NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE/CZ 7454: Deep Learning for Data Science

Li Boyang, Albert

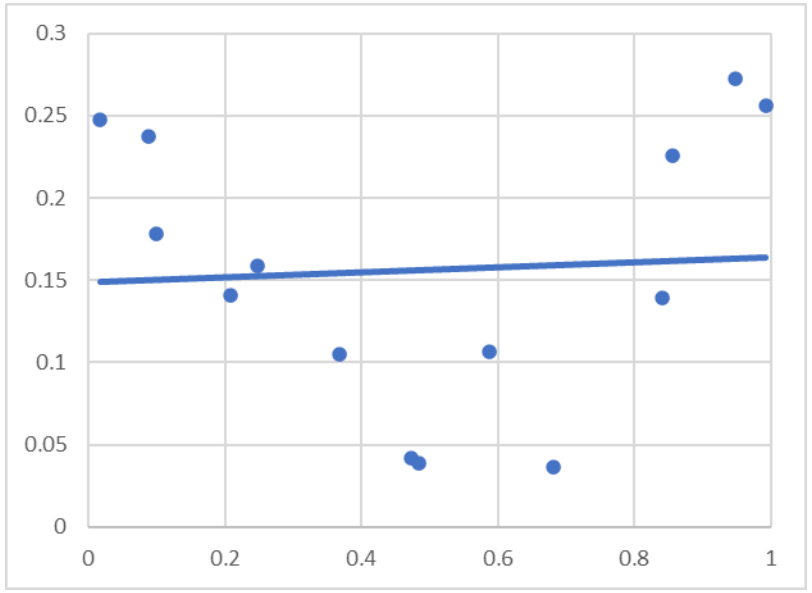
School of Computer Science and Engineering

AY2021-2022, Semester 1

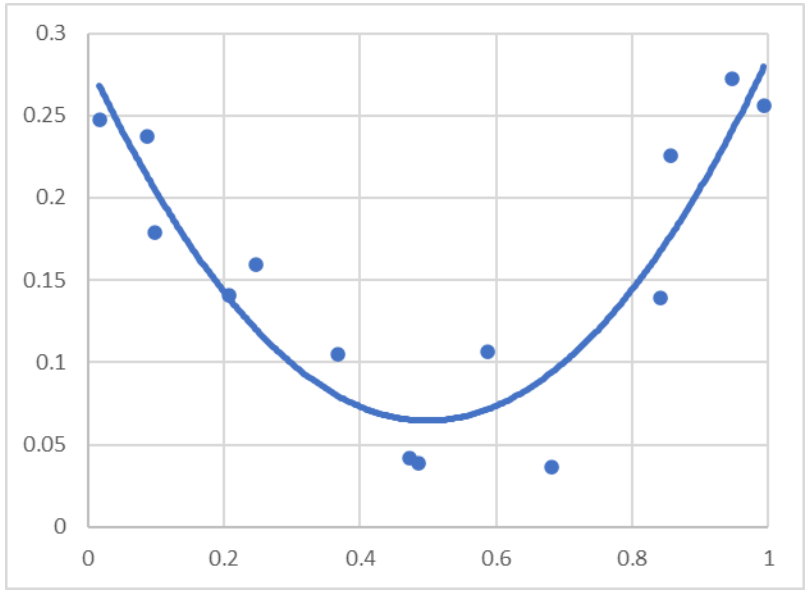
Underfitting vs. Overfitting

The Art of Balancing

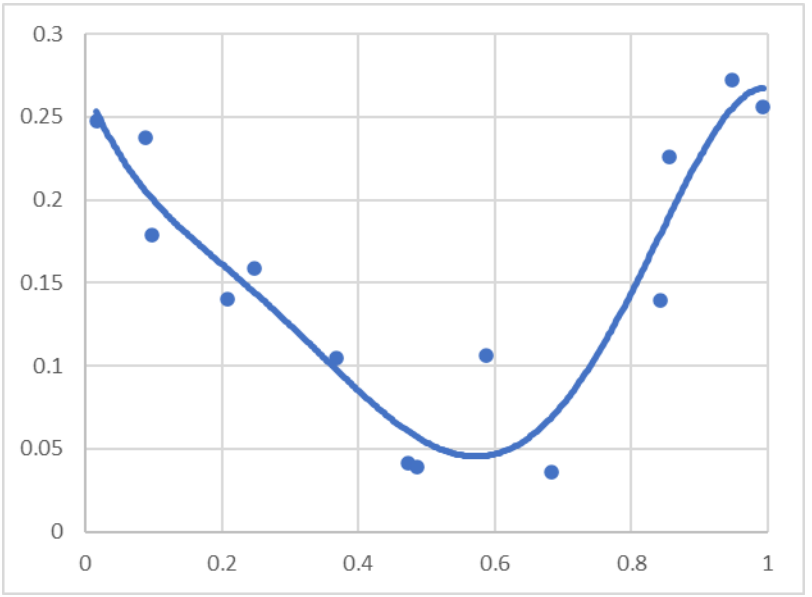
Underfitting vs. Overfitting



Not describing the data



Describing the data well



Taking the data too literally

Optimization vs. Regularization

- Optimization: fitting the data
  - Minimize the empirical loss on the training set
  - Maximize performance on the training set
- Regularization: avoid overfitting the data
  - Improve performance on future data (test set), often at the cost of training loss.
- Traditionally, these two are considered to be opposite to each other.
- In deep learning, not always.

Conventional Intuition

- Underfitting: The model is too limited and cannot represent the function that generates the data.
- Overfitting: The model is too expressive. It can represent the data-generating function as well as the noise in the data.

Deep Learning Intuition

- Underfitting: Not learning enough from the observed data.
  - Even if the model is large, the optimization could be poor.
- Overfitting: The observed data contain some random errors, some idiosyncrasies that do not apply to the general population. Overfitting is to learn from such noise.
  - Example: Learning from outlier images and thinking that they are representative of many real-world images.

Representative Images?

Bird or Kiwi Fruit?

Fish or Cucumber?

Images created by Sarah DeRemer.
Reproduced with permission.

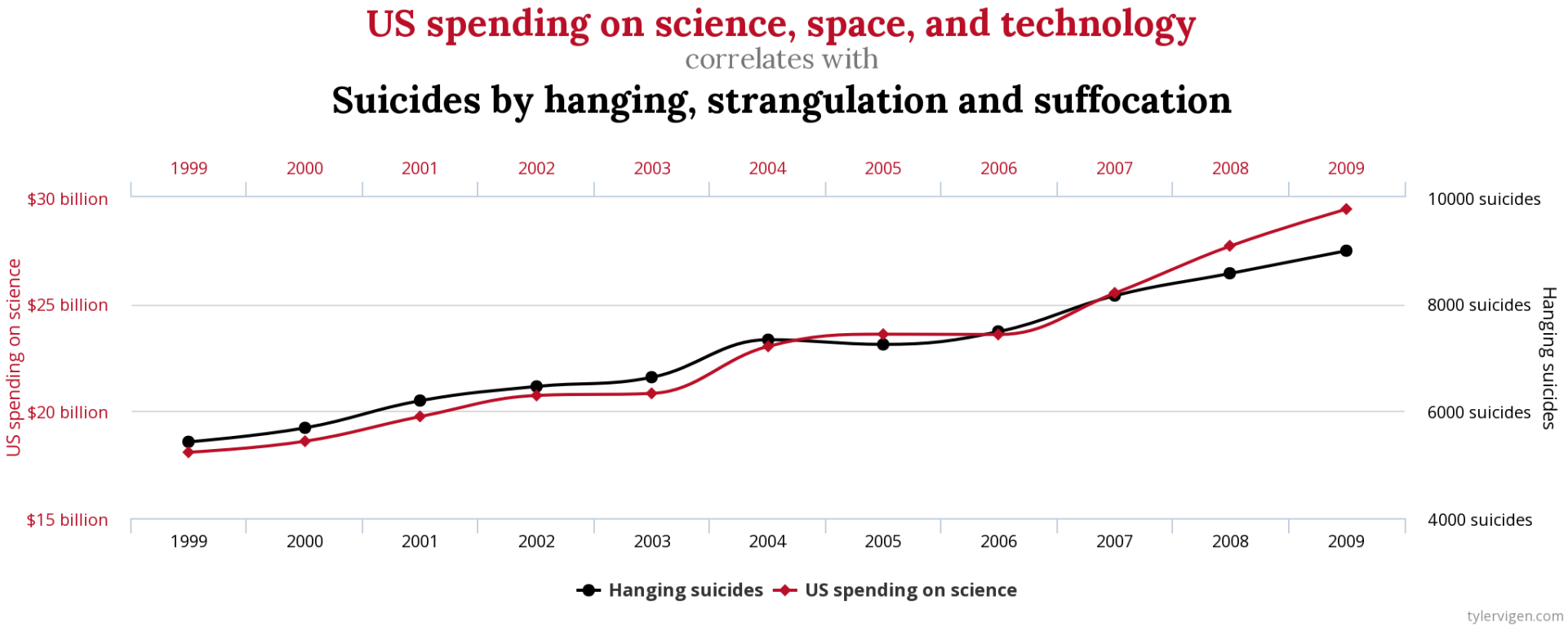
Deep Learning Intuition

- Underfitting: Not learning enough from the observed data.
  - Even if the model is large, the optimization could be poor.
- Overfitting: The observed data contain some random errors, some idiosyncrasies that do not apply to the general population. Overfitting is to learn from such noise.
  - Example: Learning from some outlier images.
- DL deals with more complex semantics than traditional ML.
  - What is noise using one set of features is signal under another set of features.
  - Quality of feature representation matters a lot.
  - Sometimes increasing capacity could reduce generalization error!

Deep Learning is Representation Learning

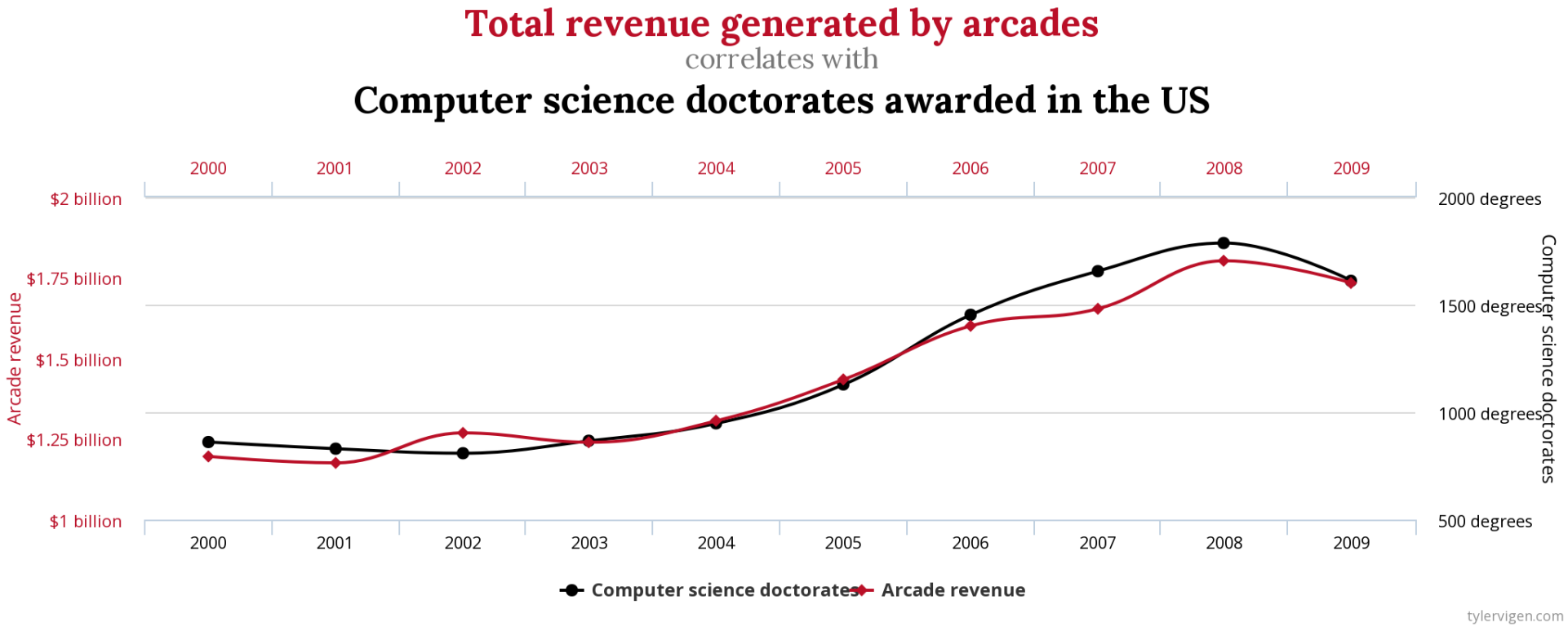
- Shallow models work with existing features
  - E.g., logistic regression: $\hat{y} = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x})$
- Deep models can be seen as learning features
  - MLP: $\hat{y} = \sigma\left(w_L \cdots \sigma\left(W_2 \sigma(W_1 \boldsymbol{x})\right)\right)$
  - Let $\boldsymbol{z} = \sigma\left(W_{L-1} \cdots \sigma\left(W_2 \sigma(W_1 \boldsymbol{x})\right)\right)$, we have $\hat{y} = \sigma(\boldsymbol{w}_L^\top \boldsymbol{z})$
- Learning the right features is critical

Bad Feature (correlation 99.79%)



Guaranteed: low training error, high test error on future data

Bad Feature (correlation 98.51%)



Guaranteed: low training error, high test error on future data

Feature Extraction: Which is more robust?

- What is the make of the car?

Overfitting: For cars in a single year, these features (wheels, fog lights, grill) can be very effective, but they will not work if Toyota makes a design change.

The logo is unlikely to change and should generalize well.

Deep Learning is Representation Learning

- Many regularization techniques in DL aim to improve the quality of features extracted by the deep network.
  - MixUp
  - Data Augmentation
  - Pretraining is effective against overfitting
- Covered in the next lecture

Optimization vs. Regularization in DL

- The most important issue in deep learning.
- Optimization is hard in DL
  - First-order optimization is hard.
  - Second-order methods are great but we can't use them.
- Regularization is hard in DL
  - Overparameterized deep nets are capable of memorizing every training instance (Zhang et al. 2017).

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. ICLR 2017

Symptoms and Diagnosis

- Effective trial-and-error requires understanding accurate diagnosis and the correct remedy
- Underfitting
  - Large training loss, large validation / test loss
  - The loss stagnates early in training
  - Solution: Improve optimization; increase model capacity
- Overfitting
  - Large gap between training set performance and val/test performance
  - Usually gap in loss values too
  - Solution: Apply stronger regularization (first priority); decrease model capacity;
- In deep learning, zero overfitting does not always give best performance.

Bias-variance Tradeoff

- We want to estimate random variables (e.g, neural network weights) from data
  - Low bias = the estimate fits data well but may overfit
  - Low variance = the estimate is not overly influenced by noise in data, but may underfit
- Adding L2 regularization to linear regression lowers the variance in the estimate of model parameter $\boldsymbol{\beta}$
  - http://cs229.stanford.edu/summer2019/BiasVarianceAnalysis.pdf

Bias-variance Tradeoff

- Dataset $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$ drawn i.i.d from a distribution $P(X, Y)$
- i.i.d = independently and identically distributed
- **Expected label**

$$\bar{y}(\mathbf{x}) = E_{y|\mathbf{x}}[Y] = \int_y y \Pr(y|\mathbf{x}) \partial y.$$

- The label y depends on x but the relationship is not 100% deterministic. So we compute its expectation from distribution $\Pr(y|\boldsymbol{x})$
- This noise in y could reflect annotation error, or simply the semantics in $\boldsymbol{x}$ that the model cannot understand. Recall: coin toss is deterministic.

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

- The machine learning algorithm A learns a model (or hypothesis) h_D from the dataset D. $h_D = A(D)$.
- Expected model

$$\bar{h} = E_{D \sim P^n}[h_D] = \int_D h_D \Pr(D) \partial D$$

- Expected test error

$$E_{(\mathbf{x},y)\sim P}\left[(h_D(\mathbf{x}) - y)^2\right] = \int_x \int_y (h_D(\mathbf{x}) - y)^2 \Pr(\mathbf{x}, y) \partial y \partial \mathbf{x}.$$

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

- Combining everything we have

$$E_{\substack{(\mathbf{x},y)\sim P \\ D\sim P^n}}\left[(h_D(\mathbf{x})-y)^2\right] = \int_D \int_{\mathbf{x}} \int_y (h_D(\mathbf{x})-y)^2 \mathrm{P}(\mathbf{x},y)\mathrm{P}(D)\partial\mathbf{x}\partial y\partial D$$

- That is, the expectation is over the observed dataset D and all possible data $(\boldsymbol{x}, y)$

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

$$E_{\substack{(\mathbf{x},y)\sim P \\ D\sim P^n}}\left[(h_D(\mathbf{x})-y)^2\right] = \int_D \int_{\mathbf{x}} \int_y (h_D(\mathbf{x})-y)^2 \mathrm{P}(\mathbf{x},y)\mathrm{P}(D)\partial\mathbf{x}\partial y\partial D$$

$$\begin{aligned} E_{\mathbf{x},y,D}\left[[h_D(\mathbf{x})-y]^2\right] &= E_{\mathbf{x},y,D}\left[\left[(h_D(\mathbf{x})-\bar{h}(\mathbf{x}))+(\bar{h}(\mathbf{x})-y)\right]^2\right] \\ &= E_{\mathbf{x},D}\left[(\bar{h}_D(\mathbf{x})-\bar{h}(\mathbf{x}))^2\right] + \underbrace{2\,E_{\mathbf{x},y,D}\left[(h_D(\mathbf{x})-\bar{h}(\mathbf{x}))\,(\bar{h}(\mathbf{x})-y)\right]}_{=0} + E_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-y)^2\right] \end{aligned}$$

$$\begin{aligned} E_{\mathbf{x},y,D}\left[(h_D(\mathbf{x})-\bar{h}(\mathbf{x}))\,(\bar{h}(\mathbf{x})-y)\right] &= E_{\mathbf{x},y}\left[E_D\left[h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right](\bar{h}(\mathbf{x})-y)\right] \\ &= E_{\mathbf{x},y}\left[\left(E_D\left[h_D(\mathbf{x})\right]-\bar{h}(\mathbf{x})\right)(\bar{h}(\mathbf{x})-y)\right] \\ &= E_{\mathbf{x},y}\left[(\bar{h}(\mathbf{x})-\bar{h}(\mathbf{x}))\,(\bar{h}(\mathbf{x})-y)\right] \\ &= E_{\mathbf{x},y}\left[0\right] \\ &= 0 \end{aligned}$$

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

$$
\begin{aligned}
E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x})-y\right)\right] &= E_{\mathbf{x},y}\left[E_D\left[h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right]\left(\bar{h}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(E_D\left[h_D(\mathbf{x})\right]-\bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{h}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[0\right] \\
&= 0
\end{aligned}
$$

Recall: The expectation operation is linear

$$E_a[f(a)b] = \int f(a)b\,P(a)\,da = b\int f(a)\,P(a)\,da = bE_a[f(a)]$$

$$
\begin{aligned}
E_a[f(a)+g(a)] &= \int\left(f(a)+g(a)\right)P(a)\,da = \int f(a)P(a)\,da + \int g(a)P(a)\,da \\
&= E_a[f(a)] + E_a[g(a)]
\end{aligned}
$$

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

$$E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right]=\underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}}+E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right]$$

- Variance of $h_D(\boldsymbol{x})$ shows how much h_D changes when the observed dataset D changes.
- If we happen to observe a few outliers in the data, is the learned function heavily affected by them?

