Deep Learning
CS60010

Abir Das
Assistant Professor
Computer Science and Engineering Department
Indian Institute of Technology Kharagpur

http://cse.iitkgp.ac.in/~adas/

Some Logistics Announcement

- Papers to be presented this Friday (Jan 10) – communicated via email
- 1. Team 1: "**Ashish Aswani** (19BM6JP53), Jhala Divyarajsinh Hanubha (19CS60R47)", - Paper - NeuronInspect: Detecting Backdoors in Neural Networks via Output Explanations 2.
- Team 2: "**Sugandha Ranjan** (19BM6JP21), K Nikhil (16CS30014)" - Paper - Deep Verifier Networks: Verification of Deep Discriminative Models with Deep Generative Models3.
- Team 3: "Vinay Singh (15CS30038), **Prateek Kukreja** (19EC65R15)" - Paper - Adversarial Examples Improve Image Recognition4.
- Team 4: "Anirban Saha (19CS60R50), Yash Gupta (19EC65R16)" - Paper - Efficient Saliency Maps For Explainable AI

Some Logistics Announcement

- Send your presentations by 5 pm tomorrow to me (abir@cse.iitkgp.ac.in)

- There will be surprise quizzes throughout the semester
  - Marks will be added towards grade (Exact mechanism to be decided later)
  - They will be 10-15 minute quizzes at the start of a lecture
  - So bring your notebooks and pens!!

Agenda

- Understand basics of Matrix/Vector Calculus and Optimization concepts to be used in the course.

- Understand different types or errors in learning

Resources

- "Deep Learning", I. Goodfellow, Y. Bengio, A. Courville. (Chapter 8)

Optimization and Deep Learning- Connections

- Deep learning (machine learning, in general) involves optimization in many contexts
- The goal is to find parameters $\boldsymbol{\theta}$ of a neural network that significantly reduce a cost function or objective function $J(\boldsymbol{\theta})$.
- Gradient based optimization is the most popular way for training Deep Neural Networks.
- There are other ways too, *e.g.*, evolutionary or derivative free optimization, but they come with issues particularly crucial for neural network training.
- Its easy to spend a semester on optimization. Thus, these few lectures will only be a scratch on a very small part of the surface.

Optimization and Deep Learning- Differences

- In learning we care about some performance measure P (*e.g.*, image classification accuracy, language translation accuracy etc.) on test set, but we minimize a different cost function $J(\boldsymbol{\theta})$ on training set, with the hope that doing so will improve P

- This is in contrast to pure optimization where minimizing $J(\boldsymbol{\theta})$ is a goal in itself

- Lets see what types of errors creep in as a result

Expected and Empirical Risk

- Training data is $\{\boldsymbol{x}^{(i)}, y^{(i)} : i = 1, \dots, N\}$ coming from probability distribution $\mathcal{D}(\boldsymbol{x}, y)$
- Let the neural network learns the output functions as $g^{\mathcal{D}}(\boldsymbol{x})$ and the loss is denoted as $l(g^{\mathcal{D}}(\boldsymbol{x}), y)$
- What we want to minimize is the expected risk

$$\mathrm{E}(g^{\mathcal{D}}) = \int l(g^{\mathcal{D}}(\boldsymbol{x}), y)\, d\mathcal{D}(\boldsymbol{x}, y) = \mathbb{E}\left(l(g^{\mathcal{D}}(\boldsymbol{x}), y)\right)$$

- If we could minimize this risk we would have got the true optimal function

$$f = \arg\min_{g^{\mathcal{D}}} \mathrm{E}(g^{\mathcal{D}})$$

- But we don't know the actual distribution that generates the data. So, what we actually minimize is the empirical risk

$$\mathrm{E_n}(g^{\mathcal{D}}) = \frac{1}{N}\sum_{i=1}^{N} l(g^{\mathcal{D}}(\boldsymbol{x_i}), y_i) = \mathbb{E}_n\left(l(g^{\mathcal{D}}(\boldsymbol{x}), y)\right)$$

- We choose a family $\mathcal{H}$ of candidate prediction functions and find the function that minimizes the empirical risk

$$g_{\mathrm{n}}^{\mathcal{D}} = \arg\min_{g^{\mathcal{D}} \in \mathcal{H}} \mathrm{E_n}(g^{\mathcal{D}})$$

- Since f may not be found in the family $\mathcal{H}$, we also define

$$g_{\mathcal{H}*}^{\mathcal{D}} = \arg\min_{g^{\mathcal{D}} \in \mathcal{H}} \mathrm{E}(g^{\mathcal{D}})$$

Sources of Error

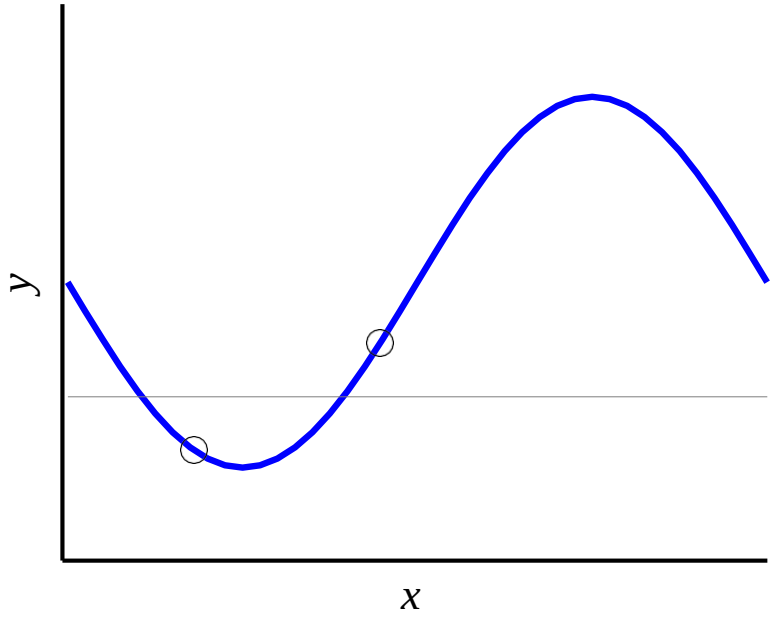
	Minimizes	Staying within	
f	expected risk	No constraints on function family	• True data distribution known • Family of functions exhaustive
$g^{\mathcal{D}}_{\mathcal{H}*}$	expected risk	family of functions $\mathcal{H}$	• True data distribution known • Family of functions not exhaustive
$g^{\mathcal{D}}_{\mathrm{n}}$	empirical risk	family of functions $\mathcal{H}$	• True data distribution not known • Family of functions not exhaustive

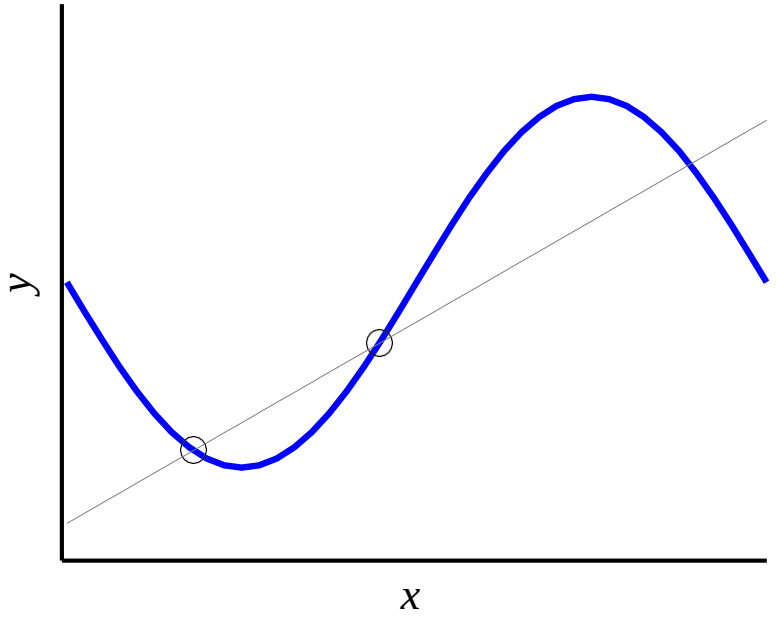
- True data generation procedure not known
- Family of functions [or hypothesis] to try is not exhaustive
- (And not to forget) we rely on a surrogate loss in place of the true classification error rate

A Simple Learning Problem

2 Data Points. 2 hypothesis sets:

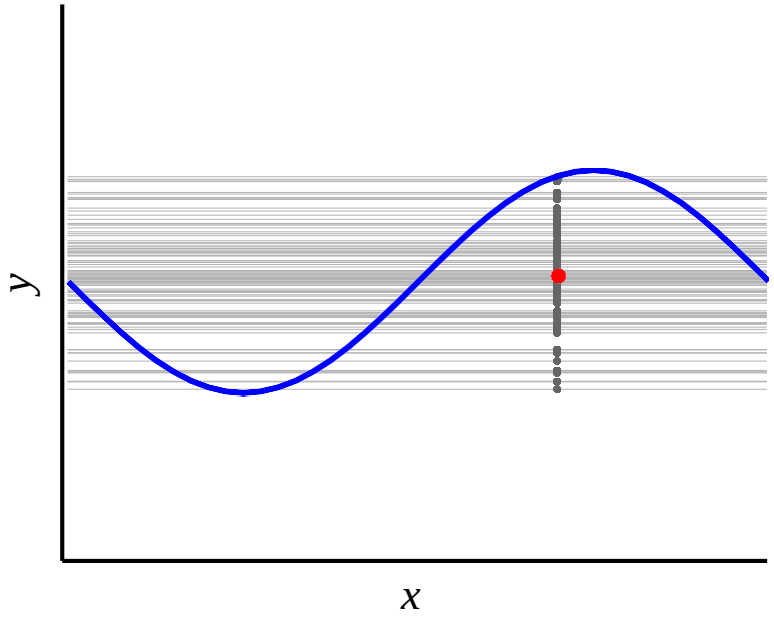
$$\mathcal{H}_0: \quad h(x) = b$$
$$\mathcal{H}_1: \quad h(x) = ax + b$$

