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

- From convex to nonconvex optimization problems
- Neural network architectures
- Backpropagation
- Convergence of SGD in nonconvex problems
- Adaptive learning algorithms (Adam, Adagrad)

Recommended reading material

Motivation: Linear classifiers

Problem (Logistic regression)

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

$$\mathbb{P}(b_i | \mathbf{a}_i, \mathbf{x}^\natural, \mu) \propto 1/(1 + e^{-b_i(\langle \mathbf{x}^\natural, \mathbf{a}_i \rangle + \mu)}),$$

where $\mathbf{x}^\natural \in \mathbb{R}^n$ *is some weight vector.*

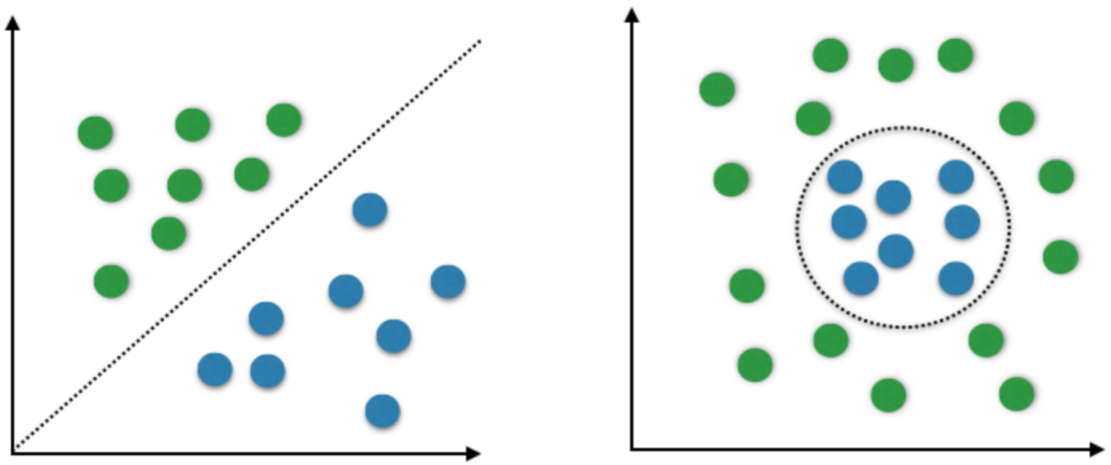


Figure: Linearly Seperable versus nonlinearly Separable dataset

Motivation: Approximation theory

Definition

A 1-hidden-layer network with m nodes is a parametric function $f(\cdot;\theta):\mathbb{R}^n \to \mathbb{R}$ of the form

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

where β and $b \in \mathbb{R}^m$, $W \in \mathbb{R}^{m\times n}$. $\sigma:\mathbb{R}\to\mathbb{R}$ is a function called the activation function. Here we denote $\sigma(\mathbf{x}) = (\sigma(x_1),\dots,\sigma(x_n))$.

