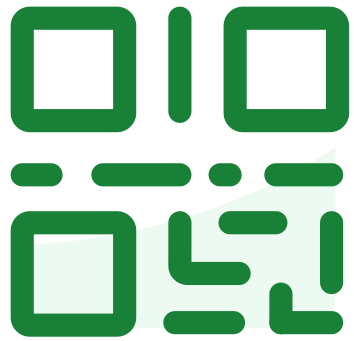


Do not edit
How to change the
design



**Join at slido.com
#3400896**

① The Slido app must be installed on every computer you're presenting from

slido

Conv. + Momentum + Adam + Stochastic Gradient Descent

The core algorithm of modern machine learning

EECS 189/289, Fall 2026 @ UC Berkeley

Joseph E. Gonzalez and Narges Norouzi



Error Minimization

Our goal is to **minimize the prediction error** of our model:

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathbb{R}^D} \frac{1}{N} \sum_{n=1}^N \mathbf{Loss}(f_{\mathbf{w}}(x_n), t_n)$$

- where $f_{\mathbf{w}}(x_n)$ is the prediction of the model $f_{\mathbf{w}}$ with parameters $\mathbf{w}$ and $\mathbf{Loss}(\hat{t}, t)$ is the loss function (e.g., L2-loss or CE-loss)

Because we are only interested in the **error as a function of the parameters $\mathbf{w}$** we typically rewrite the objective:

$$\mathbf{w}^* = \arg \min_{\mathbf{w} \in \mathbb{R}^W} E(\mathbf{w}; \mathcal{D})$$

- where $\mathcal{D}$ indicates that the error E depends on the data.



The (Batch) Gradient Descent Alg.

Update the weights by moving in the **opposite direction** of the gradient:

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\text{max}}$ ):  
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \nabla E(\mathbf{w}^{(\tau-1)})$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

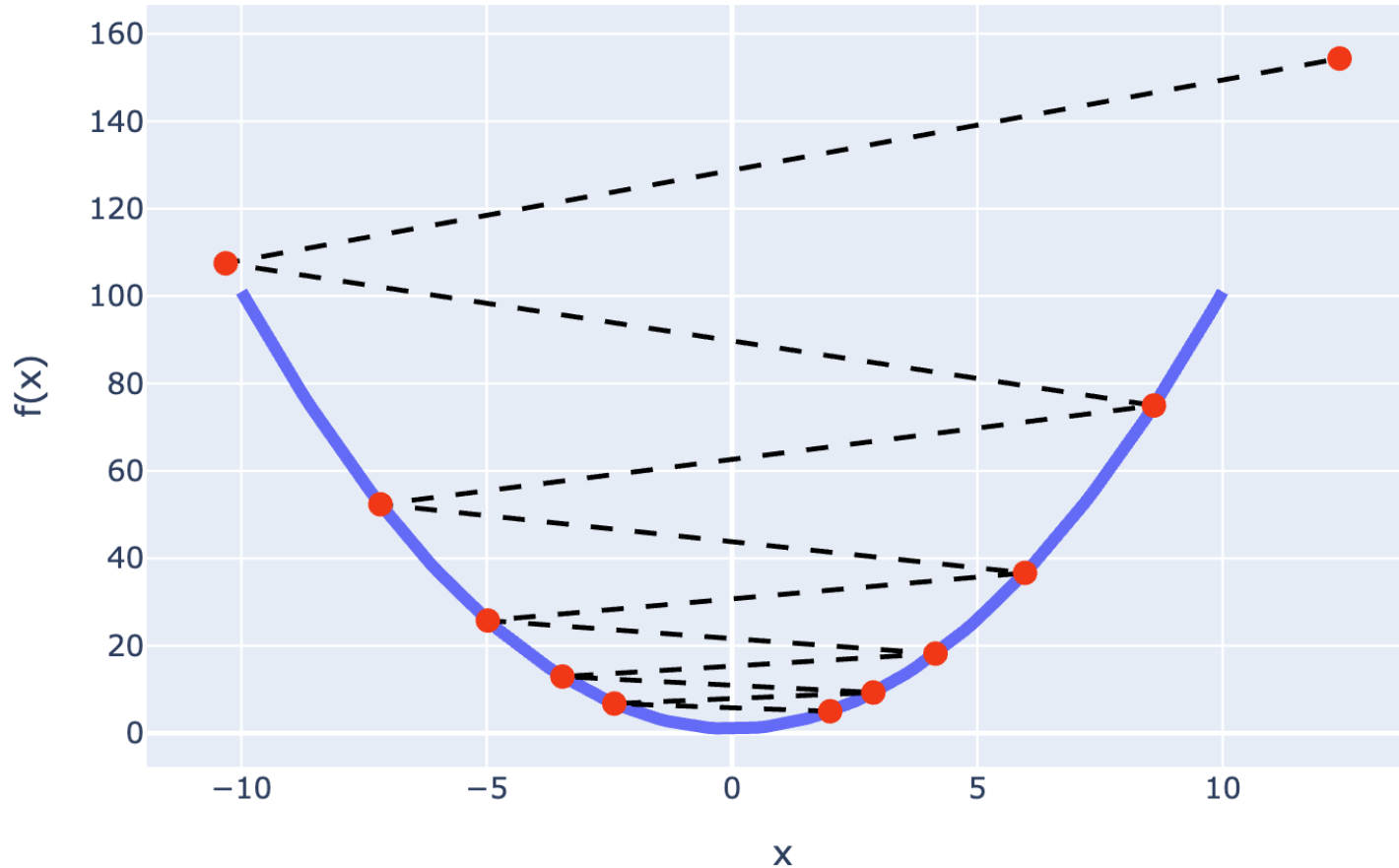
$\text{Initialize}()$ typically a small random value (more on this later)

$\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ stopping condition (e.g., $\|\Delta \mathbf{w}^{(\tau-1)}\| < \epsilon$)

Hyper Parameters:

- η is the learning rate. A common starting range is $0 < \eta < 1$, but the stable range depends on the curvature of E .
- τ_{max} is the maximum number of iterations

What are the conditions for convergence?



—●— Gradient Descent Path

Learning rate
seemed to affect
convergence even on
a simple parabola.

Convergence Assuming Quadratic Approximation

Recap: Taylor Expansion of the Error Surface



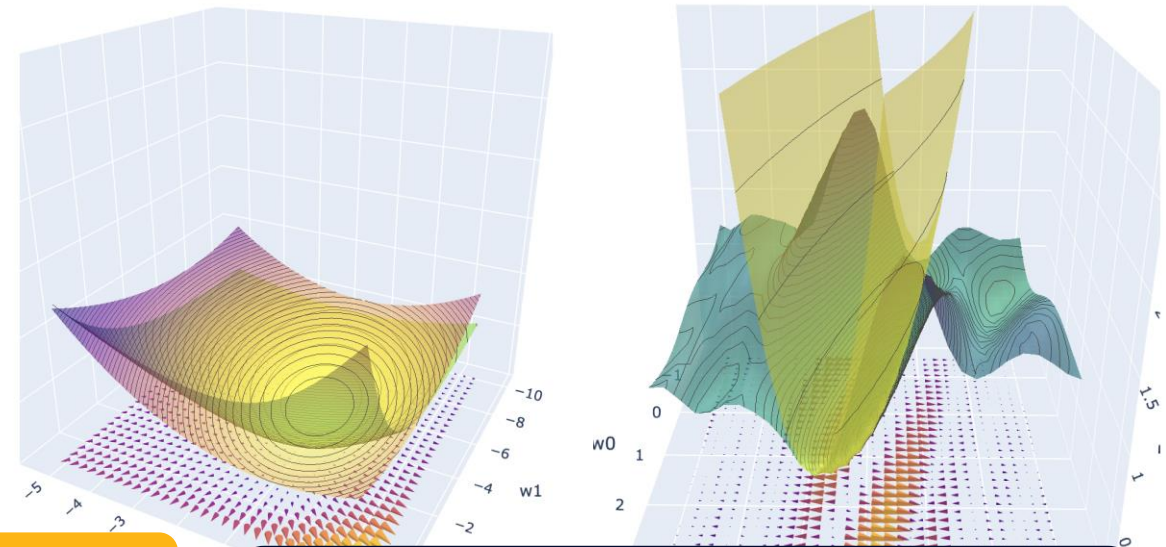
The 2nd-order Taylor expansion of the error surface at $\hat{w}$:

$$\tilde{E}_{\hat{w}}(w) = E(\hat{w}) + (w - \hat{w})^T b + \frac{1}{2} (w - \hat{w})^T H (w - \hat{w})$$

- $b = \nabla E(\hat{w})$ is the **Gradient** of E evaluated at $\hat{w}$ (vector)
- $H = \nabla \nabla E(\hat{w})$ is the **Hessian** of E evaluated at $\hat{w}$ (a matrix)

If $\hat{w}$ is a **stationary point** (w^*)
then $\nabla E(w^*) = 0 = b$

$$\tilde{E}_{w^*}(w) = E(w^*) + \frac{1}{2} (w - w^*)^T H (w - w^*)$$



Use a change of variables to better understand this part.



Recap: Quadratic Approx. Error

We constructed the **2nd-order Taylor expansion** of the error surface at the **stationary point** w^* (where $b = \nabla E|_{w=w^*} = 0$):

$$\tilde{E}_{w^*}(w) = E(w^*) + \frac{1}{2} (w - w^*)^T H (w - w^*)$$

Using the **Eigen-decomposition of the Hessian**: $Hu_i = \lambda_i u_i$ we applied a **change of variables** $\alpha_i = (w - w^*)^T u_i$ to obtain:

$$\tilde{E}_{w^*}(\alpha) = E(w^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

At $\alpha = 0$ we get the error at the stationary point $E(w^*)$.