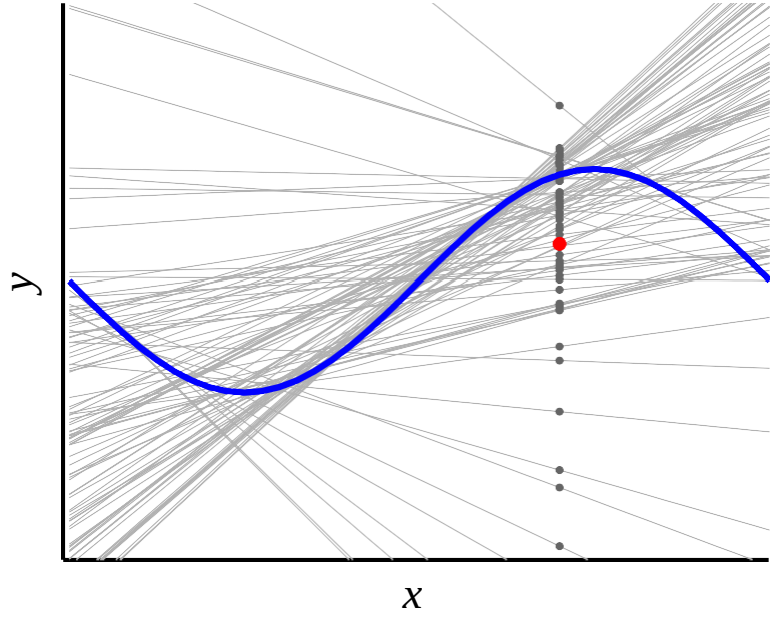




Slide courtesy: Malik Magdon-Ismail

Let's Repeat the Experiment Many Times

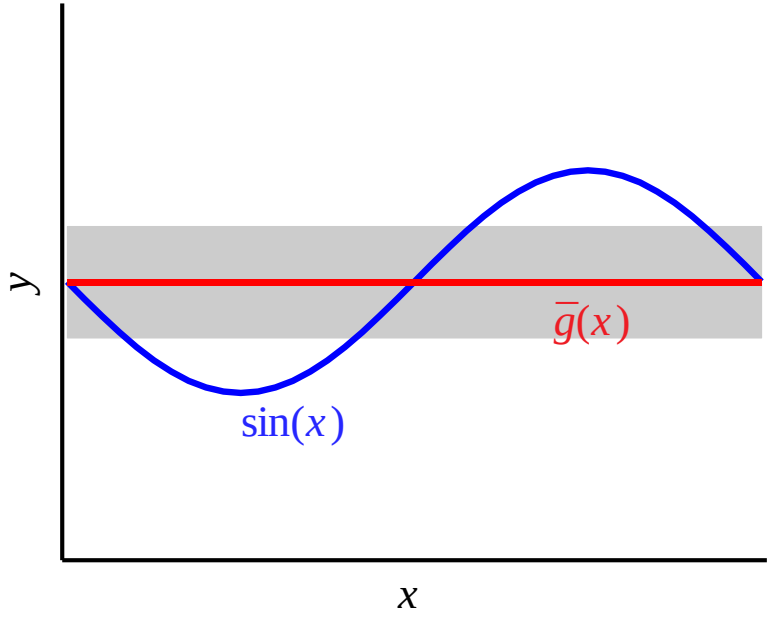




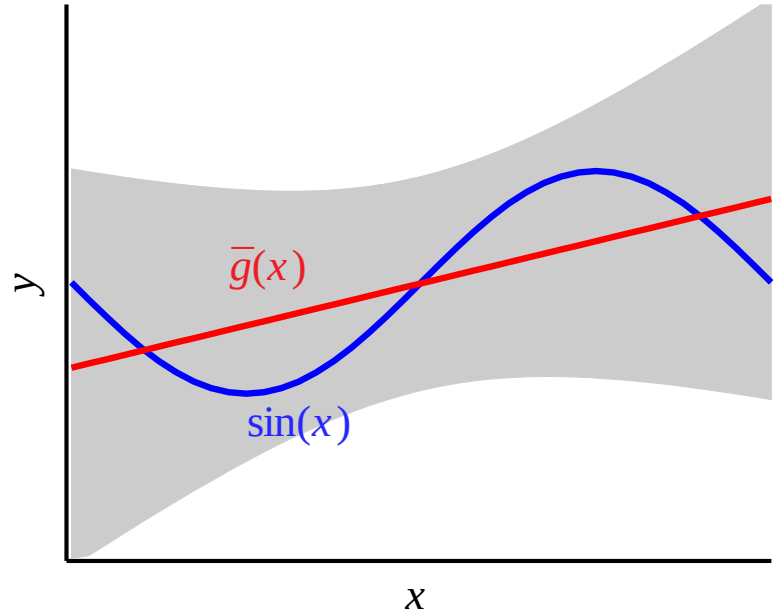
- For each data set $\mathcal{D}$ you get a different $g_n^{\mathcal{D}}$
- For a fixed $\boldsymbol{x}$, $g_n^{\mathcal{D}}(\boldsymbol{x})$ is random value, depending on $\mathcal{D}$

Slide courtesy: Malik Magdon-Ismail

What's Happening on Average



We can define



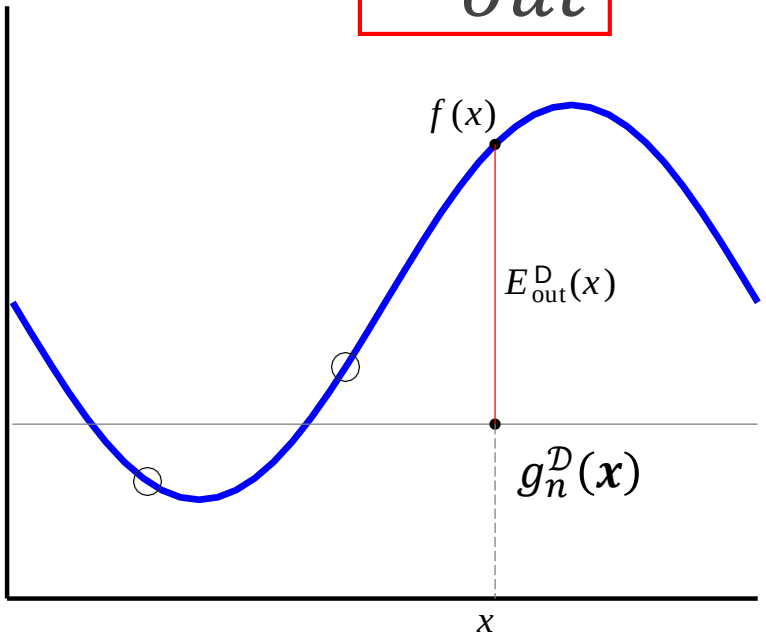
$g_n^{\mathcal{D}}(\boldsymbol{x})$ ⟵ Random value, depending on $\mathcal{D}$

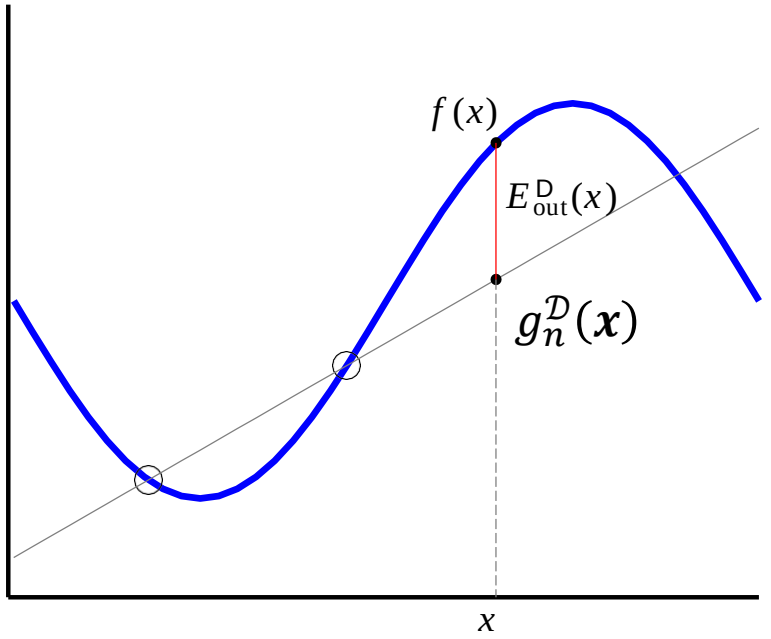
$$\bar{g}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})] = \frac{1}{K}\left(g_n^{\mathcal{D}_1}(\boldsymbol{x}) + g_n^{\mathcal{D}_2}(\boldsymbol{x}) + \cdots + g_n^{\mathcal{D}_K}(\boldsymbol{x})\right)$$ ⟵ Your average prediction on $\boldsymbol{x}$

$$\text{var}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}\left[\left(g_n^{\mathcal{D}}(\boldsymbol{x}) - \bar{g}(\boldsymbol{x})\right)^2\right] = \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2] - \bar{g}(\boldsymbol{x})^2$$ ⟵ How variable is your prediction

Slide courtesy: Malik Magdon-Ismail

E_{out} on Test Point $\boldsymbol{x}$ for Data $\mathcal{D}$





$$E_{out}^{\mathcal{D}}(\boldsymbol{x}) = \left(g_n^{\mathcal{D}}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2 \longleftarrow \textbf{Squared error}, \text{ a random value depending on } \mathcal{D}$$

$$E_{out}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}\left[E_{out}^{\mathcal{D}}(\boldsymbol{x})\right] = \mathbb{E}_{\mathcal{D}}\left[\left(g_n^{\mathcal{D}}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2\right] \longleftarrow \text{Expected value of the above random value}$$

Slide motivation: Malik Magdon-Ismail

The Bias-Variance Decomposition

$$E_{out}(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}\left[\left(g_n^{\mathcal{D}}(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2\right]$$

$$= \mathbb{E}_{\mathcal{D}}[f(\boldsymbol{x})^2 - 2f(\boldsymbol{x})g_n^{\mathcal{D}}(\boldsymbol{x}) + g_n^{\mathcal{D}}(\boldsymbol{x})^2]$$

$$= f(\boldsymbol{x})^2 - 2f(\boldsymbol{x})\mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})] + \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2]$$

$$= f(\boldsymbol{x})^2 - 2f(\boldsymbol{x})\overline{g}(\boldsymbol{x}) + \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2]$$

$$= f(\boldsymbol{x})^2 - \overline{g}(\boldsymbol{x})^2 + \overline{g}(\boldsymbol{x})^2 - 2f(\boldsymbol{x})\overline{g}(\boldsymbol{x}) + \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2]$$

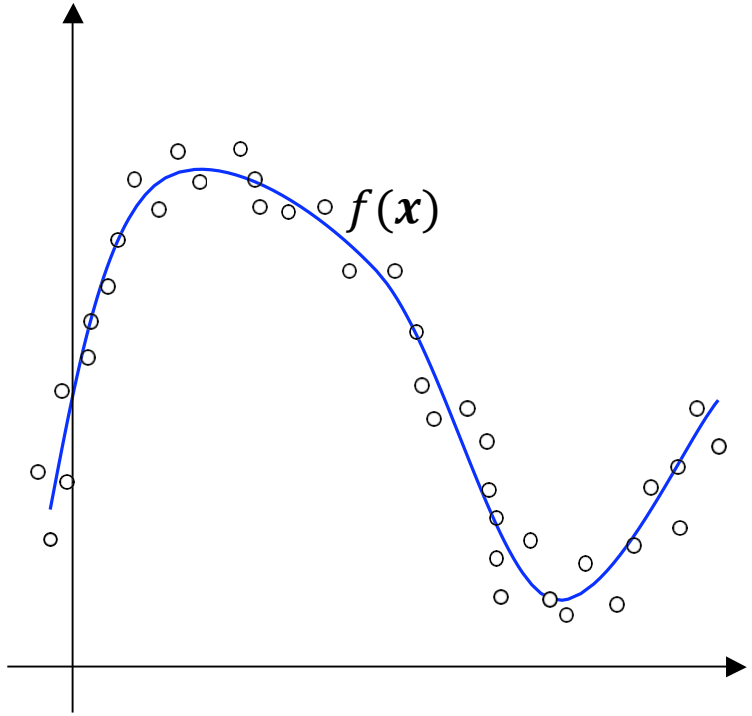
$$= \underbrace{\left(f(\boldsymbol{x}) - \overline{g}(\boldsymbol{x})\right)^2}_{\text{Bias}} + \underbrace{\mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2] - \overline{g}(\boldsymbol{x})^2}_{\text{Variance}}$$

$$\boxed{E_{out}(\boldsymbol{x}) = \text{Bias+Variance}}$$

Slide motivation: Malik Magdon-Ismail

Bias-Variance to Overfitting-Underfitting

- Suppose the true underlying function is $f(\boldsymbol{x})$.
- But the observations (data points) are noisy i.e., $f(\boldsymbol{x}) + \varepsilon$



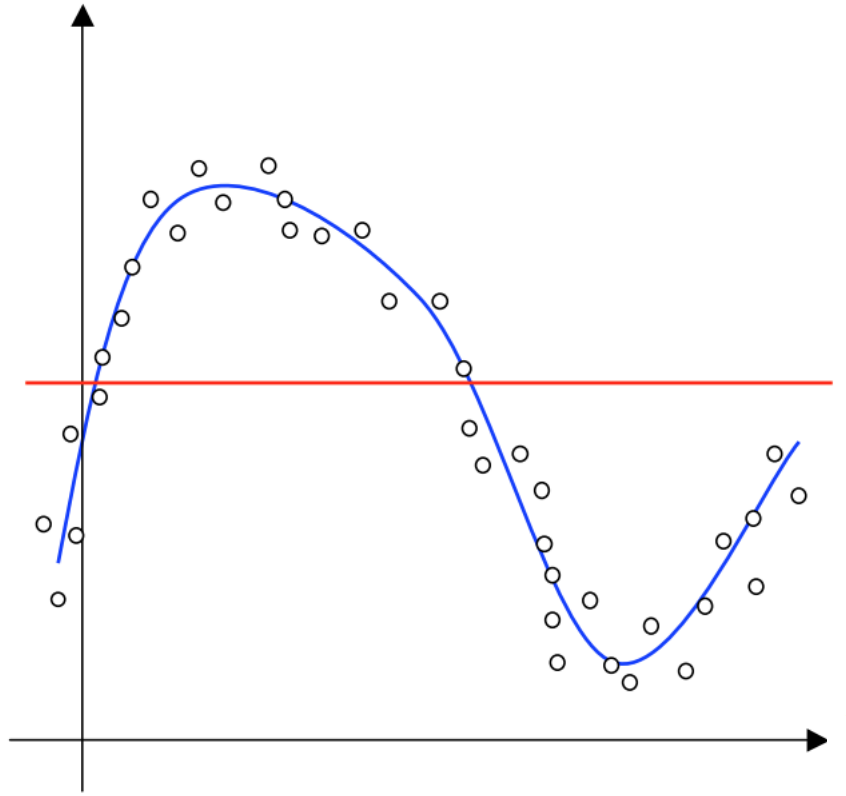
The Extreme Cases of Bias and Variance - Under-fitting

A good way to understand the concepts of bias and variance is by considering the two extreme cases of what a neural network might learn.