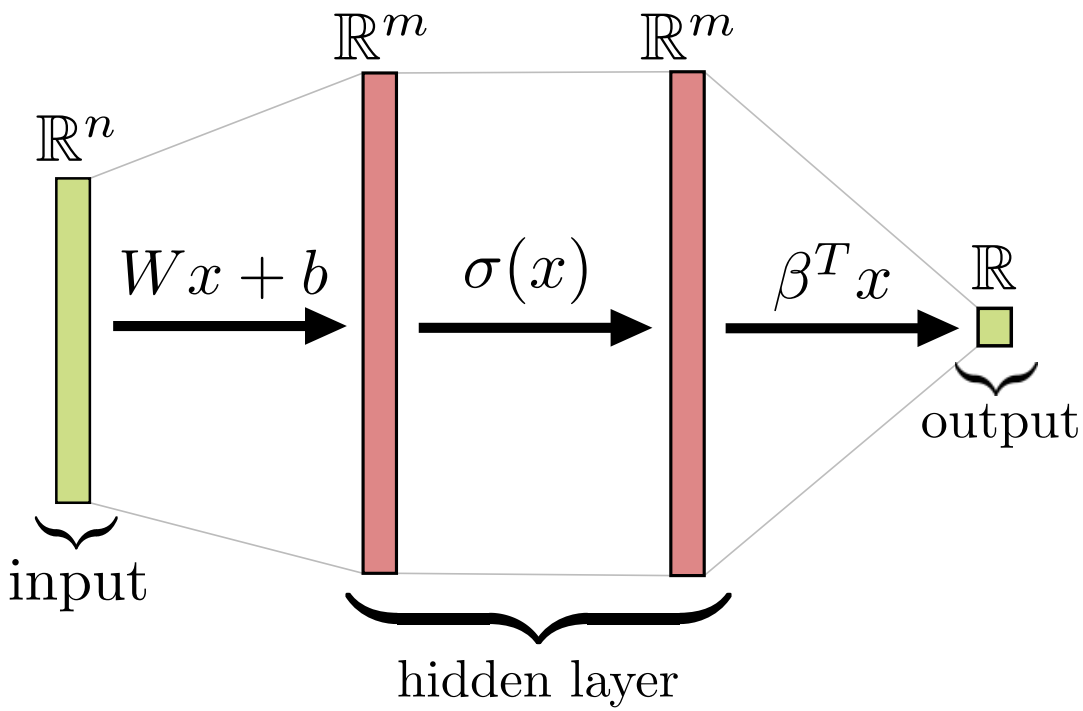


Figure: A 1-hidden-layer neural network

Motivation: Approximation theory

Theorem (Universal approximation theorem [?])

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 $C(I_n)$.

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

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

Issue: Non-convex optimization objective

Example

Consider a 1-hidden-layer neural network $f : \mathbb{R} \to \mathbb{R}$ with activation function $\sigma(x) = x$

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

with $w_1, w_2 \in \mathbb{R}$. For a sample $(\mathbf{a}_0, b_0) = (1, 1)$ the squared error is

$$(b_0 - f(\mathbf{a}_0; w_1, w_2))^2 = (1 - w_2 w_1)^2$$

Show that it is neither convex nor concave.

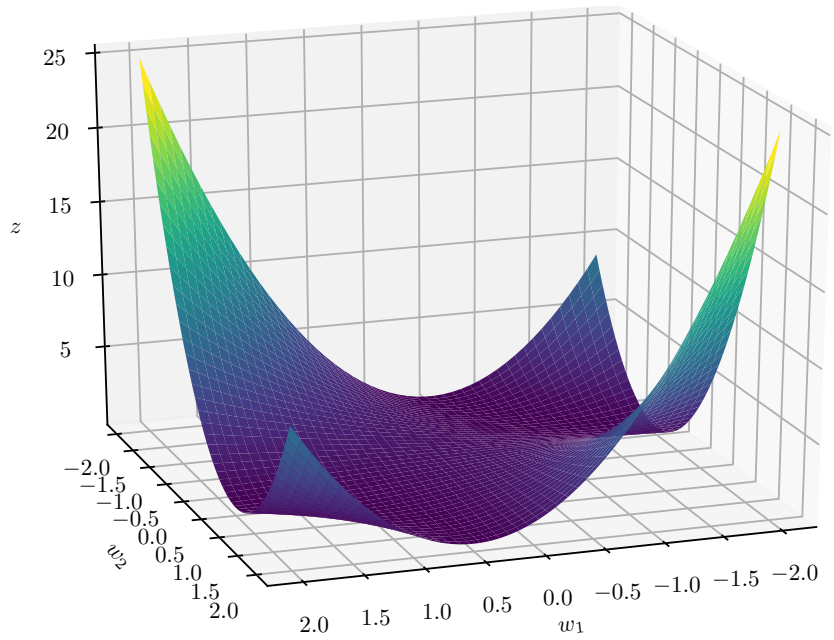


Figure: Loss surface $(1 - w_2 w_1)^2$

Why more layers?

Definition

A 2-hidden-layer network is a parametric function $f(\cdot;\theta):\mathbb{R}^m \to \mathbb{R}$ *of the form*

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

We define a k-hidden-layer network in an analogous way.

Theorem (Benefits of depth [?])

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*

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

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

End-to-end machine learning

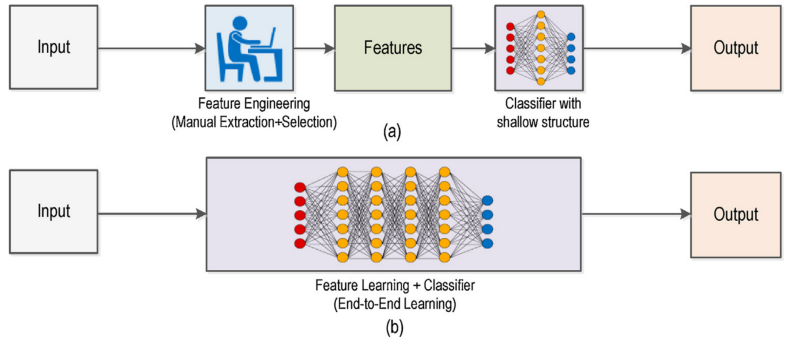


Figure: a) Traditional machine learning approaches b) End-to-end deep learning methods [**?**]

