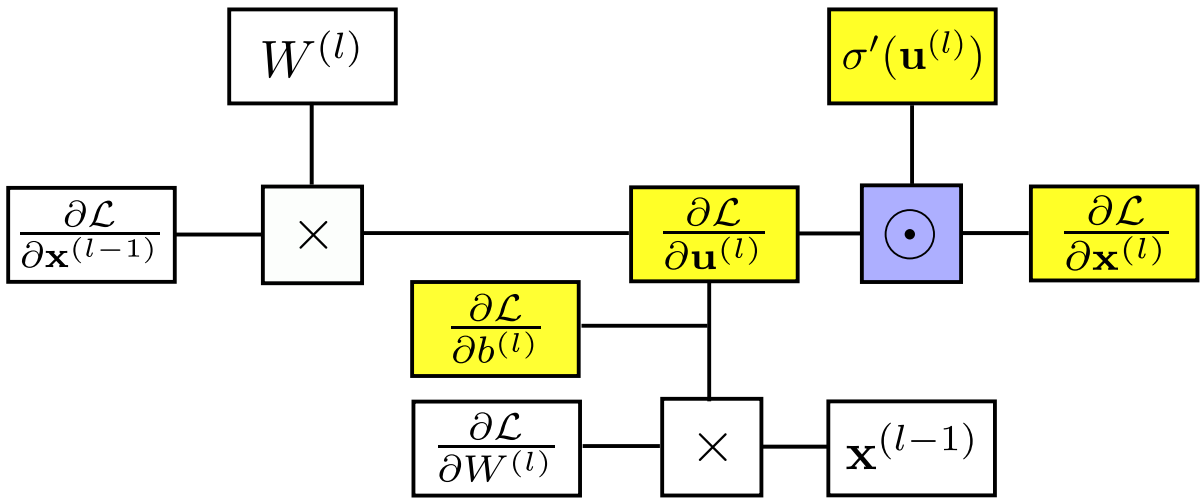


Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

Backward pass* (self-study)

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

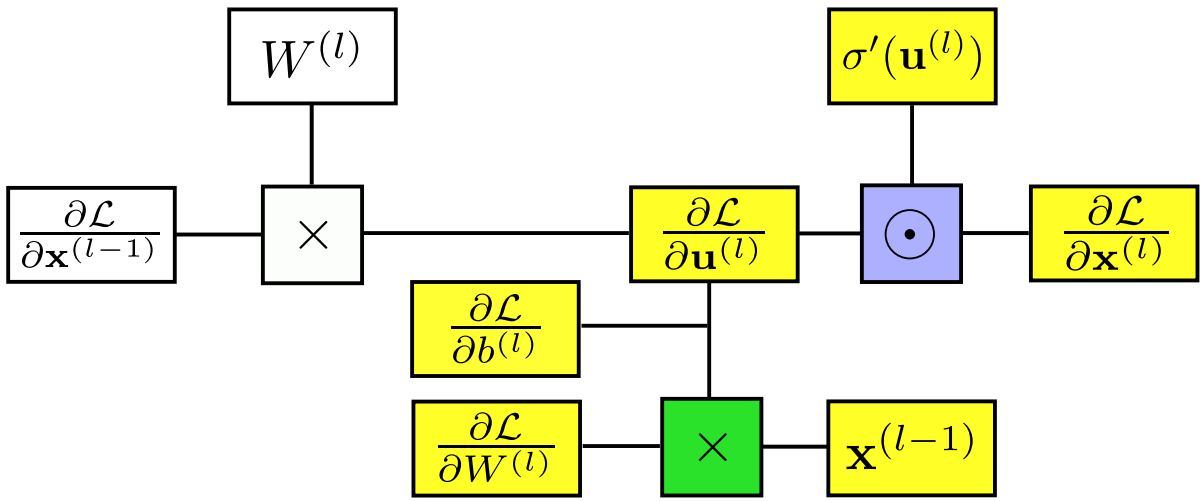


Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

Backward pass* (self-study)