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

$$E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right] = E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)+\left(\bar{y}(\mathbf{x})-y\right)^2\right]$$

$$= \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2} + 2\,E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\left(\bar{y}(\mathbf{x})-y\right)\right]$$

=0

$\underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}}$ is the variance in y

$\underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$ Bias measures how much $h(\boldsymbol{x})$ deviate from the expected label $\bar{y}$
Note that $\bar{h}$ is the average over all possible dataset D, so D has no influence here.

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

$$
\begin{aligned}
E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-y\right)^2\right] &= E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)+\left(\bar{y}(\mathbf{x})-y\right)^2\right] \\
&= \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2} + \underbrace{2\,E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\left(\bar{y}(\mathbf{x})-y\right)\right]}_{=0}
\end{aligned}
$$

$$
\begin{aligned}
E_{\mathbf{x},y}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\left(\bar{y}(\mathbf{x})-y\right)\right] &= E_{\mathbf{x}}\left[E_{y|\mathbf{x}}\left[\bar{y}(\mathbf{x})-y\right]\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[E_{y|\mathbf{x}}\left[\bar{y}(\mathbf{x})-y\right]\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[\left(\bar{y}(\mathbf{x})-E_{y|\mathbf{x}}\left[y\right]\right)\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[\left(\bar{y}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[0\right] \\
&= 0
\end{aligned}
$$

Content credit: Kilian Weinberger (https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html)

Bias-variance Tradeoff

- Combining everything,

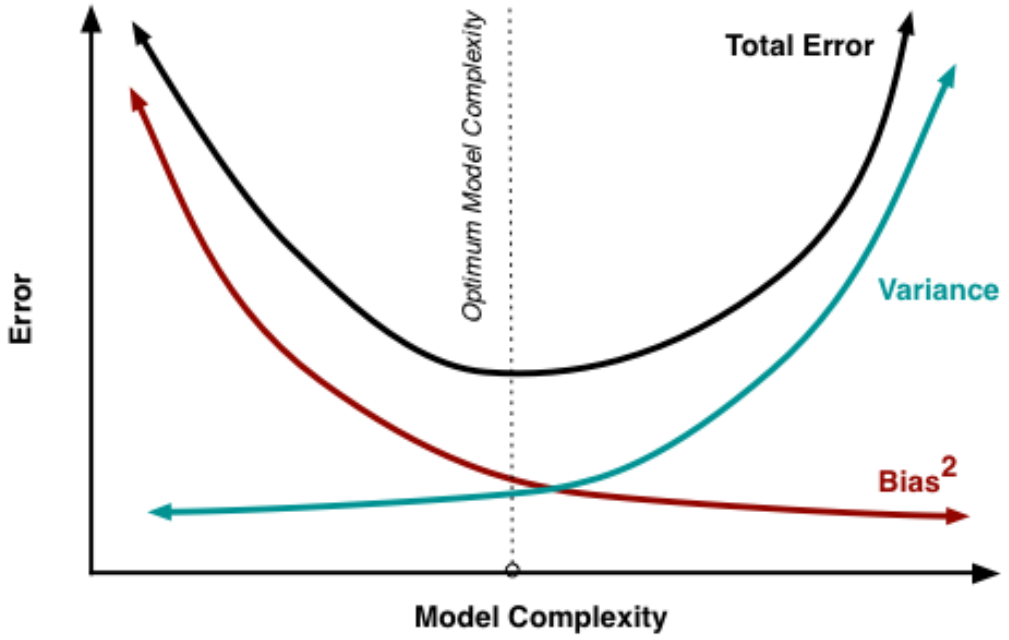
$$\underbrace{E_{\mathbf{x},y,D}\left[\left(h_D(\mathbf{x})-y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(h_D(\mathbf{x})-\bar{h}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x},y}\left[\left(\bar{y}(\mathbf{x})-y\right)^2\right]}_{\text{Noise}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x})\right)^2\right]}_{\text{Bias}^2}$$

Variance: Captures how much your classifier changes if you train on a different training set. How "over-specialized" is your classifier to a particular training set (overfitting)? If we have the best possible model for our training data, how far off are we from the average classifier?

Bias: What is the inherent error that you obtain from your classifier even with infinite training data? This is due to your classifier being "biased" to a particular kind of solution (e.g. linear classifier). In other words, bias is inherent to your model.

Noise: How big is the data-intrinsic noise? This error measures ambiguity due to your data distribution and feature representation. You can never beat this, it is an aspect of the data.

Bias-variance Tradeoff



Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

As model complexity increases, overfitting increases.

Conventional ML wisdom says that you need to have fewer parameters than training data points.

In deep learning, models are heavily overparameterized and achieve zero training loss, but still generalize well.

Still, you can have too many parameters even in deep learning. Overfitting is a severe problem and researchers developed many techniques to mitigate it .

Optimization: Gradient Descent

Gradient Descent

- We seek $\boldsymbol{w}$ that minimizes a differentiable function $L(x, \boldsymbol{w})$
- Input: loss function $L(\boldsymbol{w})$, initial position $\boldsymbol{w}_0$, learning rates η_t
- Algorithm: Repeat until termination criterion
  - Move in the direction of negative gradient
  - $\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta_t \left.\frac{\mathrm{d}L(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}}\right|_{\boldsymbol{w}_{t-1}}$

Taylor Series for Function Approximation

- We can approximate an infinitely differentiable function $f(x): \mathbb{R} \rightarrow \mathbb{R}$ with the following sum

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 + \cdots$$

- Alternatively

$$f(x) = \sum_{n=0}^{\infty} a_n (x - x_0)^n \, , a_n = \boxed{\frac{f^{(n)}(x_0)}{n!}}$$

n^{th} derivative

Taylor Series for Function Approximation

- We can approximate an infinitely differentiable function $f(x)$: $\mathbb{R} \rightarrow \mathbb{R}$ with the following sum

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \frac{1}{3!}f'''(x_0)(x - x_0)^3 + \cdots$$

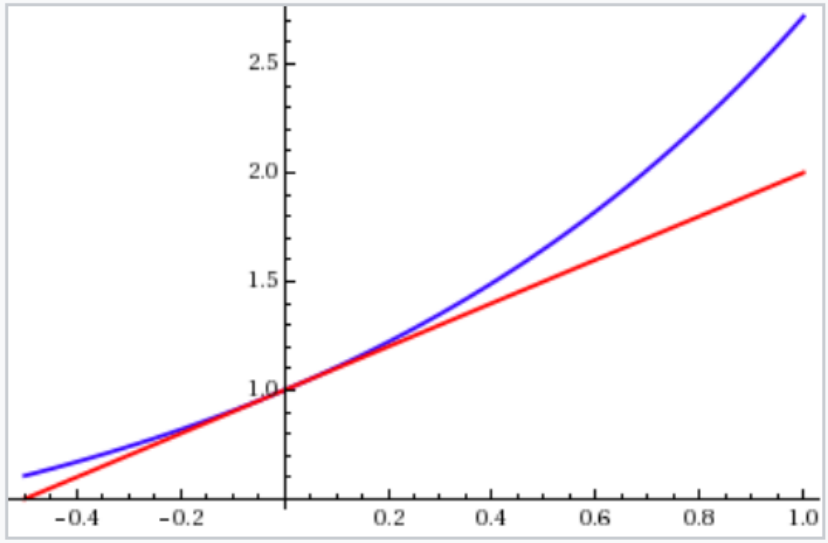
- When $|x - x_0| = \epsilon$ is small ($\epsilon \ll 1$), $(x - x_0)^2$ is $\epsilon^2 \ll \epsilon$
- Every term after that is even smaller.
- Thus, for approximation around x_0, we may take only the first two terms.

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + O(\epsilon^2)$$

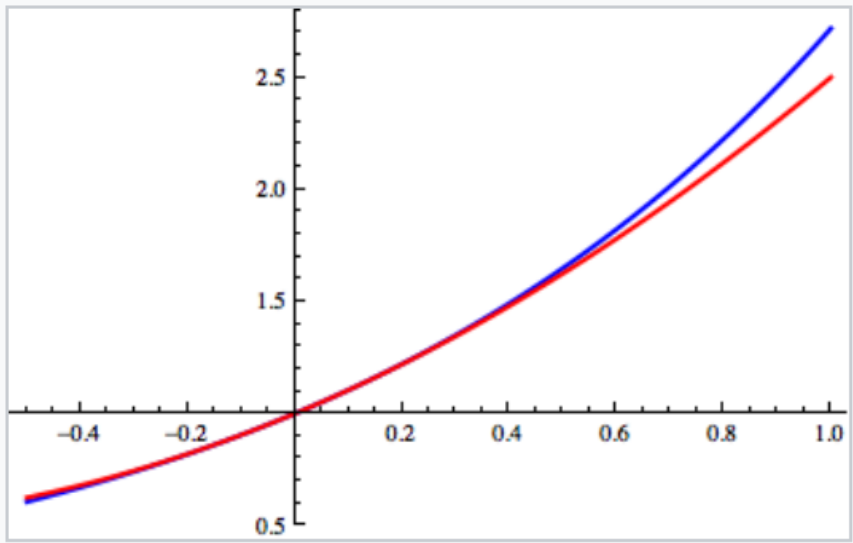
- For slightly larger ϵ, three terms allow us to be more precise

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + O(\epsilon^3)$$

Taylor Series for Function Approximation



Graph of $f(x) = e^x$ (blue) with its linear approximation $P_1(x) = 1 + x$ (red) at $a = 0$.



Graph of $f(x) = e^x$ (blue) with its quadratic approximation $P_2(x) = 1 + x + x^2/2$ (red) at $a = 0$. Note the improvement in the approximation.

Multivariate Taylor Polynomial

- For small Δ

$$f(\boldsymbol{x}+\Delta) = f(\boldsymbol{x}) + \left(\frac{df}{d\boldsymbol{x}}\right)^{\mathsf{T}} \Delta + \frac{1}{2}\Delta^{\mathsf{T}}\left(\frac{d^2 f}{d\boldsymbol{x}^2}\right)\Delta + \cdots$$

Constant term

Linear Term (in Δ)

Quadratic term with Hessian.

Additional terms are increasingly less important.

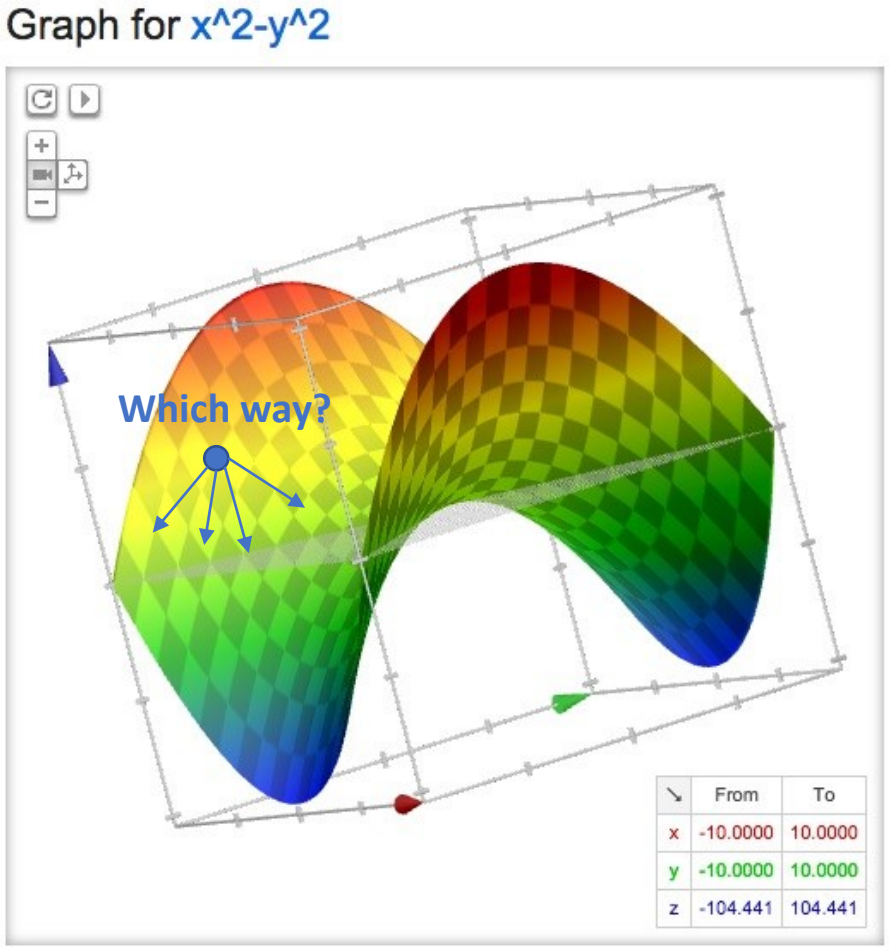
Gradient Descent is Steepest Descent

- We want to find a direction $\boldsymbol{v} \in \mathbb{R}^n$ that leads to the fastest decrease in $f(\boldsymbol{x})$.
- If we can find $\boldsymbol{v}$, we will move a little bit in that direction.

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \eta \boldsymbol{v}$$

- We look for $v_{sd} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \frac{f(\boldsymbol{x}+\boldsymbol{v})}{\|\boldsymbol{v}\|}$
- As we plan to take a small step, we can use a first-order approximation.

$$f(\boldsymbol{x}+\boldsymbol{v}) = f(\boldsymbol{x}) + \left(\frac{df}{d\boldsymbol{x}}\right)^{\mathsf{T}} \boldsymbol{v}$$



Steepest Descent under Euclidean Norm

- We want to find a direction $\boldsymbol{v} \in \mathbb{R}^n$ that leads to the fastest decrease in $f(\boldsymbol{x})$.
- If we can find $\boldsymbol{v}$, we will move a little bit in that direction.

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \eta \boldsymbol{v}$$

- We look for $v_{sd} = \underset{\boldsymbol{v}}{\operatorname{argmin}} \frac{f(\boldsymbol{x}+\boldsymbol{v})}{\|\boldsymbol{v}\|}$
- As we plan to take a small step, we can use a first-order approximation.

$$f(\boldsymbol{x}+\boldsymbol{v}) = f(\boldsymbol{x}) + \left(\frac{df}{d\boldsymbol{x}}\right)^{\top} \boldsymbol{v}$$

- Fact: $\frac{\boldsymbol{v}^{\top}\boldsymbol{g}}{\|\boldsymbol{v}\|} = \|\boldsymbol{g}\|\cos\theta$
- All it matters is $\cos\theta$
- To minimize $\frac{\boldsymbol{v}^{\top}\boldsymbol{g}}{\|\boldsymbol{v}\|}$, we can choose $\boldsymbol{v} = -\boldsymbol{g}$
- That is, $-\left(\frac{df}{d\boldsymbol{x}}\right)$ is the direction for steepest descent.
- Note: we may choose a different norm and get different results.

Further Analysis

- For the analysis of multiple GD iterations, we use a second-order approximation

$$f(\boldsymbol{x}+\Delta) = f(\boldsymbol{x}) + \left(\frac{df}{d\boldsymbol{x}}\right)^{\mathsf{T}} \Delta + \frac{1}{2}\Delta^{\mathsf{T}}\left(\frac{d^2 f}{d\boldsymbol{x}^2}\right)\Delta$$

Constant term

Linear Term (in Δ)

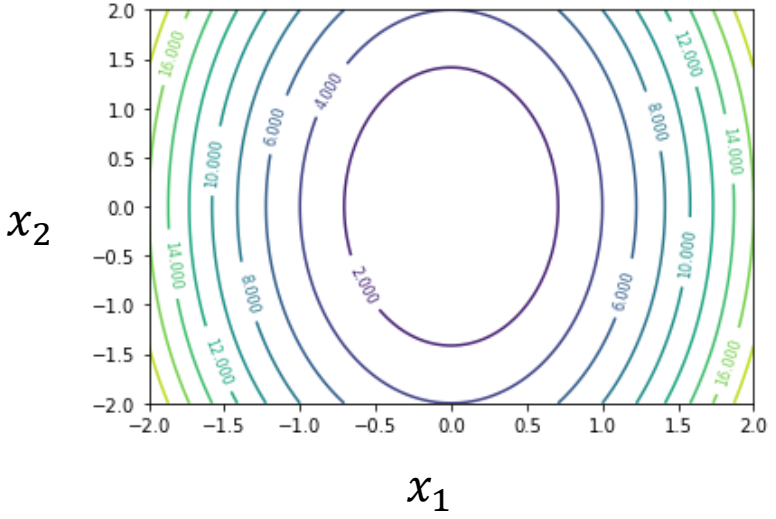
Quadratic term with Hessian.