- If we move in **direction** i (corresponding to u_i) then α_i^2 **increases**.
 - The direction of the change in error **depends on the sign of** λ_i



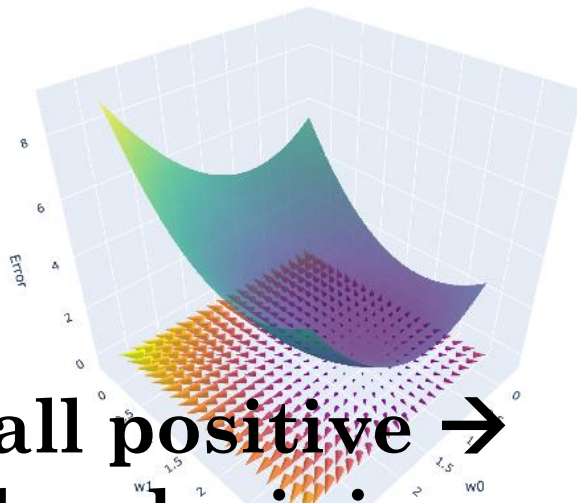
Using the **Eigen-decomposition of the Hessian**: $Hu_i = \lambda_i u_i$ we applied a **change of variables** $\alpha_i = (w - w^*)^T u_i$ to obtain:

$$\tilde{E}_{w^*}(\alpha) = E(w^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

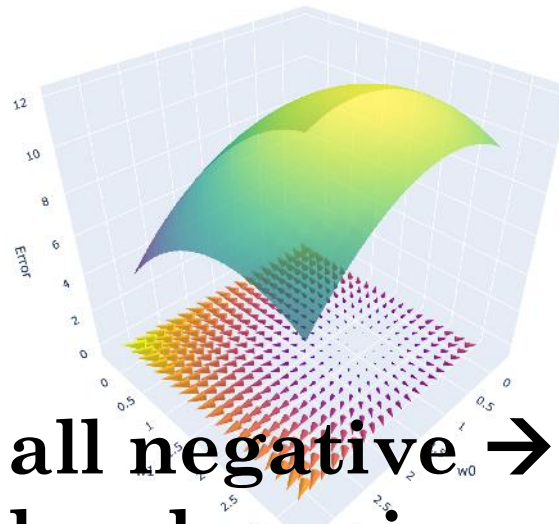
At $\alpha = 0$ we get the error at the stationary point $E(w^*)$.

- If we move in **direction i** (corresponding to u_i) then α_i^2 increases.
 - The direction of the change in error **depends on the sign of λ_i**

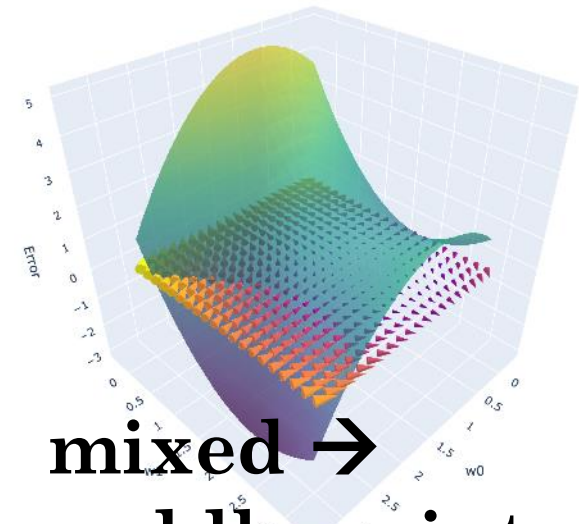
Therefore, if the **eigenvalues are**



**all positive $\rightarrow$
local minimum**



**all negative $\rightarrow$
local maximum**



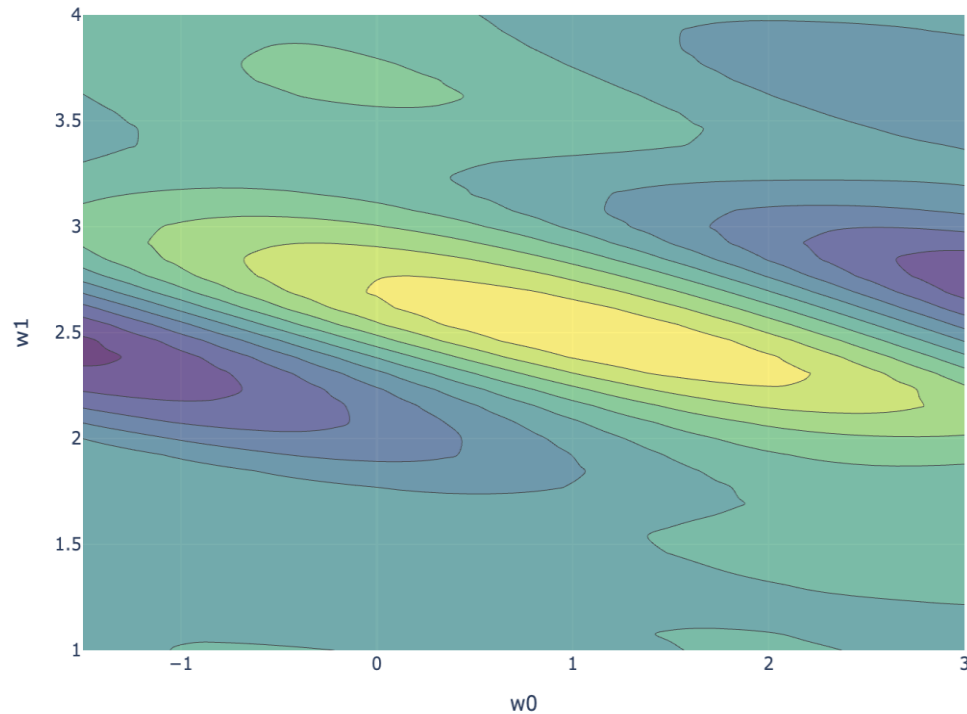
**mixed $\rightarrow$
saddle point**

Eigenvectors of the Hessian

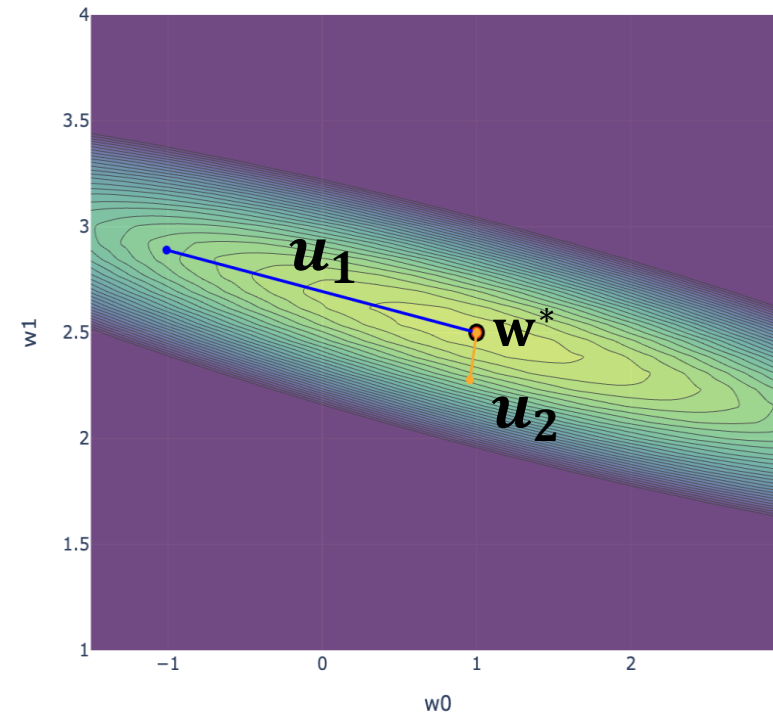


Elliptical iso-contours of constant error in the Taylor approximation align with the eigenvectors of the Hessian matrix.

Loss Surface



Taylor Approximation





Do not edit
How to change the
design



**The eigenvector u_1
corresponds to the:**

① The Slido app must be installed on every computer you're presenting from

slido



Eigenvectors of the Hessian

Elliptical iso-contours of constant error in the Taylor approximation align with the eigenvectors of the Hessian matrix.

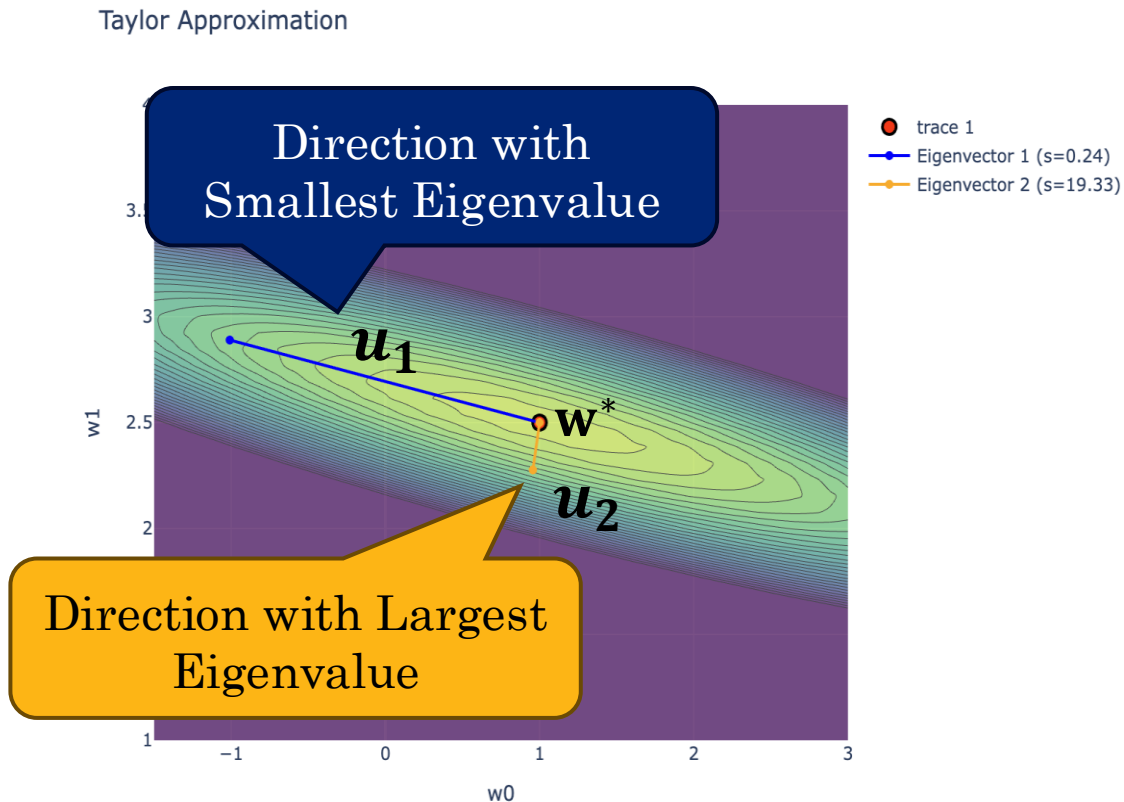
Plot the vectors so length represents a constant c change in error along $\mathbf{u}_i$

$$\tilde{E}_{\mathbf{w}^*}(\alpha) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

Then we would want:

$$\frac{1}{2} \alpha_i^2 \lambda_i = c \Rightarrow \alpha_i = \sqrt{\frac{2c}{\lambda_i}}$$

lengths proportional to $\sqrt{1/\lambda_i}$





Grad. Descent on the Quadratic Approx.