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

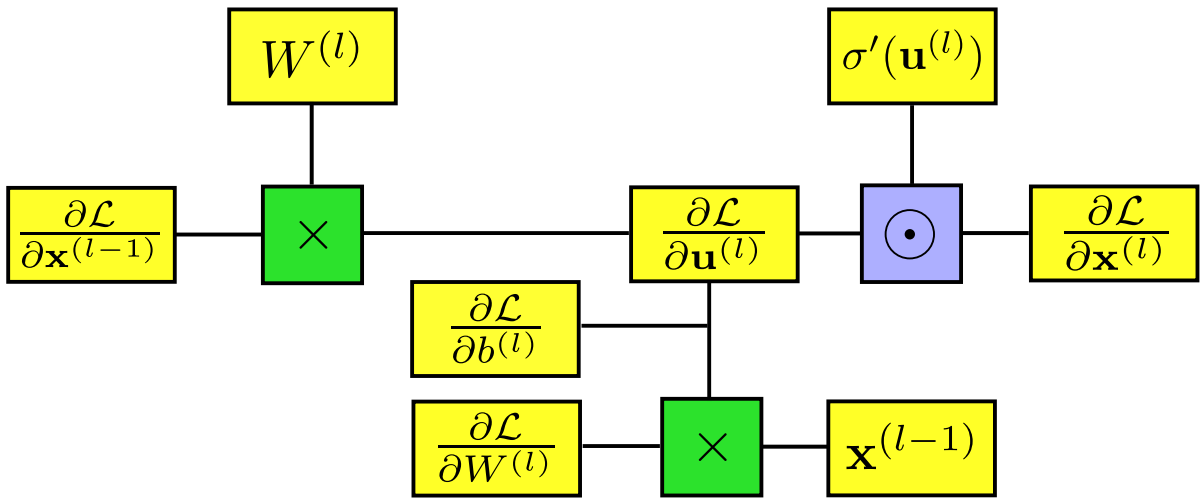


Figure: Computation of $\frac{\partial \mathcal{L}}{\partial b^{(l)}}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l-1)}}$ starting from $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(l)}}$

Complexity of Backpropagation* (self-study)

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** $l = 1, \dots, k$
   - $\mathbf{u}^{(l)} = W^{(l)}\mathbf{x}^{(l-1)} + b^{(l)}$
   - $\mathbf{x}^{(l)} = \sigma(\mathbf{u}^{(l)})$

Backward pass scheme

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

Complexity of Backpropagation* (self-study)

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

Complexity of Backpropagation* (self-study)

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 $g = \nabla \mathcal{L}(\theta)$

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
  2. Non-convexity
  3. Ill-conditioning

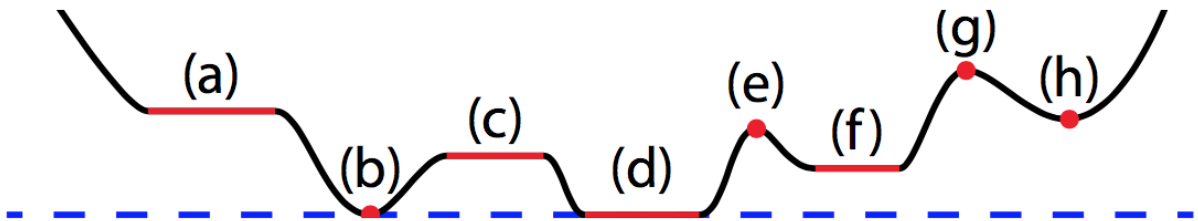


Figure: A non-convex function. (a) and (c) are plateaus, (b) and (d) are global minima, (f) and (h) are local minima, (e) and (g) are local maxima. [6]

