Convex Functions

- Any local minimizer $\mathbf{x}^*$ is a global minimizer of f if f is convex.
- Further if f is differentiable, then any stationary point $\mathbf{x}^*$ is a global minimizer of f

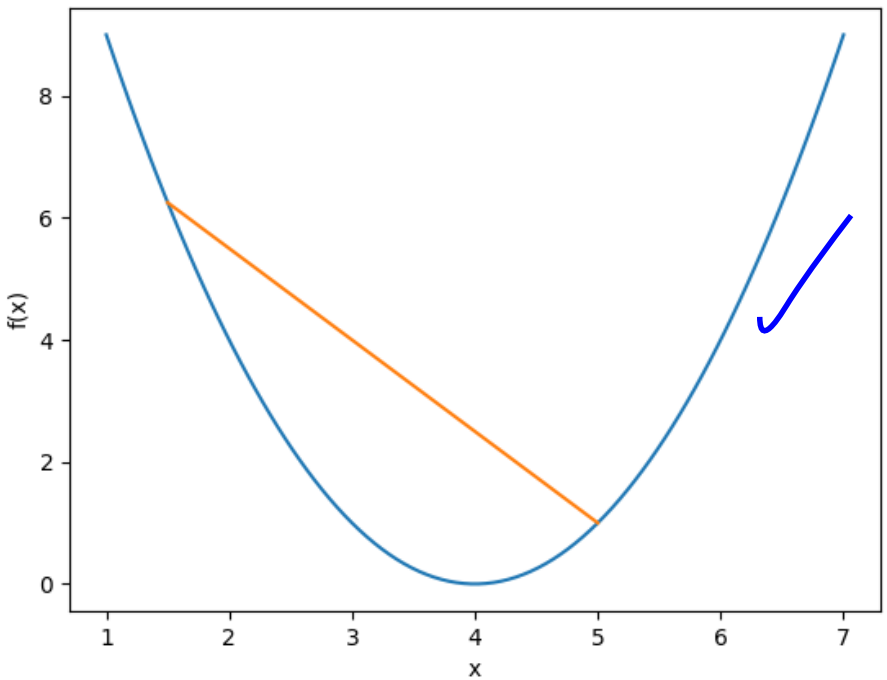


Figure: $f(x) = (x-4)^2$ is convex on $\mathbb{R}$ and $x^* = 4$ is both global and local minimizer of $f(x)$

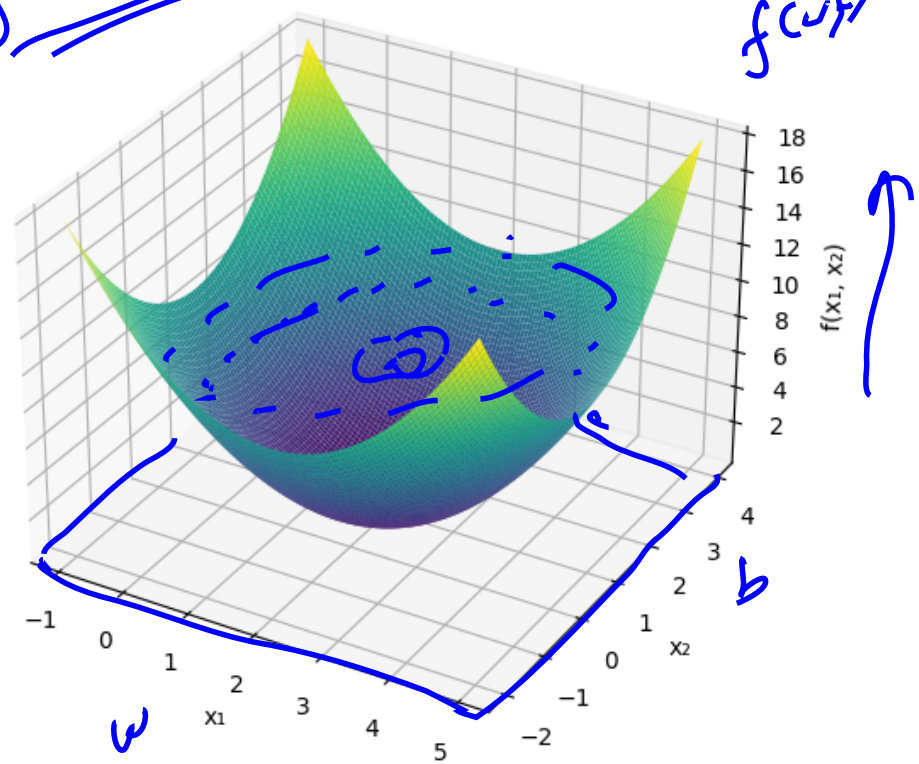


Figure: $f(x_1, x_2) = (x_1-2)^2 + (x_2-1)^2$ is convex on $\mathbb{R}^2$ and $x^* = (2,1)$ is both global and local minimizer of $f(x_1, x_2)$

Unconstrained Minimization

Unconstrained Minimization Algorithm:

1. Initialize $\mathbf{x}_0$ and $i = 0$
2. If stopping condition is not satisfied, then continue, else stop
   1. Calculate $\mathbf{x}_{i+1}$ so that $f(\mathbf{x}_{i+1}) < f(\mathbf{x}_i)$
   2. Update i with $i + 1$
3. Final $\mathbf{x}_i$ is the local minimum $\mathbf{x}^*$ of $f(\mathbf{x})$

Following questions need to be answered:

- What stopping condition can be used?
- What is the speed of convergence?
- How to calculate $\mathbf{x}_{i+1}$?