Hessian = Diagonal

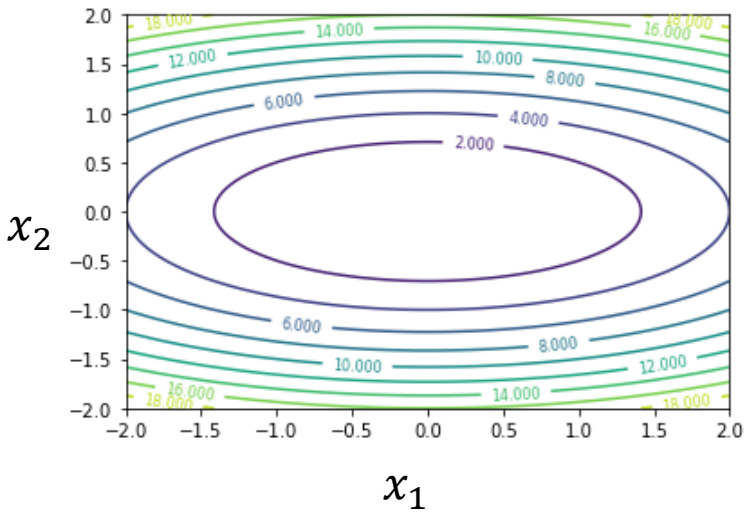
Which contour diagram correctly depicts this function?

$$f(\boldsymbol{x}) = [x_1, x_2]\begin{bmatrix}4 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

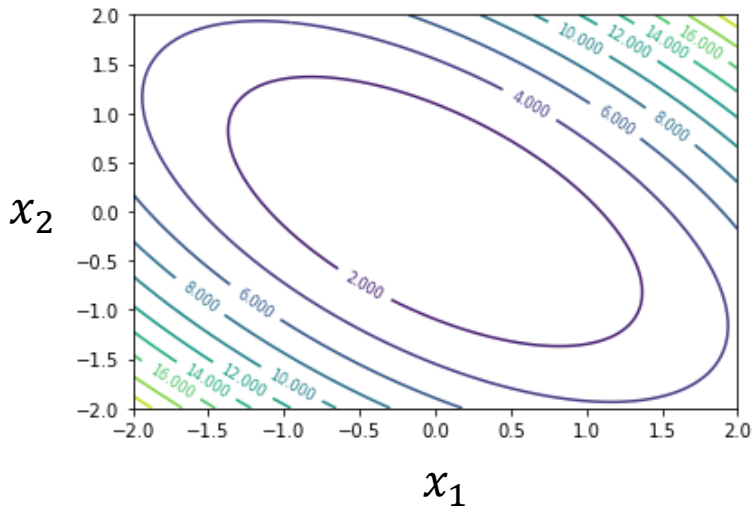
$$= 4x_1^2 + x_2^2$$



(A)



(B)



(C)

Hessian = Diagonal

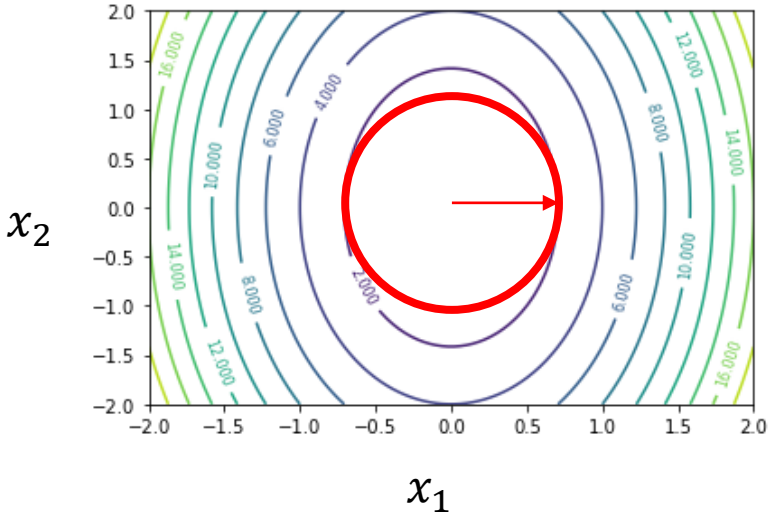
Which contour diagram correctly depicts this function?

$$f(\boldsymbol{x}) = [x_1, x_2]\begin{bmatrix}4 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

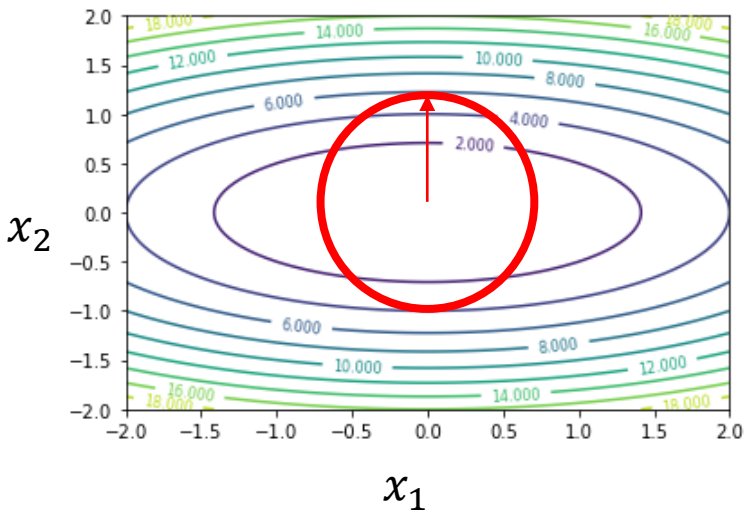
$$= 4x_1^2 + x_2^2$$

Moving by unit length in this direction increases the function value the most.

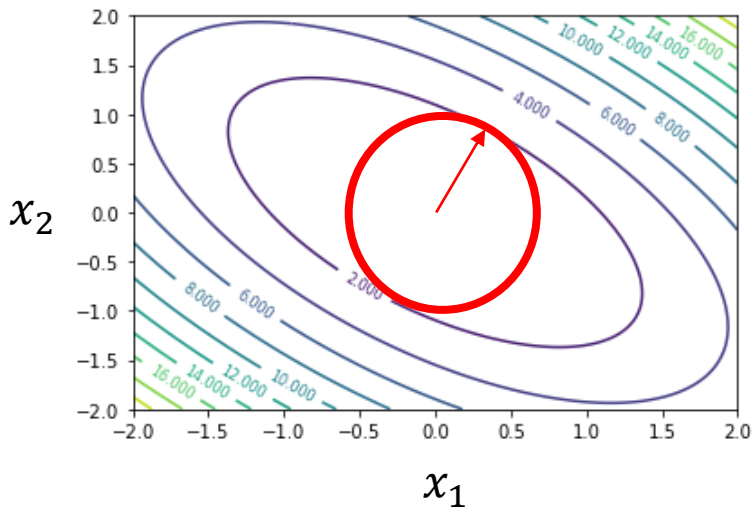
Plug in $\boldsymbol{x} = e_1 = \begin{bmatrix}1\\0\end{bmatrix}$, $e_2 = \begin{bmatrix}0\\1\end{bmatrix}$, or $\begin{bmatrix}\cos\theta\\ \sin\theta\end{bmatrix}$, which one has higher function value?



(A)



(B)



(C)

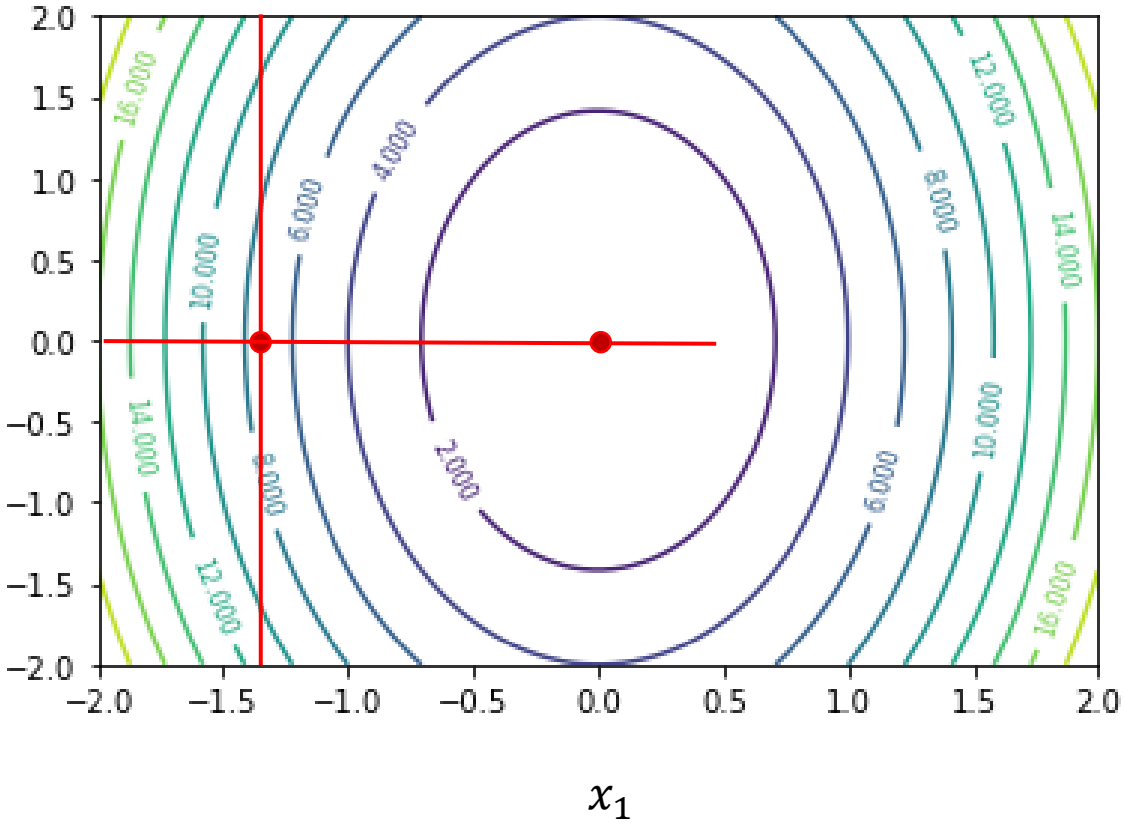
Diagonal Hessian

$$f(\boldsymbol{x}) = [x_1, x_2]\begin{bmatrix}4 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

$$= 4x_1^2 + x_2^2$$

All scaling happens along the coordinate axes.

Easy to optimize. First find the best x_2 on any x_1, and find the best x_1 (coordinate descent).



x_2

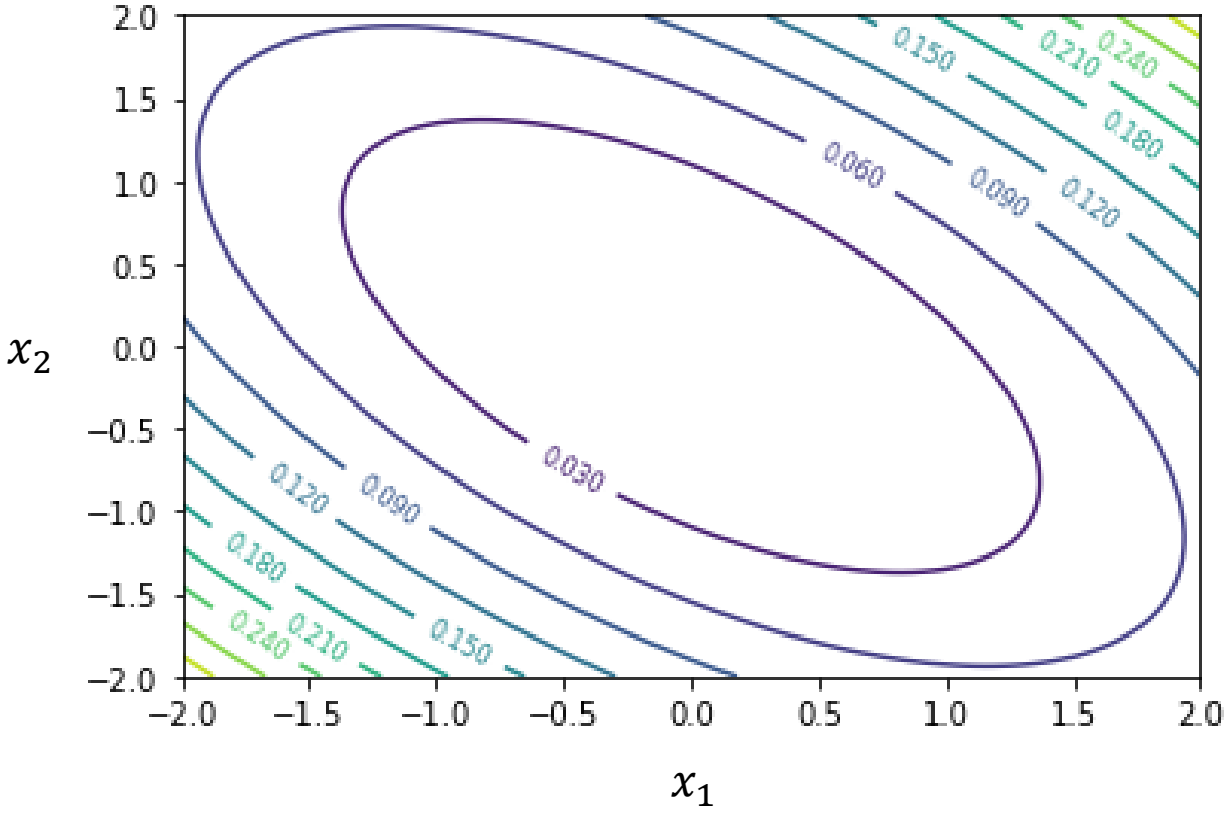
x_1

Hessian = Diagonal × Rotation

$$f(\boldsymbol{x}) = [x_1, x_2] \underbrace{\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}}_{\text{Clockwise } 45°} \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}}_{\text{Counter-clockwise } 45°} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top Q \Lambda Q^\top \boldsymbol{x}$$

The eigenvalues of Hessian determine the ratio of the two axes of the ellipse.



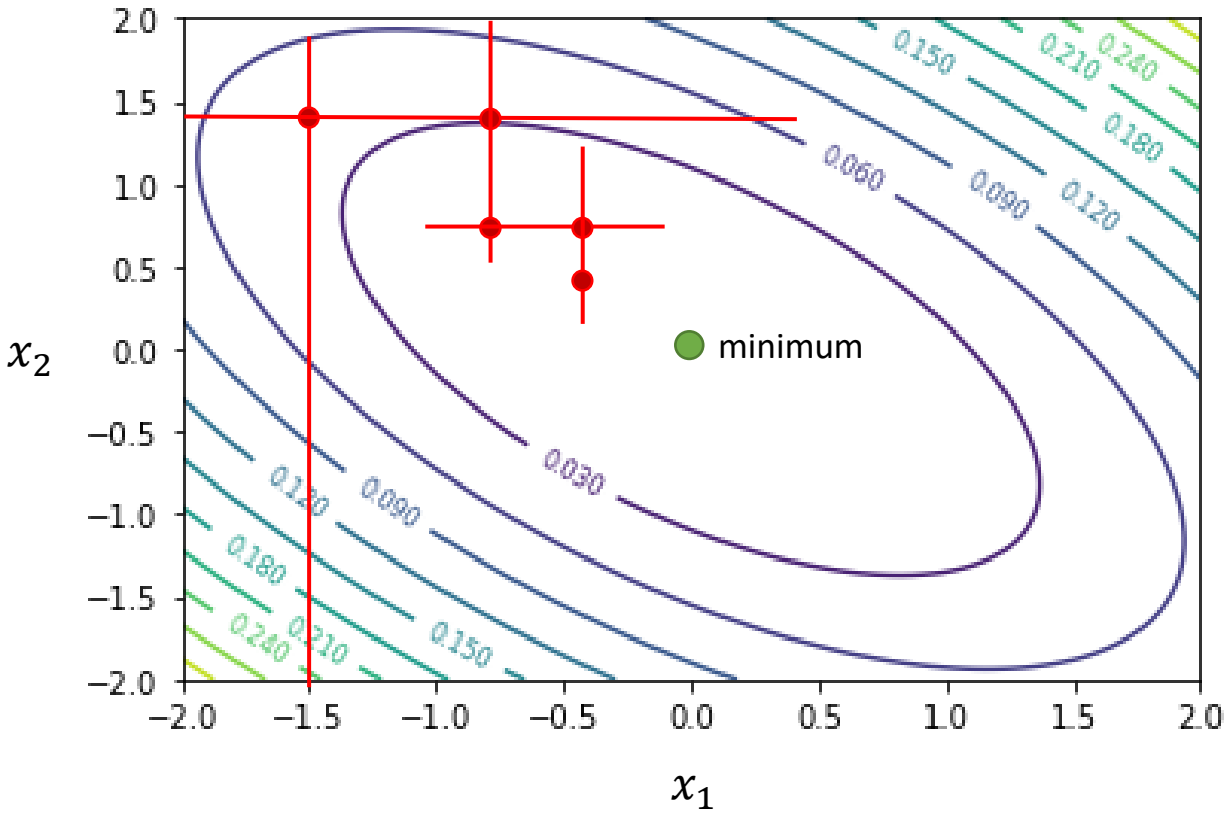
Hessian = Diagonal × Rotation

Clockwise 45°

Counter-clockwise 45°

$$f(\boldsymbol{x}) = [x_1, x_2]\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Coordinate descent converges slowly in this scenario.
  - Not rotational invariant.
- Gradient descent is rotational invariant.