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

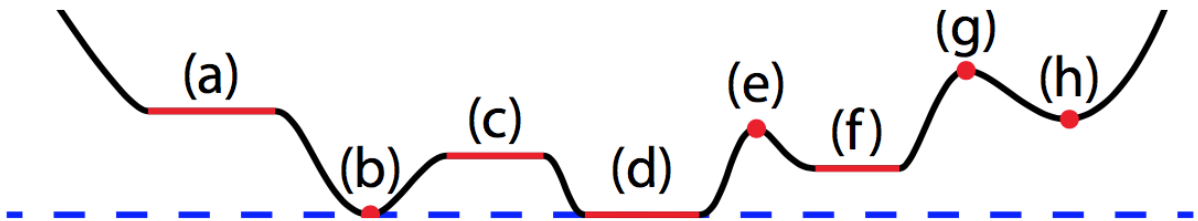


Figure: A non-convex function. (a) and (c) are plateaus, (b) and (d) are global minima, (f) and (h) are local minima, (e) and (g) are local maxima. [6]

Stochastic Gradient Descent (SGD)

Vanilla Minibatch SGD
Input: learning rate $\{\gamma\}_{t=0}^{N-1}$
1. initialize θ_0 **2.** **For** t = 0, 1,... ,N-1: obtain the minibatch gradient $\hat{\mathbf{g}}_t$ update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \hat{\mathbf{g}}_t$

Stochastic Gradient Descent (SGD)

Vanilla Minibatch SGD

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

1. initialize θ_0
2. **For** t = 0, 1,... ,N-1:
   obtain the minibatch gradient $\hat{\mathbf{g}}_t$
   update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \hat{\mathbf{g}}_t$

- Heavy ball is more frequently used vs Nesterov accelerated SGD

Heavy Ball Minibatch SGD (HB SGD)

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

1. initialize θ_0, $\mathbf{m_0} \leftarrow \mathbf{0}$
2. **For** t = 0, 1,... ,N-1:
   obtain the minibatch gradient $\hat{\mathbf{g}}_t$
   update $\mathbf{m}_{t+1} \leftarrow \rho \mathbf{m}_t + \hat{\mathbf{g}}_t$
   update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \mathbf{m}_{t+1}$

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 L-continous 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])

Run minibatch SGD with assumptions above for N iterations with constant learning rate $\gamma_t = \frac{1}{L\sqrt{N}}$. The expectation of the gradients' average squared norm converges at a rate of $O(\frac{1}{\sqrt{N}})$ i.e., $\mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N-1} \|\mathbf{g}_t\|_2^2\right] \sim \mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$

- Convergence is captured by the gradient norm

Convergence of SGD* (Self-study)

Proof

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

$$\begin{aligned} \mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t) &\leq (\theta_{t+1} - \theta_t)^T \mathbf{g}_t + \frac{L}{2} \|\theta_{t+1} - \theta_t\|_2^2 \\ &= -\gamma_t \hat{\mathbf{g}}_t{}^T \mathbf{g}_t + \frac{\gamma_t^2 L}{2} \|\hat{\mathbf{g}}_t\|_2^2 \end{aligned} \tag{8}$$

Take the expectation and use the assumption 3

$$\mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)] = -\gamma_t \|\mathbf{g}_t\|_2^2 + \frac{\gamma_t^2 L}{2} (\|\mathbf{g}_t\|_2^2 + \sigma^2) \tag{9}$$

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

$$\begin{aligned} \mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)] &= -\frac{1}{L\sqrt{N}} \|\mathbf{g}_t\|_2^2 + \frac{1}{2LN} (\|\mathbf{g}_t\|_2^2 + \sigma^2) \\ &\leq -\frac{1}{2L\sqrt{N}} \|\mathbf{g}_t\|_2^2 + \frac{\sigma^2}{2LN} \end{aligned} \tag{10}$$

Convergence of SGD* (Self-study)

Proof (Cont'd).