Suppose the neural network is lazy and just produces the same constant output whatever training data we give it, i.e. $g_n^{\mathcal{D}}(\boldsymbol{x})$=c, then

$$\begin{aligned} E_{out}(\boldsymbol{x}) &= \left(f(\boldsymbol{x}) - \overline{g}(\boldsymbol{x})\right)^2 + \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2] - \overline{g}(\boldsymbol{x})^2 \\ &= (f(\boldsymbol{x}) - c)^2 + \mathbb{E}_{\mathcal{D}}[\boldsymbol{c}^2] - \boldsymbol{c}^2 \\ &= (f(\boldsymbol{x}) - c)^2 + 0 \end{aligned}$$

In this case the variance term will be zero, but the bias will be large, because the network has made no attempt to fit the data. We say we have extreme under-fitting



Ignore the data ⇒
Big approximation error (high bias)
No variation between data sets (no variance)

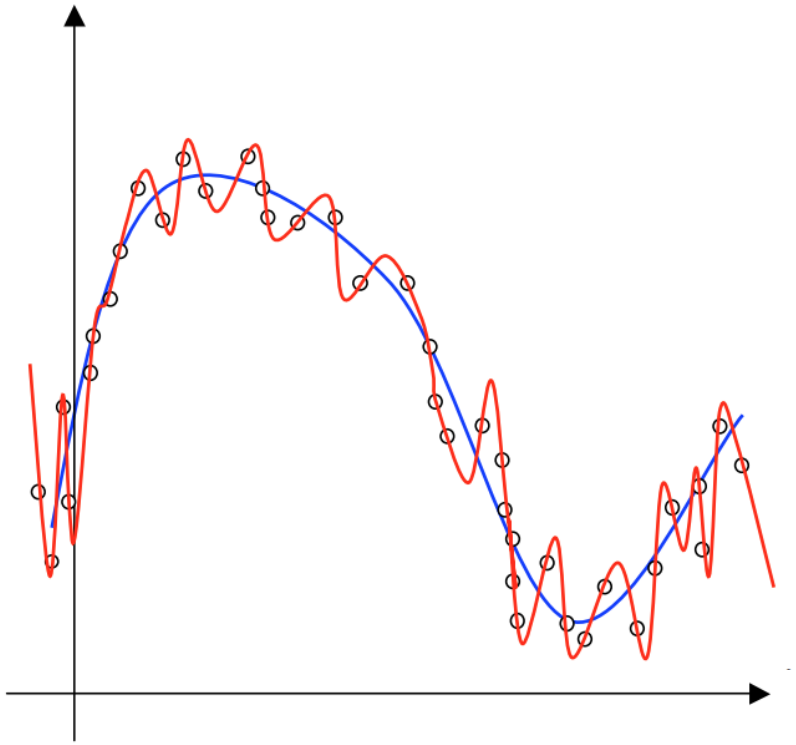
Slide motivation: John A. Bullinaria

The Extreme Cases of Bias and Variance - Over-fitting

On the other hand, suppose the neural network is very hard working and makes sure that it exactly fits every data point, i.e. $g_n^{\mathcal{D}}(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon$, then, $\overline{g}(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\mathcal{D}}}[g_n^{\mathcal{D}}(\boldsymbol{x})] = \mathbb{E}_{\boldsymbol{\mathcal{D}}}[f(\boldsymbol{x}) + \varepsilon] = \mathbb{E}_{\boldsymbol{\mathcal{D}}}[f(\boldsymbol{x})] + 0 = f(\boldsymbol{x})$

$$
\begin{aligned}
E_{out}(\boldsymbol{x}) &= \left(f(\boldsymbol{x}) - \overline{g}(\boldsymbol{x})\right)^2 + \mathbb{E}_{\mathcal{D}}[g_n^{\mathcal{D}}(\boldsymbol{x})^2] - \overline{g}(\boldsymbol{x})^2 \\
&= \left(f(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2 + \mathbb{E}_{\mathcal{D}}\left[\left(g_n^{\mathcal{D}}(\boldsymbol{x}) - \overline{g}(\boldsymbol{x})\right)^2\right] \\
&= 0 + \mathbb{E}_{\mathcal{D}}\left[\left(f(\boldsymbol{x}) + \varepsilon - f(\boldsymbol{x})\right)^2\right] \\
&= 0 + \mathbb{E}_{\mathcal{D}}[\boldsymbol{\varepsilon}^2]
\end{aligned}
$$

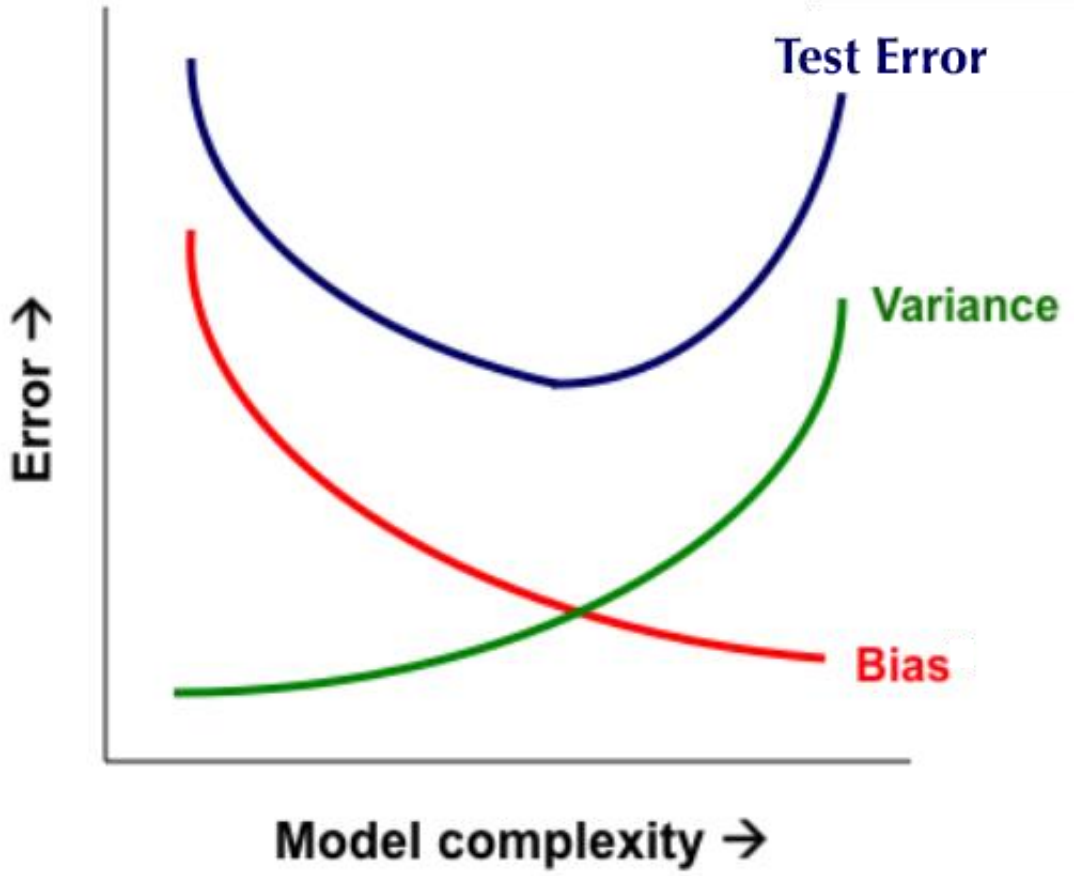
In this case the bias term will be zero, but the variance is the square of the noise on the data, which could be substantial. In this case we say we have extreme over-fitting.

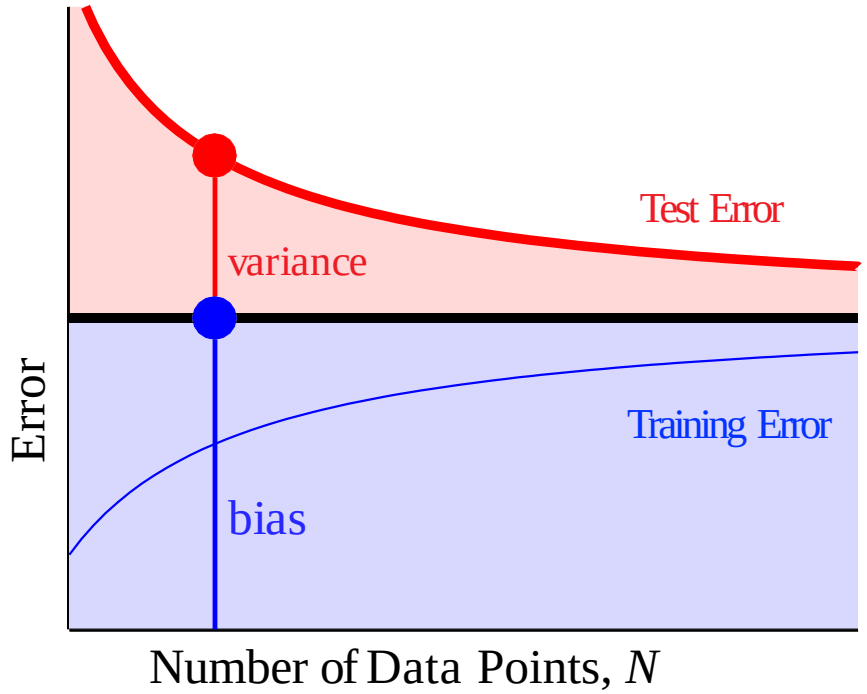


Fit every data point ⇒
No approximation error (zero bias)
Variation between data sets (high variance)

Slide motivation: John A. Bullinaria

Bias-Variance Trade-off





Some Logistics Announcement

- Office hours will be 5-6 pm on Thursdays instead of 4-5 pm on Thursdays

Vector/Matrix Calculus

- If $f(\boldsymbol{x}) \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$, then $\nabla_{\boldsymbol{x}} f \triangleq \left[\frac{\partial f}{\partial x_1} \quad \frac{\partial f}{\partial x_2} \quad \cdots \quad \frac{\partial f}{\partial x_n}\right]^T$ is called the gradient of $f(\boldsymbol{x})$
- If $f(\boldsymbol{x}) = [f_1(x) \quad f_2(x) \quad \cdots \quad f_m(x)]^T \in \mathbb{R}^m$ and $\boldsymbol{x} \in \mathbb{R}^n$, then
- $$\nabla_{\boldsymbol{x}} f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$
- is called "Jacobian matrix" of $f(\boldsymbol{x})$ w.r.t. $\boldsymbol{x}$

Vector/Matrix Calculus

- If $f(\boldsymbol{x}) \in \mathbb{R}$ and $\boldsymbol{x} \in \mathbb{R}^n$, then

- $$\nabla_{\boldsymbol{x}}^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_1 x_n} & \frac{\partial^2 f}{\partial x_2 x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- is called the "Hessian matrix" of $f(\boldsymbol{x})$ w.r.t. $\boldsymbol{x}$

Vector/Matrix Calculus

- Some standard results:
- $\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{b}^T \boldsymbol{x} = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{b} = \boldsymbol{b}$
- $\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}$
- $\frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = \left(\boldsymbol{A} + \boldsymbol{A}^T\right)\boldsymbol{x}$; if $\boldsymbol{A} = \boldsymbol{A}^T, \frac{\partial}{\partial \boldsymbol{x}} \frac{1}{2} \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x}$
- Product rule: $\boldsymbol{u} \in \mathbb{R}^m, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{x} \in \mathbb{R}^n$
- $$\left[\frac{\partial \boldsymbol{u}^T \boldsymbol{v}}{\partial \boldsymbol{x}}\right]^T_{1\times n} = [\boldsymbol{u}^T]_{1\times m} \left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{x}}\right]_{m\times n} + [\boldsymbol{v}^T]_{1\times m} \left[\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}}\right]_{m\times n}$$

Vector/Matrix Calculus

- Some standard results:

- If $F(f(\boldsymbol{x})) \in \mathbb{R}, f(\boldsymbol{x}) \in \mathbb{R}, \boldsymbol{x} \in \mathbb{R}^n$, then
- $$\left[\frac{\partial F}{\partial \boldsymbol{x}}\right]_{n\times 1} = \left[\frac{\partial f}{\partial \boldsymbol{x}}\right]_{n\times 1} \left[\frac{\partial F}{\partial f}\right]_{1\times 1}$$

- If $F(\boldsymbol{f}(\boldsymbol{x})) \in \mathbb{R}, \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^m, \boldsymbol{x} \in \mathbb{R}^n$, then
- $$\left[\frac{\partial F}{\partial \boldsymbol{x}}\right]_{n\times 1} = \left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right]^T_{n\times m} \left[\frac{\partial F}{\partial \boldsymbol{f}}\right]_{m\times 1}$$

Vector/Matrix Calculus

- Derivatives of norms:

- For two norm of a vector

$$\frac{\partial}{\partial \boldsymbol{x}} ||\boldsymbol{x}||_2^2 = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{x}^T \boldsymbol{x} = 2\boldsymbol{x}$$