Unconstrained Minimization

Stopping Conditions:

- $||\nabla f(\mathbf{x}_i)|| \leq \epsilon$
- $\frac{f(\mathbf{x}_i) - f(\mathbf{x}_{i+1})}{|f(\mathbf{x}_i)|} \leq \epsilon$

Speed of Convergence:

The sequence $\{\mathbf{x}_i\}$ converges to $\mathbf{x}^*$ with order p and convergence rate β if

$$\lim_{i \to \infty} \frac{||\mathbf{x}_{i+1} - \mathbf{x}^*||}{||\mathbf{x}_i - \mathbf{x}^*||^p} = \beta, \qquad \beta \in \mathbb{R}$$

- Convergence is faster for higher p
- Linear Convergence: $p = 1, 0 < \beta < 1$
- Quadratic Convergence: $p = 2, \beta > 0$

$x_0, x_1, x_2, \ldots \to x^*$

$||x_{i+1} - x^*|| \leq \beta ||x_i - x^*||$

Unconstrained Minimization

Calculating $\mathbf{x}_{i+1}$ for the Unconstrained Minimization Algorithm:

- First Order Methods
  - Gradient Descent
  - Stochastic Gradient Descent
  - Mini-Batched Gradient Descent
  - Stochastic Average Gradient
  - Optimizers - AdaGrad, RMSProp & Adam
- Second Order Methods
  - Newton's Method
  - Quasi-Newton Method

Nonlinear Least Squares Problem

The problem is to fit $g(x) = \frac{1}{1+e^{-(wx+b)}}$ to a training set with observations $(x_1, x_2, \ldots x_n)$ and corresponding estimated responses $(y_1, y_2, \ldots y_n)$ using least squares.

The objective function (or the loss function) to be minimized is:

$$f(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(g(x_i) - y_i)^2$$

$$f(w, b) = \frac{1}{2}\sum_{i=1}^{n}\left(\frac{1}{1+e^{-(wx_i+b)}} - y_i\right)^2$$

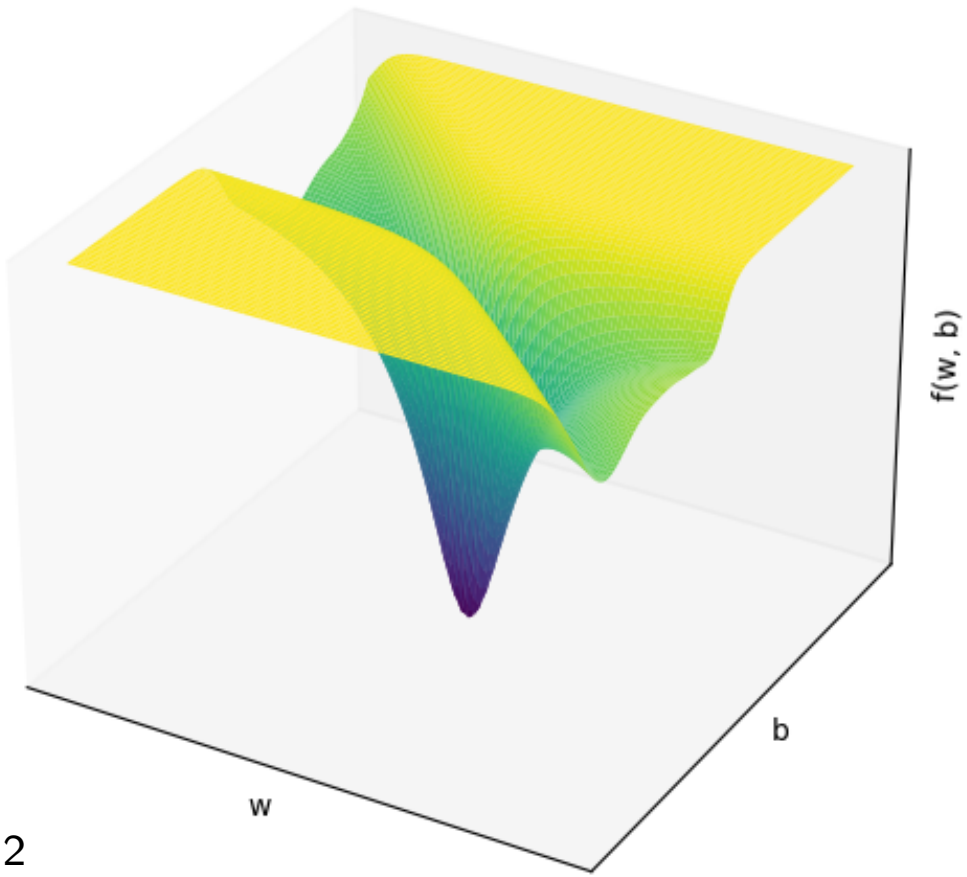


Figure: Surface plot of Loss Function $f(w, b)$

Gradient Descent

Descent Direction:

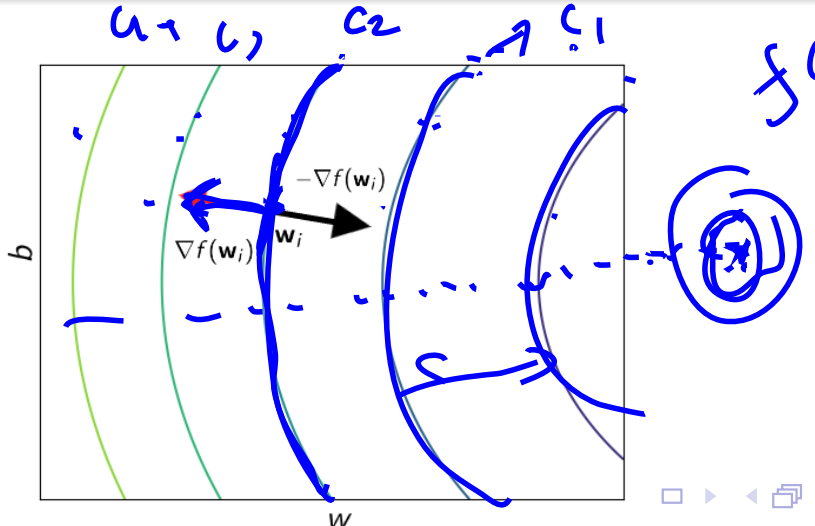
$\mathbf{d} \in \mathbb{R}^n$ is a *descent direction* of $f(\mathbf{w})$ at $\mathbf{w}^* \in \mathbb{R}^n$ if $f(\mathbf{w}^* + \eta\mathbf{d}) < f(\mathbf{w}^*)$ for all $\eta \in (0, \delta)$, $\delta > 0$