Sum the inequality of N steps together and use assumption 1

$$\begin{aligned}\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) &\geq \mathcal{L}(\theta_0) - \mathbb{E}[\mathcal{L}(\theta_N)] \\ &= \mathbb{E}\left[\sum_{t=0}^{N-1} (\mathcal{L}(\theta_t) - \mathcal{L}(\theta_{t+1}))\right] \\ &\geq \frac{1}{2L}\mathbb{E}\left[\sum_{t=0}^{N-1} (\frac{\|\mathbf{g}_t\|_2^2}{\sqrt{N}} - \frac{\sigma^2}{N})\right]\end{aligned} \tag{11}$$

Rearrange the inequality, we have the following

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

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

- A preview of what is next

	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

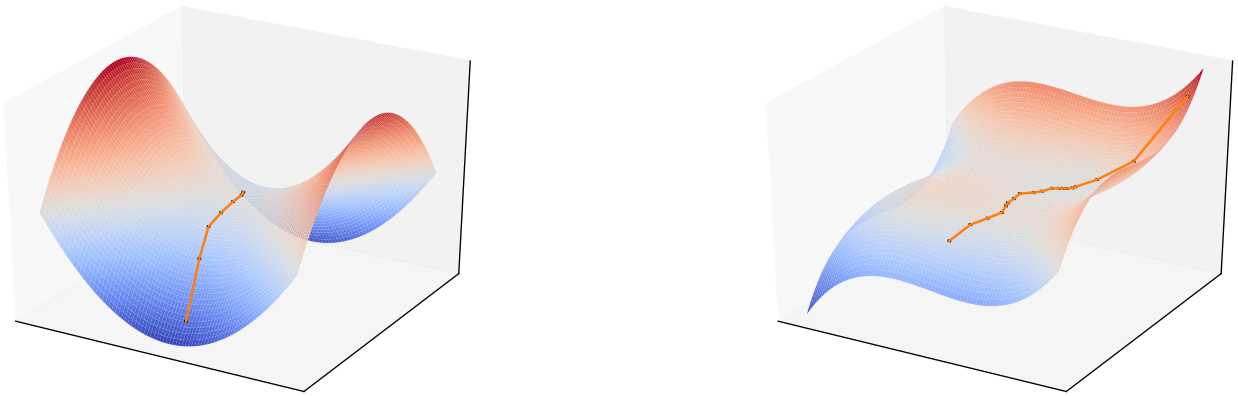


Figure: Stochasticity introduced by minibatch can help scape saddle points (Left). Momentum can help scape local minima (Right).

Escaping from saddle points

Recall (classification of stationary points)

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a twice differentiable function 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

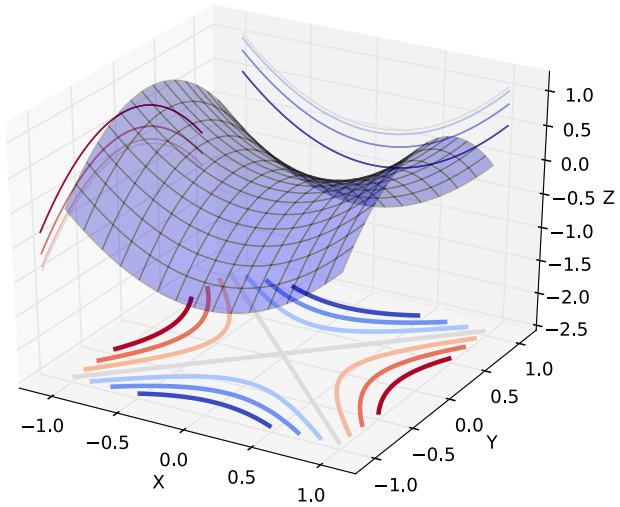


Figure: Minmax saddle ($\lambda_i \neq 0$ for all i)

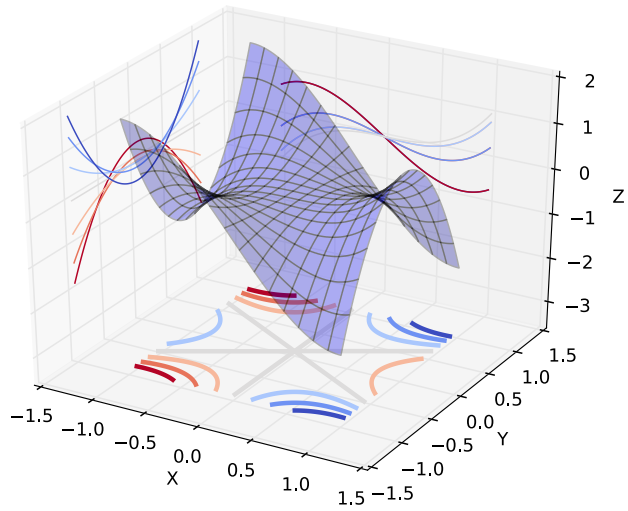


Figure: Monkey saddle ($\lambda_i = 0$ for some i)