- For Frobenius norm of a matrix

$$\frac{\partial}{\partial \boldsymbol{X}} ||\boldsymbol{X}||_F^2 = \frac{\partial}{\partial \boldsymbol{X}} tr(\boldsymbol{X}\boldsymbol{X}^T) = 2\boldsymbol{X}$$

Optimization Problem

- Problem Statement:

$$\min_{s.t.\ \boldsymbol{x}\in\chi} f(\boldsymbol{x})$$

- Problem statement of convex optimization:

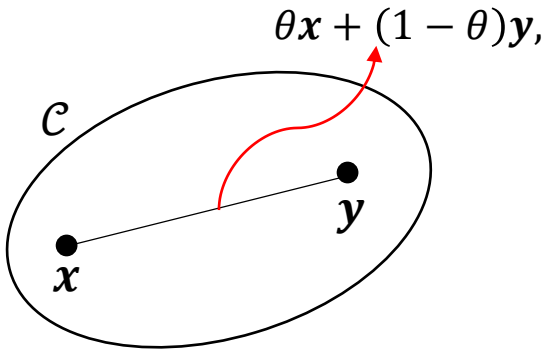
$$\min_{s.t.\ \boldsymbol{x}\in\chi} f(\boldsymbol{x})$$

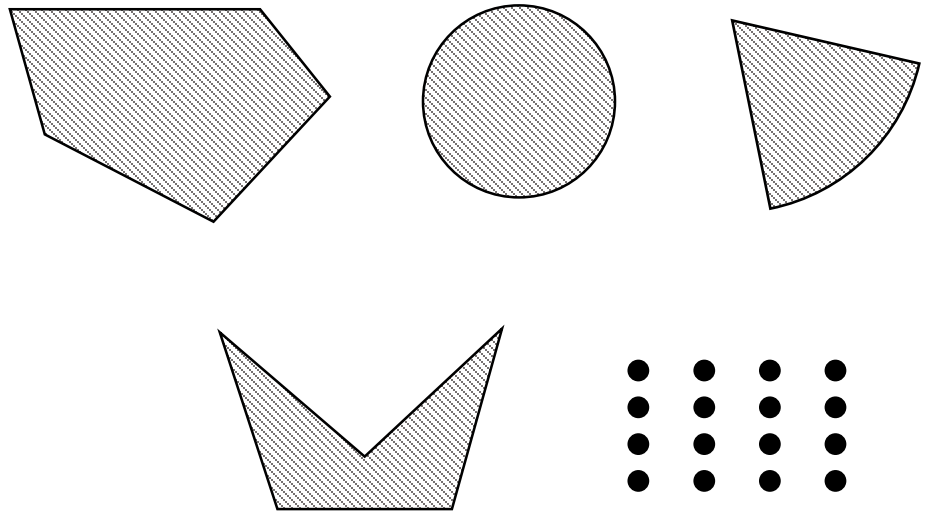
  - with $f\colon \mathbb{R}^n \to \mathbb{R}$ is a **convex function** and
  - χ is a **convex set**

Convex Sets and Functions

- **Convex Set**: A set $\mathcal{C} \subseteq \mathbb{R}^n$ is a convex set if for all $x, y \in \mathcal{C}$, the line segment connecting x and y is in $\mathcal{C}$, i.e.,

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}, \theta \in [0,1], \theta \boldsymbol{x} + (1-\theta)\boldsymbol{y} \in \mathcal{C}$$





Convex Sets and Functions

- **Convex Function**: A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function if its domain $dom(f)$ is a convex set and for all $x, y \in dom(f)$, and $\theta \in [0,1]$, we have

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y})$$

All norms are convex functions

$$\|\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}\| \leq \|\theta \boldsymbol{x}\| + \|(1-\theta)\boldsymbol{y}\|$$
$$= \theta \|\boldsymbol{x}\| + (1-\theta)\|\boldsymbol{y}\|$$

Alternative Definition of Convexity for differentiable functions

Theorem: (first order characterization) Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be a differentiable function whose dom(f) is convex. Then, *f* is convex *iff*

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \quad \forall \boldsymbol{x}, \boldsymbol{y} \in dom(f) \qquad \ldots(1)$$

Proof *(part 1)*: $Eq^n\ (1) \Rightarrow Convex$

Given $Eq^n\ (1)$: $f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle$

Let us consider $\boldsymbol{x}, \boldsymbol{y} \in dom(f)$ and $\theta \in [0,1]$

Let $\boldsymbol{z} = (1-\theta)\boldsymbol{x} + \theta\boldsymbol{y}$

Now, By $eq^n\ (1)$,

$f(\boldsymbol{x}) \geq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \rangle$ ... (2), taking $\boldsymbol{x}, \mathbf{z}$

$f(\boldsymbol{y}) \geq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{y} - \boldsymbol{z} \rangle$ ... (3), taking $\boldsymbol{y}, \mathbf{z}$

Multiplying $Eq^n\ (2)$ with $(1-\theta)$, we get,

$$(1-\theta)f(\boldsymbol{x}) \geq (1-\theta)f(\boldsymbol{z}) + (1-\theta)\langle \nabla f(\boldsymbol{z}), \boldsymbol{x}-\boldsymbol{z}\rangle \quad \ldots(4)$$

and $Eq^n\ (3)$ with θ

$$\theta f(\boldsymbol{y}) \geq \theta f(\boldsymbol{z}) + \theta\langle \nabla f(\boldsymbol{z}), \boldsymbol{y}-\boldsymbol{z}\rangle \quad \ldots(5)$$

Now combining $Eq^n\text{s}\ (4)\ and\ (5)$, we have,

$$(1-\theta)f(\boldsymbol{x}) + \theta f(\boldsymbol{y}) \geq (1-\theta)f(\boldsymbol{z}) + \theta f(\boldsymbol{z}) + (1-\theta)\langle \nabla f(\boldsymbol{z}), \boldsymbol{x}-\boldsymbol{z}\rangle + \theta\langle \nabla f(\boldsymbol{z}), \boldsymbol{y}-\boldsymbol{z}\rangle$$

$$(1-\theta)f(\boldsymbol{x})+\theta f(\boldsymbol{y}) \geq (1-\theta)f(\boldsymbol{z})+\theta f(\boldsymbol{z})+(1-\theta)\langle \nabla f(\boldsymbol{z}), \boldsymbol{x}-\boldsymbol{z}\rangle + \theta\langle \nabla f(\boldsymbol{z}), \boldsymbol{y}-\boldsymbol{z}\rangle$$

$$\Rightarrow (1-\theta)f(\boldsymbol{x})+\theta f(\boldsymbol{y}) \geq f(\boldsymbol{z}) \cancel{-\theta f(\boldsymbol{z})} \cancel{+\theta f(\boldsymbol{z})} + \langle \nabla f(\boldsymbol{z}), (1-\theta)(\boldsymbol{x}-\boldsymbol{z})\rangle + \langle \nabla f(\boldsymbol{z}), \theta(\boldsymbol{y}-\boldsymbol{z})\rangle$$

$$\Rightarrow (1-\theta)f(\boldsymbol{x})+\theta f(\boldsymbol{y}) \geq f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \underbrace{(1-\theta)(\boldsymbol{x}-\boldsymbol{z})+\theta(\boldsymbol{y}-\boldsymbol{z})}_{=0 \text{ (Why ?)}}\rangle$$

$$\left[\begin{aligned}(1-\theta)(\boldsymbol{x}-\boldsymbol{z})+\theta(y-z) &= (1-\theta)\boldsymbol{x}-(1-\theta)\boldsymbol{z}+\theta\boldsymbol{y}-\theta\boldsymbol{z} \\ &= (1-\theta)\boldsymbol{x}-\boldsymbol{z}+\theta\boldsymbol{z}+\theta\boldsymbol{y}-\theta\boldsymbol{z} \\ &= (1-\theta)\boldsymbol{x}+\theta\boldsymbol{y}-\boldsymbol{z} = \boldsymbol{z}-\boldsymbol{z} = \boldsymbol{0}\end{aligned}\right]$$

$$= f(\boldsymbol{z}) = f((1-\theta)\boldsymbol{x}+\theta\boldsymbol{y})$$

Proof *(part 2): Convex* $\Rightarrow Eq^n$ (1)

Suppose f is convex, i.e., $\forall \boldsymbol{x}, \boldsymbol{y} \in dom(f)$ and $\forall \theta \in [0,1]$,

$$f\big((1-\theta)\boldsymbol{x} + \theta\boldsymbol{y}\big) \leq (1-\theta)f(\boldsymbol{x}) + \theta f(\boldsymbol{y})$$

Equivalently,

$$f\left(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})\right) \leq f(\boldsymbol{x}) + \theta(f(\boldsymbol{y}) - f(\boldsymbol{x}))$$

$$\Rightarrow f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \frac{f\big(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})\big) - f(\boldsymbol{x})}{\theta}$$

By taking the limit as $\theta \to 0$ on both sides and using the definition of derivative, we obtain,

$$f(\boldsymbol{y}) - f(\boldsymbol{x}) \geq \lim_{\theta \to 0} \frac{f\big(\boldsymbol{x} + \theta(\boldsymbol{y} - \boldsymbol{x})\big) - f(\boldsymbol{x})}{\theta} = \langle \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle$$

(*How?*)

$$g = \lim_{\theta \to 0} \frac{f\big(x + \theta(y - x)\big) - f(x)}{\theta}$$

[Let us see in case of 1-D x, y.]

$$g = \lim_{\theta \to 0} \frac{f\big(x + \theta(y - x)\big) - f(x)}{\theta(y - x)} (y - x)$$

[Multiplying numerator and denominator by $(y - x)$]

Let, $\theta(y - x) = h$, As $\theta \to 0, h \to 0$

$$\therefore g = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h} (y - x) = f'(x)(y - x)$$

Gradient Descent

$$\min_{\boldsymbol{w},\boldsymbol{b}} \text{loss function}$$

More formally, $\min_{\theta} J(\theta)$

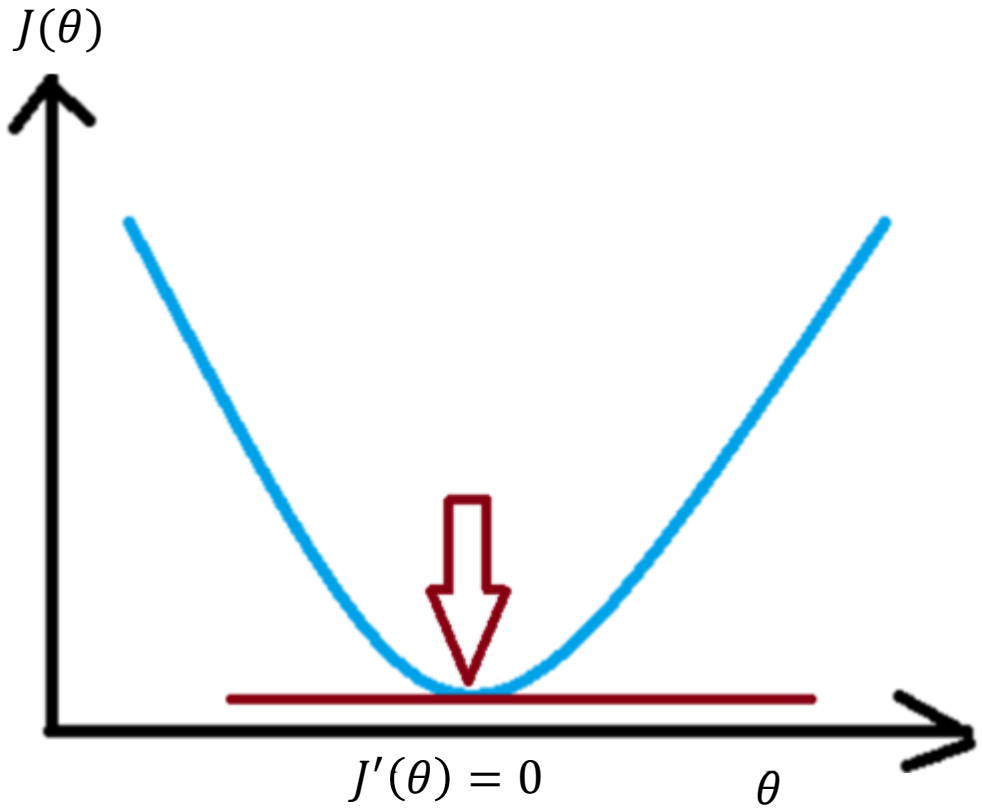
For scalar θ, the condition is $J'(\theta) = \frac{\partial J}{\partial \theta} = 0$

For higher dimensional $\boldsymbol{\theta}$, the condition boils down to

$$J'(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} J = \mathbf{0}$$

$$\Rightarrow \left[\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \ldots, \frac{\partial J}{\partial \theta_N}\right] = \mathbf{0}$$

$$\Rightarrow \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_N} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