Suppose we apply **gradient descent** to our **Taylor Approximation**:

$$\tilde{E}_{w^*}(\alpha) = E(w^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

Construct the **gradient** with respect to α_i :

$$\frac{\partial}{\partial \alpha_i} \tilde{E}_{w^*}(\alpha) = \alpha_i \lambda_i$$

The **gradient descent update** recurrence:

$$\begin{aligned} \alpha_i^{(\tau)} &= \alpha_i^{(\tau-1)} - \eta \left[\frac{\partial}{\partial \alpha_i} \tilde{E}_{w^*}(\alpha) \right]_{\alpha=\alpha^{(\tau-1)}} = \alpha_i^{(\tau-1)} - \eta \left(\alpha_i^{(\tau-1)} \lambda_i \right) \\ \alpha_i^{(\tau)} &= (1 - \eta \lambda_i) \alpha_i^{(\tau-1)} \end{aligned}$$



$$\tilde{E}_{w^*}(\alpha) = E(w^*) + \frac{1}{2} \sum_i \alpha_i^2 \lambda_i$$

Construct the **gradient** with respect to α_i :

$$\frac{\partial}{\partial \alpha_i} \tilde{E}_{w^*}(\alpha) = \alpha_i \lambda_i$$

The **gradient descent update** recurrence:

$$\alpha_i^{(\tau)} = \alpha_i^{(\tau-1)} - \eta \left[\frac{\partial}{\partial \alpha_i} \tilde{E}_{w^*}(\alpha) \right]_{\alpha=\alpha^{(\tau-1)}} = \alpha_i^{(\tau-1)} - \eta \left(\alpha_i^{(\tau-1)} \lambda_i \right)$$
$$\alpha_i^{(\tau)} = (1 - \eta \lambda_i) \alpha_i^{(\tau-1)}$$

If we **iteratively** apply the gradient descent update, we obtain:

$$\alpha_i^{(1)} = (1 - \eta \lambda_i) \alpha_i^{(0)} \Rightarrow \alpha_i^{(2)} = (1 - \eta \lambda_i) \left((1 - \eta \lambda_i) \alpha_i^{(0)} \right) \Rightarrow \dots \Rightarrow$$

$$\alpha_i^{(\tau)} = (1 - \eta \lambda_i)^\tau \alpha_i^{(0)}$$



Grad. Descent on the Quadratic Approx.

Assuming a **quadratic approximation** around the minimum, after τ iterations of gradient descent:

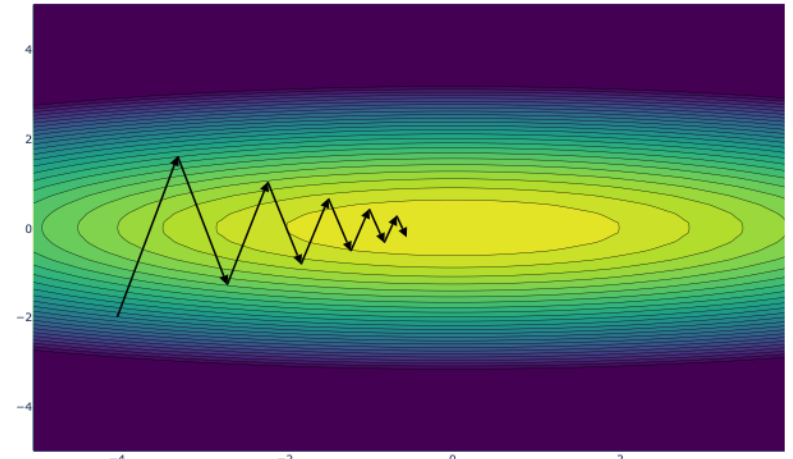
$$\alpha_i^{(\tau)} = (1 - \eta\lambda_i)^\tau \alpha_i^{(0)}$$

and the **error is minimized** when $\alpha_i = 0$.

To ensure $\alpha_i^{(\tau)} \rightarrow 0$ as $\tau \rightarrow \infty$ the learning rate should satisfy:

$$|1 - \eta\lambda_i| < 1 \quad (\text{Convergence Condition})$$

- Smaller $|1 - \eta\lambda_i|$ is better (converges faster)
- Negative $1 - \eta\lambda_i < 0$ oscillatory behavior





Learning Rate Impact on Convergence

To maximize the **progress on each step** of gradient descent we want to make $(1 - \eta\lambda_i)^\tau \rightarrow 0$ quickly as τ increases

However, to **ensure convergence** we require $|1 - \eta\lambda_i| < 1$ for all λ_i

Let $\lambda_{\min}$ and $\lambda_{\max}$ be the smallest and largest **eigenvalues** of the Hessian

- **Convergence:** $0 < \eta < \frac{2}{\lambda_{\max}}$ because if $\eta \geq \frac{2}{\lambda_{\max}}$ then $|1 - \eta\lambda_{\max}| \geq 1$
- To **accelerate the contraction** along the direction of $\lambda_{\min}$ we choose the largest learning rate $\eta < 2/\lambda_{\max}$, and the contraction is bounded below:

$$\left(1 - \frac{2}{\lambda_{\max}} \lambda_{\min}\right)$$



Condition Number and Loss Surface

The **reduction** in $|\alpha_{\min}^{(\tau)}|$ along the eigenvector $u_{\min}$ for any $\eta < 2/\lambda_{\max}$:

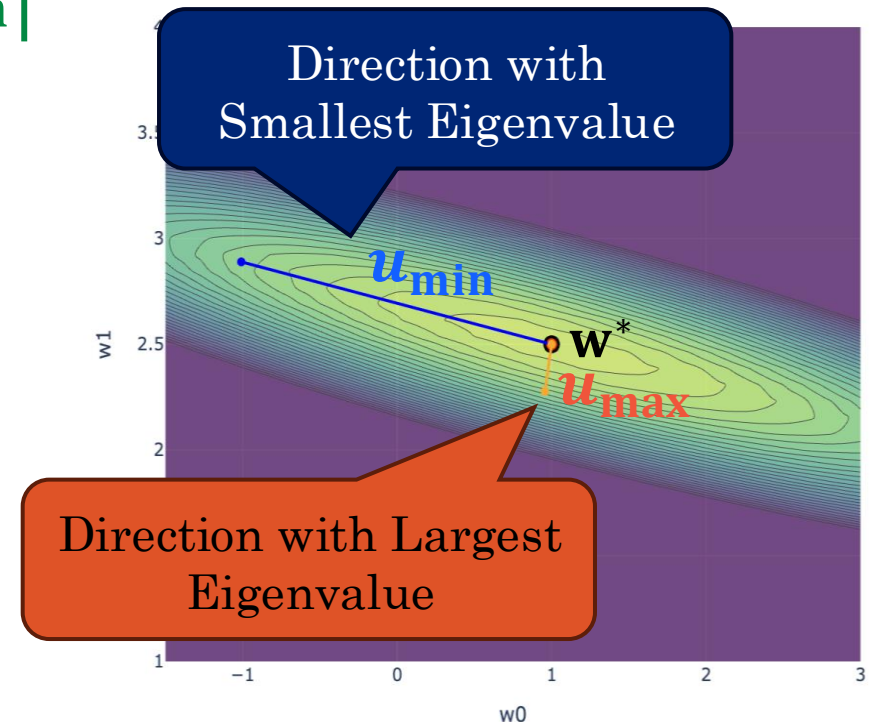
$$|\alpha_{\min}^{(\tau)}| \geq \left(1 - \frac{2\lambda_{\min}}{\lambda_{\max}}\right)^{\tau} |\alpha_{\min}^{(0)}|$$

The **condition number** is

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \geq 1$$

- Large κ implies **slow convergence**.

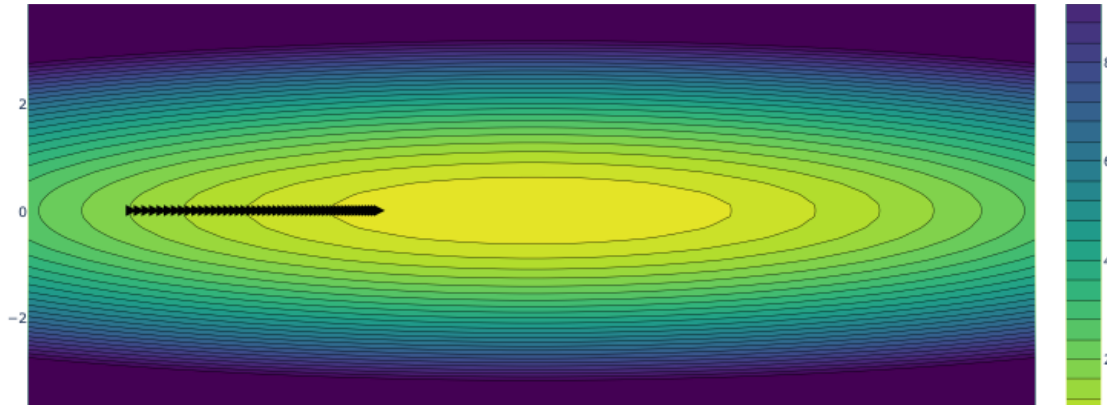
Gradient descent converges quickly on **well conditioned** problems ($\kappa \approx 1$).