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 robust local minimum, or is a saddle point (or local maximum) with a significant negative eigenvalue.

Perturbed SGD algorithm

Perturbed Stochastic Gradient Descent [5]
Input: Stochastic Gradient Oracle $SG(\mathbf{x})$, initial point $\mathbf{x}_0$, number of iterations T, step size η
1. For t = 0 to T - 1: sample noise ξ uniformly from unit sphere update $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta(SG(\mathbf{x}_t) + \xi)$

Perturbed SGD algorithm

Perturbed Stochastic Gradient Descent [5]
Input: Stochastic Gradient Oracle $SG(\mathbf{x})$, initial point $\mathbf{x}_0$, number of iterations T, step size η
1. For t = 0 to T - 1: sample noise ξ 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.

Perturbed SGD scapes saddle points

Theorem (Convergence of PSGD [5])

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 $\mathbf{x}^*$.*

Ill-conditioned curvature

Definition (Condition Number)

For a matrix M, the condition number is the ratio of its largest singular value to its smallest singular value: $\kappa(M) = \frac{\sigma_{max}(M)}{\sigma_{min}(M)}$.

- We say a matrix is ill-conditioned if it has large condition number
- Ill-conditioned Hession corresponds to a very long 'ellipsoids' contour.

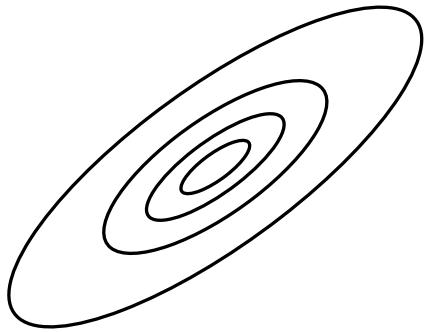


Figure: Ill-conditioned contour.

Ill-conditioned curvature

Definition (Condition Number)

For a matrix M, the condition number is the ratio of its largest singular value to its smallest singular value: $\kappa(M) = \frac{\sigma_{max}(M)}{\sigma_{min}(M)}$.

- We say a matrix is ill-conditioned if it has large condition number
- Ill-conditioned Hession corresponds to a very long 'ellipsoids' contour.

- SGD zigzags to the minimum
- Optimizer needs to adapt to the landscape
- Different step size in different directions can help

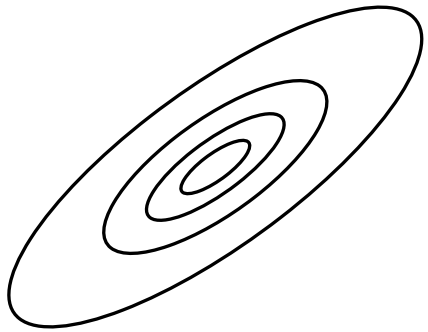


Figure: Ill-conditioned contour.

AdaGrad [4]

AdaGrad

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

1. initialize θ_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 $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$

AdaGrad [4]

AdaGrad

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

1. initialize θ_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 $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$

- Upshots
  - Adaptive learning rate in different directions
  - 'Effective learning rate' decreases monotonically if γ_t is a constant

RMSProp [12]

- Average used in place of accumulation with a decaying τ
- Recent gradients have more weight when calculating the average

AdaGrad
Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ
1. initialize θ_0, $\mathbf{r} \leftarrow \mathbf{0}$ **2.** **For** t $= 0, 1, \ldots, N\text{-}1$: 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 θ_0, $\mathbf{r} \leftarrow \mathbf{0}$ **2.** **For** t $= 0, 1, \ldots, N\text{-}1$: 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 [8]

- Two approaches to speed up training
  1. Momentum e.g., HB SGD (rate implications in convex)
  2. Adaptive learning rate e.g., AdaGrad, RMSProp (variable metric in convex)
- How about mixing them together?