One Way to Find Minima – Gradient Descent

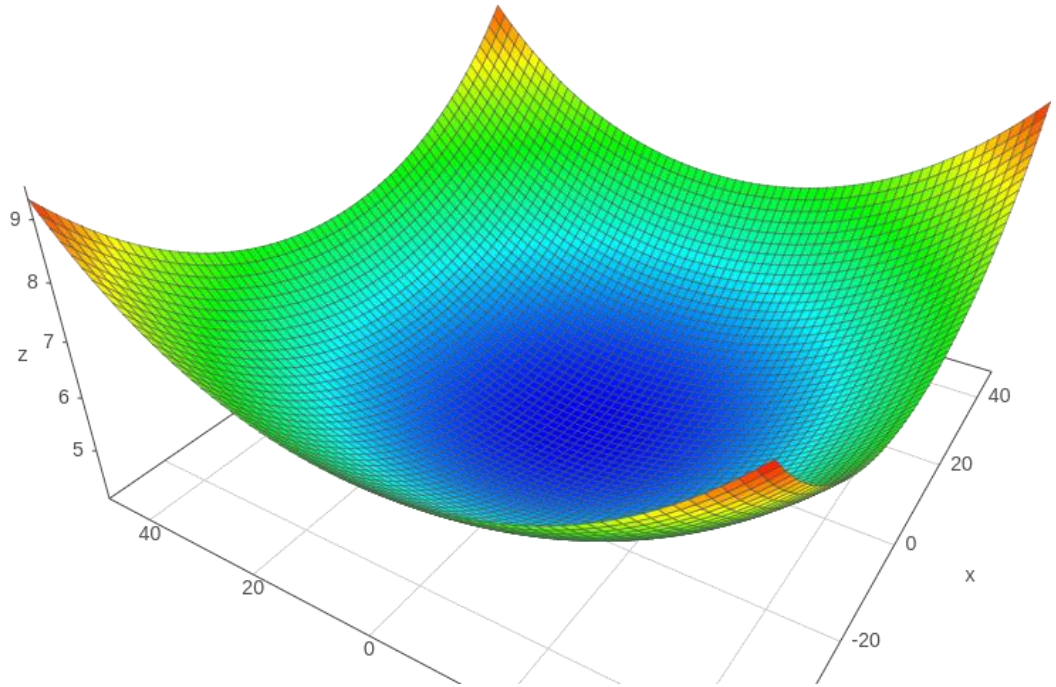
This is helpful but not always useful.

For example $J(\theta) = \log \sum_{i=1}^{m} e^{(\boldsymbol{a_i^T \theta} + b_i)}$ is a convex function with clear minima, but finding analytical solution is not easy.

$$\nabla_{\boldsymbol{\theta}} J = \mathbf{0}$$

$$\Rightarrow \frac{1}{\sum_{i=1}^{m} e^{(\boldsymbol{a_i^T \theta} + b_i)}} \sum_{i=1}^{m} e^{(\boldsymbol{a_i^T \theta} + b_i)} \boldsymbol{a_i} = 0$$

$$\Rightarrow \sum_{i=1}^{m} e^{(\boldsymbol{a_i^T \theta} + b_i)} \boldsymbol{a_i} = 0$$



So, a numerical iterative solution is sought for.

One Way to Find Minima – Gradient Descent

Start with an initial guess $\boldsymbol{\theta}^0$

Repeatedly update $\boldsymbol{\theta}$ by taking a small step: $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} + \eta\Delta\boldsymbol{\theta}$ … (1)
so that $J(\boldsymbol{\theta})$ gets smaller with each update i.e., $J(\boldsymbol{\theta}^k) \leq J(\boldsymbol{\theta}^{k-1})$ … (2)

(1) implies, $J(\boldsymbol{\theta}^k) = J(\boldsymbol{\theta}^{k-1} + \eta\Delta\boldsymbol{\theta})$

$= J(\boldsymbol{\theta}^{k-1}) + \eta(\Delta\boldsymbol{\theta})^T J'(\boldsymbol{\theta}^{k-1}) + h.o.t.$ [Using Taylor series expansion]

$\approx J(\boldsymbol{\theta}^{k-1}) + \eta(\Delta\boldsymbol{\theta})^T J'(\boldsymbol{\theta}^{k-1})$ [Neglecting $h.o.t.$] … (3)

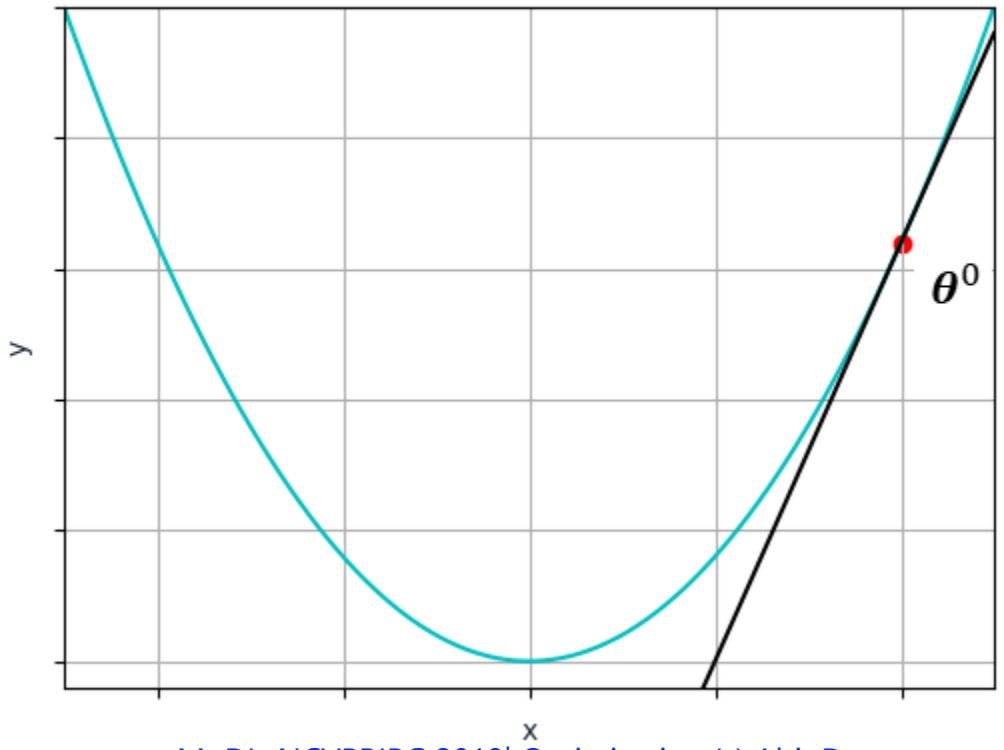
Combining (2) and (3),
$\eta(\Delta\boldsymbol{\theta})^T J'(\boldsymbol{\theta}^{k-1}) \leq 0\ i.e., (\Delta\boldsymbol{\theta})^T J'(\boldsymbol{\theta}^{k-1}) \leq 0$ [as η is positive]

So, for $\boldsymbol{\theta}$ to minimize $J(\boldsymbol{\theta}), i.e.,$ to satisfy (2) we have to choose some $\Delta\boldsymbol{\theta}$ that is negative when a dot product of it is done with the gradient $J'(\boldsymbol{\theta}^{k-1})$.
Then why not choose, $\Delta\boldsymbol{\theta} = -J'(\boldsymbol{\theta}^{k-1})$!!
Then, $(\Delta\boldsymbol{\theta})^T J'(\boldsymbol{\theta}^{k-1}) = -||J'(\boldsymbol{\theta}^{k-1})||^2$ surely is a negative quantity and satisfies the condition.

Gradient Descent

Start with an initial guess $\boldsymbol{\theta}^0$

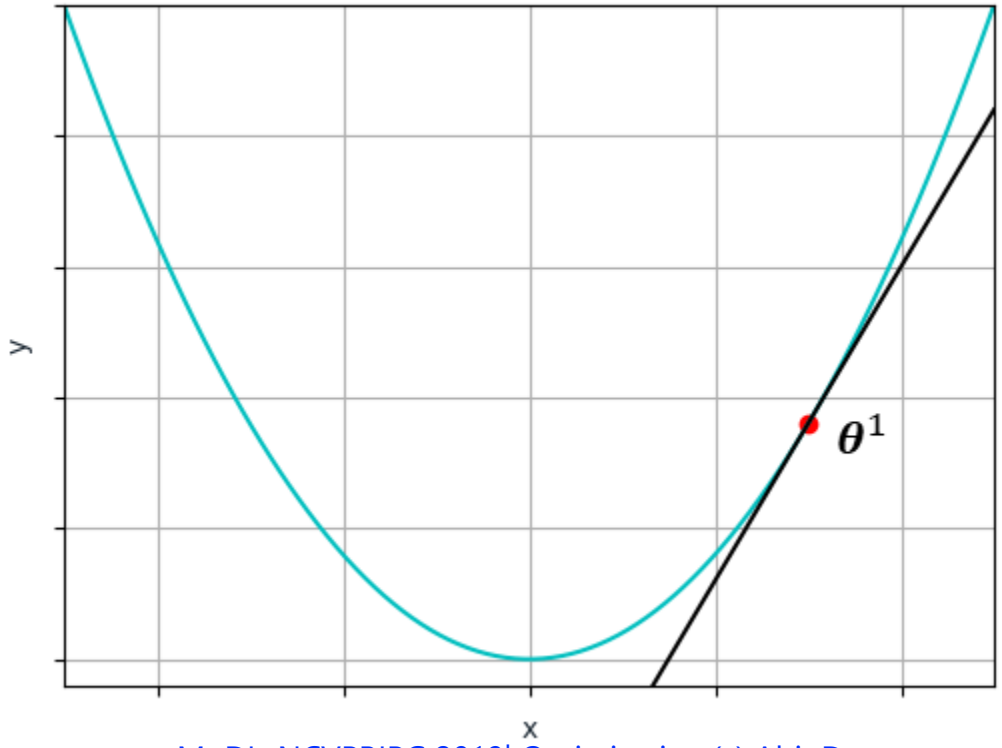
Repeatedly update $\boldsymbol{\theta}$ by taking a small step: $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \eta J'(\boldsymbol{\theta}^{k-1})$ until convergence ($J'(\boldsymbol{\theta}^{k-1})$ is very small)



Gradient Descent

Start with an initial guess $\boldsymbol{\theta}^0$

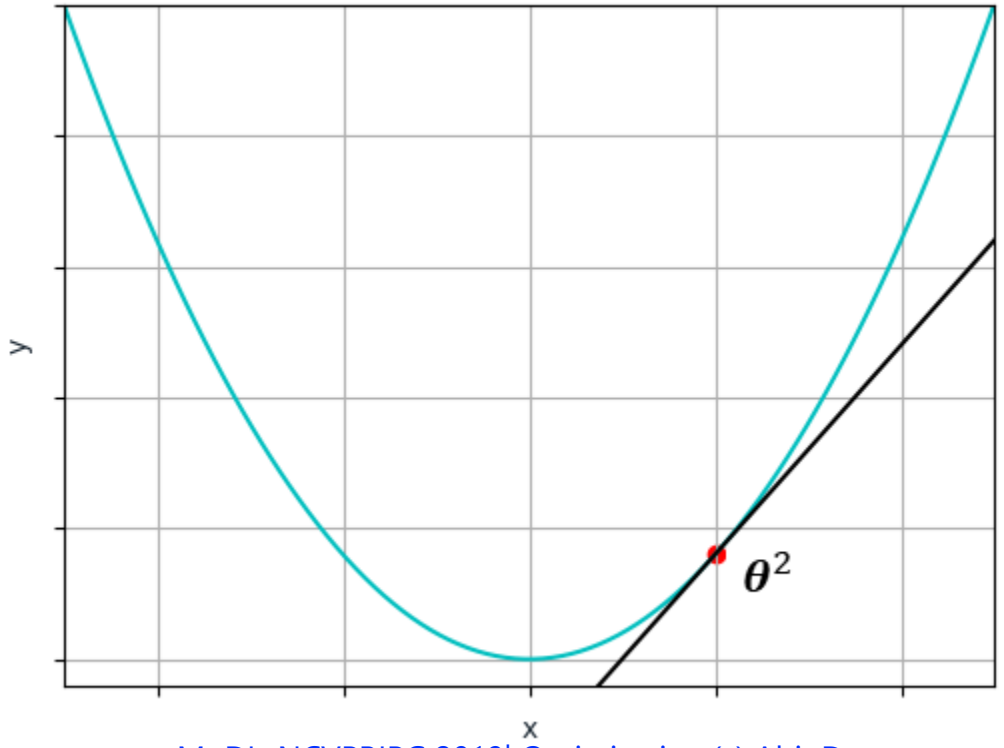
Repeatedly update $\boldsymbol{\theta}$ by taking a small step: $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \eta J'\left(\boldsymbol{\theta}^{k-1}\right)$ until convergence ($J'\left(\boldsymbol{\theta}^{k-1}\right)$ is very small)