- In terms of minimization, $f(\mathbf{w}_{i+1}) < f(\mathbf{w}_i)$, where $\mathbf{w}_{i+1} = \mathbf{w}_i + \eta_i\mathbf{d}_i$
- $\eta_i =$ minimize $f(\mathbf{w}_i + \eta\mathbf{d}_i)$, $\eta > 0$
- $\eta_i =$ minimize $h(\eta)$, $\eta > 0$
- $\mathbf{d}_i = -\nabla f(\mathbf{w}_i)$ in gradient descent algorithm

Gradient Descent

Gradient Descent Algorithm:

1. Initialize $\mathbf{w}^0$ and $k = 0$
2. If stopping condition like $\left|\left|\nabla f(\mathbf{w}^k)\right|\right| \leq \epsilon$ is satisfied, then stop, else continue
   1. $\mathbf{d}^k = -\nabla f(\mathbf{w}^k)$
   2. Calculate η^k along $\mathbf{d}^k$ so that $f(\mathbf{w}^k + \eta^k \mathbf{d}^k) < f(\mathbf{w}^k)$
   3. $\mathbf{w}^{k+1} = \mathbf{w}^k + \eta^k \mathbf{d}^k$
   4. Update k with $k + 1$
3. Final $\mathbf{w}^k$ is the local minimum $\mathbf{w}^*$ of $f(\mathbf{w})$



Gradient Descent

- The objective function (or the loss function) to be minimized is:

$$f(\mathbf{w}) = f(w, b) = \frac{1}{2}\sum_{i=1}^{n}(g(x_i) - y_i)^2 = \frac{1}{2}\sum_{i=1}^{n}\left(\frac{1}{1 + e^{-(wx_i+b)}} - y_i\right)^2$$

- The update with learning rate η:

$$\begin{bmatrix} w^{k+1} \\ b^{k+1} \end{bmatrix} := \begin{bmatrix} w^{k} \\ b^{k} \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial}{\partial w}\left(\frac{1}{2}\sum_{i=1}^{n}(g(x_i) - y_i)^2\right) \\ \frac{\partial}{\partial b}\left(\frac{1}{2}\sum_{i=1}^{n}(g(x_i) - y_i)^2\right) \end{bmatrix}$$

$$= \begin{bmatrix} w^{k} \\ b^{k} \end{bmatrix} - \eta \begin{bmatrix} \sum_{i=1}^{n}(g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) * x_i \\ \sum_{i=1}^{n}(g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) \end{bmatrix}$$

- In Gradient Descent, gradients have to be evaluated for every training set before updating which can be expensive for large training.
- Convergence rate: Linear
- Iteration cost: $O(n)$