Example

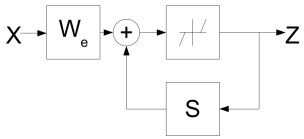
In sparse coding, given a an overcomplete family basis vectors, we would like to find the optimal sparse code vector $Z^* \in \mathbb{R}^m$ to reconstruct the input vector $X \in \mathbb{R}^n$

$$Z^* = \arg\min_Z \frac{1}{2}\|X - W_d Z\|_2^2 + \alpha\|Z\|_1, \tag{1}$$

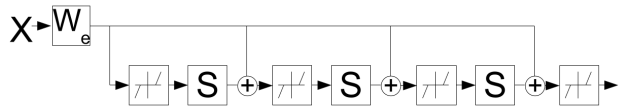
where W_d is $n \times m$ dictionary matrix, whose columns are normalized basis vectors.

- Too slow for real-time applications

End-to-end machine learning



ISTA method for sparse coding



Learned ISTA - time unfolded version of the ISTA block diagram

Example

$$\mathcal{L}(W) = \frac{1}{P} \sum_{p=0}^{(P-1)} \underbrace{\frac{1}{2} \|Z^{*p} - f_e(W, X^p)\|^2}_{L(W, X^p)} \tag{2}$$

where $(X^1, \ldots, X^P)$ are training set, and Z^{*p} is the optimal code for example X^p. Parameters of the encoder are optimized with SGD:

$$W(j+1) = W(j) - \eta(j) \frac{dL(W, X^{(j \bmod P)})}{dW} \tag{3}$$

Back propagation

Training loss

Consider the loss over the training set:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(\mathbf{x^{(i)}}; \theta), \mathbf{y^{(i)}}), \tag{4}$$

where $\ell(f(\mathbf{x^{(i)}}; \theta), \mathbf{y^{(i)}})$ is the loss function on the sample $(\mathbf{x^{(i)}}, \mathbf{y^{(i)}})$. The goal of backpropagation is to:

- compute the gradient of the loss $\mathcal{L}$ w.r.t to the parameters of the network θ for a single data point $(\mathbf{x}', \mathbf{y}')$
- update the parameters of the network using for instance first order methods (SGD)

$$\theta^{\mathbf{t}} = \theta^{\mathbf{t-1}} - \eta^t \nabla \ell(f(\mathbf{x}'; \theta), \mathbf{y}') \tag{5}$$

Neural network architectures

- Fully connected layers

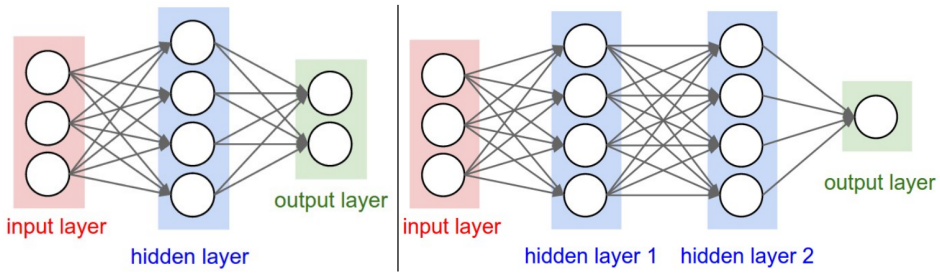


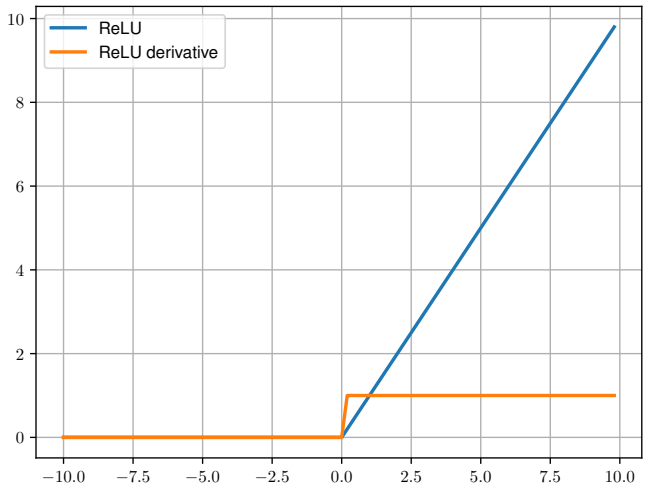
Figure: Fully connected layers

- If the activation function is $\sigma(x) = x$, a network can only do an affine transformation.
- Multiple layers are equivalent to a single affine map $W_1(W_2x + b_2) + b_1 = Wx + b$.
- With a continuous, bounded and nonconstant activation function, neural networks can approximate any function.