Gradient Descent

Start with an initial guess $\boldsymbol{\theta}^0$

Repeatedly update $\boldsymbol{\theta}$ by taking a small step: $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \eta J'\left(\boldsymbol{\theta}^{k-1}\right)$ until convergence ($J'\left(\boldsymbol{\theta}^{k-1}\right)$ is very small)

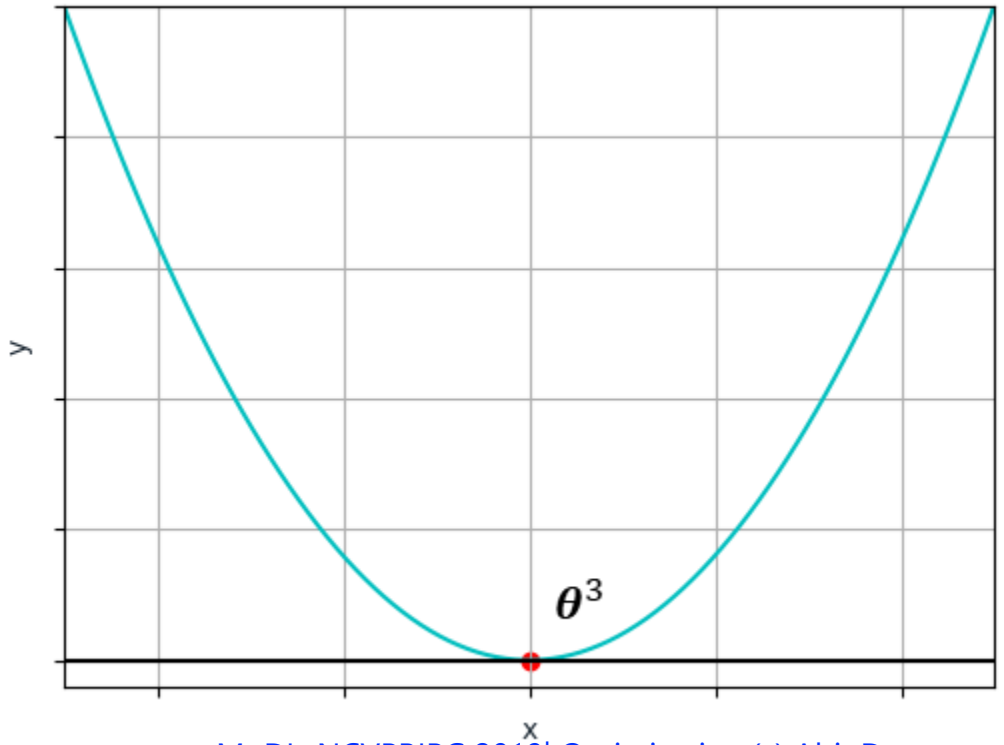


Gradient Descent

Start with an initial guess $\boldsymbol{\theta}^0$

Repeatedly update $\boldsymbol{\theta}$ by taking a small step: $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{k-1} - \eta J'(\boldsymbol{\theta}^{k-1})$ until convergence ($J'(\boldsymbol{\theta}^{k-1})$ is very small)

Remember, in Neural Networks, the loss is computed by averaging the losses for all examples



But Imagine This

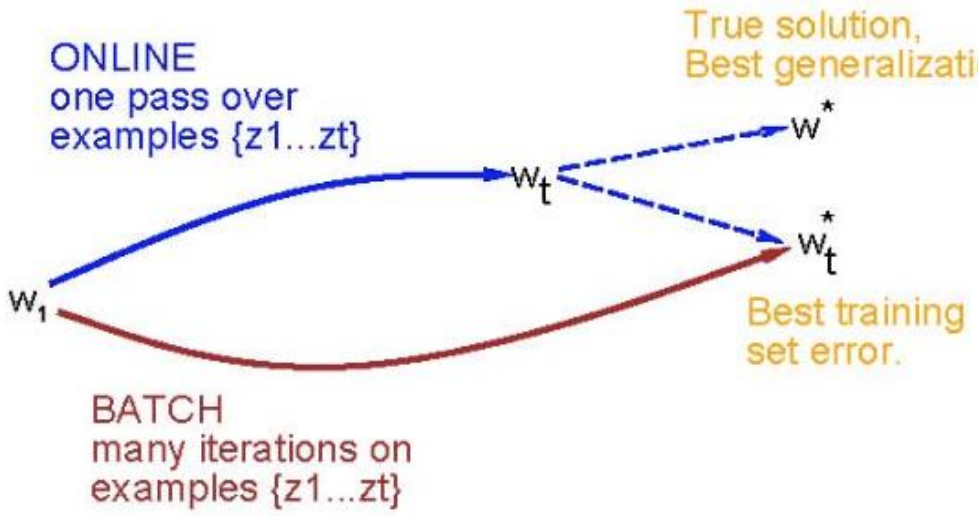
Speed Advantage.

Assume training set contains
10 copies of the 100 same examples.

- Batch — Blindly computes redundant gradients.
  1 epoch on large set ≡ 1 epochs on small set.
- Online — Take advantage of redundancy.
  1 epoch on large set ≡ 10 epochs on small set.

In practice, stochastic gradient
can be orders of magnitude faster.

Online vs. Batch



Adapted from Olivier Bousquet's invited talk at NeurIPS 2018 after winning Test of Time Award for NIPS 2007 paper: "The Trade-Offs of Large Scale Learning" by Leon Bottou and Olivier Bousquet. Link of the talk.

Batch, Stochastic and Minibatch

- Optimization algorithms that use the entire training set to compute the gradient are called batch or deterministic gradient methods. Ones that use a single training example for that task are called stochastic or online gradient methods

- Most of the algorithms we use for deep learning fall somewhere in between!

- These are called minibatch or minibatch stochastic methods

Batch, Stochastic and Mini-batch Stochastic Gradient Descent

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**

2: Compute gradient estimate over N examples:

3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$

4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$

5: **end while**

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**

2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

3: Compute gradient estimate:

4: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$

5: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$

6: **end while**

Mini-batch

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.

Require: Initial parameter $\boldsymbol{\theta}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

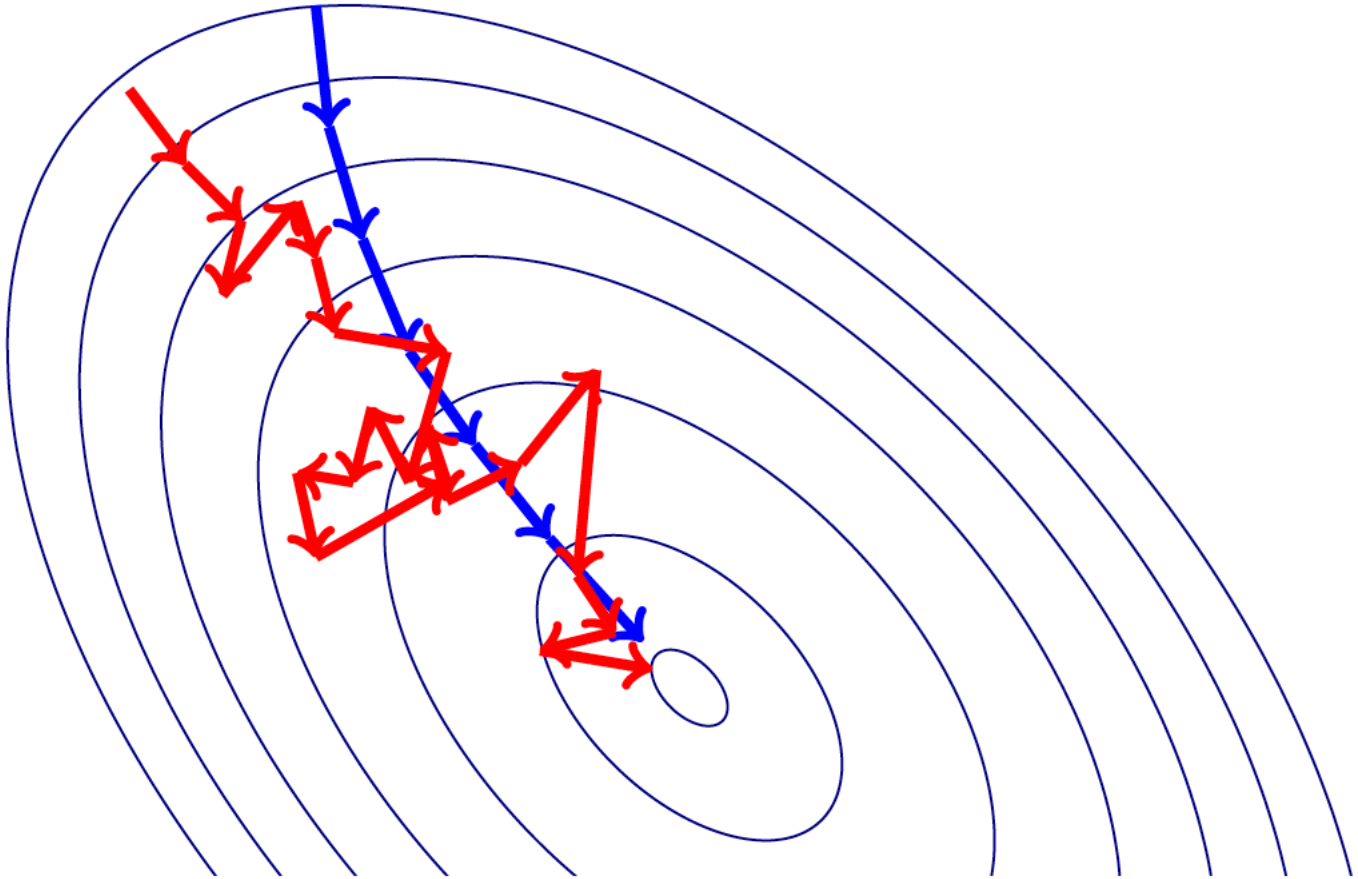
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\hat{\boldsymbol{g}}$

end while

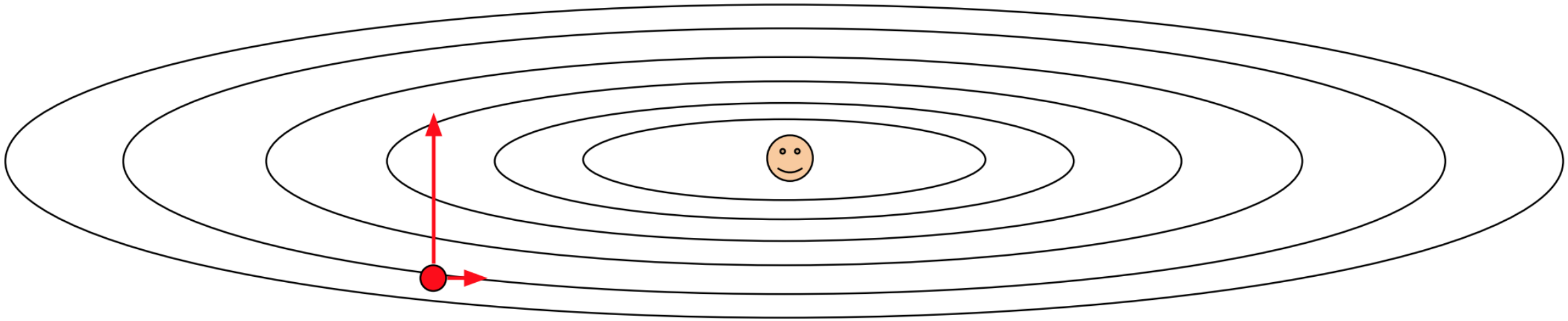
Images courtesy: Shubhendu Trivedi et. Al, Goodfellow et. al..

Batch and Stochastic Gradient Descent



Images courtesy: Shubhendu Trivedi et. al.

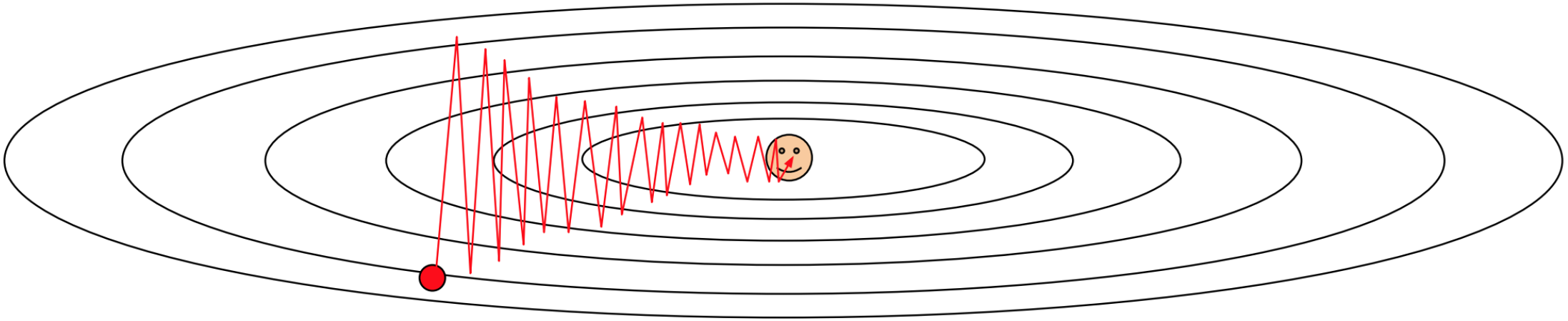
Suppose loss function is steep vertically but shallow horizontally:



Q: What is the trajectory along which we converge towards the minimum with SGD?