Gradient Descent

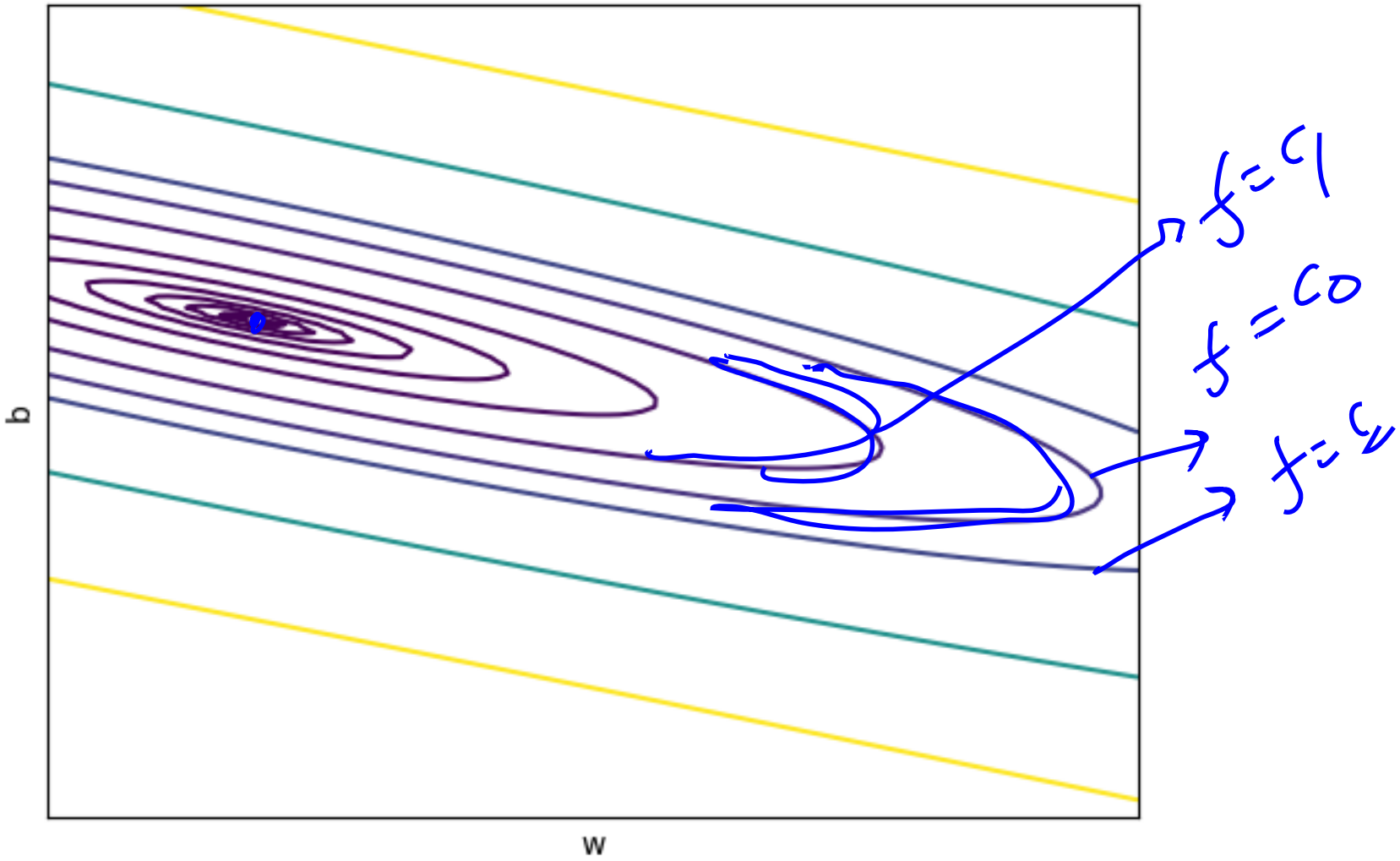


Figure: Contour plot of Loss function $f(w, b)$

Gradient Descent

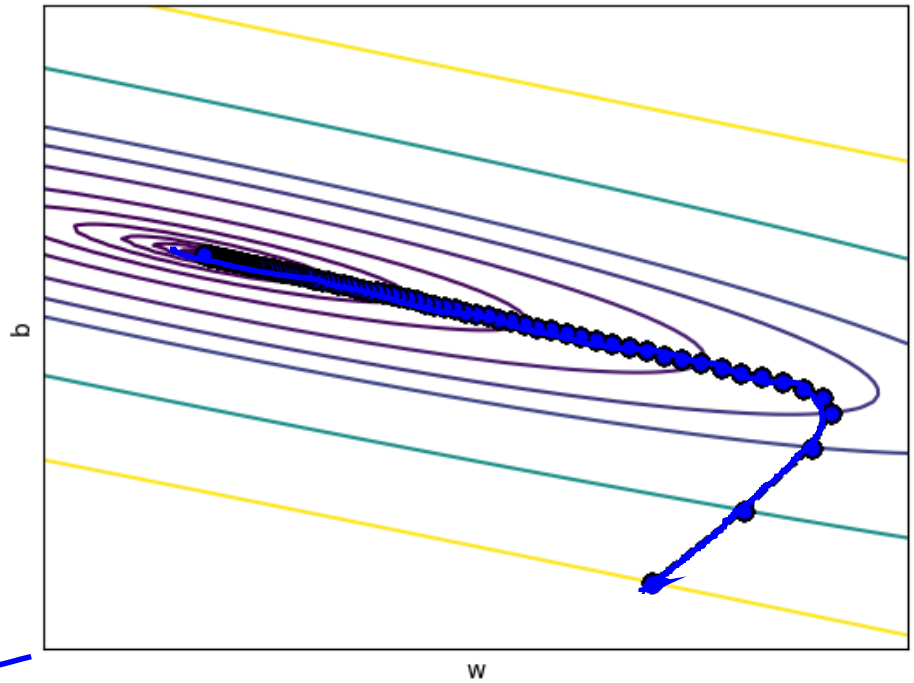


Figure: Epoch: 175

Total Gradient Evaluations: 175000

Stochastic Gradient Descent

Stochastic Gradient Descent:

1. Initialize $\mathbf{w}_0$
2. Repeat until an approximate minimum is obtained:
   1. Randomly shuffle examples in the training set.
   2. For $i = 1, 2, \cdots, n$, do:
      1. $\mathbf{w} := \mathbf{w} - \eta \nabla f_i(\mathbf{w})$
3. Final $\mathbf{w}$ is the local minimum $\mathbf{w}^*$ of $f(\mathbf{w})$

- Here $f_i(\mathbf{w}) = \frac{1}{2}(g(x_i) - y_i)^2$
- The update with learning rate η:

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \eta \begin{bmatrix} (g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) * x_i \\ (g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) \end{bmatrix}$$

- In Stochastic Gradient Descent, update is done with gradients of just one data point at a time. This significantly reduces computation compared to Gradient Descent.
- Convergence rate: Sub-Linear; Iteration cost: $O(1)$