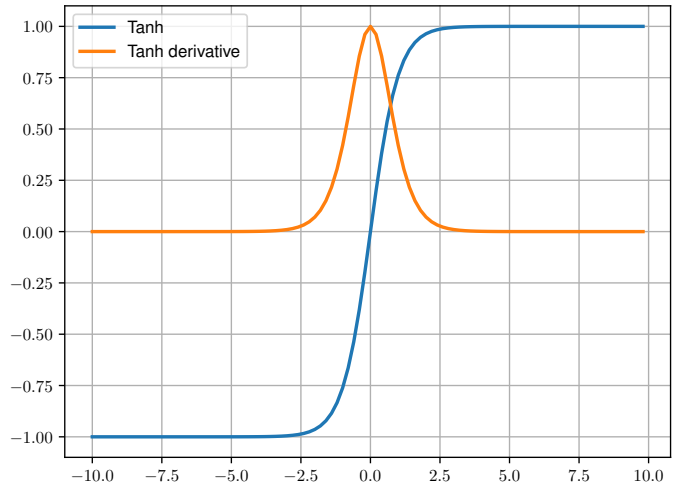
Activation functions

- Nonlinear activation functions
  - used to separate nonlinearly separable data
  - Allow neural networks to approximate any arbitrarily complex functions
  - without nonlinearity, a network is equivalent to a linear classifier.

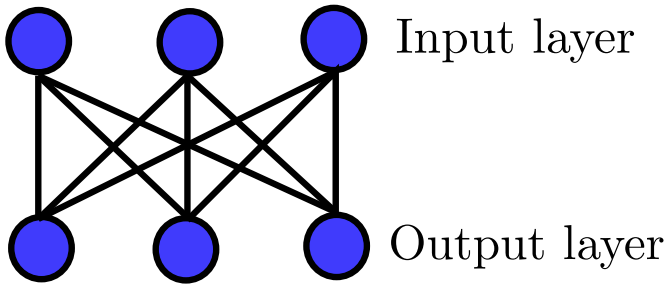
Rectified Linear Unit (ReLU)$(f(x) = \max(0, x))$:



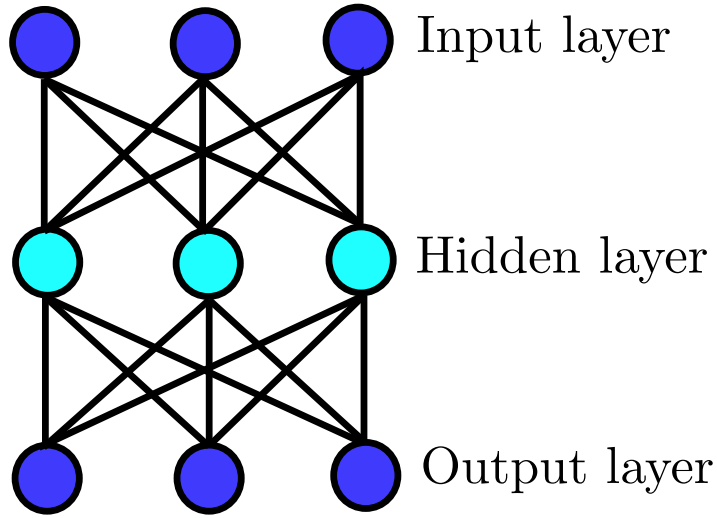
Tanh:



one-layer versus multi-layer networks



One-layer network



Multi-layer network

There are no desired values for the hidden layers $\Longrightarrow$ Backpropagation method creates the desired values for the hidden layers.