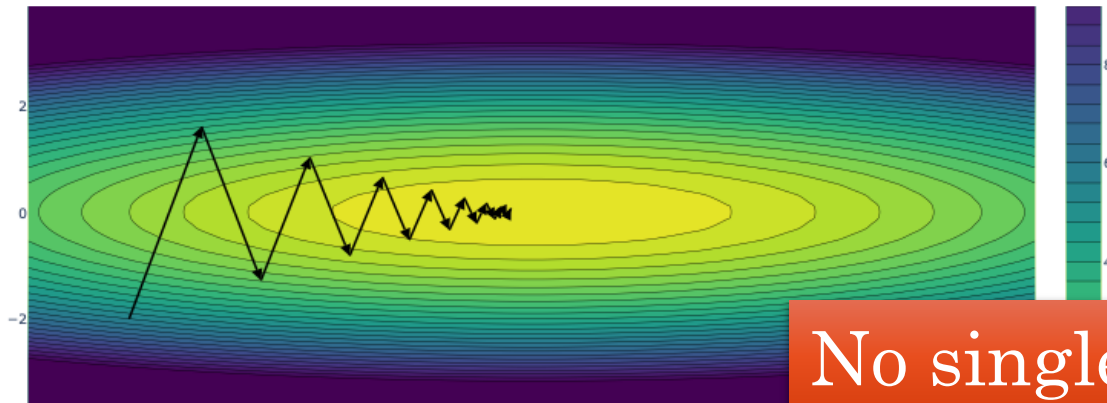
Issues with Gradient Descent (GD)

Along directions of **low curvature** (small λ_i), GD converges slowly:



Step size η is
too small for this direction.

Along directions of **high curvature** (large λ_i), GD oscillates:



Step size η is
close to $2/\lambda_{\max}$.

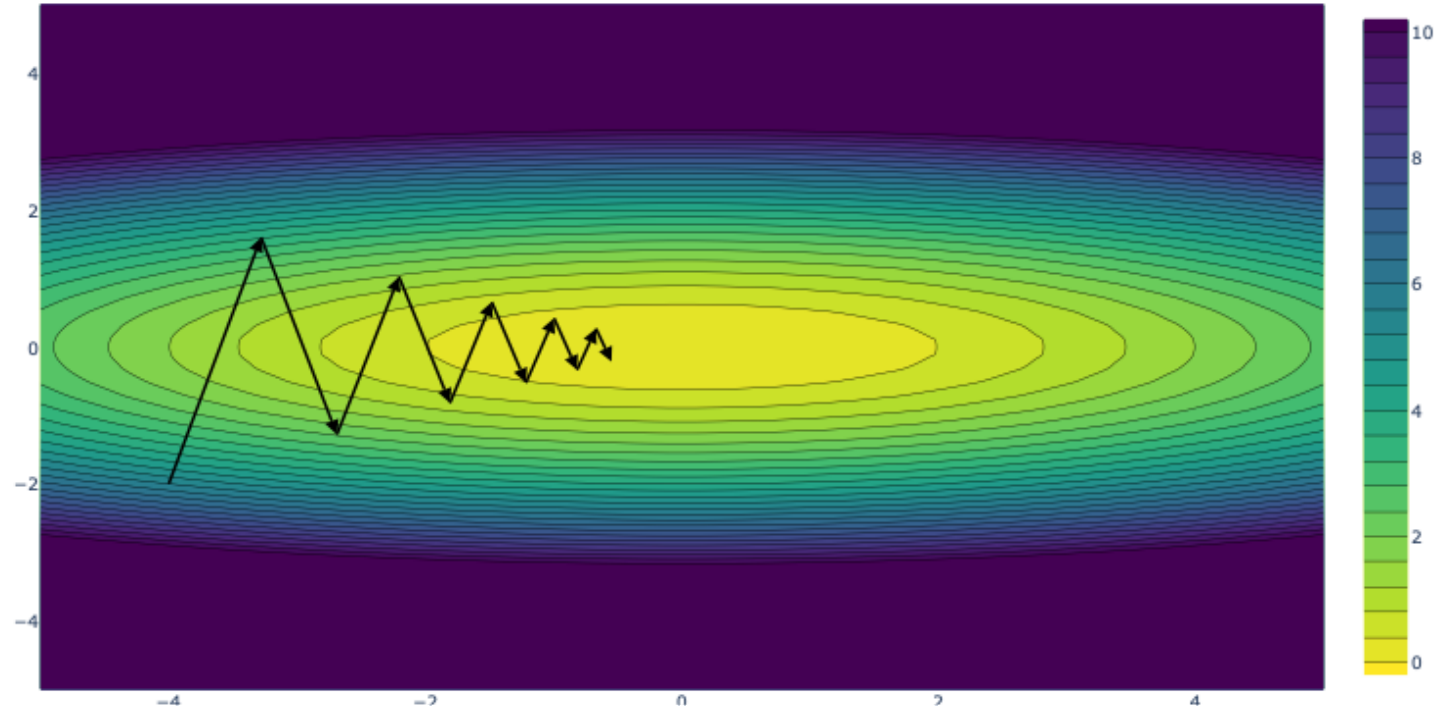
No single η is ideal for all directions.



3400896

Demo

Oscillating convergence



Momentum



Momentum can be used to improve convergence by **accumulating common update directions**

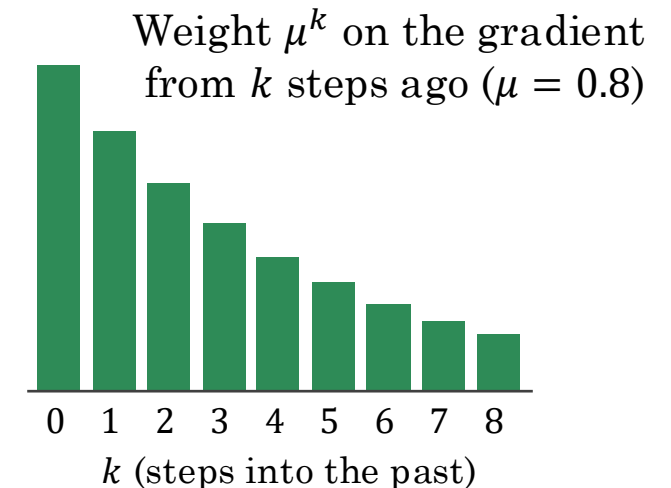
$$\begin{aligned}\Delta \mathbf{w}^{(\tau-1)} &= -\eta \nabla E(\mathbf{w}^{(\tau-1)}) + \mu \Delta \mathbf{w}^{(\tau-2)} \\ \mathbf{w}^{(\tau)} &= \mathbf{w}^{(\tau-1)} + \Delta \mathbf{w}^{(\tau-1)}\end{aligned}$$

- $0 \leq \mu < 1$ ($\mu = 0.9$) is the **momentum hyperparameter**

Substituting for $\Delta \mathbf{w}^{(\tau-1)}$ repeatedly (with $\Delta \mathbf{w} = 0$ before the first step):

$$\Delta \mathbf{w}^{(\tau-1)} = -\eta \sum_{k=0}^{\tau-1} \mu^k \nabla E(\mathbf{w}^{(\tau-1-k)})$$

- The step is an **exponentially weighted sum of past gradients**. A gradient from k steps ago has weight μ^k
- **Intuition:** The solution “**picks up speed**” along the direction of repeated updates.



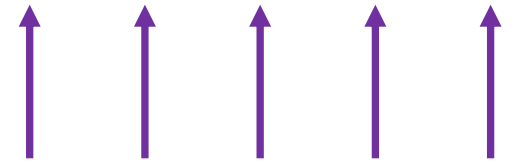


Consistent and Oscillating Gradients

Consistent gradient (low curvature): ∇E is approximately the same at each step.

$$\Delta \mathbf{w} \approx -\eta \sum_{k=0}^{\infty} \mu^k \nabla E = -\eta \nabla E (1 + \mu + \mu^2 + \dots) = -\frac{\eta}{1 - \mu} \nabla E$$

Sign of ∇E over the last 5 steps

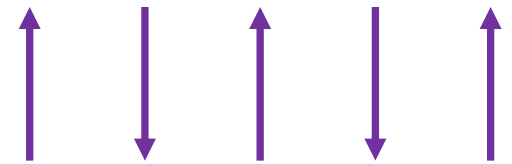


agree: accumulate

The effective learning rate **increases** to $\eta/(1 - \mu)$, e.g., 10η for $\mu = 0.9$.

Oscillating gradient (high curvature): ∇E changes sign at each step.

$$\Delta \mathbf{w} \approx -\eta \sum_{k=0}^{\infty} (-1)^k \mu^k \nabla E = -\eta \nabla E (1 - \mu + \mu^2 - \dots) = -\frac{\eta}{1 + \mu} \nabla E$$



alternate: cancel

The effective learning rate **decreases** to $\eta/(1 + \mu)$, e.g., 0.53η for $\mu = 0.9$.