Hessian = Diagonal × Rotation

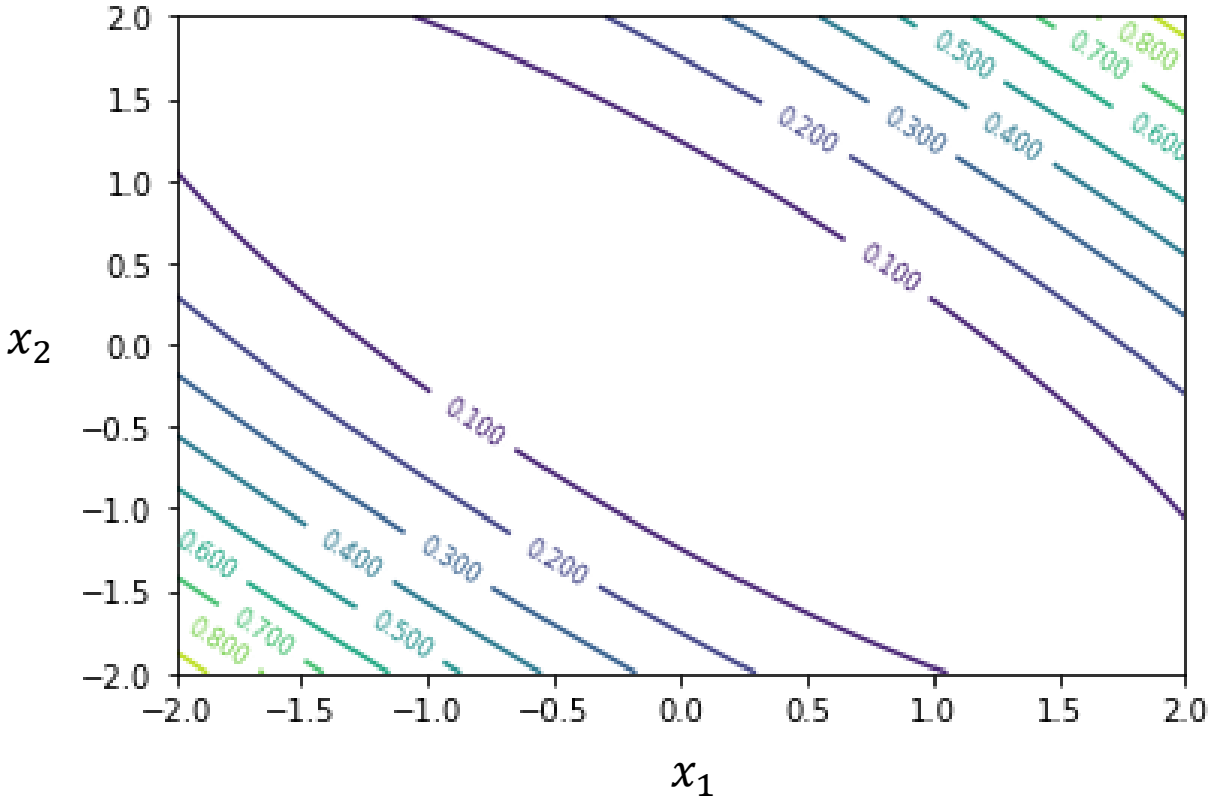
$$f(\boldsymbol{x}) = [x_1, x_2] \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} 12 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Clockwise 45°

Counter-clockwise 45°

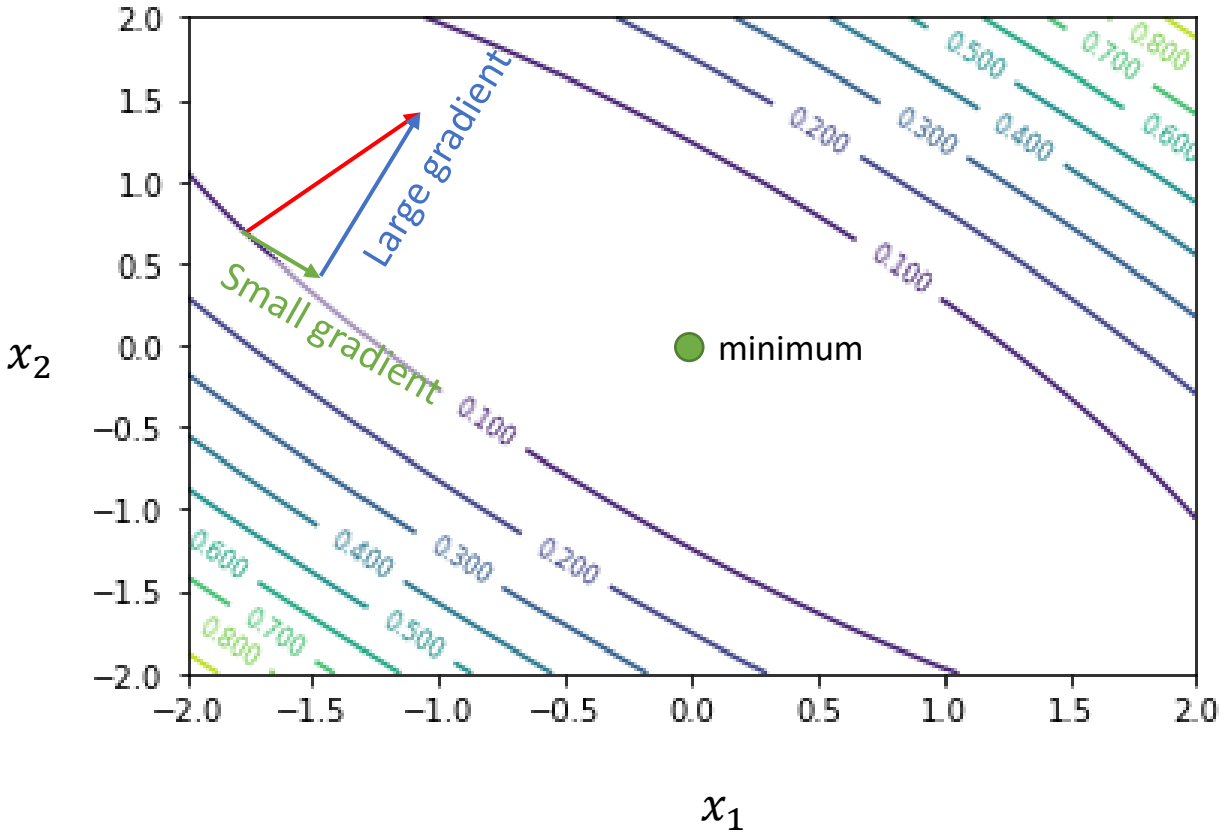
$$f(\boldsymbol{x}) = \boldsymbol{x}^\top Q \Lambda Q^\top \boldsymbol{x}$$

The eigenvalues of Hessian determine the ratio of the two axes of the ellipse.



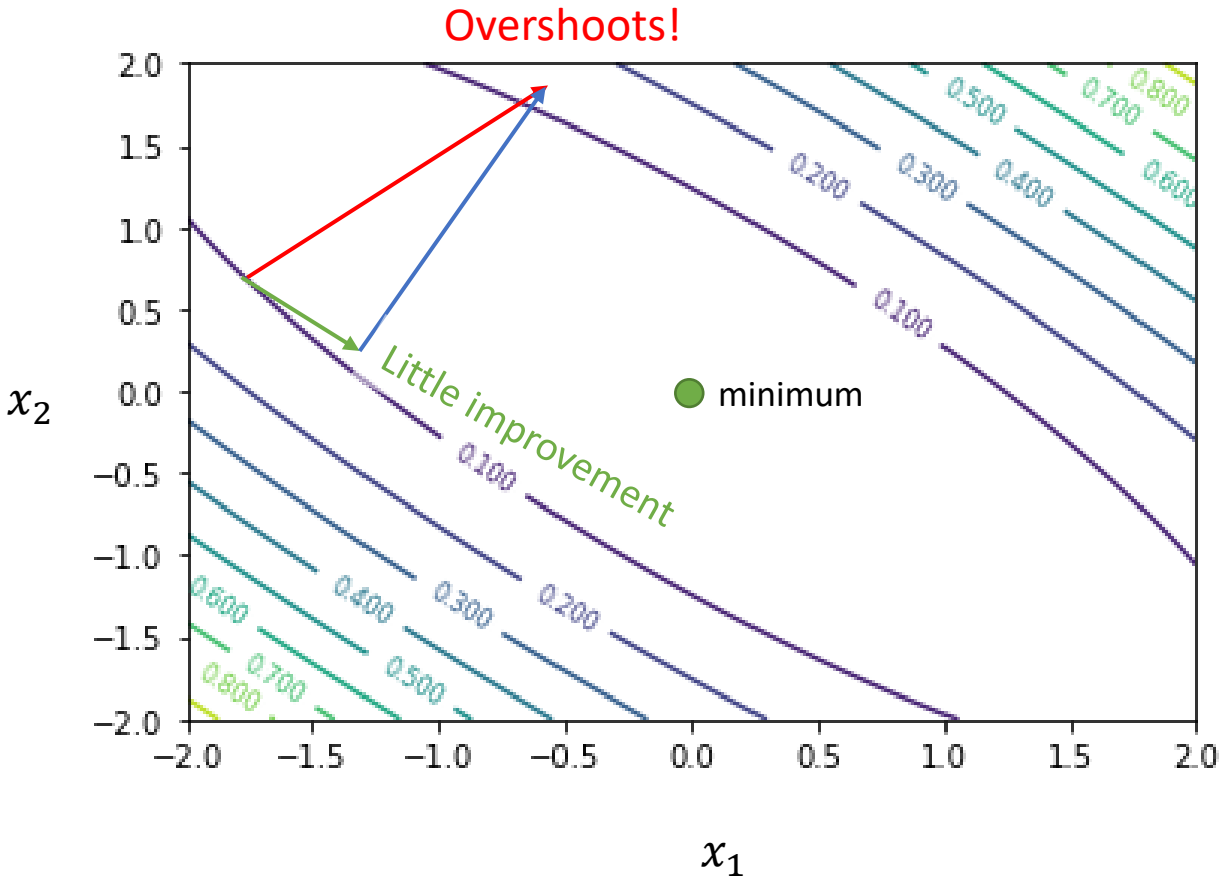
Hessian = Poorly Conditioned

- When the ellipse is really “squashed”, the gradient on one direction is much greater than the other dimension.
- We need to set a learning rate λ such that we don’t overshoot in one dimension.
- But it would be too small for fast convergence on the other dimension.
- A large condition number leads to slow convergence for GD.



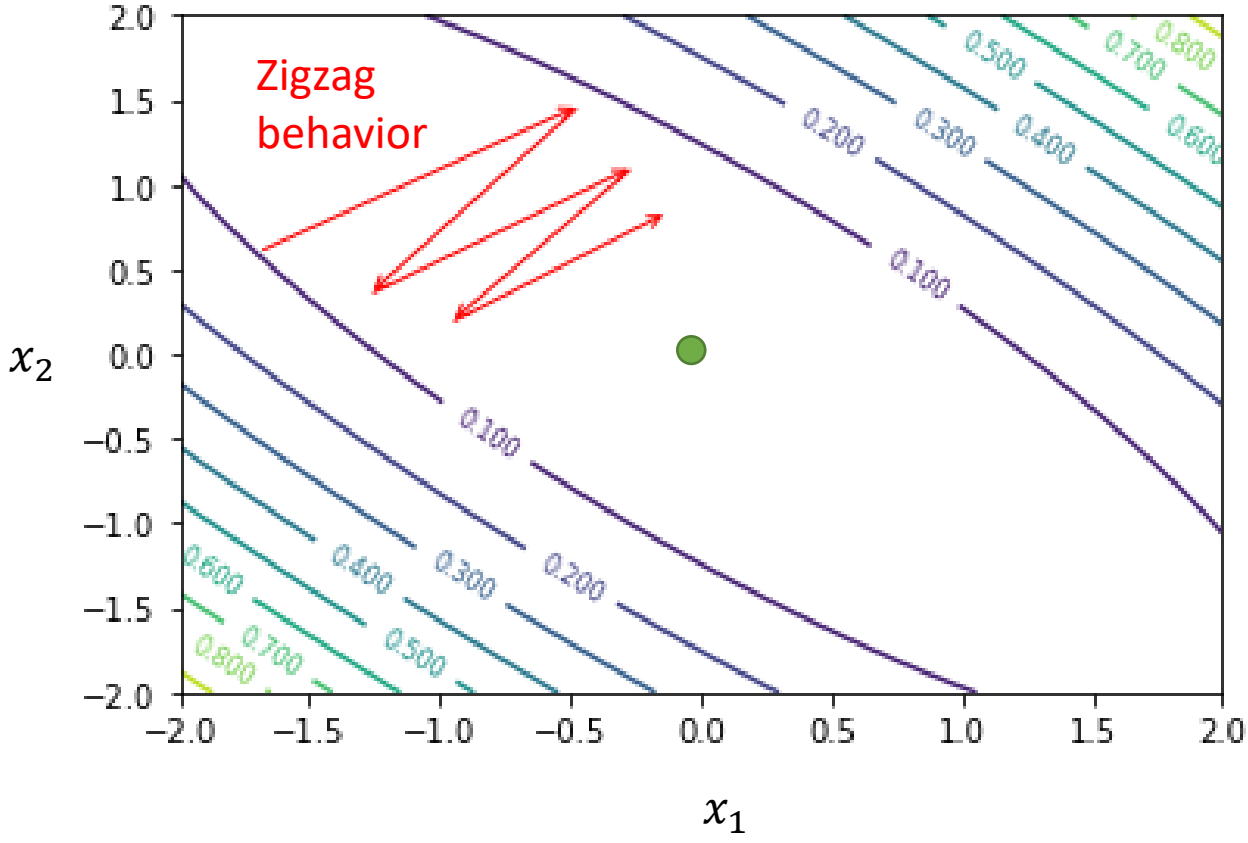
Hessian = Poorly Conditioned

- When the ellipse is really "squashed", the gradient on one direction is much greater than the other dimension.
- We need to set a learning rate λ such that we don't overshoot in one dimension.
- But it would be too small for fast convergence on the other dimension.
- A large condition number leads to slow convergence for GD.



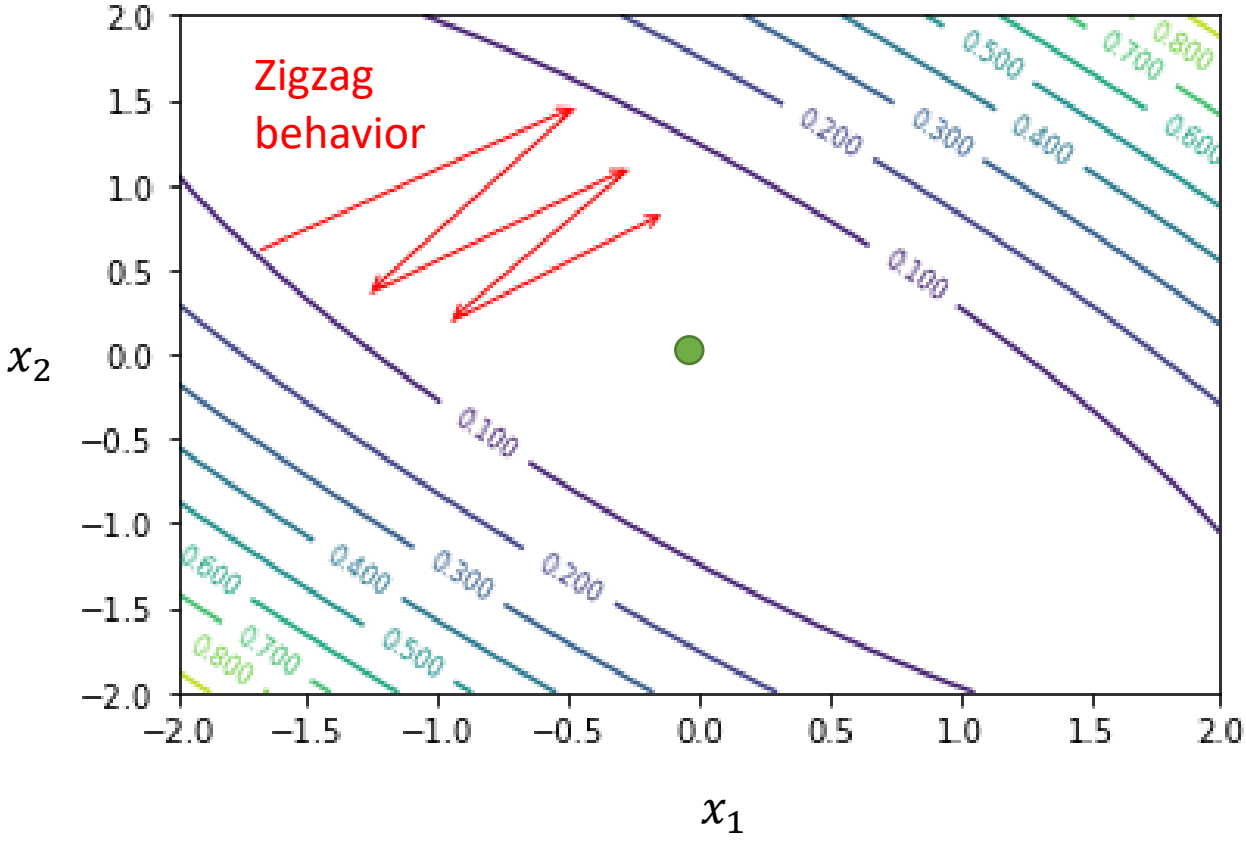
Hessian = Poorly Conditioned

- When the ellipse is really “squashed”, the gradient on one direction is much greater than the other dimension.
- We need to set a learning rate λ such that we don’t overshoot in one dimension.
- But it would be too small for fast convergence on the other dimension.
- A large condition number leads to slow convergence for GD.



Hessian = Poorly Conditioned

- This phenomenon is characterized by the ratio $\frac{\lambda_{\max}}{\lambda_{\min}}$ of the Hessian matrix, known as the condition number.
- A large condition number leads to slow convergence for GD.



Momentum

Initial condition:

$$\boldsymbol{v}_0 = \mathbf{0}$$

Velocity update:

$$\boldsymbol{v}_t = \alpha \boldsymbol{v}_{t-1} - \eta \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}, 0 < \alpha < 1$$

Parameter update:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \boldsymbol{v}_t$$

- Let $\boldsymbol{g}_t = \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}$
- $\boldsymbol{v}_0 = \mathbf{0}$
- $\boldsymbol{v}_1 = -\eta \boldsymbol{g}_1$
- $\boldsymbol{v}_2 = -\eta(\alpha \boldsymbol{g}_1 + \boldsymbol{g}_2)$
- $\boldsymbol{v}_t = -\eta\left(\sum_{i=1}^{t-1} \alpha^{t-k} \boldsymbol{g}_k + \boldsymbol{g}_t\right)$
- The contribution of $\boldsymbol{g}_k$ decreases exponentially as t increases.
- This is known as exponential moving average (EMA).