multi-layer networks

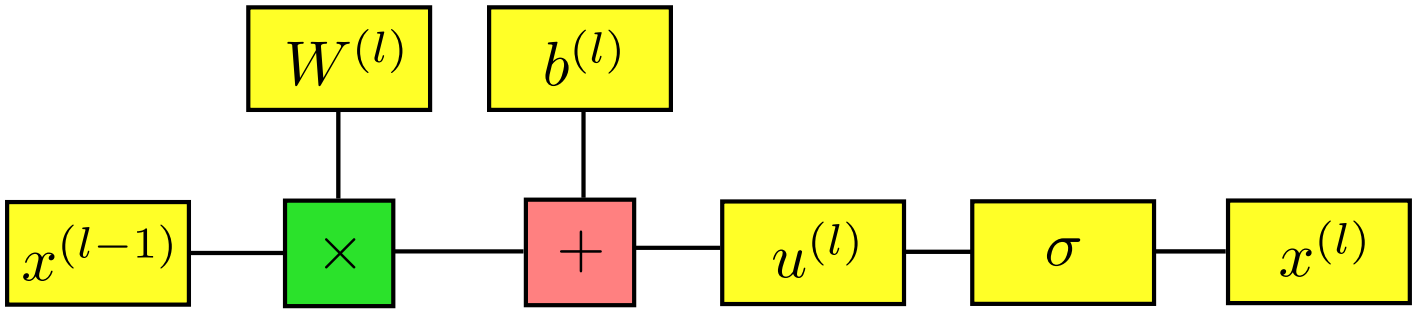


Figure: Multi-layer network

- L layers of weights ($\mathbf{W}^{(\mathbf{l})}$) and biases ($\mathbf{b}^{(\mathbf{l})}$):

$$\forall l = 1, \dots, L, \quad \begin{cases} \mathbf{u}^{(\mathbf{l})} & = \mathbf{W}^{(\mathbf{l})}\mathbf{x}^{(\mathbf{l-1})} + \mathbf{b}^{(\mathbf{l})} \\ \mathbf{x}^{(\mathbf{l})} & = \sigma(\mathbf{u}^{(\mathbf{l})}) \end{cases}$$

- We need to compute the gradient of the loss with respect to the parameters: $\frac{\partial \mathcal{L}}{\partial b_i}$ and $\frac{\partial \mathcal{L}}{\partial W_{i,j}}$

Forward pass

Forward pass scheme	
Set $\mathbf{x}^{(0)} = \mathbf{x}$, and initialize $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$	$\forall l = 1, \dots, L.$
1. For $l = 1, \dots, L$	
Compute $\mathbf{u}^{(l)} = \mathbf{W}^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}$	
Compute $\mathbf{x}^{(l)} = \sigma(\mathbf{u}^{(l)})$	

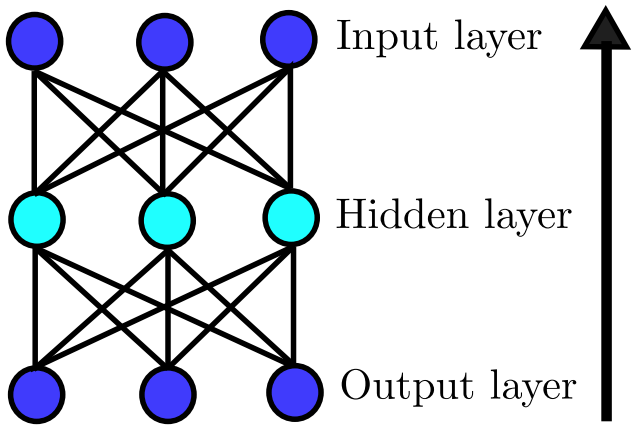


Figure: Forward pass

Backward pass

Then, we can compute the gradient of the loss w.r.t to the parameters of the network:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(\mathbf{l})}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(\mathbf{l})}} (\mathbf{x}^{(\mathbf{l-1})})^T \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(\mathbf{l})}} = \frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(\mathbf{l})}} \tag{7}$$

Update the parameters using gradient step:

$$\mathbf{W}^{(\mathbf{l+1})} = \mathbf{W}^{(\mathbf{l})} - \gamma \frac{\partial \mathcal{L}}{\partial \mathbf{w}^{(\mathbf{l})}} \tag{8}$$

$$\mathbf{b}^{(\mathbf{l+1})} = \mathbf{b}^{(\mathbf{l})} - \gamma \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{(\mathbf{l})}} \tag{9}$$

Chain rule (1)

We use chain rule to compute derivative of the composition of functions.

- **Scalar case:** Suppose $f, h : \mathbb{R} \to \mathbb{R}$ and $y = f(x), z = h(y)$ or $z = (h \circ f)(x)$, using chain rule we have:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \tag{10}$$

- **Vector in, scalar out:** Suppose $f : \mathbb{R}^N \to \mathbb{R}$, then we have:

$$\frac{\partial y}{\partial \mathbf{x}} = (\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_N}), \tag{11}$$

  where x_i is the ith coordinate of the vector $\mathbf{x}$.

Chain rule (2)

- **Vector in, Vector out:** Suppose $f : \mathbb{R}^N \to \mathbb{R}^M$, and $h : \mathbb{R}^M \to \mathbb{R}^K$. then derivative of $\mathbf{y}$ with respect to $\mathbf{x}$, also called *Jacobian* is:

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{bmatrix}$$

We use the chain rule again:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \tag{12}$$

Now $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is $M \times N$ matrix, and $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is $K \times N$ matrix, so $\frac{\partial \mathbf{z}}{\partial \mathbf{y}}$ is $K \times M$ matrix.