Stochastic Gradient Descent

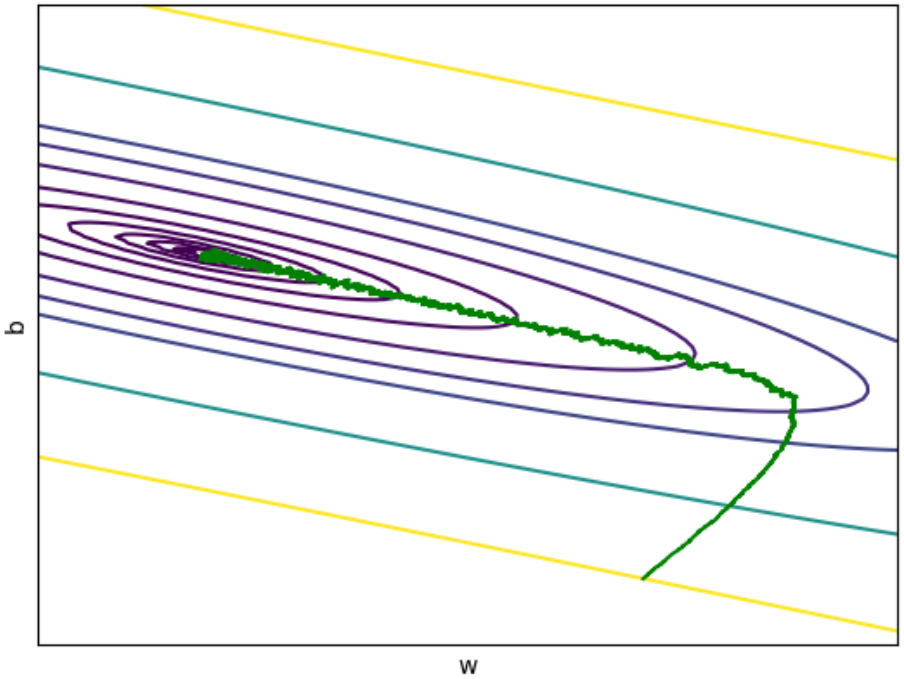
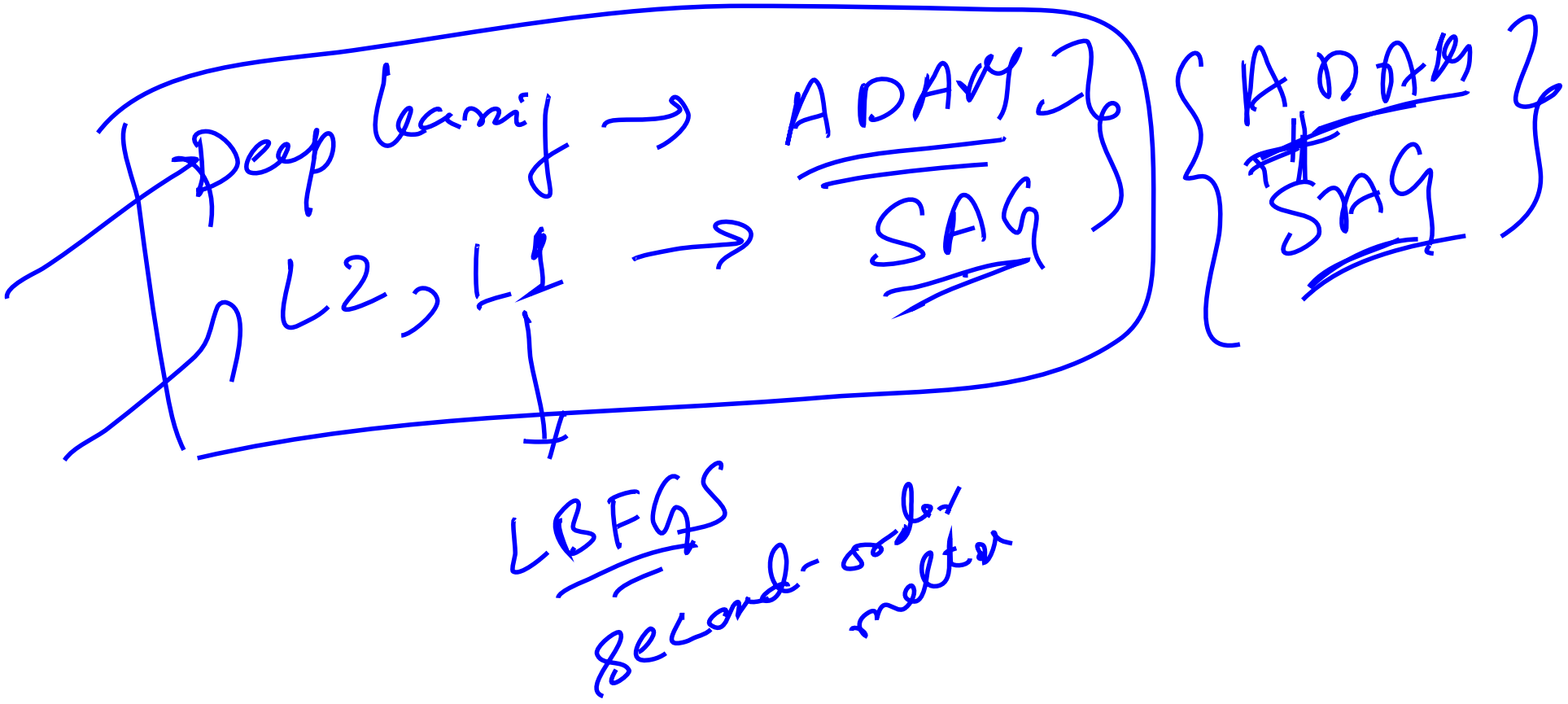


Figure: Epoch: 110
Total Gradient Evaluations: 110000


Deep learning → ADAM
L2, L1 → SAG
ADAM
SAG
LBFGS
second-order
method

Mini-Batched Gradient Descent

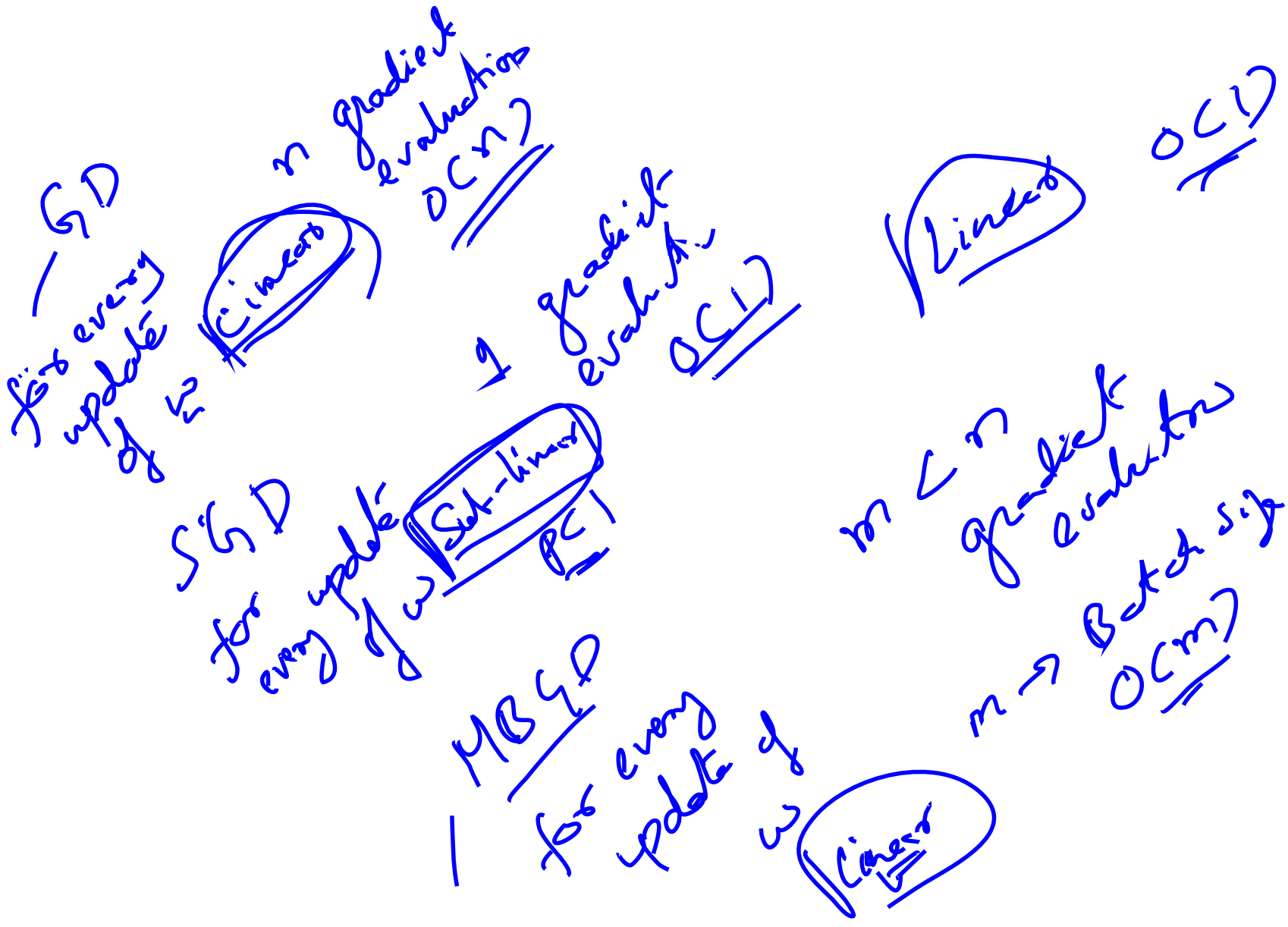
Algorithm:

1. Initialize $\mathbf{w}_0$
2. Repeat until an approximate minimum is obtained:
   1. Randomly shuffle examples in the training set.
   2. Repeat for n/m times:
      1. Calculate $\sum_i \nabla f_i(\mathbf{w})$ for a batch size of m
      2. $\mathbf{w} := \mathbf{w} - \eta * \sum_i \nabla f_i(\mathbf{w})$
3. Final $\mathbf{w}$ is the local minimum $\mathbf{w}^*$ of $f(\mathbf{w})$

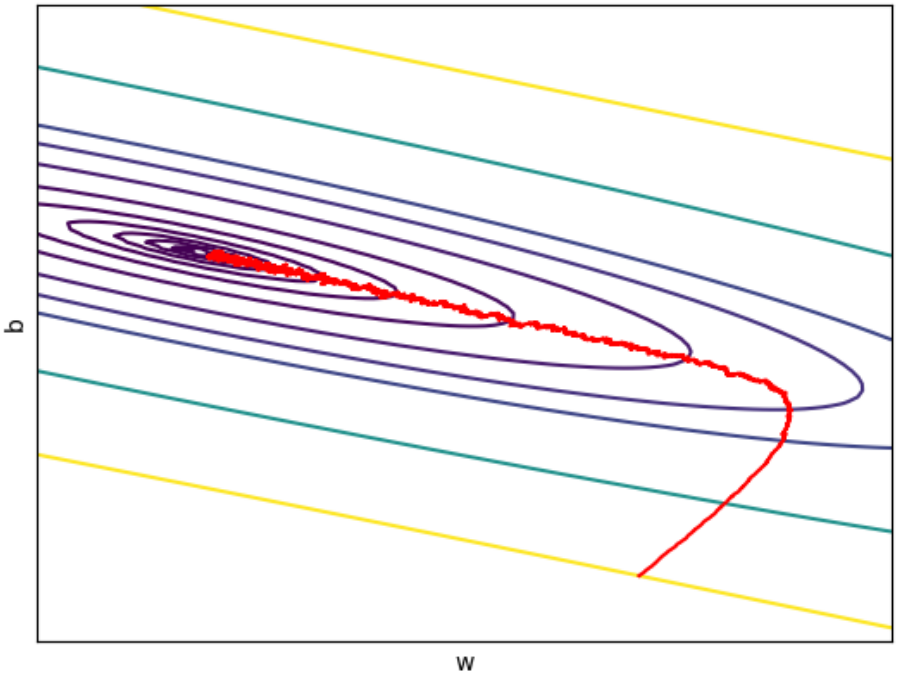
- Here $f_i(\mathbf{w}) = \frac{1}{2}\left(g(x_i) - y_i\right)^2$
- The update with learning rate η:

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \eta \begin{bmatrix} \sum_i (g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) * x_i \\ \sum_i (g(x_i) - y_i) * g(x_i) * (1 - g(x_i)) \end{bmatrix}$$

- In Mini-Batched Gradient Descent, gradients are evaluated for a batch of data training (size m) , $m < n$.
- Convergence rate: Linear, Iteration cost: $O(m)$


GD
for every update of w
linear
n gradient evaluations
O(n)
SGD
for every update of w
Sub-linear
1 gradient evaluation
O(1)
linear
O(1)
MBGD
for every update of w
linear
m < n gradient evaluations
m → Batch size
O(m)

Mini-Batched Gradient Descent



$n = 1000$
$m = 50$

Figure: Epoch: 95
Total Gradient Evaluations: 95000

Stochastic Average Gradient (SAG)

- Like stochastic gradient (SG) methods, the SAG method's iteration cost is independent of the number of terms in the sum.
- However, by incorporating a memory of previous gradient values the SAG method achieves a faster convergence rate than black-box SG methods.
- The update at k^{th} iteration for i^{th} training data:

$$\mathbf{v}_i^k = \begin{cases} \nabla f_i(\mathbf{w}^k) & \text{if } i = i_k \\ \mathbf{v}_i^{k-1} & \text{otherwise} \end{cases}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \eta \begin{bmatrix} \sum_i (v_i)_w \\ \sum_i (v_i)_b \end{bmatrix}$$