Images courtesy: Karpathy et. al.

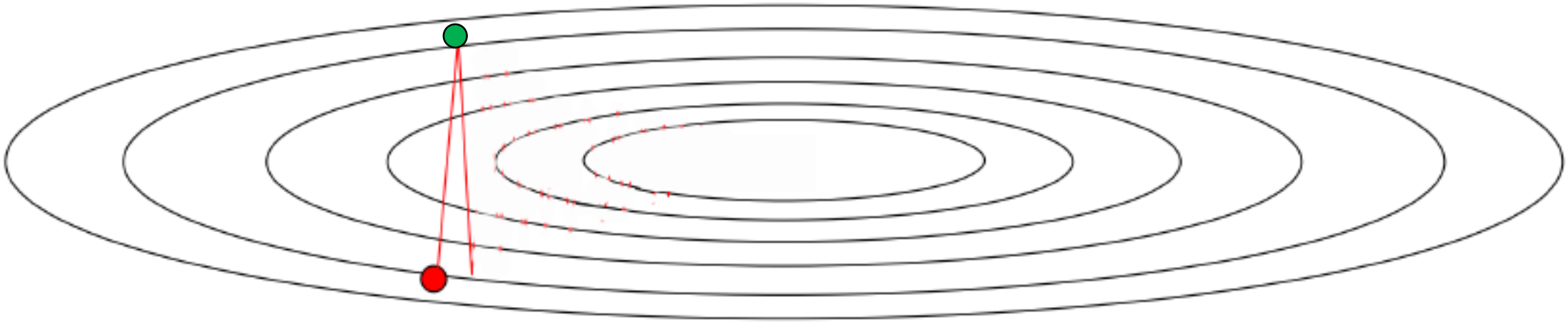
Suppose loss function is steep vertically but shallow horizontally:



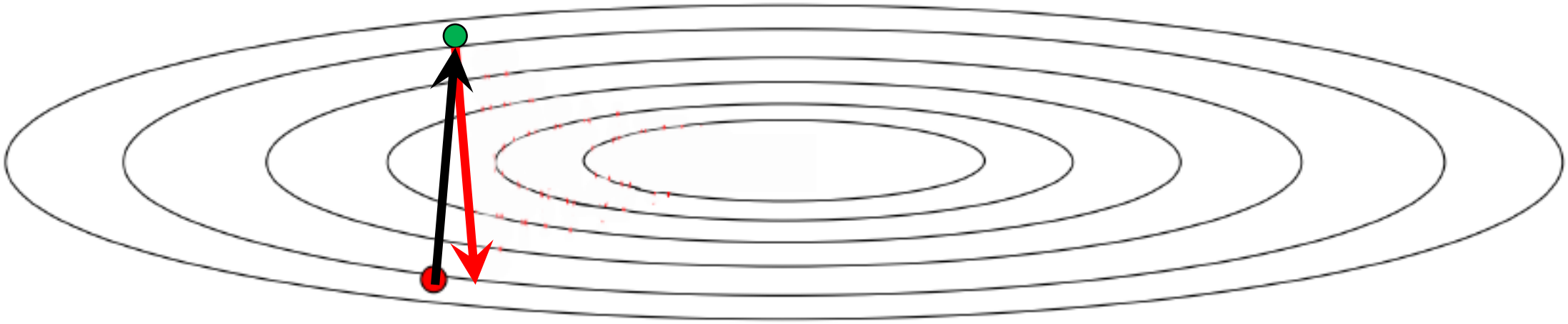
Q: What is the trajectory along which we converge towards the minimum with SGD? very slow progress along flat direction, jitter along steep one

Images courtesy: Karpathy et. al.

Suppose loss function is steep vertically but shallow horizontally:

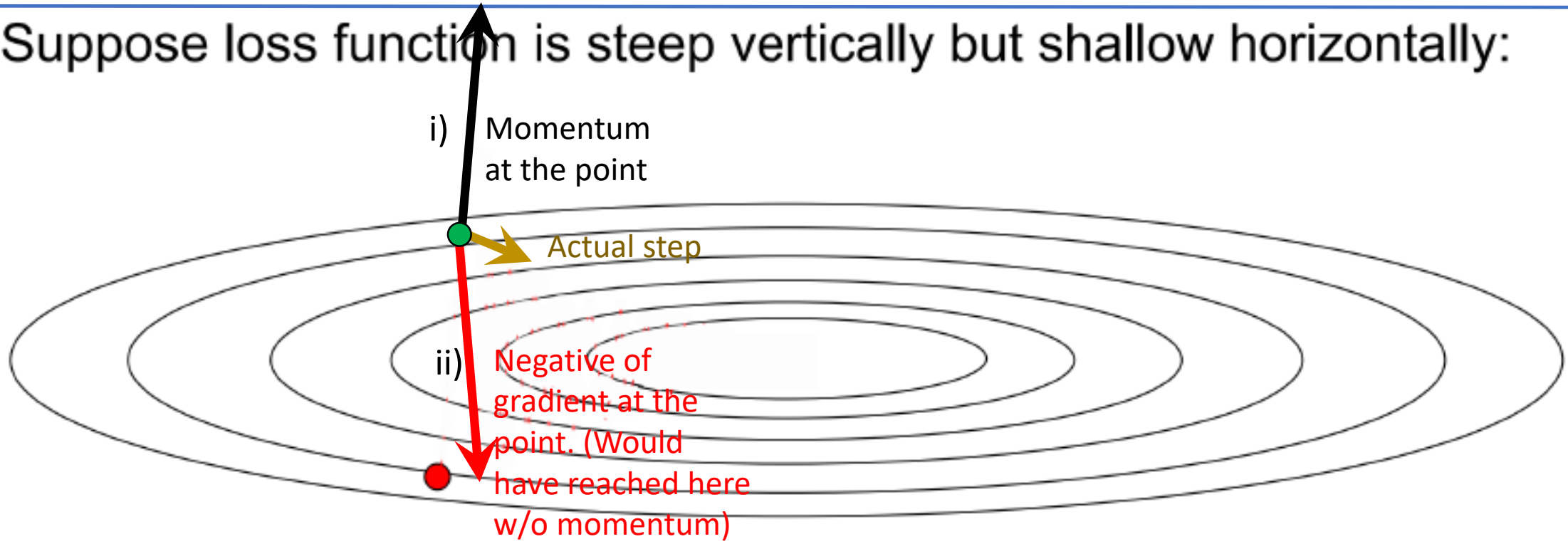


Suppose loss function is steep vertically but shallow horizontally:



Gradient Descent: only one force acting at any point.

Suppose loss function is steep vertically but shallow horizontally:



Momentum: Two forces acting at any point.

i) Momentum built up due to gradients pushing the particle at that point

ii) Gradient computed at that point

Stochastic Gradient Descent with Momentum

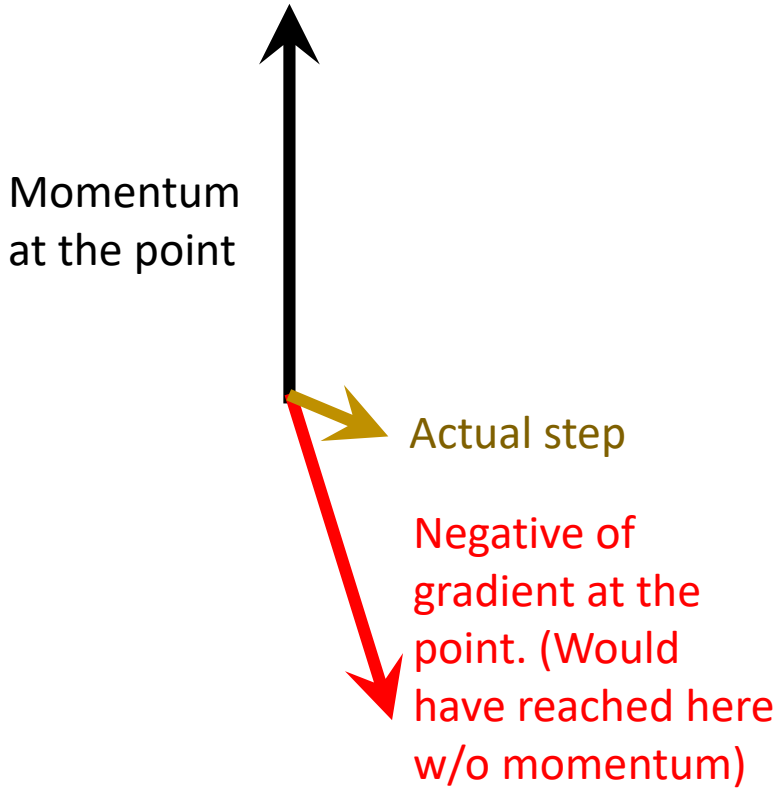
Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

$\alpha \in [0, 1)$

What is the role of α?

- If α is larger than ϵ the current update is more affected by the previous gradients
- Usually values for α are set high $\approx 0.8, 0.9$



Images courtesy: Goodfellow et. al.

Nesterov Momentum

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

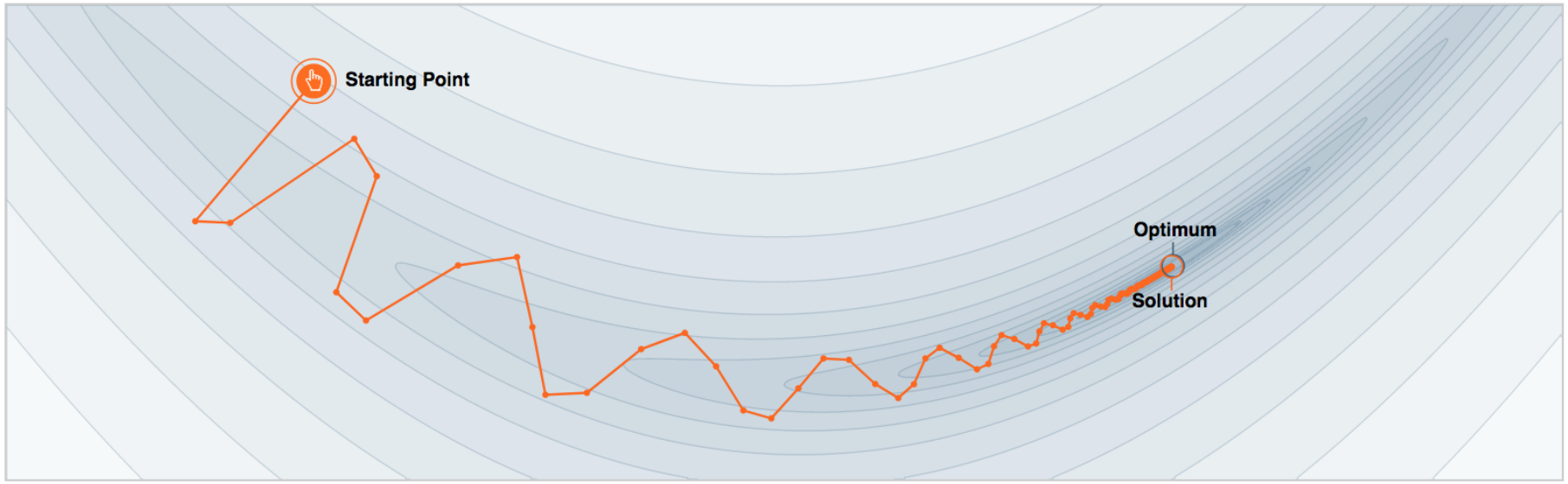
Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

- The difference between Nesterov momentum and standard momentum is where the gradient is evaluated.
- With Nesterov momentum the gradient is evaluated after the current velocity is applied.
- In practice, Nesterov momentum speeds up the convergence only for well behaved loss functions (convex with consistent curvature)

Images courtesy: Goodfellow et. al.