Adam [8]

- Two approaches to speed up training
  1. Momentum e.g., HB SGD (rate implications in convex)
  2. Adaptive learning rate e.g., AdaGrad, RMSProp (variable metric in convex)
- 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 θ_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	$\frac{\beta_1(1-\beta_1^{t-1})}{1-\beta_1^t}$

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 $\hat{\mathbf{g}}_t$.

Adaptivity may lead to overfit

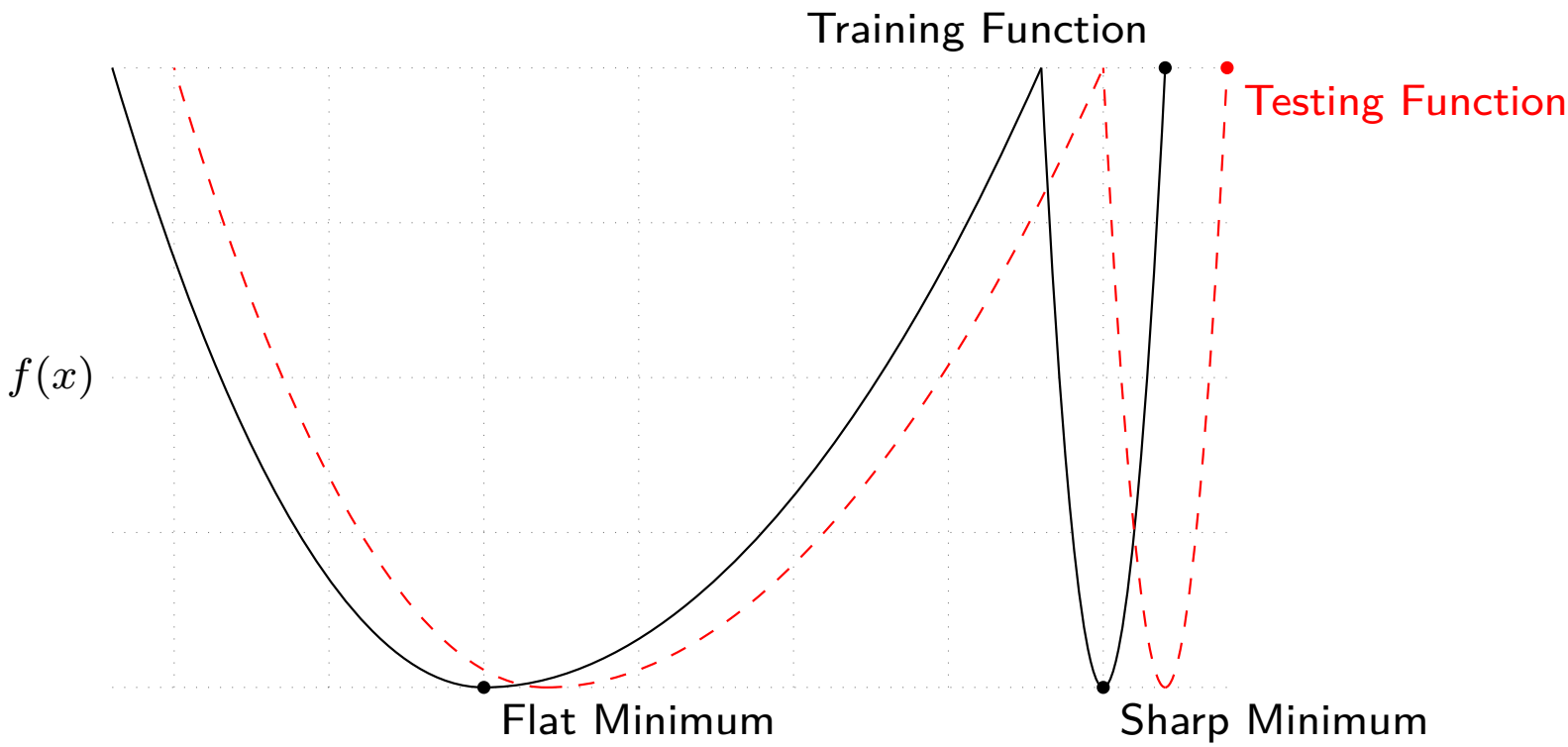


Figure: Sharp Minima vs Flat Minima [7]