Momentum

Initial condition:

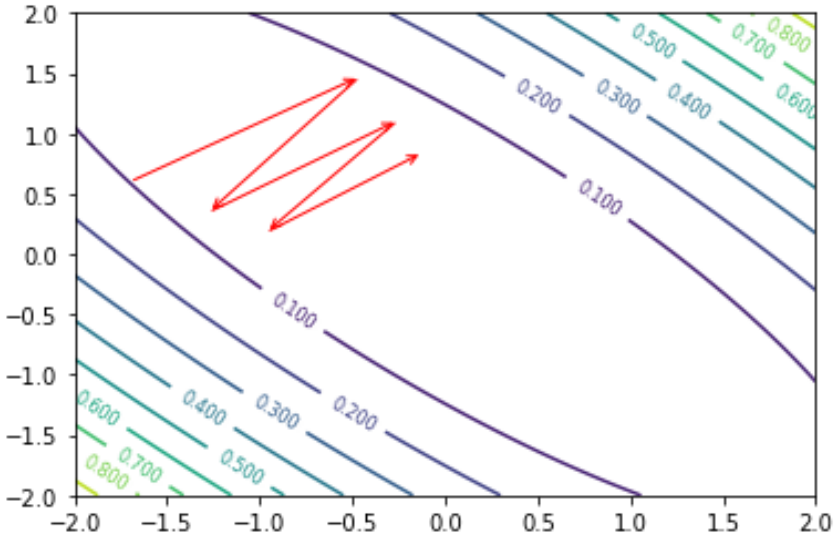
$$\boldsymbol{v}_0 = \mathbf{0}$$

Velocity update:

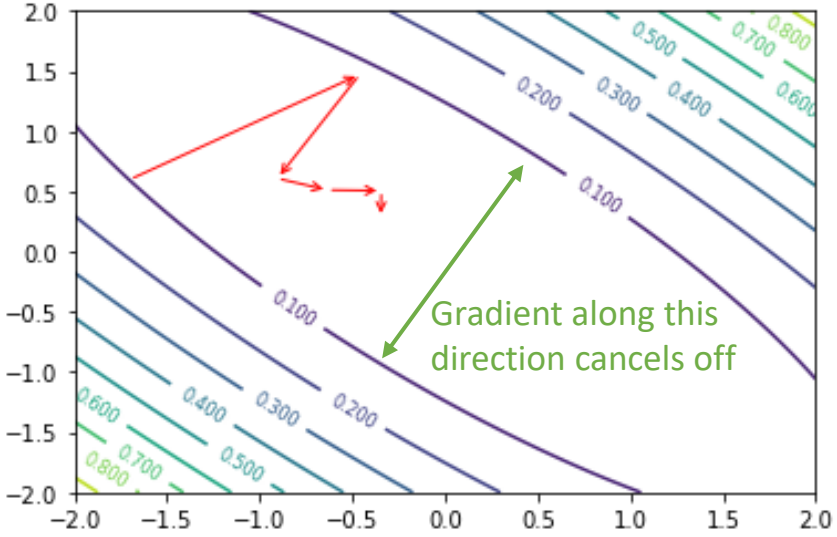
$$\boldsymbol{v}_t = \alpha \boldsymbol{v}_{t-1} - \eta \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}, 0 < \alpha < 1$$

Parameter update:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \boldsymbol{v}_t$$



$\alpha = 0$
(No momentum)



$\alpha = 0.3$

Momentum

Initial condition:

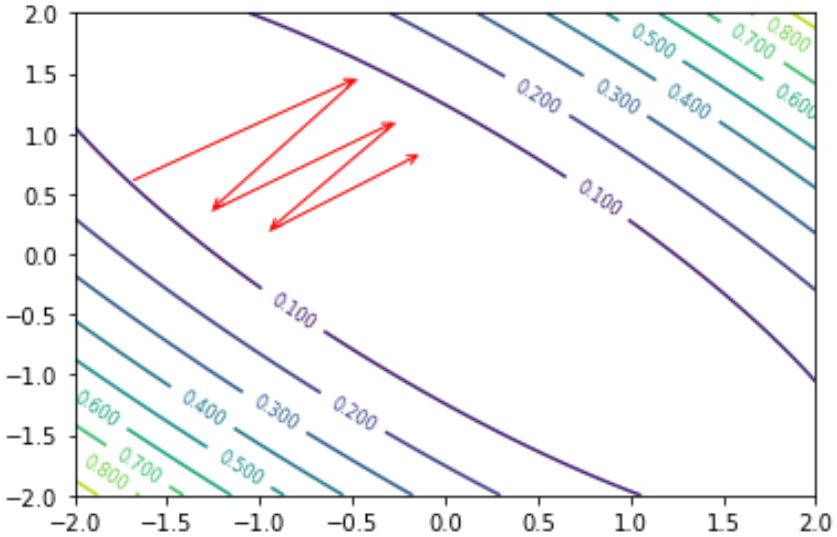
$$\boldsymbol{v}_0 = \mathbf{0}$$

Velocity update:

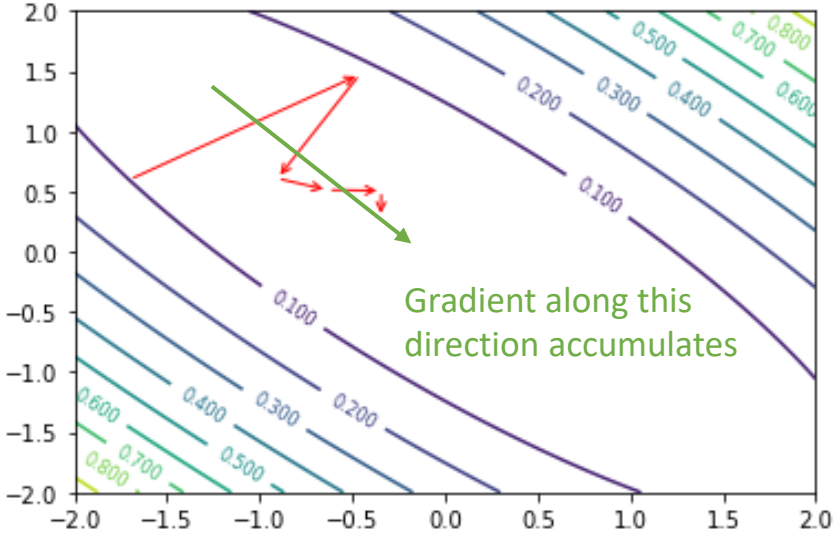
$$\boldsymbol{v}_t = \alpha \boldsymbol{v}_{t-1} - \eta \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}, 0 < \alpha < 1$$

Parameter update:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \boldsymbol{v}_t$$



$\alpha = 0$
(No momentum)



$\alpha = 0.3$

Momentum

Initial condition:

$$\boldsymbol{v}_0 = \mathbf{0}$$

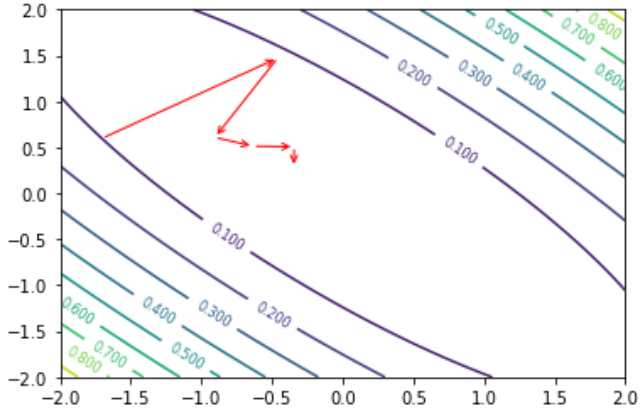
Velocity update:

$$\boldsymbol{v}_t = \alpha \boldsymbol{v}_{t-1} - \eta \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}, 0 < \alpha < 1$$

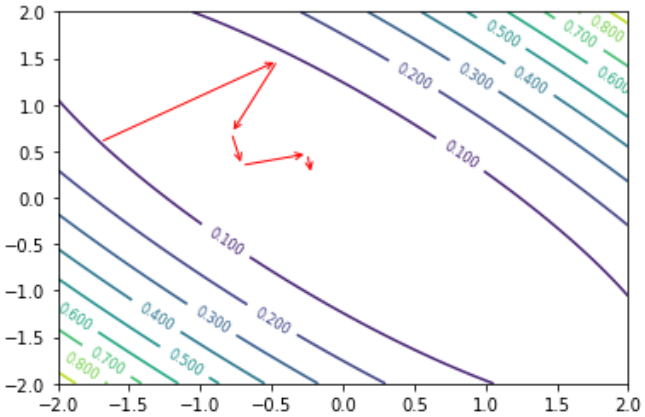
Parameter update:

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} + \boldsymbol{v}_t$$

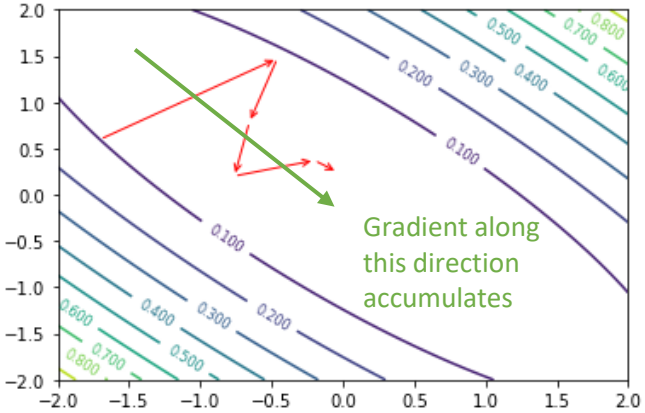
$\alpha = 0.3$



$\alpha = 0.4$



$\alpha = 0.5$



Effects of Data Whitening

- Whitening data = setting per-feature mean to 0 and variance to 1
- For each feature, we subtract its mean and divide by its standard deviation. These two values are computed from the entire training set.
- It is very commonly used in machine learning and has modern variants in deep learning.
- Why does it work?

Effects of Data Whitening

- Consider the loss function of linear regression with data matrix $X \in \mathbb{R}^{n\times(p+1)}$, label vector $\boldsymbol{y} \in \mathbb{R}^{n\times 1}$, and model parameter $\boldsymbol{\beta}$

$$\mathrm{L} = \frac{1}{n}(X\beta - y)^{\top}(X\beta - y)$$

$$\mathrm{L} = \frac{1}{n}\beta^{\top}X^{\top}X\beta - 2y^{\top}X\beta + \boldsymbol{y}^{\top}\boldsymbol{y}$$

$$\frac{d^2L}{d\beta^2} = \frac{1}{n}X^{\top}X \in \mathbb{R}^{(p+1)\times(p+1)}$$

The (uncentered) feature covariance matrix

- The condition number of the feature covariance matrix affects the speed of convergence.

Effects of Data Whitening

- Whitening data has the effect of lowering the condition number and improving optimization.
- Modern deep neural networks utilize a variety of normalization operations (e.g., BatchNorm, LayerNorm, GroupNorm, etc)
  - We will see them later on.
- [Optional] See detailed discussion at https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2021/readings/L01_intro.pdf

Adam

- Sometimes the gradient is too large and overshoots.
- Sometimes the gradient is too small, and optimization stagnates.
- What if we adjust the gradient so it's neither too large nor too small?
- Adam often works well for recurrent networks (introduced later).

- Input: loss function $L(\boldsymbol{w})$ and initial position $\boldsymbol{w}_0$
- Initialize: $\boldsymbol{s}_0 = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{0}, t = 0$
- Hyperparameters: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$
- Repeat until termination condition
  - $t = t + 1$
  - $\boldsymbol{g}_t = \frac{\mathrm{d}L(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}}\Big|_{\boldsymbol{w}_{t-1}}$
  - $\boldsymbol{s}_t = \beta_1 \boldsymbol{s}_{\mathrm{t}-1} + (1-\beta_1)\boldsymbol{g}_t$ // EMA for $\boldsymbol{g}_t$
  - $\boldsymbol{r}_t = \beta_2 \boldsymbol{r}_{\mathrm{t}-1} + (1-\beta_2)\|\boldsymbol{g}_t\|^2$ // EMA for $\|\boldsymbol{g}_t\|^2$
  - $\hat{\boldsymbol{s}}_t = \boldsymbol{s}_t/(1-\beta_1^t)$ // bias correction for $\boldsymbol{s}_t$
  - $\hat{\boldsymbol{r}}_t = \boldsymbol{r}_t/(1-\beta_2^t)$ // bias correction for $\boldsymbol{r}_t$
  - $\Delta\boldsymbol{w} = -\frac{\eta \hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t}+\epsilon}$ // component-wise division
  - $\boldsymbol{w}_{\mathrm{t}} = \boldsymbol{w}_{\mathrm{t}-1} + \Delta\boldsymbol{w}$

Adam: Bias Correction

- Let $\boldsymbol{g}_t = \frac{dL(\boldsymbol{w}_{t-1})}{d\boldsymbol{w}_{t-1}}$
- $\boldsymbol{s}_0 = \boldsymbol{0}$
- $\boldsymbol{s}_1 = (1-\beta_1)\boldsymbol{g}_1$
- $\boldsymbol{s}_2 = (1-\beta_1)(\beta_1\boldsymbol{g}_1 + \boldsymbol{g}_2)$
- $\boldsymbol{s}_t = (1-\beta_1)\left(\sum_{i=1}^{t-1} {\beta_1}^{t-k}\boldsymbol{g}_k + \boldsymbol{g}_t\right)$

The first few $\boldsymbol{s}$ are too small

- Thus, we introduce bias correction.
- $\hat{\boldsymbol{s}}_t = \boldsymbol{s}_t/(1-\beta_1^t)$
- Since $0 < 1-\beta_1^t < 1$, $\|\hat{\boldsymbol{s}}_t\| > \|\boldsymbol{s}_t\|$
- When $t \longrightarrow \infty$, $\hat{\boldsymbol{s}}_t \longrightarrow \boldsymbol{s}_t$

Adam

- Adam may not converge well near the minimum.
- Near the minimum, the gradients have small magnitudes. So Adam scales the gradients up, which may lead to overshooting.
- Can be handled by
  - decaying the learning rate
  - or switching to SGD with momentum near the end.

- Input: loss function $L(\boldsymbol{w})$ and initial position $\boldsymbol{w}_0$
- Initialize: $\boldsymbol{s}_0 = \boldsymbol{0}, \boldsymbol{r}_0 = \boldsymbol{0}, t = 0$
- Hyperparameters: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$
- Repeat until termination condition is met
  - $t = t + 1$
  - $\boldsymbol{g}_t = \frac{\mathrm{d}L(\boldsymbol{w})}{\mathrm{d}\boldsymbol{w}}\Big|_{\boldsymbol{w}_{t-1}}$
  - $\boldsymbol{s}_t = \beta_1 \boldsymbol{s}_{\mathrm{t}-1} + (1-\beta_1)\boldsymbol{g}_t$ // EMA for $\boldsymbol{g}_t$
  - $\boldsymbol{r}_t = \beta_2 \boldsymbol{r}_{\mathrm{t}-1} + (1-\beta_2)\|\boldsymbol{g}_t\|^2$ // EMA for $\|\boldsymbol{g}_t\|^2$
  - $\hat{\boldsymbol{s}}_t = \boldsymbol{s}_t/(1-\beta_1^t)$ // bias correction for $\boldsymbol{s}_t$
  - $\hat{\boldsymbol{r}}_t = \boldsymbol{r}_t/(1-\beta_2^t)$ // bias correction for $\boldsymbol{r}_t$
  - $\Delta\boldsymbol{w} = -\eta \frac{\hat{\boldsymbol{s}}_t}{\sqrt{\hat{\boldsymbol{r}}_t}+\epsilon}$ // component-wise division
  - $\boldsymbol{w}_{\mathrm{t}} = \boldsymbol{w}_{\mathrm{t}-1} + \Delta\boldsymbol{w}$

Newton's Method

- Newton's method is a second-order optimization technique that models the objective function as quadratic.
- Starting from $\boldsymbol{w}_0$, we seek Δ that minimizes

$$f(\boldsymbol{w}_0 + \Delta) = f(\boldsymbol{w}_0) + \left(\frac{df}{d\boldsymbol{w}}\right)^{\mathsf{T}} \Delta + \frac{1}{2}\Delta^{\mathsf{T}} \left(\frac{d^2 f}{d\boldsymbol{w}^2}\right) \Delta$$

- Setting the derivative to zero, we get

$$\frac{\mathrm{d}f(\boldsymbol{w}_0 + \Delta)}{d\Delta} = \left(\frac{df}{d\boldsymbol{w}}\right) + \left(\frac{d^2 f}{d\boldsymbol{w}^2}\right)\Delta = 0 \;\Rightarrow\; \Delta = -\left(\frac{d^2 f}{d\boldsymbol{w}^2}\right)^{-1} \left(\frac{df}{d\boldsymbol{w}}\right) = -H^{-1} g$$

Newton's Method

- Newton's step

$$\Delta = -\left(\frac{d^2 f}{d\boldsymbol{w}^2}\right)^{-1}\left(\frac{df}{d\boldsymbol{w}}\right) = -H^{-1}g$$

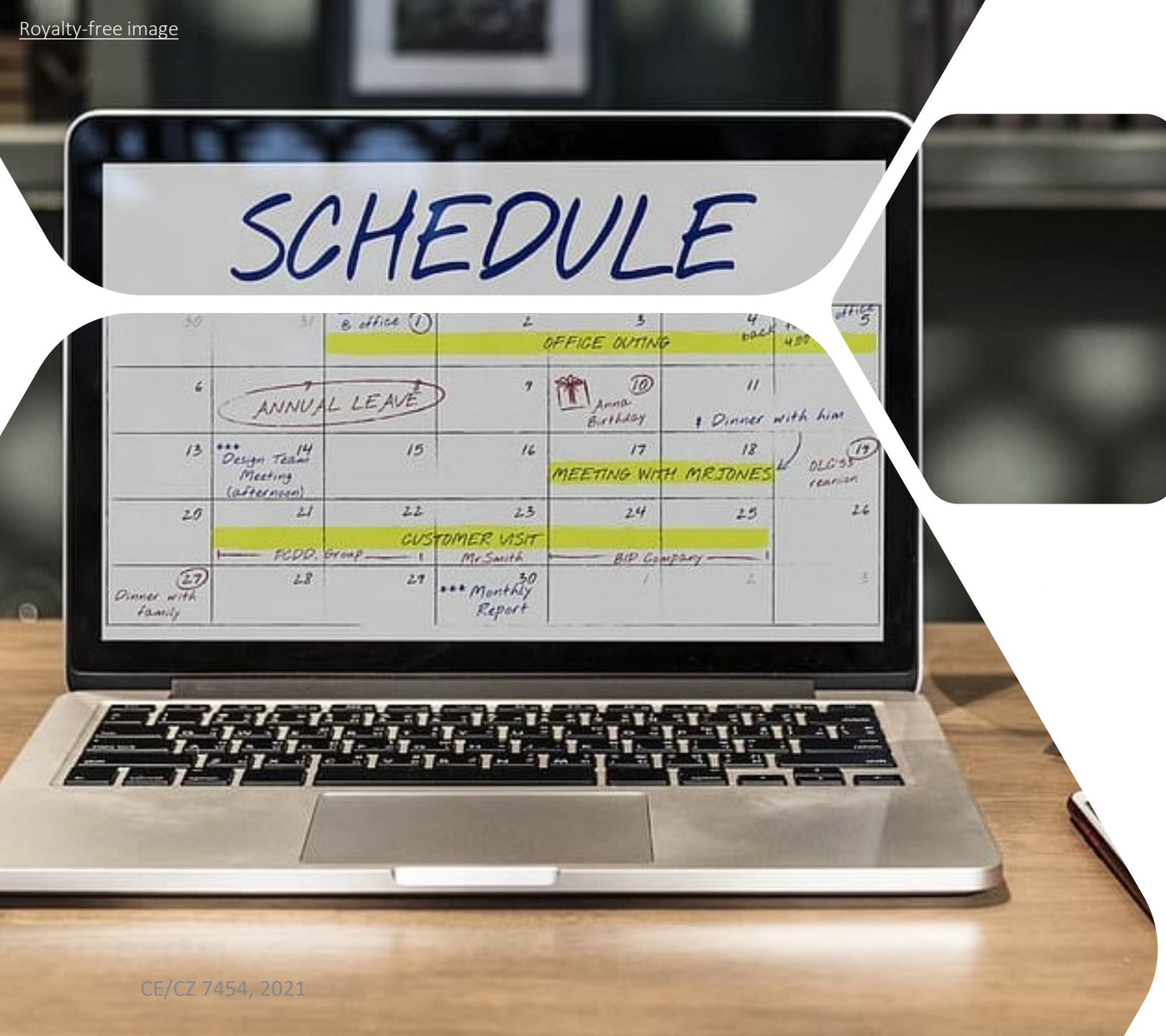
- If the function is indeed quadratic, Newton's method converges in a single step.
- Superior convergence speed compared to first-order gradient descent.