Distill Pub Article on Why Momentum Really Works

Why Momentum Really Works



Step-size α = 0.02

0 0.003 0.006

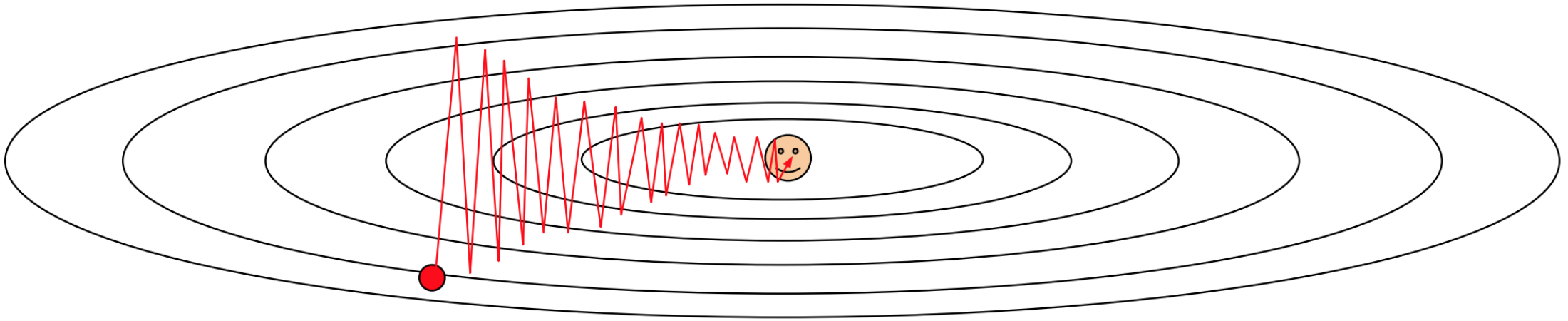
Momentum β = 0.85

0.00 0.500 0.990

We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

Adaptive Learning Rate Methods

- Till now we assign the same learning rate in all directions.
- Would it not be a good idea if we can move slowly in a steeper direction whereas move fast in a shallower direction?



Images courtesy: Karpathy et. al.

AdaGrad

- Downscale learning rate by square-root of sum of squares of all the historical gradient values
- Parameters that have large partial derivative of the loss – learning rates for them are rapidly declined

Algorithm 4 AdaGrad

Require: Global Learning rate ϵ, Initial Parameter θ, δ

Initialize $\mathbf{r} = 0$

1: **while** stopping criteria not met **do**

2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

4: Accumulate: $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

5: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

6: Apply Update: $\theta \leftarrow \theta + \Delta\theta$

7: **end while**

Images courtesy: Shubhendu Trivedi et. al.

AdaGrad

- $\hat{\boldsymbol{g}}_{(1)} = \begin{bmatrix} g_{(1),1} \\ g_{(1),2} \\ \vdots \\ g_{(1),d} \end{bmatrix}, \hat{\boldsymbol{g}}_{(2)} = \begin{bmatrix} g_{(2),1} \\ g_{(2),2} \\ \vdots \\ g_{(2),d} \end{bmatrix}, \ldots, \hat{\boldsymbol{g}}_{(t)} = \begin{bmatrix} g_{(t),1} \\ g_{(t),2} \\ \vdots \\ g_{(t),d} \end{bmatrix}$
- $\boldsymbol{r}_{(1)} = \begin{bmatrix} g_{(1),1}^2 \\ g_{(1),2}^2 \\ \vdots \\ g_{(1),d}^2 \end{bmatrix}, \boldsymbol{r}_{(2)} = \begin{bmatrix} g_{(1),1}^2 + g_{(2),1}^2 \\ g_{(1),2}^2 + g_{(2),2}^2 \\ \vdots \\ g_{(1),d}^2 + g_{(2),d}^2 \end{bmatrix}, \ldots, \boldsymbol{r}_{(t)} = \begin{bmatrix} g_{(1),1}^2 + g_{(2),1}^2 + \cdots + g_{(t),1}^2 \\ g_{(1),2}^2 + g_{(2),2}^2 + \cdots + g_{(t),2}^2 \\ \vdots \\ g_{(1),d}^2 + g_{(2),d}^2 + \cdots + g_{(t),d}^2 \end{bmatrix}$
- $\Delta\boldsymbol{\theta}_{(t)} = -\begin{bmatrix} \frac{\epsilon . g_{(t),1}}{\delta + \sqrt{g_{(1),1}^2 + g_{(2),1}^2 + \cdots + g_{(t),1}^2}} \\ \frac{\epsilon . g_{(t),2}}{\delta + \sqrt{g_{(1),2}^2 + g_{(2),2}^2 + \cdots + g_{(t),2}^2}} \\ \vdots \\ \frac{\epsilon . g_{(t),d}}{\delta + \sqrt{g_{(1),d}^2 + g_{(2),d}^2 + \cdots + g_{(t),d}^2}} \end{bmatrix}$

RMSProp

- One problem of AdaGrad is that the 'r' vector continues to build up and grow its value.
- This shrinks the learning rate too aggressively.
- RMSProp strikes a balance by exponentially decaying contributions from past gradients.

Algorithm 5 RMSProp

Require: Global Learning rate ϵ, decay parameter ρ, δ

Initialize $\mathbf{r} = 0$

1: **while** stopping criteria not met **do**

2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set

3: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

4: Accumulate: $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$

5: Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$

6: Apply Update: $\theta \leftarrow \theta + \Delta\theta$

7: **end while**

Images courtesy: Shubhendu Trivedi et. al.

RMSProp

- $\hat{\boldsymbol{g}}_{(1)}, \hat{\boldsymbol{g}}_{(2)}, \ldots, \hat{\boldsymbol{g}}_{(t)} \quad \boldsymbol{r}_{(t)} = \rho \boldsymbol{r}_{(t-1)} + (1-\rho)\hat{\boldsymbol{g}}_{(t)} \odot \hat{\boldsymbol{g}}_{(t)}$

- $\boldsymbol{r}_{(0)} = 0$
- $\boldsymbol{r}_{(1)} = \rho \boldsymbol{r}_{(0)} + (1-\rho)\hat{\boldsymbol{g}}_{(1)} \odot \hat{\boldsymbol{g}}_{(1)} = (1-\rho)\hat{\boldsymbol{g}}_{(1)} \odot \hat{\boldsymbol{g}}_{(1)}$
- $\boldsymbol{r}_{(2)} = \rho \boldsymbol{r}_{(1)} + (1-\rho)\hat{\boldsymbol{g}}_{(2)} \odot \hat{\boldsymbol{g}}_{(2)} = \rho(1-\rho)\hat{\boldsymbol{g}}_{(1)} \odot \hat{\boldsymbol{g}}_{(1)} + (1-\rho)\hat{\boldsymbol{g}}_{(2)} \odot \hat{\boldsymbol{g}}_{(2)}$
- $\boldsymbol{r}_{(3)} = \rho \boldsymbol{r}_{(2)} + (1-\rho)\hat{\boldsymbol{g}}_{(3)} \odot \hat{\boldsymbol{g}}_{(3)} = \rho^2(1-\rho)\hat{\boldsymbol{g}}_{(1)} \odot \hat{\boldsymbol{g}}_{(1)} + \rho(1-\rho)\hat{\boldsymbol{g}}_{(2)} \odot \hat{\boldsymbol{g}}_{(2)} + (1-\rho)\hat{\boldsymbol{g}}_{(3)} \odot \hat{\boldsymbol{g}}_{(3)}$

- RMSProp uses an exponentially decaying average to discard history from the extreme past so that the accumulation of gradients do not stall the learning.
- AdaDelta is another variant where instead of exponentially decaying average, a moving window average of the past gradients is taken
- Nesterov acceleration can also be applied to both these variants by computing the gradients at a ‘look ahead’ position (i.e., at a place where the momentum would have taken the parameters).

ADAM (Adaptive Moments)

- Variant of the combination of RMSProp and Momentum.
- Incorporates first order moment of the gradient which can be thought of as equivalent to taking advantage of the momentum strategy. Here momentum is also added with exponential averaging.
- It also incorporates the second order term which can be thought of as the RMSProp like exponential averaging of the past gradients.
- Both first and second moments are corrected for bias to account for their initialization to zero.

ADAM (Adaptive Moments)

- Biased first order moment $\boldsymbol{s}_{(t)} = \rho_1 \boldsymbol{s}_{(t-1)} + (1-\rho_1)\boldsymbol{g}_t$
- Biased first order moment $\boldsymbol{r}_{(t)} = \rho_2 \boldsymbol{r}_{(t-1)} + (1-\rho_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t$
- Bias corrected first order moment $\hat{\boldsymbol{s}}_{(t)} = \frac{\boldsymbol{s}_{(t)}}{1-\rho_1}$
- Bias corrected second order moment $\hat{\boldsymbol{r}}_{(t)} = \frac{\boldsymbol{r}_{(t)}}{1-\rho_2}$
- Weight update $\Delta\boldsymbol{\theta}_{(t)} = -\epsilon \frac{\hat{\boldsymbol{s}}_{(t)}}{\sqrt{\hat{\boldsymbol{r}}_{(t)}+\delta}}$ (operations applied elementwise)

ADAM (Adaptive Moments)

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})

Require: Initial parameters $\boldsymbol{\theta}$

Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

$t \leftarrow t + 1$

Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

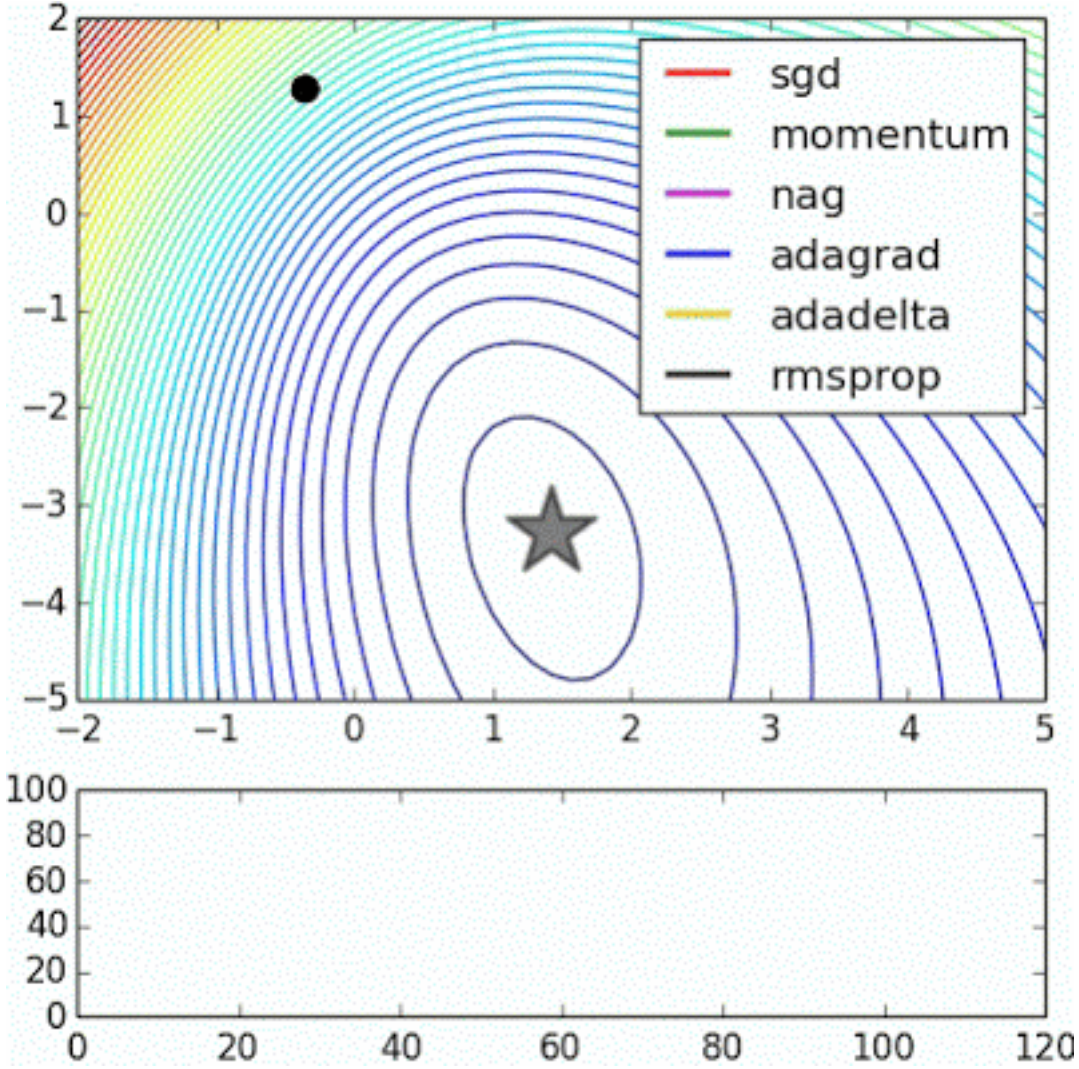
Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

Images courtesy: Shubhendu Trivedi et. al.

Visualization



Find more animations at
https://tinyurl.com/y6tkf4f8

animation courtesy: Alec Radford

Thank You!!