- Intuitively, flat minima has better generalization property than sharp minima. [7]
- Empirically, adaptive methods tend to find sharper minima than ones found by SGD.
- Theoratically, the relationship between landscape around minima and generalization is an open problem. [3]

Adaptivity may lead to overfit

Adaptive learning method may converge fast but generalize worse.

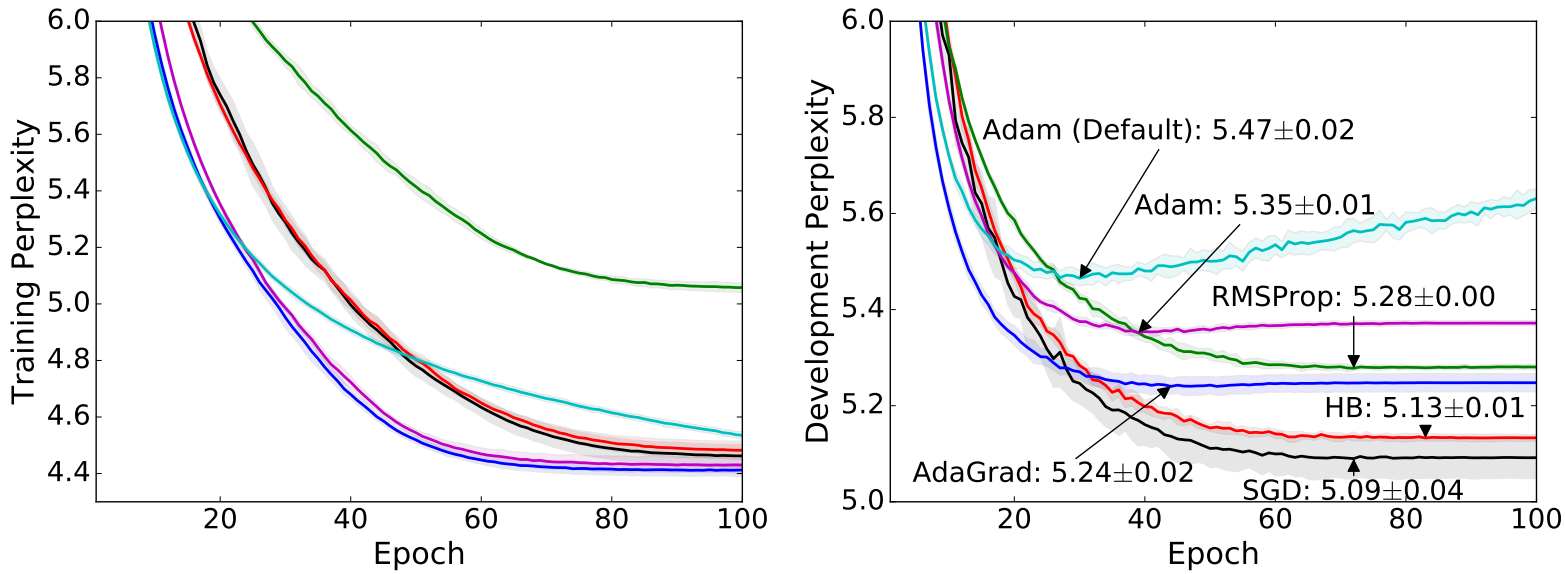


Figure: Performance of different optimizers in training and development set of a language modeling problem. The training and test perplexity are the exponential values of training and test losses.[13]

Curse of dimensionality and nonconvexity

- Complicated models, better performance.
- No universal optimizers.
- A long way to go ...