Backward pass

Backward pass

1. Compute gradient of the loss w.r.t. to the predicted value $\frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(\mathbf{L})}}$

2. **For** $l = L-1, \ldots, 2$

Compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}^{(\mathbf{l})}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(\mathbf{l})}} \odot \sigma'(\mathbf{u}^{(\mathbf{l})})$$

Compute

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

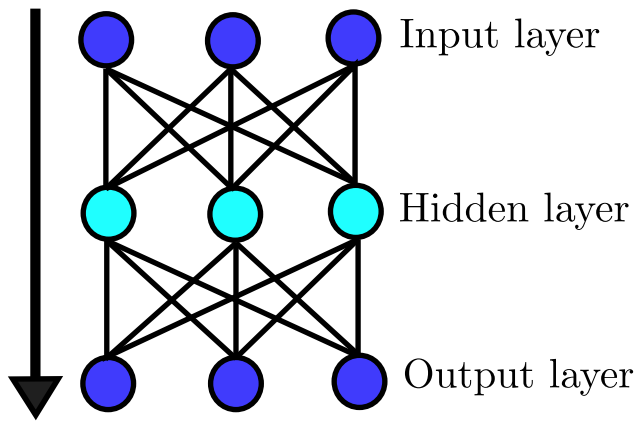


Figure: Backward pass

Optimizers for Neural Networks

What do we have now?

- Loss function $\mathcal{L}(\theta)$.
- First-order gradient from back-propagation $g = \nabla \mathcal{L}(\theta)$

What are the difficulties?

- Super high dimension. Second-order methods like Newton Method is not feasible.
- Numerous poor local minimas and saddle points.
- Ill-conditioned curvature.

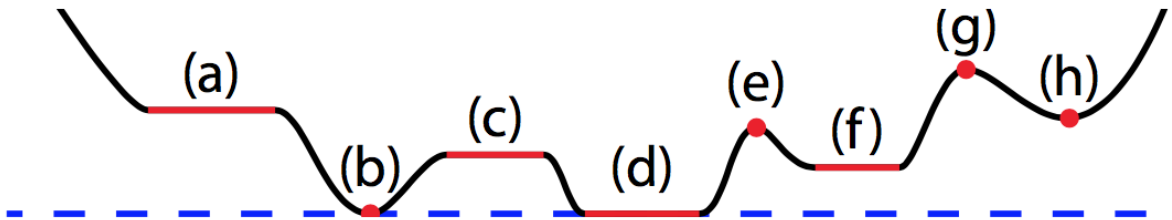


Figure: The curvature of an example nonconvex function. (a) and (c) are plateaus, (b) and (d) are global minima, (f) and (h) are local minima, (e) and (g) are local maxima. [**?**]

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 minibatch gradient $\hat{\mathbf{g}}_t$
   update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \hat{\mathbf{g}}_t$

Minibatch SGD with Momentum (Heavy Ball Method)

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 minibatch gradient $\hat{\mathbf{g}}_t$
   update $\mathbf{m}_{t+1} \leftarrow \rho \mathbf{m}_t + (1-\rho)\hat{\mathbf{g}}_t$
   update $\theta_{t+1} \leftarrow \theta_t - \gamma_t \mathbf{m}_t$

Stochastic Gradient Descent (SGD)

Remark

- SGD is fast, unbiased and needs no extra memory.
- Stochasticity can help escape saddle points.
- SGD might get stuck in local minima.
- Momentum help smooth out variations.
- Momentum can escape the poor local minima.
- Momentum might cause overshoot when ρ and γ are big.

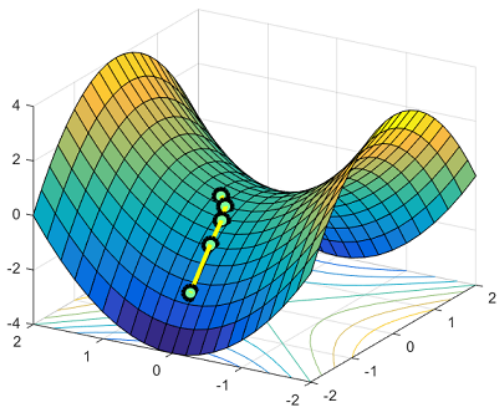


Figure: Noise introduced by stochasticity can help escape saddle points

Convergence of SGD [?]