Momentum in Code

Deep learning libraries (e.g., PyTorch) store a **velocity** $\mathbf{v}$, as a running sum of gradients:

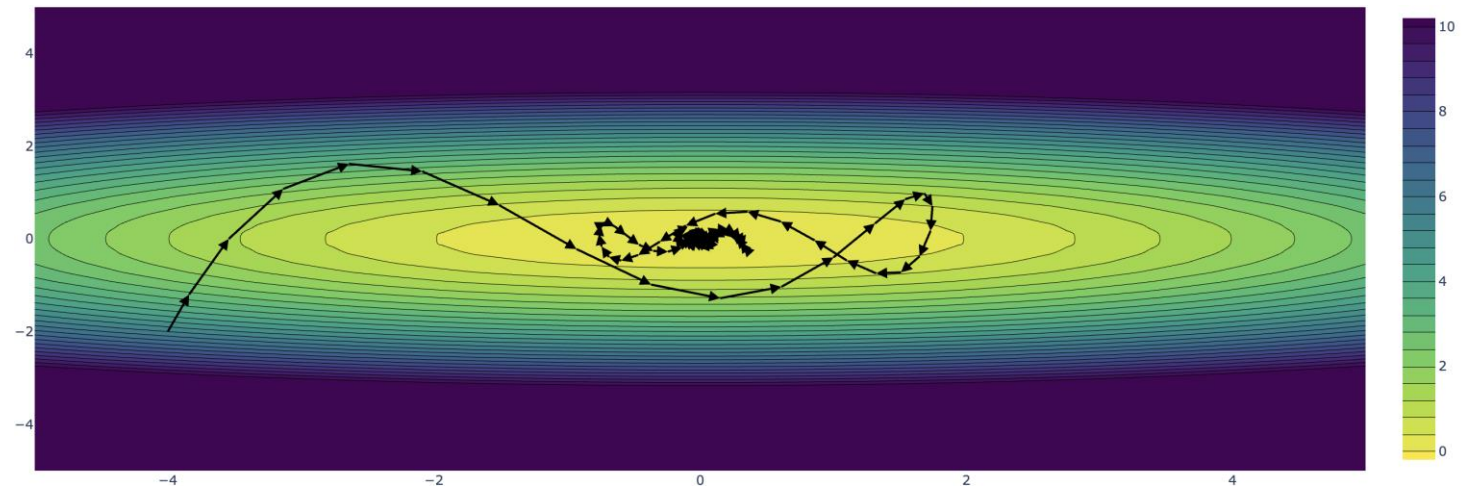
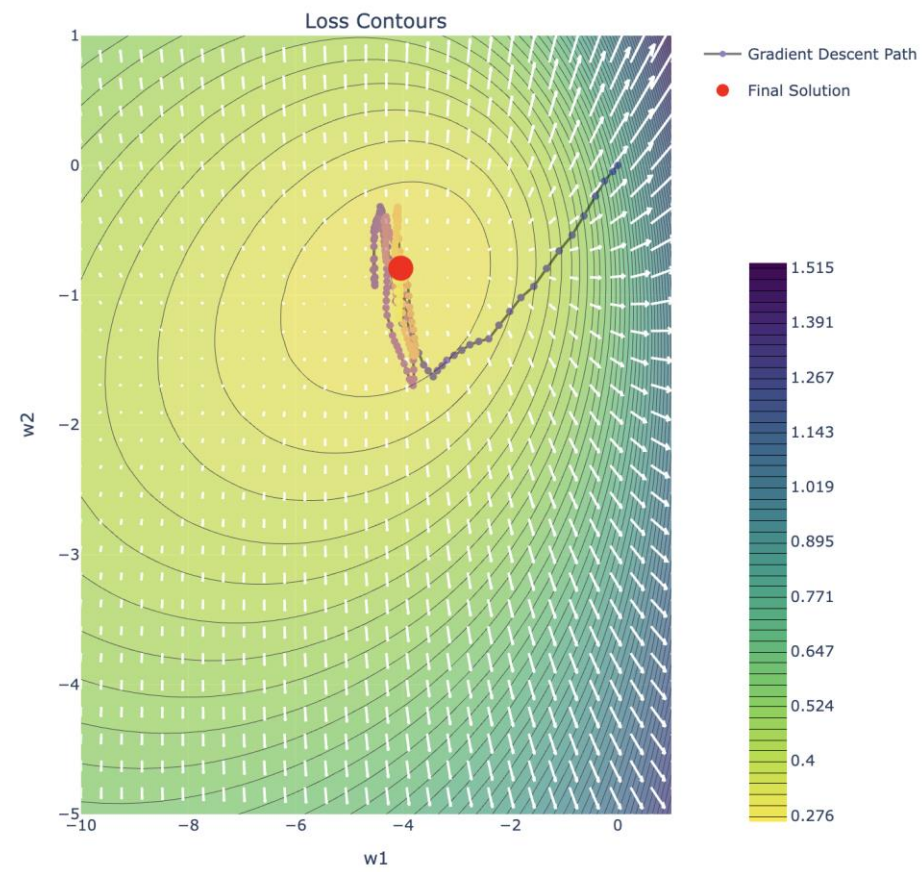
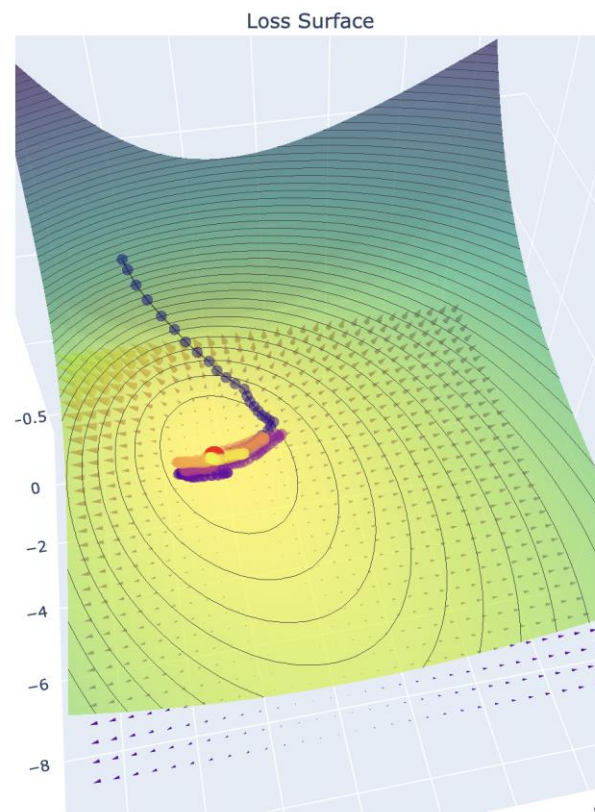
$$\mathbf{v}^{(\tau)} = \mu \mathbf{v}^{(\tau-1)} + \nabla E(\mathbf{w}^{(\tau-1)})$$

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \mathbf{v}^{(\tau)}$$

- With $\Delta \mathbf{w}^{(\tau-1)} = -\eta \mathbf{v}^{(\tau)}$ this is the update on the previous slides.
- The two forms differ only when η changes during training

Demo

Momentum





Adaptive Learning Rates – AdaGrad

AdaGrad* (Adaptive Gradient) reduces learning rate on dimensions with greatest curvature (largest gradient magnitude).

Scale the learning rate by the running sum of squared gradients.

$$r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)^2$$

Tracking Magnitude
(terms squared)

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)} + \delta}} \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)$$

Scaled Learning Rate

Issue:

Large initial gradients result in overly small learning rates near convergence.

- $\delta \approx 10^{-8}$ is a small constant for numerical stability.

*Developed at UC Berkeley



Adaptive Learning Rates – RMSProp

RMSProp focuses on the **most recent gradients** by replacing the sum with an **exponentially weighted average**:

AdaGrad $\rightarrow r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)^2$

RMSProp $\rightarrow r_i^{(\tau)} = \beta r_i^{(\tau-1)} + (1 - \beta) \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)^2$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{(\tau)} + \delta}} \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)$$

- Initialize $r_i^{(0)} = 0$. The **hyperparameter** β is typically $\beta = 0.9$.



Adam = RMSProp + Momentum

Adam (adaptive moments) combines RMSProp with momentum.

Momentum

$$s_i^{(\tau)} = \beta_1 s_i^{(\tau-1)} + (1 - \beta_1) \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)$$

RMSProp

$$r_i^{(\tau)} = \beta_2 r_i^{(\tau-1)} + (1 - \beta_2) \left(\frac{\partial}{\partial w_i} E(\mathbf{w}) \right)^2$$

Gradient Update

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \eta \frac{s_i^{(\tau)}}{\sqrt{r_i^{(\tau)} + \delta}}$$

- Initialize $s_i^{(0)} = 0$ and $r_i^{(0)} = 0$.
- Typical values: $\beta_1 = 0.9$ and $\beta_2 = 0.99$ (Bishop) or $\beta_2 = 0.999$ (PyTorch).

Adam (variants) are the most widely used gradient descent update.