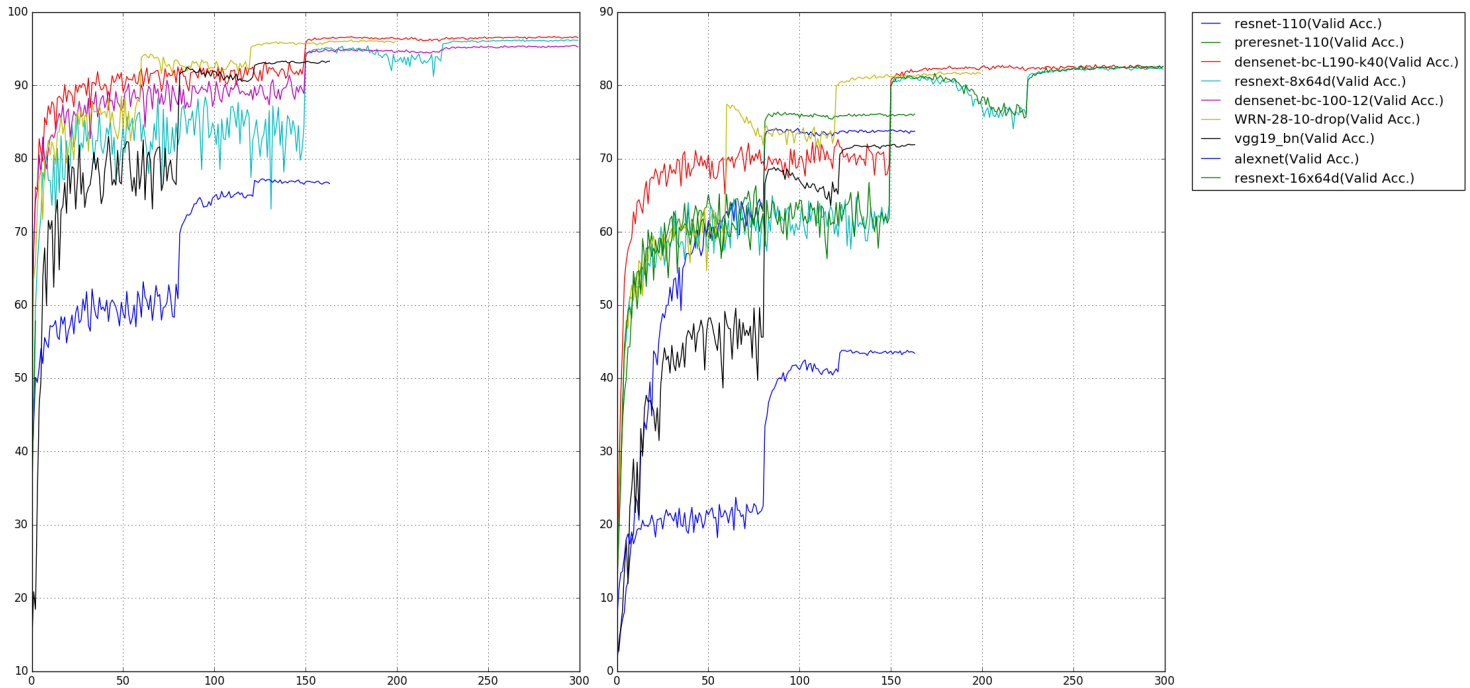


Figure: Performance of different popular architectures on test set in CIFAR10 (left) and CIFAR100 (right). [1]

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

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] 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.

[5] 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

[6] 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.

[7] 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.

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

[9] 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.

[10] B. Neyshabur, R. Tomioka, R. Salakhutdinov, and N. Srebro.
Geometry of Optimization and Implicit Regularization in Deep Learning.
ArXiv e-prints, May 2017.

References III

[11] M. Telgarsky.
Benefits of depth in neural networks.
ArXiv e-prints, February 2016.

[12] 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.

[13] 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.