Stochastic Average Gradient (SAG)

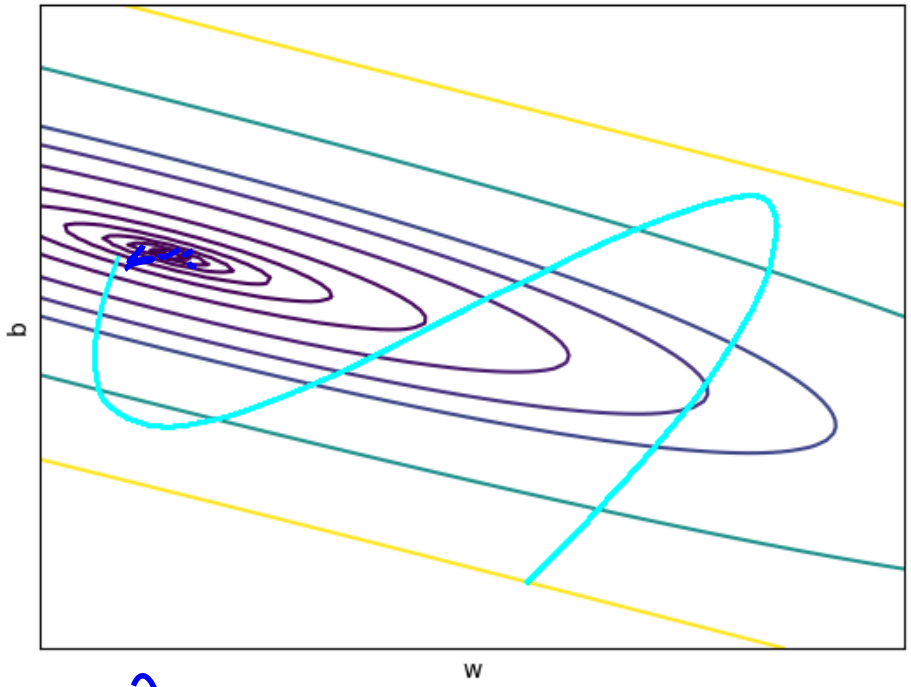
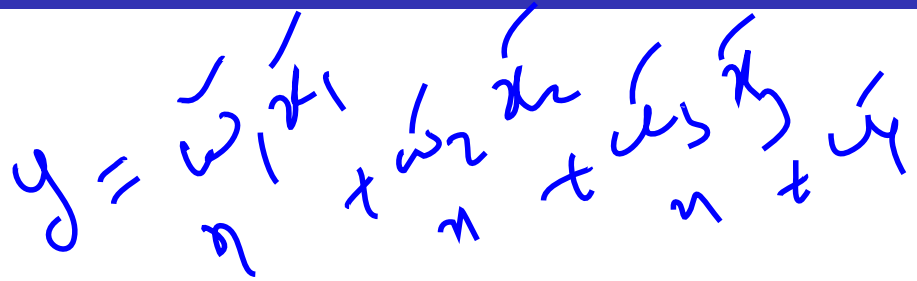


Figure: Epoch: 365
Total Gradient Evaluations: 365

Challenges in gradient descent type methods



- Choice of proper learning rate can be difficult. (Too small $\rightarrow$ small convergence, too large $\rightarrow$ can hinder convergence)
- Having a same learning rate for all parameter updates is problematic when the data is sparse.
- Lot of research has been focused on developing adaptive learning rate for Gradient Descent type methods widely used in Deep learning community.

Adpative learning rate method: AdaGrad (Adaptive Gradient Descent)

- Perform larger updates (higher learning rate) for parameters corresponding to infrequent features and smaller updates (smaller learning rate) for parameters corresponding to frequent features.
- Update rule:

$$\begin{bmatrix} v_w \\ v_b \end{bmatrix} := \begin{bmatrix} v_w \\ v_b \end{bmatrix} + \begin{bmatrix} \left(\frac{\partial f(w,b)}{\partial w}\right)^2 \\ \left(\frac{\partial f(w,b)}{\partial b}\right)^2 \end{bmatrix}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{v_w+\epsilon}} * \frac{\partial f(w,b)}{\partial w} \\ \frac{\eta}{\sqrt{v_b+\epsilon}} * \frac{\partial f(w,b)}{\partial b} \end{bmatrix}$$

- For sparse training data, w corresponding to infrequent feature will have higher learning rate and b will smaller updates. Over time the effective learning rate for b will decay to an extent that there will be no further updates to b.

$$\begin{pmatrix} \omega^{k+1} \\ b^{k+1} \end{pmatrix} := \begin{pmatrix} \omega^{k} \\ b^{k} \end{pmatrix} - \eta \begin{bmatrix} \frac{\partial f}{\partial \omega} \\ \frac{\partial f}{\partial b} \end{bmatrix}$$

$$f = \frac{1}{2} \sum_{i=1}^{n} (g(x_i) - y_i)^2$$

$$= \begin{pmatrix} \omega^{k} \\ b^{k} \end{pmatrix} - \eta \begin{bmatrix} \sum_{i=1}^{n} (g(x_i) - y_i)\, g(x_i)(1 - g(x_i))\, x_i \\ \sum_{i=1}^{n} (g(x_i) - y_i)\, g(x_i)(1 - g(x_i))\, 1 \end{bmatrix}$$

$$x = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ x_4 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}$$

$$\frac{\eta}{\sqrt{v_\omega + \varepsilon}} \qquad \eta_\omega \qquad \eta_b \qquad \frac{\eta}{\sqrt{v_b + \varepsilon}}$$