Bias Corrected Version:

$$\hat{s}_i^{(\tau)} = \frac{s_i^{(\tau)}}{1 - \beta_1^\tau}$$

$$\hat{r}_i^{(\tau)} = \frac{r_i^{(\tau)}}{1 - \beta_2^\tau}$$

$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{\hat{r}_i^{(\tau)} + \delta}} \hat{s}_i^{(\tau)}$$

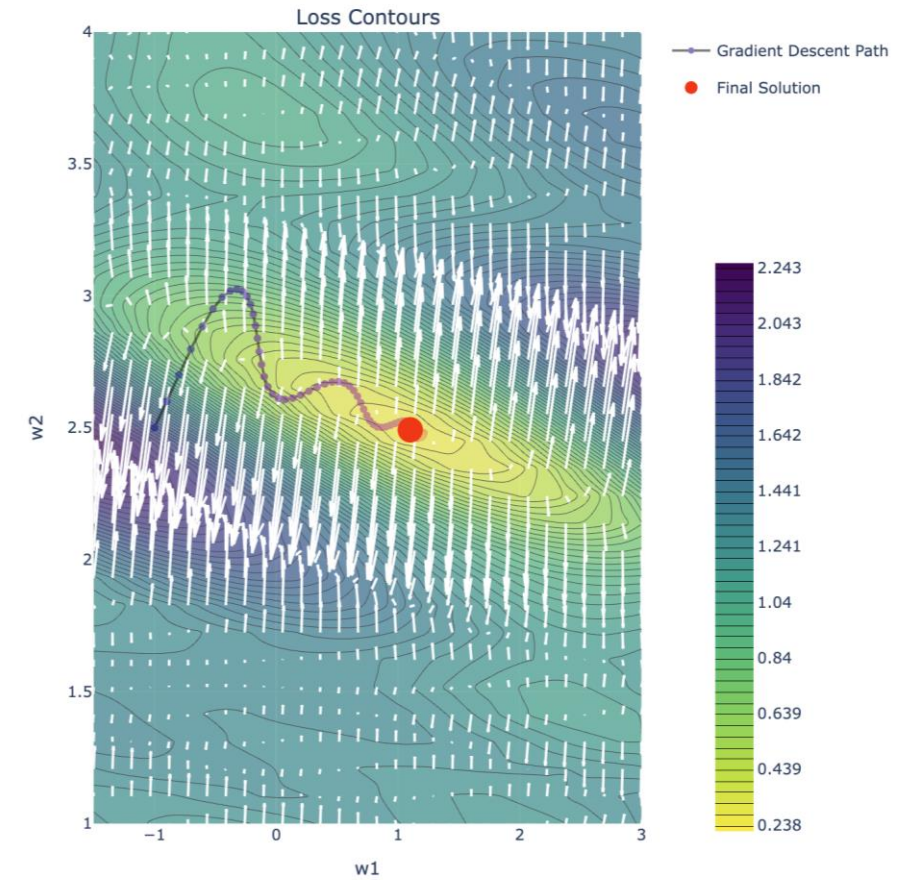
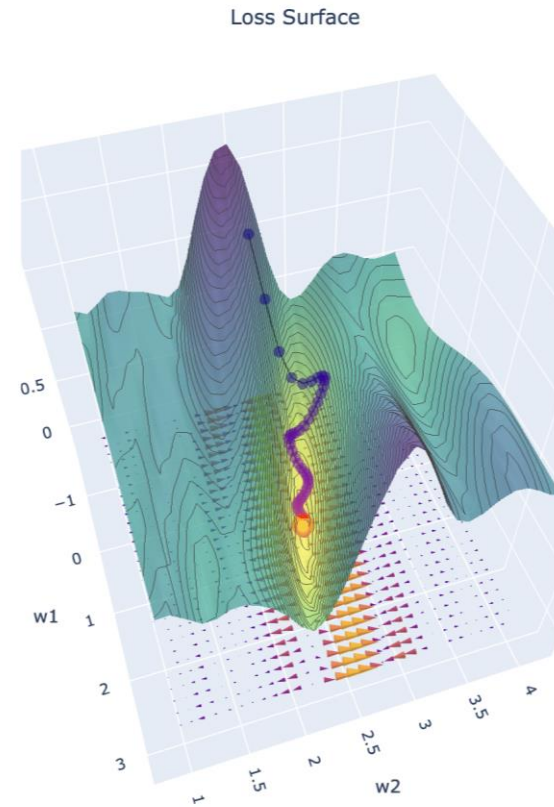
Corrects the initialization bias of the moment estimates toward zero.

Demo

Adam



3400896





Learning Rate Schedules

A larger initial learning rate can accelerate convergence but to ensure convergence we often reduce the learning rate across iterations.

- **Linear Schedule:** interpolates between η_{start} and η_{end} over T iterations

$$\eta^{(\tau)} = \left(1 - \frac{\tau}{T}\right) \eta_{\text{start}} + \left(\frac{\tau}{T}\right) \eta_{\text{end}}$$

- **Power-law Schedule:** with η_{start} , s , and c hyperparameters

$$\eta^{(\tau)} = \eta_{\text{start}} \left(1 + \frac{\tau}{s}\right)^{-c}$$

- **Step Decay:** with η_{start} , s , and c hyperparameters

$$\eta^{(\tau)} = \eta_{\text{start}} c^{\lfloor \frac{\tau}{s} \rfloor}$$

- **Cosine Schedule:** common for large models, with η_{min} , η_{max} , and T hyperparameters

$$\eta^{(\tau)} = \eta_{\text{min}} + \frac{1}{2} (\eta_{\text{max}} - \eta_{\text{min}}) \left(1 + \cos\left(\frac{\tau}{T} \pi\right)\right)$$



Learning Rate Warmup

Large models are commonly trained with a short **warmup**:

- **Learning rate increases** linearly from 0 to $\eta_{\max}$ over the first T_w iterations
- **Learning rate decays**, for example with a cosine schedule:

$$\eta^{(\tau)} = \eta_{\max} \frac{\tau}{T_w}, \quad \text{for } \tau < T_w$$

$$\eta^{(\tau)} = \eta_{\min} + \frac{(\eta_{\max} - \eta_{\min})}{2} \left(1 + \cos \left(\pi \frac{\tau - T_w}{T - T_w} \right) \right), \quad \text{for } T_w \leq \tau \leq T$$

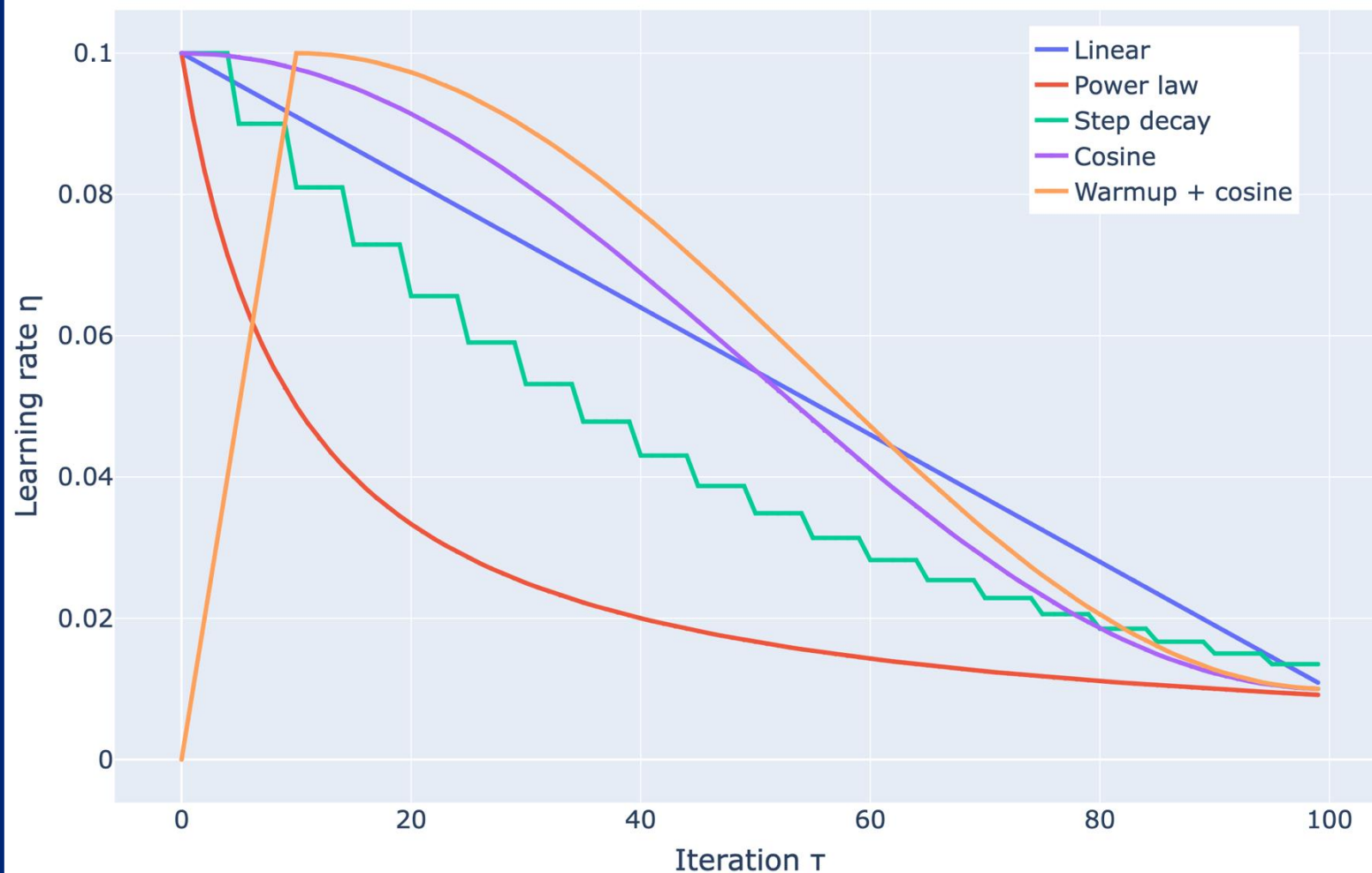
- Early in training **the gradients can be large**, and the **moving averages in Adam** are based on only a few steps. Small initial steps **prevent a single large**, poorly directed update.
- T_w is typically a small fraction of the number of iterations T .

Demo

Learning Rate Schedules



3400896





Recap: Batch Gradient Descent

Update the weights by moving in the **opposite direction** of the gradient:

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\text{max}}$ ):  
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

$\text{Initialize}()$ typically a small random value (more on this later)

$\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ stopping condition (e.g., $\|\Delta \mathbf{w}^{(\tau-1)}\| < \epsilon$)

$\text{Step}(\nabla E(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$ defines how the gradient is used to compute $\Delta \mathbf{w}^{(\tau-1)}$

- This is where the update rule (e.g., momentum or Adam) is implemented

Approximating Gradient Descent

Stochastic Gradient Descent



The Cost of Batch Gradient Descent

What is the cost of each iteration of batch gradient descent?

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\text{max}}$ ):  
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

If we have N training data points and a model with W parameters:

$$\nabla E(\mathbf{w}^{(\tau-1)}) = \nabla E(\mathbf{w}^{(\tau-1)}; \{(x_n, t_n)\}_{n=1}^N) = \frac{1}{N} \sum_{n=1}^N \nabla L(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n))$$



What is the cost for a single gradient update assuming N data points in W dimensions and the basic gradient update equation.