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:

$$\|\nabla \mathcal{L}(\theta_1) - \nabla \mathcal{L}(\theta_2)\|_2 \leq L \|\theta_1 - \theta_2\|_2 \tag{13}$$

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

$$\mathbb{E}(\hat{\mathbf{g}}) = \mathbf{g} \tag{14}$$

$$\mathbb{E}(\|\hat{\mathbf{g}} - \mathbf{g}\|_2^2) \leq \sigma^2 \tag{15}$$

Convergence of SGD

Proof.

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

$$\begin{aligned} \mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t) &\leq (\theta_{t+1} - \theta_t)^T \mathbf{g}_t + \frac{L}{2} \|\theta_{t+1} - \theta_t\|_2^2 \\ &= -\gamma_t \hat{\mathbf{g}}_t{}^T \mathbf{g}_t + \frac{\gamma_t^2 L}{2} \|\hat{\mathbf{g}}_t\|_2^2 \end{aligned} \tag{16}$$

Take the expectation over θ_t and use the assumption 3

$$\mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)|\theta_t] = -\gamma_t \|\mathbf{g}_t\|_2^2 + \frac{\gamma_t^2 L}{2} (\|\mathbf{g}_t\|_2^2 + \sigma^2) \tag{17}$$

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

$$\begin{aligned} \mathbb{E}[\mathcal{L}(\theta_{t+1}) - \mathcal{L}(\theta_t)|\theta_t] &= -\frac{1}{L\sqrt{N}} \|\mathbf{g}_t\|_2^2 + \frac{1}{2LN} (\|\mathbf{g}_t\|_2^2 + \sigma^2) \\ &\leq -\frac{1}{2L\sqrt{N}} \|\mathbf{g}_t\|_2^2 + \frac{\sigma^2}{2LN} \end{aligned} \tag{18}$$

□

Convergence of SGD

Proof.

Sum the inequality of N steps together and use assumption 1

$$\begin{aligned}\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) &\geq \mathcal{L}(\theta_0) - \mathcal{L}(\theta_N) \\ &= \mathbb{E}\left[\sum_{t=0}^{N-1} (\mathcal{L}(\theta_t) - \mathcal{L}(\theta_{t+1}))\right] \\ &\geq \frac{1}{2L}\mathbb{E}\left[\sum_{t=0}^{N-1} (\frac{\|\mathbf{g}_t\|_2^2}{\sqrt{N}} - \frac{\sigma^2}{N})\right]\end{aligned} \tag{19}$$

Rearrange the inequality, we have the following

$$\mathbb{E}\left[\frac{1}{N}\sum_{t=0}^{N-1} \|\mathbf{g}_t\|_2^2\right] \leq \frac{1}{\sqrt{N}}[2L(\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*) + \sigma^2)] \tag{20}$$

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. □

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 high gradients or slow convergence in ones of low gradients.

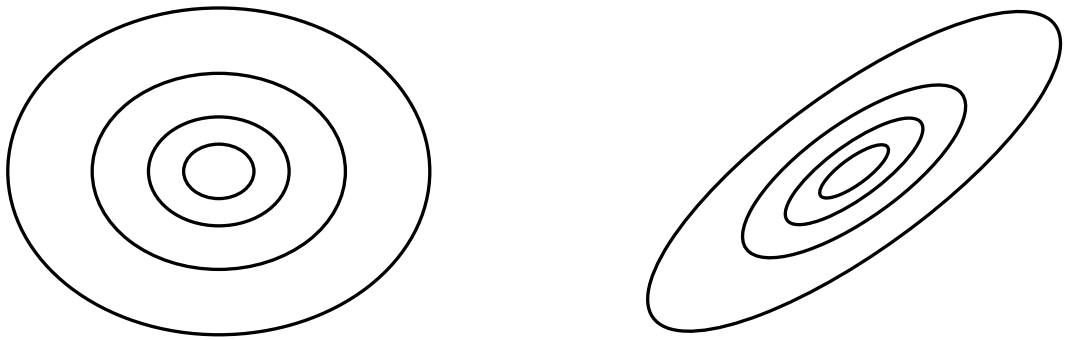


Figure: Examples of well-conditioned curvature (left) and ill-conditioned curvature (right).

AdaGrad [?]

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 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 [?]

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 minibatch gradient $\hat{\mathbf{g}}_t$
   update $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}}_t \odot \hat{\mathbf{g}}_t$
   update $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$

Remark

- AdaGrad uses different local learning rate for different parameter dimensions.
- The local learning rate decreases monotonically in AdaGrad, this might cause early stop.
- AdaGrad is particularly suitable for the sparse gradient cases.