Newton's Method

- The Newton's method has many advantages, but it does not work well for deep neural networks
  - H is $n \times n$. Storage is hard for large n. For a network with 1M parameters, H takes a few TB to store
  - Finding H^{-1} naively take $O(n^3)$ operations, which very expensive for large n. (though can be simplified).
  - Estimating $H^{-1}g$ with sufficient accuracy requires a lot of data.

Royalty-free image

Li Boyang, Albert



Learning Rate Schedule

Learning Rate Schedule

- First-order methods critically relies on the learning rate η.
- Typically, we gradually decrease η during the optimization.
- Commonly used learning rate schedules
  - Exponential
  - Step
  - Cosine
  - Plateau (dynamic schedule)
  - Many more…

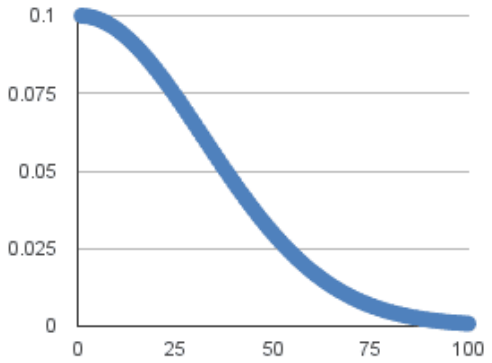
Learning Rate Schedule

- Exponential
  - $\eta_t = \alpha^t \eta_0, 0 < \alpha < 1$
- Step
  - Decrease by the factor α at predefined iteration / epoch

- This step function is popular in the training of convolutional networks for image recognition.
  - Total number of epochs = T
  - Multiply η by 1/10 at $\frac{T}{2}$, and another 1/10 at $\frac{3T}{4}$

Learning Rate Schedule

- Cosine
  - $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{\max} - \eta_{min})\left(1 + \cos\frac{t}{T}\pi\right)$
  - A half cycle of the cosine function
  - Slow decrease at the beginning and at the end, but fast drop in the middle.



- Plateau
  - Decrease the learning rate by α when the training loss does not decrease by at least β in S epochs.
  - Useful as an exploration strategy when no prior knowledge is available.
  - May create difficulties for reproduction.

Li Boyang, Albert

Adding Stochasticity …

Stochastic Gradient Descent

- GD computes the gradient on the loss function

$$\mathcal{L} = \frac{1}{N}\sum_{i} \ell(x^{(i)}, y^{(i)}, w)$$

Summation over all training data

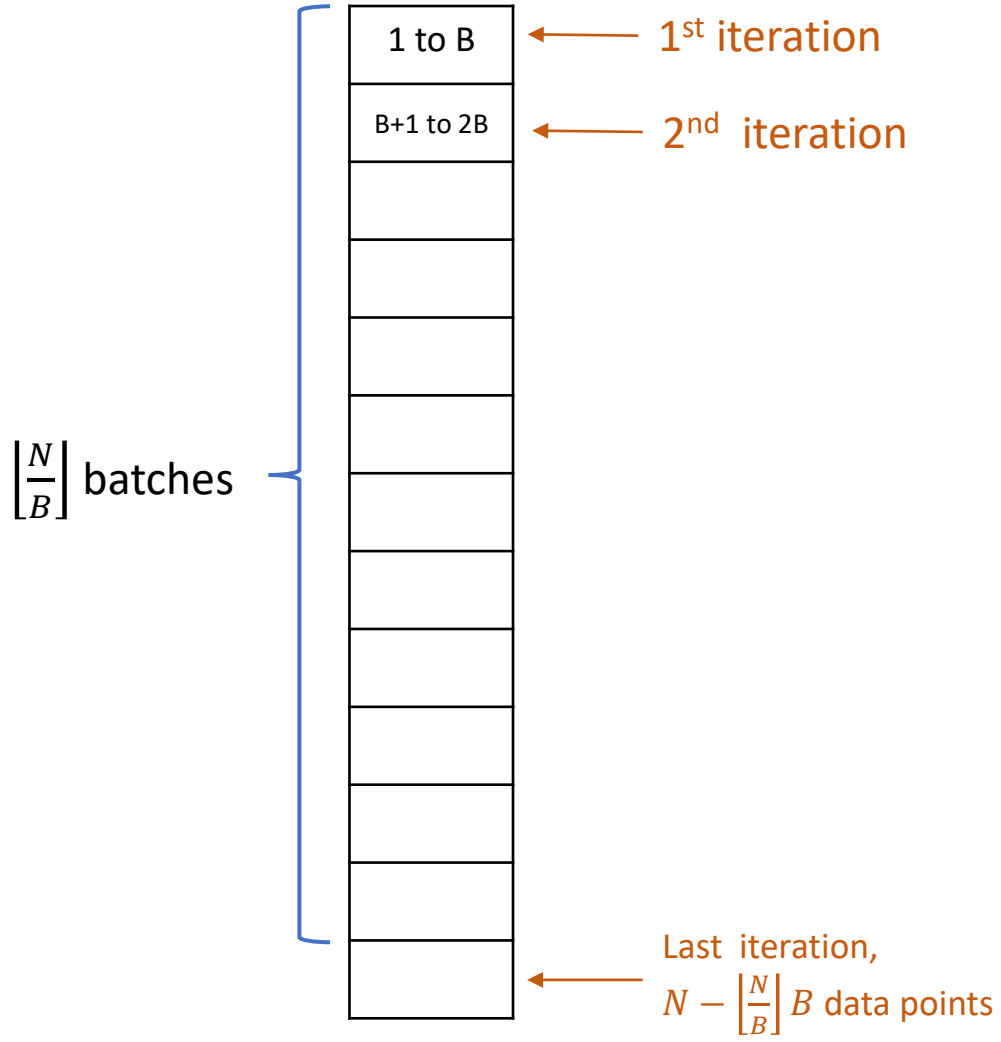
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{i} \frac{\partial \ell\left(x^{(i)}, y^{(i)}, \boldsymbol{w}\right)}{\partial \boldsymbol{w}}$$

- When there are too many training data, the above average is too expensive to compute.
- We use only a small subset randomly sampled from the training dataset to compute the gradient.

Implementation

1. Shuffle all training data into a random permutation.
2. Start from the beginning of the shuffled sequence.
3. In every iteration, take the next B data points, which form a mini-batch, and perform training on the mini-batch.
4. When the sequence is exhausted, go back to Step 1. This is one epoch of training.

Random shuffling is necessary for unbiased gradient estimates.



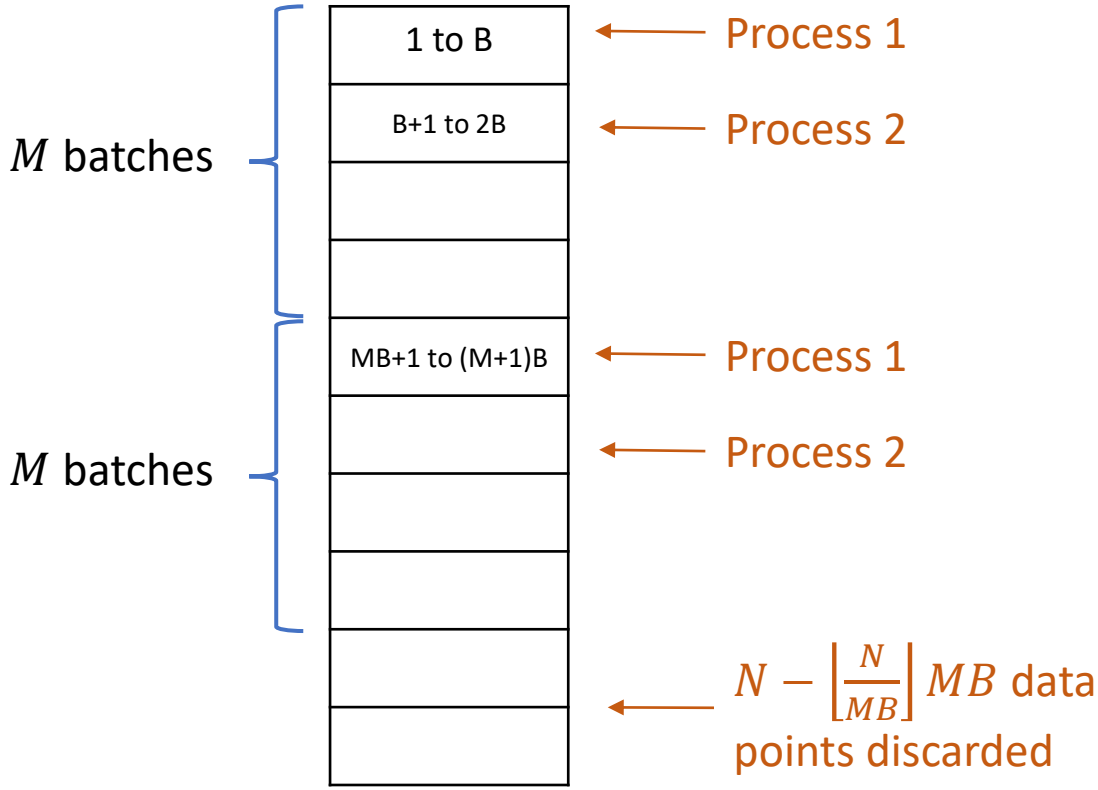
The last batch with fewer than B data points may be discarded if small batches cause problems (e.g., for BatchNorm)

Pseudo-random Number Generator (PNG)

- Pseudo-random number generators are not really “random”.
  - What is really random? A philosophical discussion that we will not get into.
- It is a completely deterministic function of the random seed.
  - Always generates the same sequence if the seed is the same.
  - A.k.a deterministic random bit generator
- However, the sequence of numbers satisfy certain statistical properties that they can be considered random if the seed is not known.
  - Do not change seeds if you want good randomness. Use the sequence.
- If we give the same seed to two generators w/ the same algorithm, they will create the same sequence.
  - We can align multiple processes or multiple training sessions.

Synchronous Training on Multiple GPUs / Nodes

- Multiple processes (total number = M) use PNGs sharing the same seed when shuffling the dataset.
- Based on its process id, a process uses one batch of size B every M batches.
- The gradients from M processes are averaged and the model is updated once (synchronization).
- Equivalent to a global batch size of MB



Stochastic Gradient Descent

- From a probabilistic perspective, $\left(x^{(i)}, y^{(i)}\right)$ is drawn from the data distribution $\mathcal{D}$. $\bar{g}$ is the expected value of the gradient.

Recall: The variance of the mean estimator is inversely proportional to sample size.

$$\bar{g} = E_{\left(x^{(i)}, y^{(i)}\right)\sim\mathcal{D}}\left[\frac{\partial \ell\left(x^{(i)}, y^{(i)}, \boldsymbol{w}\right)}{\partial \boldsymbol{w}}\right]$$

Expected gradient

$$\hat{g}_{\text{all-data}} = \frac{1}{N}\sum_{i}\frac{\partial \ell\left(x^{(i)}, y^{(i)}, \boldsymbol{w}\right)}{\partial \boldsymbol{w}}$$

Estimated gradient using all available data

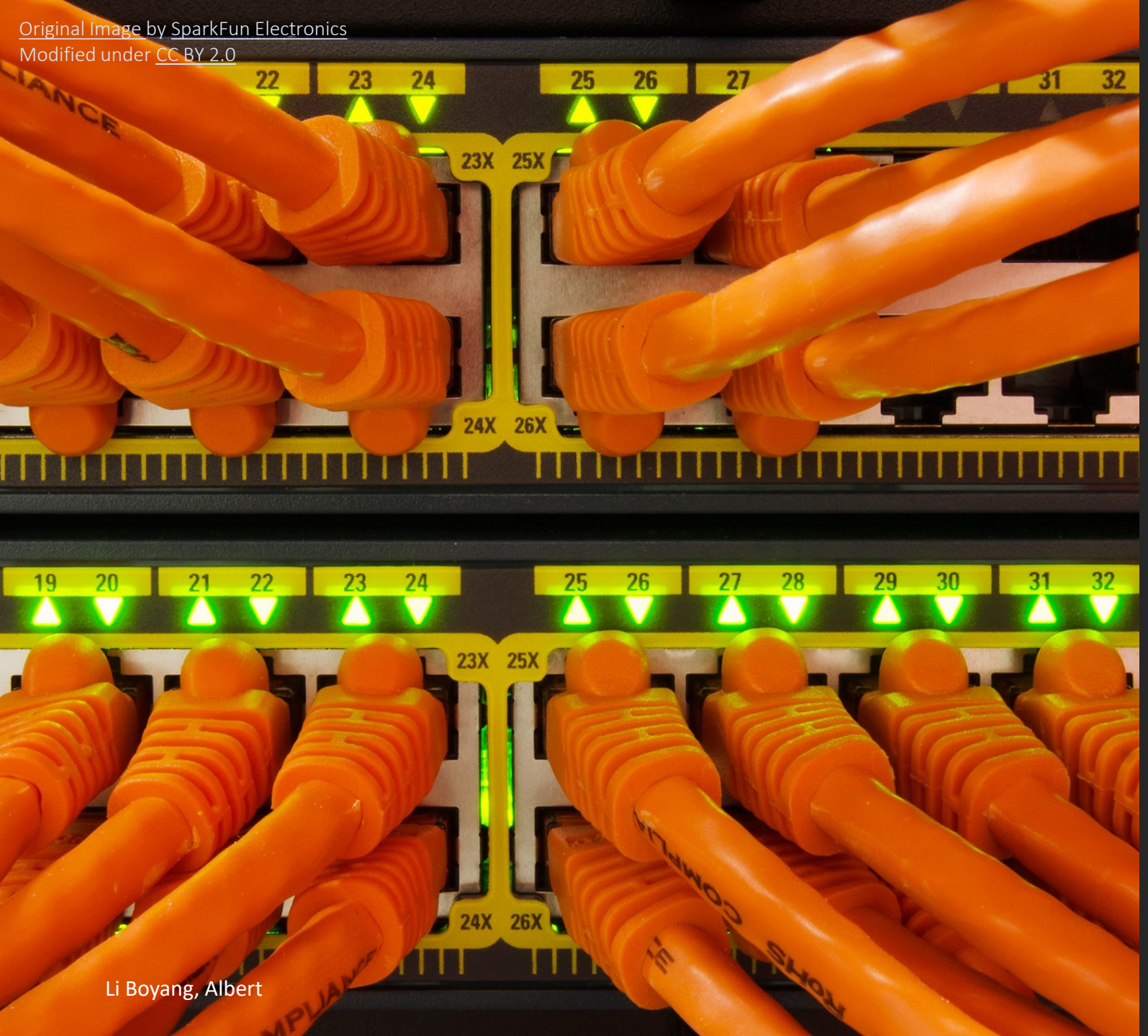
$$\hat{g}_{\text{mini-batch}} = \frac{1}{B}\sum_{i\in\mathcal{B}}\frac{\partial \ell\left(x^{(i)}, y^{(i)}, \boldsymbol{w}\right)}{\partial \boldsymbol{w}}$$

Estimated gradient using all data from a mini-batch $\mathcal{B}$

Stochastic Gradient Descent

$$\hat{g}_{\text{mini-batch}} = \frac{1}{B}\sum_{i\in\mathcal{B}} \frac{\partial \ell\left(x^{(i)}, y^{(i)}, \boldsymbol{w}\right)}{\partial \boldsymbol{w}} = \bar{g} + \epsilon$$

- The mini-batch estimate of the gradient contains some random noise. Is that bad?
  - Bias is bad. We need to shuffle datasets to get unbiased estimates.
  - Variance may not be too bad. As long as the angle between the estimated gradient $\hat{g}$ and true gradient $\bar{g}$ is less than 90 degrees, we are still roughly in the right direction.
  - Getting more accurate gradient means a bigger batch, which can be more expensive



Batch Size and Learning Rate

Effects of Batch Size

- $g_i(w_t) = \frac{d\ell(x^{(i)}, y^{(i)}, w_t)}{w_t}$ is the per-sample gradient.
- The mini-batch update $w_t = w_{t-1} - \eta\hat{g} = w_{t-1} - \frac{\eta}{B}\sum_{i\in\mathcal{B}} g_i(w_{t-1})$
- We can understand this as sequentially movements: $\frac{\eta g_1(w_{t-1})}{B} + \frac{\eta g_2(w_{t-1})}{B} + \cdots + \frac{\eta g_B(w_{t-1})}{B}$
- The contribution of the i^{th} data point to the model parameter w is: η/B
- We vary B while keeping $\alpha = \frac{\eta}{B}$ fixed (linear scaling of η).