The Cost of Batch Gradient Descent

What is the cost of each iteration of batch gradient descent?

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\max}$ ):  
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

If we have N training data points and a model with W parameters:

$$\nabla E(\mathbf{w}^{(\tau-1)}) = \nabla E(\mathbf{w}^{(\tau-1)}; \{(x_n, t_n)\}_{n=1}^N) = \frac{1}{N} \sum_{n=1}^N \nabla L(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n))$$

The cost per update is $\mathcal{O}(NW)$

$\mathcal{O}(N)$

$\mathcal{O}(W)$



The Training Error Estimates the Test Error

As an **optimizer**, our goal has been to minimize the **average training error**:

$$E_{\text{train}}[\mathbf{w}] = \frac{1}{N} \sum_{n=1}^N L(t_n; f_{\mathbf{w}}(x_n))$$

However, we would have preferred to minimize the **expected test error**:

$$E_{\text{test}}[\mathbf{w}] = \mathbb{E}_{p(t_*, x_*)}[L(t_*, f_{\mathbf{w}}(x_*))]$$

- Where $p(t_*, x_*)$ is the **unknown distribution over test examples**.

If the training data are drawn from $p(t_*, x_*)$ and $\mathbf{w}$ is not chosen using the data, then for large N :

$$E_{\text{train}}[\mathbf{w}] = \frac{1}{N} \sum_{n=1}^N L(t_n; f_{\mathbf{w}}(x_n)) \approx \mathbb{E}_{p(t_*, x_*)}[L(t_*, f_{\mathbf{w}}(x_*))] = E_{\text{test}}[\mathbf{w}]$$

For fixed $\mathbf{w}$ the **training error is an unbiased estimate of the test error**. For the minimizer $\mathbf{w}^*$ of E_{train} , it typically underestimates $E_{\text{test}}[\mathbf{w}^*]$ (overfitting).



Approximating the Gradient

The error (and the gradient) are **empirical expectations** of the **test error**:

$$E(\mathbf{w}^{(\tau-1)}) = \frac{1}{N} \sum_{n=1}^N L(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n)) \approx \mathbb{E}_{p(t,x)} \left[L(t, f_{\mathbf{w}^{(\tau-1)}}(x)) \right]$$

The error at a **single point** is *also* a (**noisy**) estimate:

$$E_n(\mathbf{w}^{(\tau-1)}) = L(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n)) \approx \mathbb{E}_{p(t,x)} \left[L(t, f_{\mathbf{w}^{(\tau-1)}}(x)) \right]$$

Compute the gradient for a **randomly (stochastically)** sampled **single point**:

$$\underbrace{\nabla E_n(\mathbf{w}^{(\tau-1)})}_{\text{Cheap approx.}} \approx \underbrace{\nabla E(\mathbf{w}^{(\tau-1)}; \{(x_n, t_n)\}_{n=1}^N)}_{\text{Best estimate (but expensive)}} \approx \underbrace{\mathbb{E}_{p(t,x)} \left[\nabla L(t, f_{\mathbf{w}^{(\tau-1)}}(x)) \right]}_{\text{Ideal but don't know } p(t,x)}$$



Stochastic Gradient Descent

Gradient descent but with the gradient computed on a **single** randomly sampled training data point

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\text{max}}$ ):  
     $n = \text{RandomInteger}(0, N)$   
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E_n(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

Gradient of the error on the n^{th} data point.

$$\nabla E_n(\mathbf{w}^{(\tau-1)}) = \nabla L(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n))$$

- Stochasticity comes from **random gradient estimates** $\nabla E_n(\mathbf{w}^{(\tau-1)})$
- **Unbiased**: $\mathbb{E}_n[\nabla E_n(\mathbf{w})] = \nabla E(\mathbf{w})$ for n uniform on $\{1, \dots, N\}$
- **Cost per step is $\mathcal{O}(W)$** instead of $\mathcal{O}(NW)$ in batch gradient descent



Stochastic Gradient Descent (Shuffling)

In practice, we often **shuffle** and then **iterate** over the entire dataset

```
 $\mathbf{w}^{(0)} = \text{Initialize}();$ 
```

```
 $\tau = 1$ 
```

```
for  $e$  in range(0,  $e_{\text{max}}$ ): # Epochs
```

```
    ShuffleData() # Randomize the Data Ordering
```

```
    for  $n$  in range(0,  $N$ ):
```

```
         $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E_n(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$ 
```

```
        if Converged( $\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon$ ): terminate
```

```
     $\tau = \tau + 1$ 
```

One **epoch** is a complete pass over all the data points.

Some code shuffles only once.

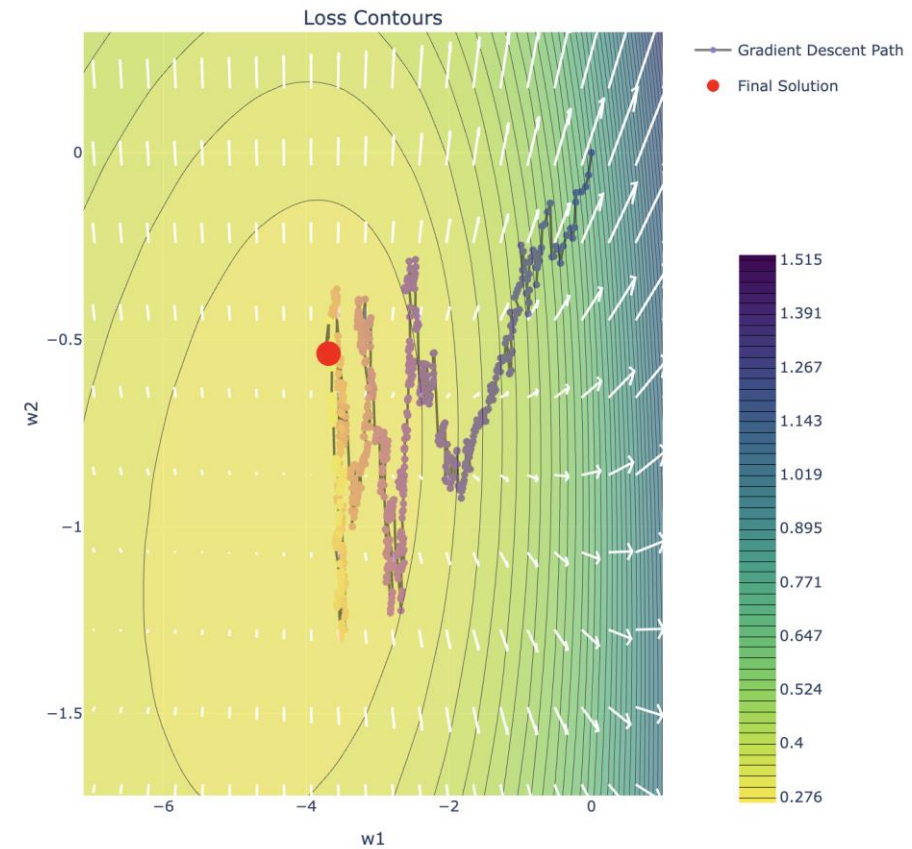
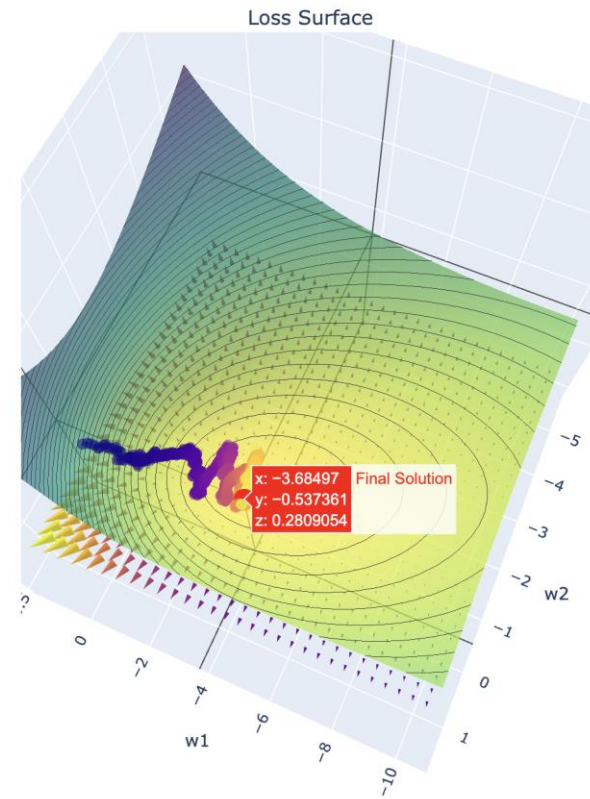
Gradient of the error on the n^{th} data point.