RMSProp [?]

RMSProp
Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, decaying parameter τ
1. initialize θ_0, $\mathbf{r} \leftarrow \mathbf{0}$ **2.** **For** t = 0, 1,... ,N-1: obtain 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 $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$

RMSProp [?]

RMSProp
Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, decaying parameter τ
1. initialize θ_0, $\mathbf{r} \leftarrow \mathbf{0}$ **2.** **For** t = 0, 1,... ,N-1: obtain 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 $\theta_{t+1} \leftarrow \theta_t - \frac{\gamma_t}{\delta + \sqrt{\mathbf{r}}} \hat{\mathbf{g}}_t$

Remark

• Adapt the learning rate devisor by accumulating an exponentially decaying average of the gradient.

Adam [?]

Adam

Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, first order decaying paramter 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 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 $\theta_{t+1} \leftarrow \theta_t - \gamma_t \frac{\hat{\mathbf{m}}_1}{\delta + \sqrt{\hat{\mathbf{m}}_2}}$

Adam [?]

Adam

Input: global learning rate $\{\gamma\}_{t=0}^{N-1}$, damping coefficient δ, first order decaying paramter 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 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 $\theta_{t+1} \leftarrow \theta_t - \gamma_t \frac{\hat{\mathbf{m}}_1}{\delta + \sqrt{\hat{\mathbf{m}}_2}}$

Remark

• Adam can be considered as the momentum version of RMSProp but with bias correction terms for the first and second moments.

Comparison

Adaptive learning method may converge fast and generalize worse.

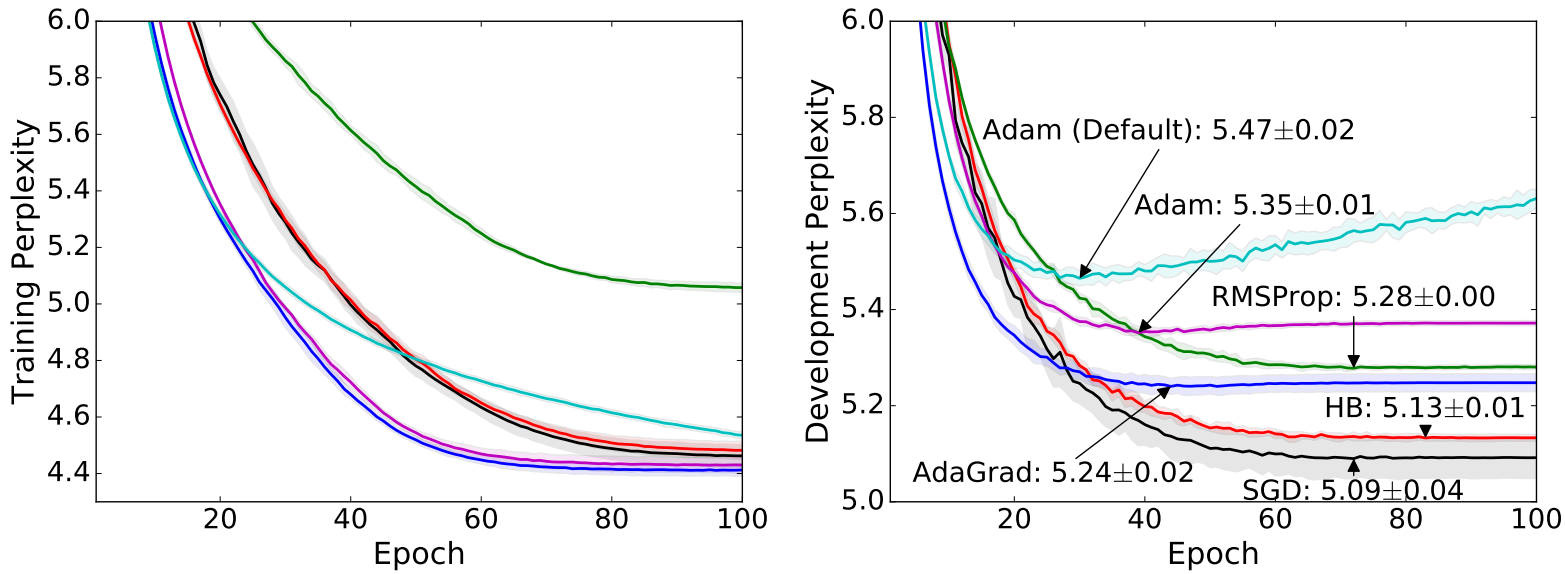


Figure: Performance of different optimizers in training and development set of a language modeling problem. [?]

References I