If $B = 1$

$$w_1 = w_0 - \alpha g_1(w_0)$$
$$w_2 = w_1 - \alpha g_2(w_1)$$
$$w_3 = w_2 - \alpha g_3(w_2)$$

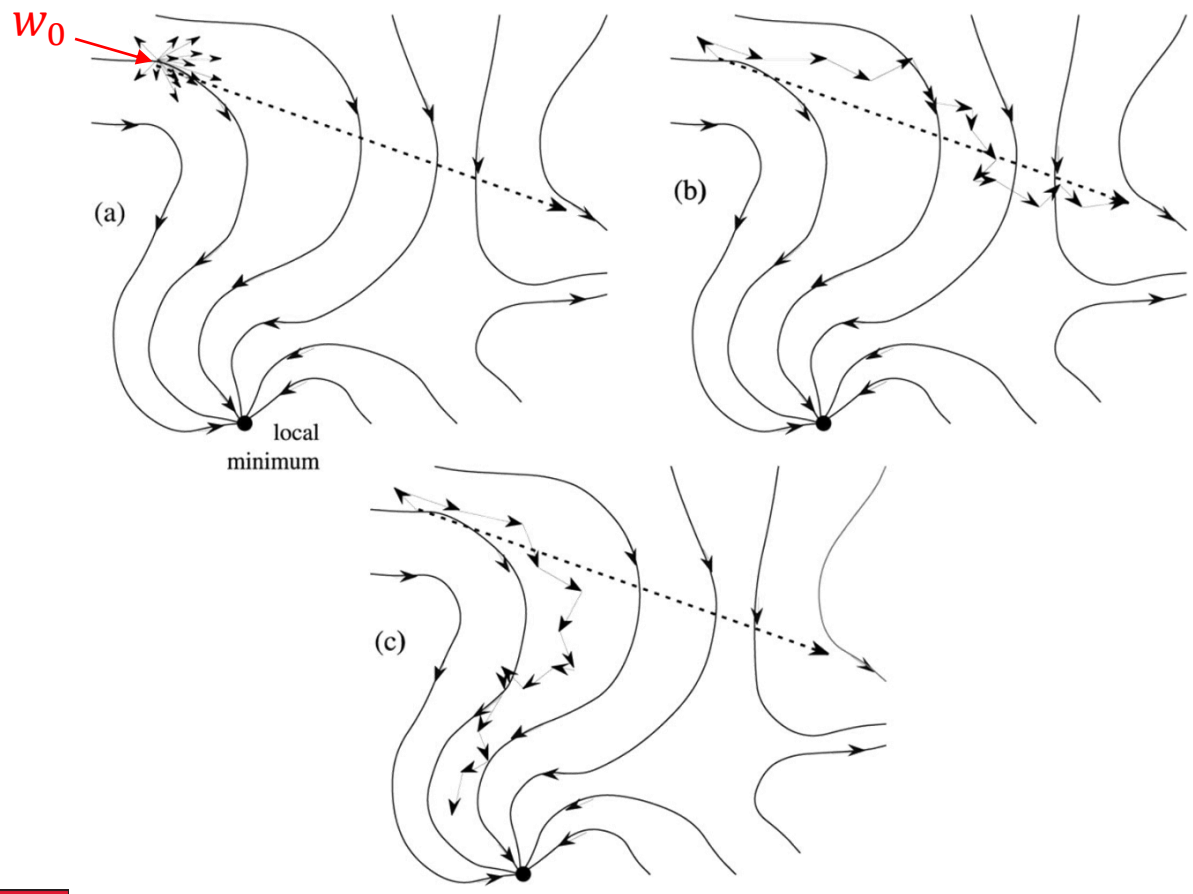
If $B = 3$

$$w_2 = w_0 - \alpha g_1(w_0) - \alpha g_2(w_0) - \alpha g_3(w_0)$$

$$w_{0,1} = w_0 - \alpha g_1(w_0)$$
$$w_{0,2} = w_{0,1} - \alpha g_2(w_0)$$
$$w_1 = w_{0,2} - \alpha g_3(w_0)$$

With a large B, we use stale gradients computed at an old w.

Effects of Batch Size



The directed curves indicate the underlying true gradient of the error surface.

(a) Batch training. Several weight change vectors and their sum.

(b) Batch training with weight change vectors placed end-to-end. As all vectors are computed at w_0, they do not reflect the landscape (i.e., "stale" gradients). We must use a smaller learning rate (sub-linear scaling of η).

(c) On-line training ($B = 1$). The local gradient influences the direction of each weight change vector, allowing it to follow curves.

Image from: D. Randall Wilsona and Tony R. Martinezb. The general inefficiency of batch training for gradient descent learning. 2003

Effects of Batch Size

- In practice, setting $B = 1$ often leads to very effective optimization.
- The variance does not matter much as long as the angle between the estimated gradient $\hat{g}$ and true gradient $\bar{g}$ is less than 90 degrees.
- Drawbacks of online learning ($B = 1$)
  - It fails to utilize modern GPU hardware, which is massively parallel.
  - Terrible statistics for BatchNorm
- It is usually desirable to use large batch sizes with the support of multiple GPU cards / nodes.

Large Batch Training

If $B = 1$

$$w_1 = w_0 - \alpha g_i(w_0)$$
$$w_2 = w_1 - \alpha g_{i+1}(w_1)$$
$$\vdots$$
$$w_K = w_{K-1} - \alpha g_{i+K}(w_{K-1})$$

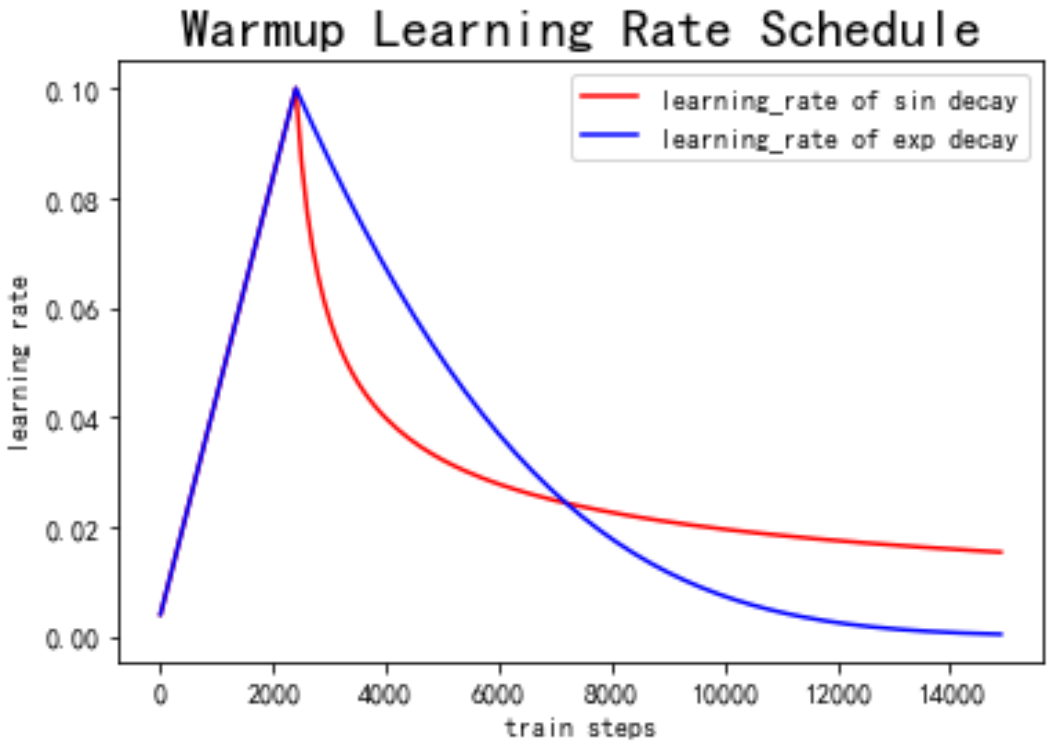
If $B = K$

$$w_{0,1} = w_0 - \alpha g_i(w_0)$$
$$w_{1,2} = w_{0,1} - \alpha g_{i+1}(w_0)$$
$$\vdots$$
$$w_1 = w_{0,K-1} - \alpha g_{i+K}(w_0)$$

- Under what conditions are the left side and the right side equivalent?
  - $\alpha = \frac{\eta}{B}$ is kept constant (Assumption 1)
  - The loss function is smooth enough, $g_i(w_{0,K-1}) \approx g_i(w_0)$ (Assumption 2)
- From Assumption 1, η and B should increase proportionally (linear scaling).
- Assumption 2 usually does not hold at the beginning of the optimization. We must use small learning rates at the beginning.

Learning Rate Warm-up

- Linear growth of learning rate from 0 to $\eta_{\max}$ in the first few epochs.
- $\eta_{\max}$ grows proportionally with batch size.
- Proposed in
  - Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017
- This technique allows batch size up to 8,000 for image recognition



LR Warm-up

LR decreases by factor of 10

- ResNet-50
- kn = batch size
- η = learning rate
- How to increase batch size even further?

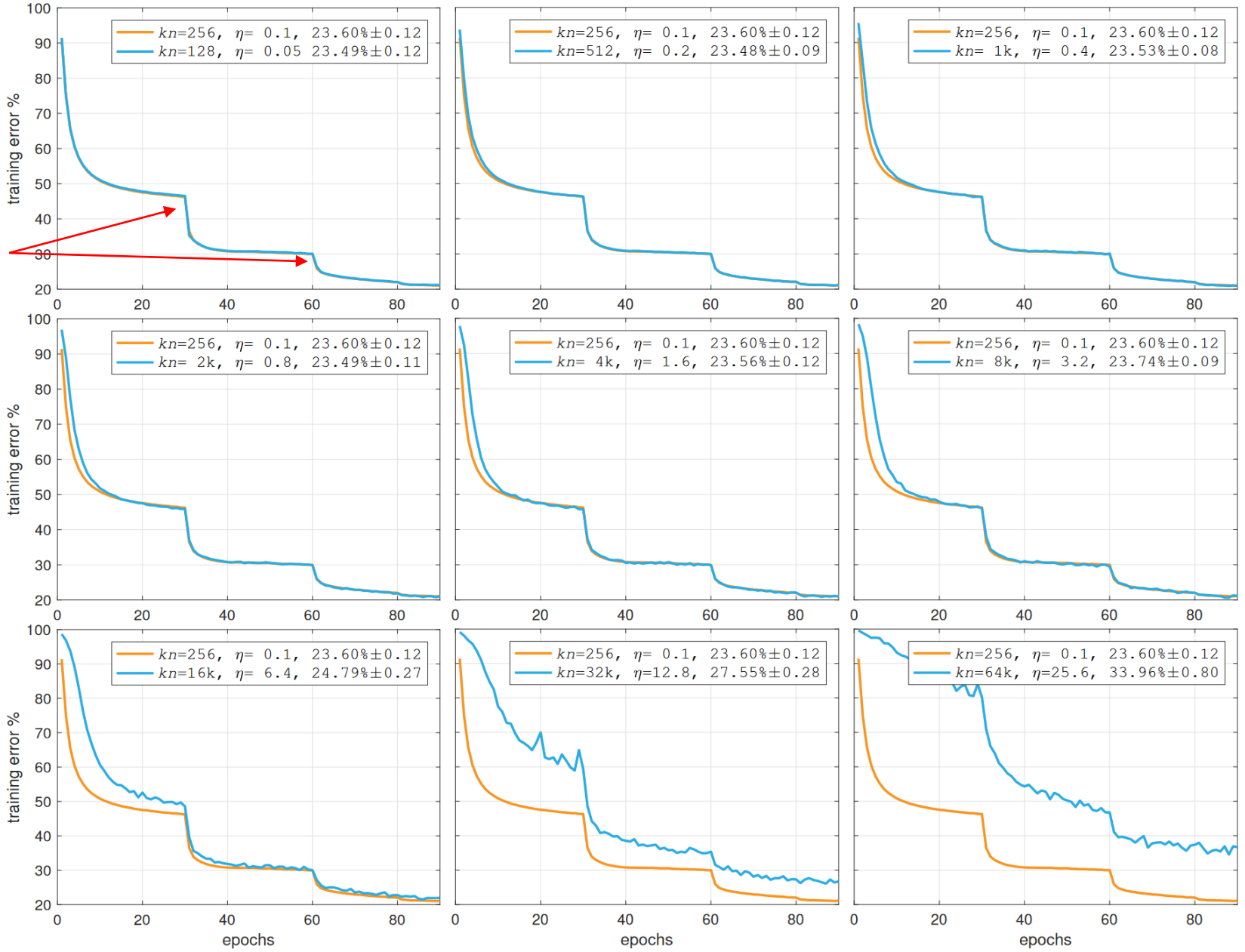
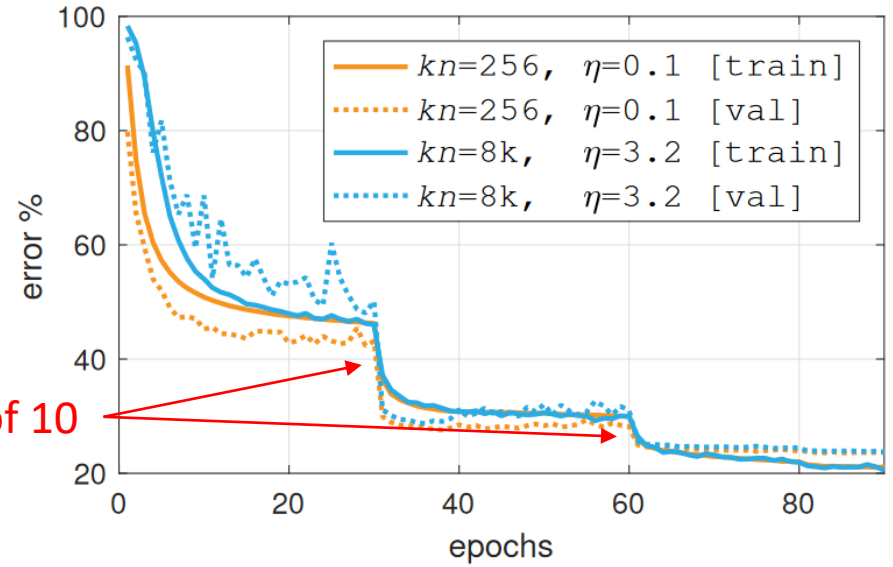


Figure 3. **Training error *vs.* minibatch size.** Training error curves for the 256 minibatch baseline and larger minibatches using gradual warmup and the linear scaling rule. Note how the training curves closely match the baseline (aside from the warmup period) up through 8k minibatches. *Validation* error (mean±std of 5 runs) is shown in the legend, along with minibatch size kn and reference learning rate η.

Image from: Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017

LR Warm-up



LR decreases by factor of 10

Figure 4. **Training and validation curves** for large minibatch SGD with gradual warmup *vs*. small minibatch SGD. Both sets of curves match closely after training for sufficient epochs. We note that the BN statistics (for inference only) are computed using *running* average, which is updated less frequently with a large minibatch and thus is noisier in early training (this explains the larger variation of the validation error in early epochs).

Image from Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. 2017

LARS

- Observation: gradient magnitudes differ at different network locations. Thus, optimization happens at different rates.
- Notation: $\nabla L(w) = \frac{\partial L}{\partial w}$

Table 2: AlexNet-BN: The norm of weights and gradients at 1st iteration.

Layer	conv1.b	conv1.w	conv2.b	conv2.w	conv3.b	conv3.w	conv4.b	conv4.w
$\|\|w\|\|$	1.86	0.098	5.546	0.16	9.40	0.196	8.15	0.196
$\|\|\nabla L(w)\|\|$	0.22	0.017	0.165	0.002	0.135	0.0015	0.109	0.0013
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	8.48	**5.76**	33.6	83.5	69.9	127	74.6	148
Layer	conv5.b	conv5.w	fc6.b	fc6.w	fc7.b	fc7.w	fc8.b	fc8.w
$\|\|w\|\|$	6.65	0.16	30.7	6.4	20.5	6.4	20.2	0.316
$\|\|\nabla L(w)\|\|$	0.09	0.0002	0.26	0.005	0.30	0.013	0.22	0.016
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	73.6	69	117	**1345**	68	489	93	19

Table from: Yang You, Igor Gitman, and Boris Ginsburg. *Large batch training of convolutional networks.* arXiv 1708.03888, 2017

Layer-wise Adaptive Rate Scaling (LARS)

- Proposal: Synchronize the learning speed of different layers and components.
- First, divide parameters into groups, $\mathbf{w}^1, \ldots, \mathbf{w}^l, \ldots, \mathbf{w}^L$
- For each group, apply updates with normalized gradients

$$w_{t+1}^l = w_t^l - \eta \frac{\nabla_{w_t} \mathcal{L}(\mathbf{w}_t^l) \, \|\mathbf{w}_t^l\|}{\|\nabla_{w_t} \mathcal{L}(\mathbf{w}_t^l)\|}$$

- Effect: $\frac{\|\Delta \mathbf{w}_t^l\|}{\|\mathbf{w}_t^l\|}$ is a constant η. At every iteration, a fixed portion of $\mathbf{w}_t^l$ is updated. This alleviates vanishing and exploding gradients.
- Scales batch size to 32K for ResNet-50.