Demo

Stochastic Gradient Descent



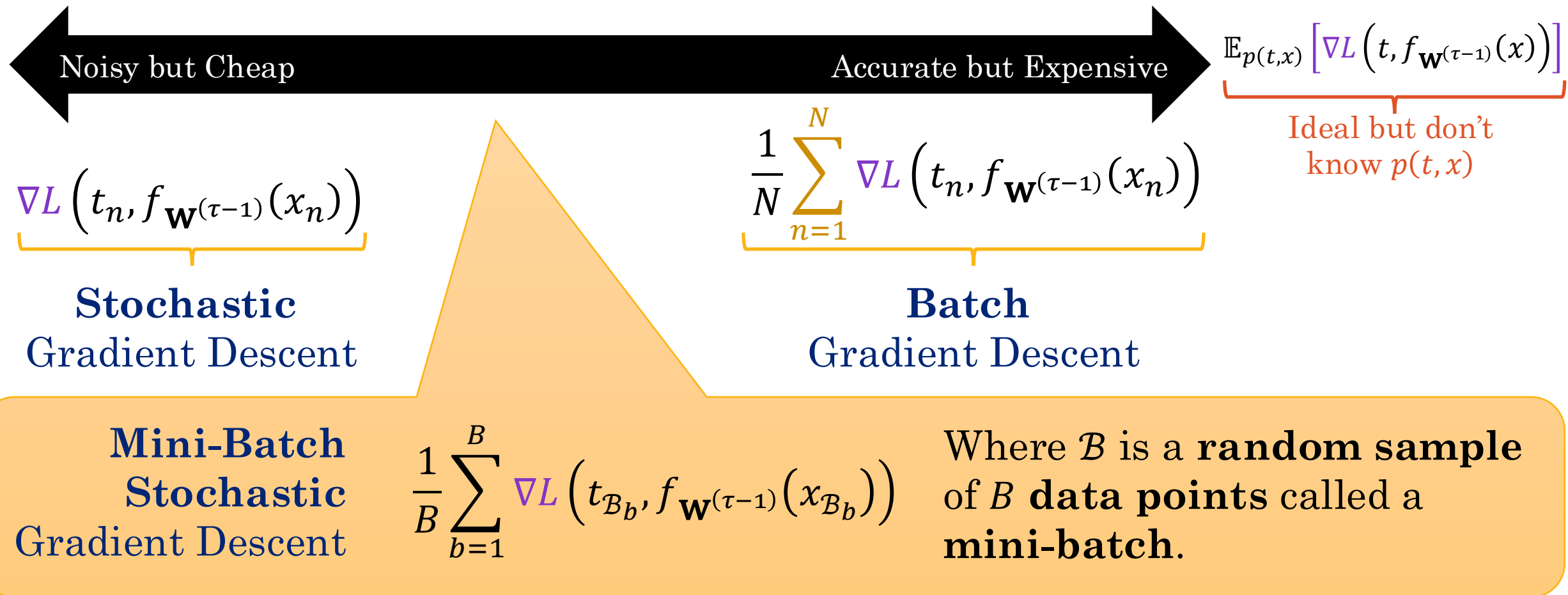
3400896





A Better Gradient Approximation

Estimating the gradient using a single data point results in **noisy estimates**.





A Spectrum of Stochastic Gradients

We typically refer to any of the sampling based gradient estimation procedures as **Stochastic Gradient Descent (SGD)**

Noisy but Cheap

Accurate but Expensive

$$\underbrace{\nabla L\left(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n)\right)}$$

**Stochastic
Gradient Descent**

$$\mathcal{O}(W)$$

$$\underbrace{\frac{1}{B} \sum_{b=1}^B \nabla L\left(t_{\mathcal{B}_b}, f_{\mathbf{w}^{(\tau-1)}}(x_{\mathcal{B}_b})\right)}$$

**Mini-Batch Stochastic
Gradient Descent**

$$\mathcal{O}(BW)$$

$$\underbrace{\frac{1}{N} \sum_{n=1}^N \nabla L\left(t_n, f_{\mathbf{w}^{(\tau-1)}}(x_n)\right)}$$

**Batch
Gradient Descent**

$$\mathcal{O}(NW)$$

Stochastic Gradient Methods



Mini-batch Stochastic Gradient Descent

Gradient descent but with the gradient computed on a **batch of randomly sampled training points**

```
 $\mathbf{w}^{(0)} = \text{Initialize}()$   
for  $\tau$  in range(1,  $\tau_{\text{max}}$ ):  
     $\mathcal{B} = \text{RandBatch}(0, N, \text{size} = B)$   
     $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E_{\mathcal{B}}(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$   
    if  $\text{Converged}(\mathbf{w}^{(\tau)}, \mathbf{w}^{(\tau-1)}, \epsilon)$ : terminate
```

$$\begin{aligned}\nabla E_{\mathcal{B}}(\mathbf{w}^{(\tau-1)}) \\ &= \frac{1}{B} \sum_{b=1}^B \nabla L(t_{\mathcal{B}_b}, f_{\mathbf{w}^{(\tau-1)}}(x_{\mathcal{B}_b}))\end{aligned}$$

- **Cost per step is $\mathcal{O}(BW)$** instead of $\mathcal{O}(NW)$ in batch gradient descent
- Note that at batch size $B = 1$ **mini-batch** is the same as **SGD**
 - We often call both algorithms **SGD** (just with a **batch-size hyperparameter**)

Shuffling for Minibatch Gradient Descent



In practice, we often **shuffle** and then **iterate** over the entire dataset

```
 $\mathbf{w}^{(0)} = \text{Initialize}();$ 
```

```
 $\tau = 1$ 
```

```
for  $e$  in range(0,  $e_{\text{max}}$ ): # Epochs
```

```
    ShuffleData() # Randomize the Data Ordering
```

```
    for  $n$  in range(0,  $N$ , step= $B$ ):
```

```
         $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \text{Step}(\nabla E_{n, \dots, n+B-1}(\mathbf{w}^{(\tau-1)}), \tau, \beta, \eta)$ 
```

```
        if Converged( $\mathbf{w}^{(\tau)}$ ,  $\mathbf{w}^{(\tau-1)}$ ,  $\epsilon$ ): terminate
```

```
         $\tau = \tau + 1$ 
```

One **epoch** is a complete pass over all the data points.

Some code shuffles only once.

Gradient of the error on the next B data points.



If I have N data points and I choose a batchsize $B=2$. In one epoch how many gradient steps will I take?



How do I Pick the Batch Size?

Larger batch size:

- **Better gradient estimate**, but the noise decreases only as $1/\sqrt{B}$
- **Enables hardware parallelism** – can compute gradient for each point in parallel

Smaller batch size:

- **Cheap to compute** → take more steps
- **Greater stochasticity** can help with local minima

Typical design decision:

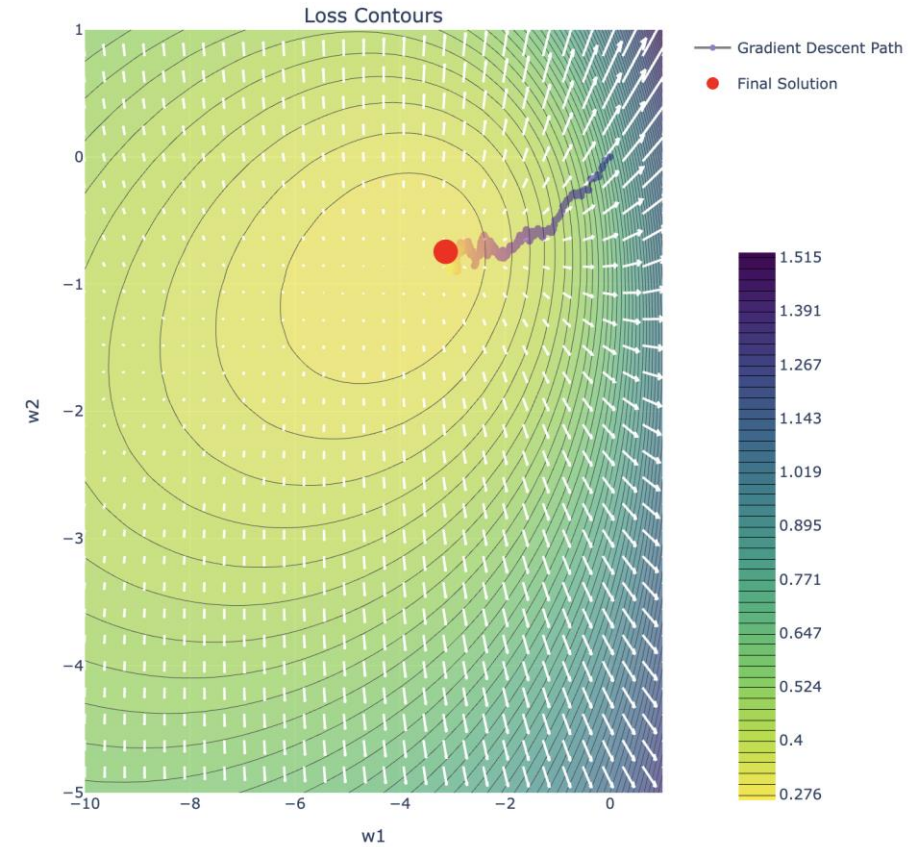
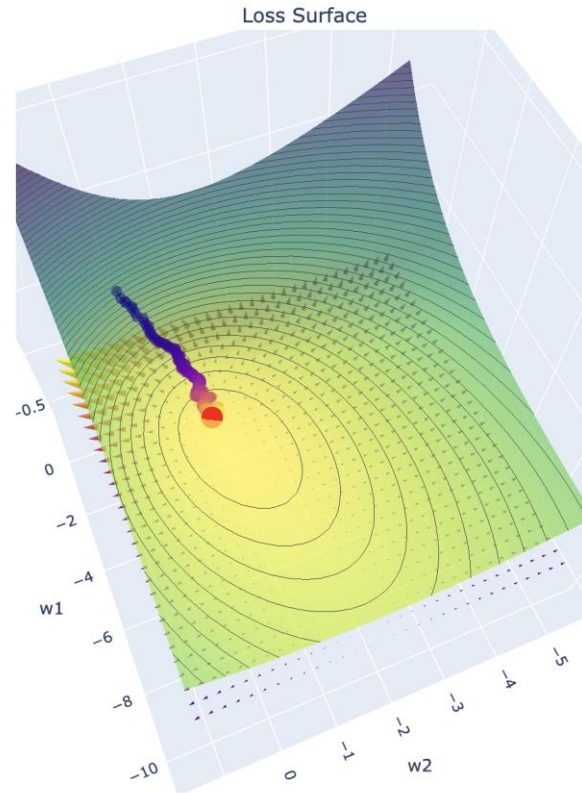
- Increase batch size to **saturate hardware parallelism**
- **Increase the learning rate** in proportion to B (a heuristic for SGD, used with warmup; it breaks down for very large B)

Demo

Mini-batch Gradient Descent



3400896



Lecture 11

Stochastic Gradient Descent

Credit: Joseph E. Gonzalez and Narges Norouzi

Reference Book Chapters: Chapter 7