Yang You, Igor Gitman, and Boris Ginsburg. *Large batch training of convolutional networks.* arXiv 1708.03888, 2017

Optimal Batch Size in Practice

- Too small a batch size
  - Fails to fully utilize parallel hardware (GPU).
  - Creates excessive noise for Batch Normalization, leading to unstable training
- Too large a batch size
  - Could create problems for optimization.
- Given enough GPUs, we usually use a large batch + Linear LR Scaling + LR warm-up in order to accelerate training

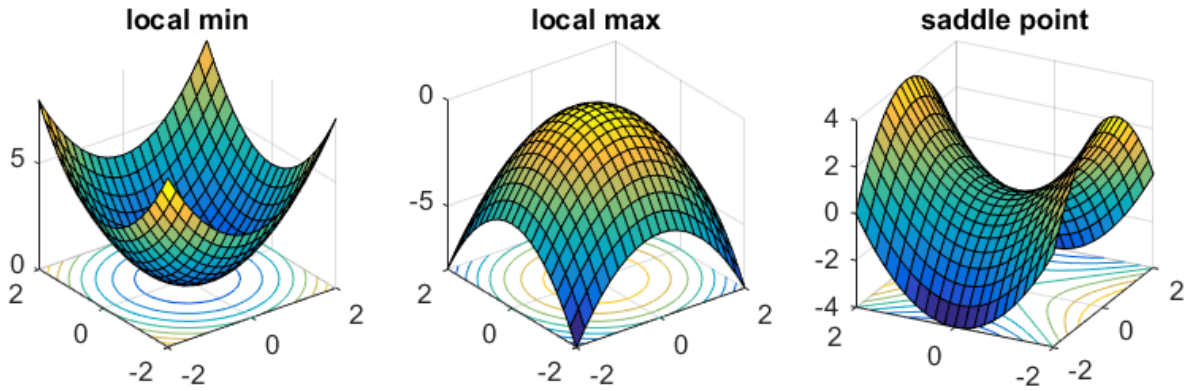
Li Boyang, Albert

Implicit Regularization of SGD

Benefits of SGD

- Could the noise provide some benefits to machine learning?
- Escaping from saddle points, local maxima, and (sometimes) bad local minima.
- Implicit regularization: finding flat optima
  - Machine learning is about generalization performance. Optimization is a means to an end.

Second-order Conditions



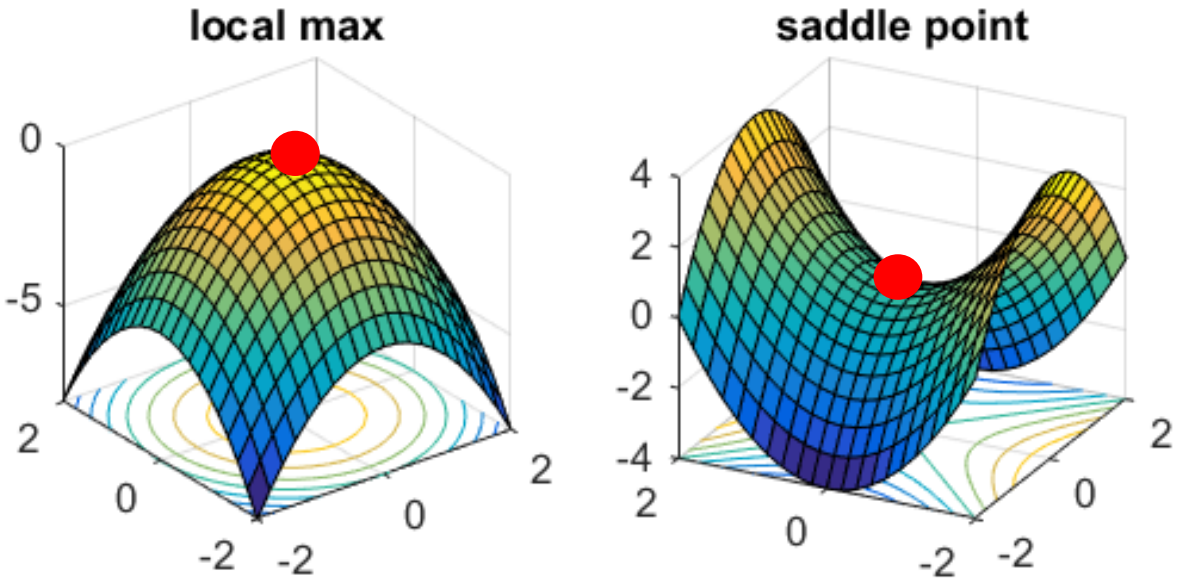
Hessian is PSD

Hessian is NSD

Hessian is neither PSD nor NSD

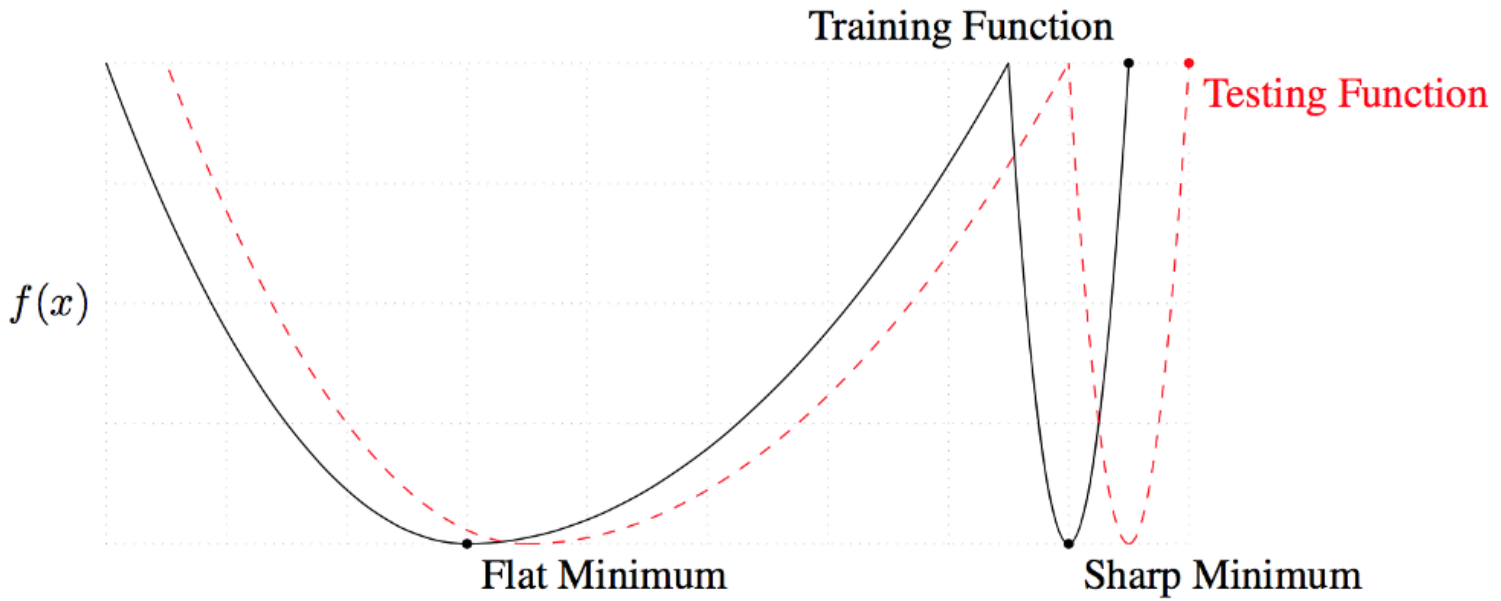
Image credit: Rong Ge (http://www.offconvex.org/2016/03/22/saddlepoints/)

Escaping with Random Noise



- The gradient is exactly zero at a local maximum and a saddle point.
- A small random error added to $\boldsymbol{w}$ allows escape from those points.

Convergence of SGD to Flat Minimum

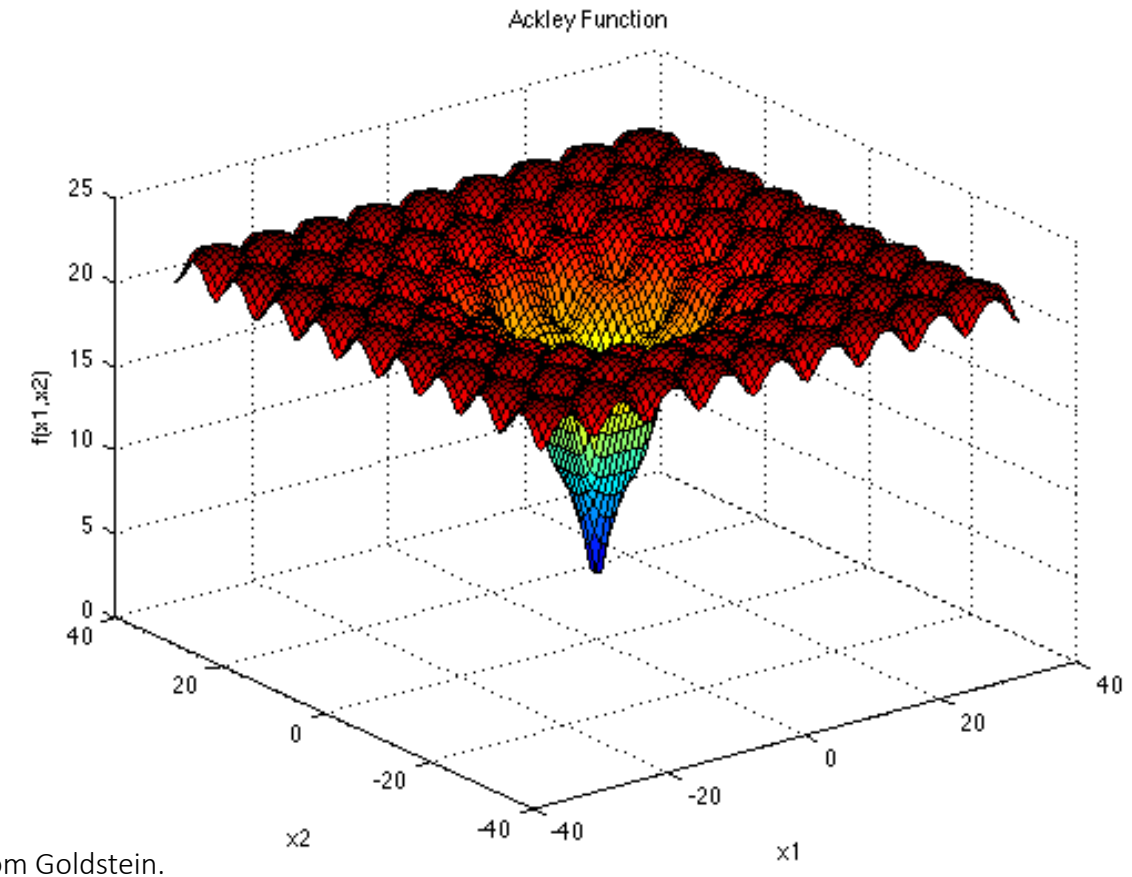


To fall into a sharp minimum, the gradient must be precise. The noise in SGD prevents that. It is hypothesized that flat minima generalize better.

Image credit: 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. ICLR 2017.

Effects of Multiple Local Minima

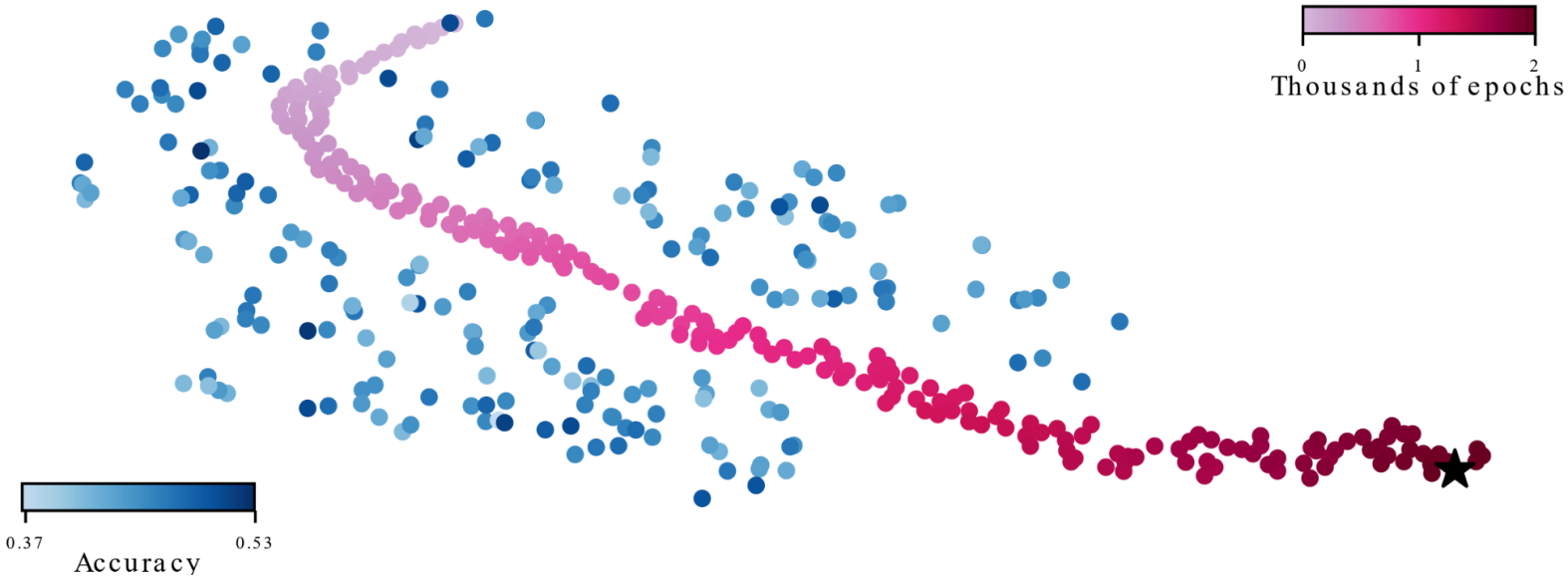
- How can we expect to find the global optimum?
- Answer 1: (circa 2000) Impossible. Therefore, deep learning cannot work.
- Answer 2: Every local minimum is as good as the global minimum (cf. [1]).



[1] Micah Goldblum, Jonas Geiping, Avi Schwarzschild, Michael Moeller, and Tom Goldstein. Truth or Backpropaganda? An Empirical Investigation of Deep Learning Theory. ICLR 2020.

Image from: W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K. Terry, Furong Huang, Tom Goldstein. Understanding Generalization through Visualizations. 2019

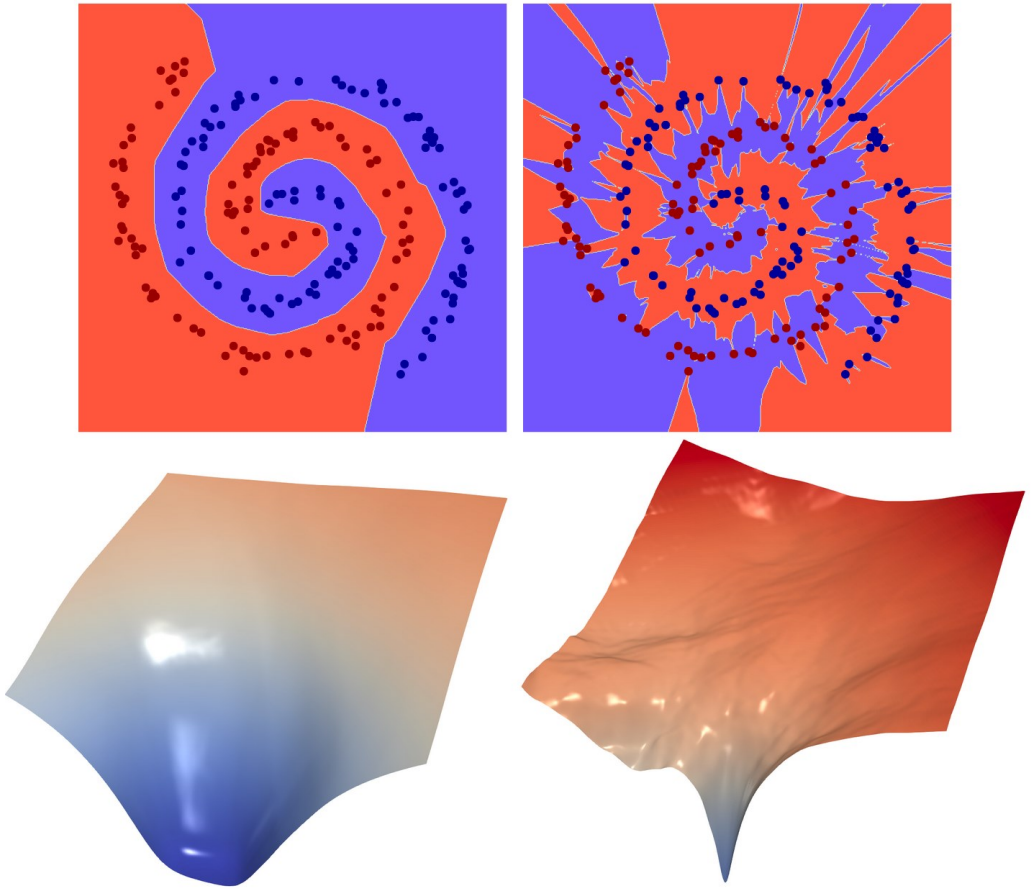
Dancing through a minefield of bad minima



- Red dots: the iterates of SGD after each tenth epoch.
- Blue dots: locations of nearby "bad" minima with poor generalization. They have near perfect train accuracy, but with test accuracy below 53% (random chance is 50%).
- The final iterate of SGD (black star) also achieves perfect train accuracy, but with 98.5% test accuracy. Miraculously, SGD always finds its way through a landscape full of bad minima, and lands at a minimizer with excellent generalization.

Images from: W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K. Terry, Furong Huang, Tom Goldstein. Understanding Generalization through Visualizations. 2019

Flat vs. Sharp Minima



- Decision boundaries of the same network with different parameters. Red and blue dots represent training data.
- The network on the left generalizes well. The network on the right generalizes poorly (perfect train accuracy, bad test accuracy).
- The flatness and large volume of the good minimizer make it likely to be found by SGD, while the sharpness and tiny volume of the bad minimizer make it unlikely to be found.

Image from: W. Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K. Terry, Furong Huang, Tom Goldstein. Understanding Generalization through Visualizations. 2019

Comparing Volume of Minima

- As the dimension of parameter space increases, the ratio between good and bad minima rises exponentially.
- On the SVHN data, Huang *et al.* find that bad minima are over 10,000 orders of magnitude smaller than good minima, making them nearly impossible to find.