$i = 3$

$i = 4$

$i = 5$

AdaGrad

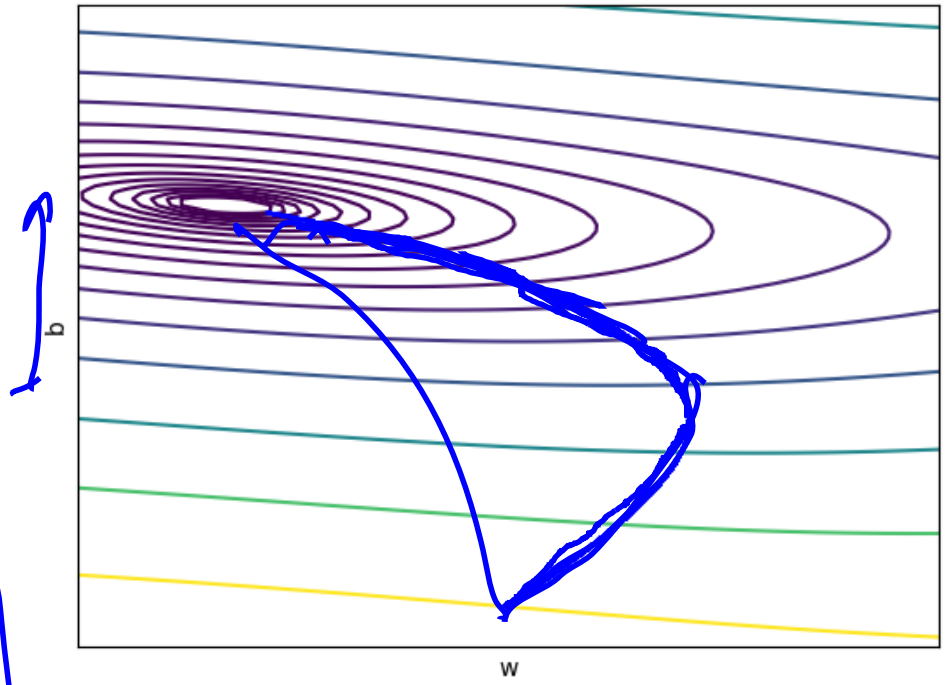


Figure: Epoch: 250
Total Gradient Evaluations: 250000

Adaptive Learning Rate Method:RMSProp

- In AdaGrad, the parameters corresponding to frequent features will start receiving very small updates because of the decayed learning rate. To avoid this the denominator is decayed in RMSProp to prevent its rapid growth.
- Update rule:

$$\begin{bmatrix} v_w \\ v_b \end{bmatrix} := \beta \begin{bmatrix} v_w \\ v_b \end{bmatrix} + (1-\beta) \begin{bmatrix} \left(\frac{\partial f(w,b)}{\partial w}\right)^2 \\ \left(\frac{\partial f(w,b)}{\partial b}\right)^2 \end{bmatrix}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{v_w+\epsilon}} * \frac{\partial f(w,b)}{\partial w} \\ \frac{\eta}{\sqrt{v_b+\epsilon}} * \frac{\partial f(w,b)}{\partial b} \end{bmatrix}$$

- RMSProp overcomes the problem of AdaGrad by being less aggressive on the decay.

$$\begin{pmatrix} v_w \\ v_b \end{pmatrix} := \beta \begin{pmatrix} v_w \\ v_b \end{pmatrix} + (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}$$

$k=0$

$$\begin{pmatrix} v_w^{(0)} \\ v_b^{(0)} \end{pmatrix} := (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(0)}$$

$k=1$

$$\begin{pmatrix} v_w^{(1)} \\ v_b^{(1)} \end{pmatrix} = \beta \left[(1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(0)} \right] + (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(1)}$$

$k=2$

$$\begin{pmatrix} v_w^{(2)} \\ v_b^{(2)} \end{pmatrix} = \beta \left[\beta (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(0)} + (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(1)} \right] + (1-\beta) \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(2)}$$

$$\underbrace{\beta^2 (1-\beta)}_{0.009} \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(0)} + \underbrace{\beta (1-\beta)}_{0.08} \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(1)} + \underbrace{(1-\beta)}_{0.1} \begin{bmatrix} \left(\frac{\partial f}{\partial w}\right)^2 \\ \left(\frac{\partial f}{\partial b}\right)^2 \end{bmatrix}^{(2)}$$

$\beta = 0.9$

RMSProp

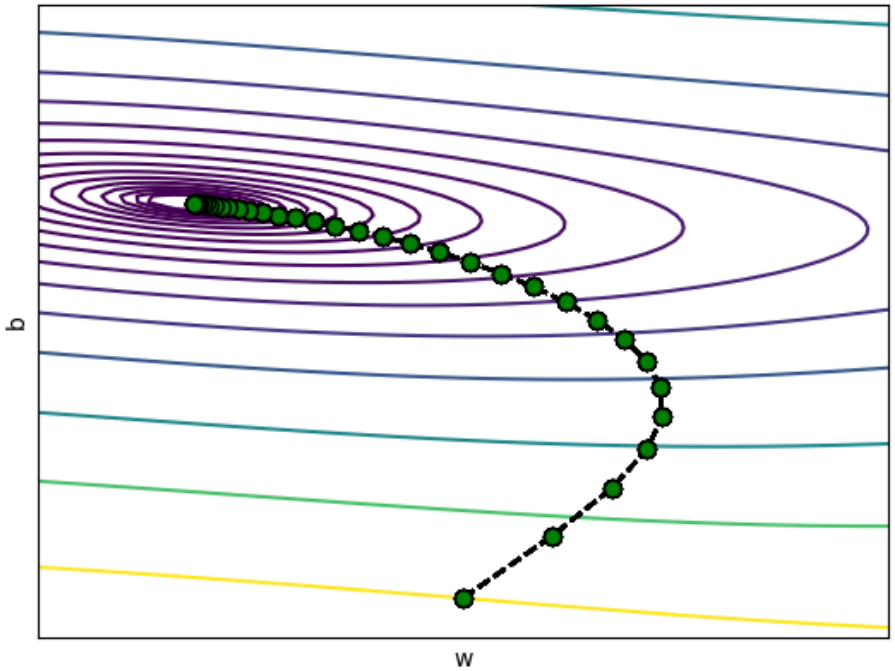


Figure: Epoch: 50
Total Gradient Evaluations: 50000

AdaGrad & RMSProp

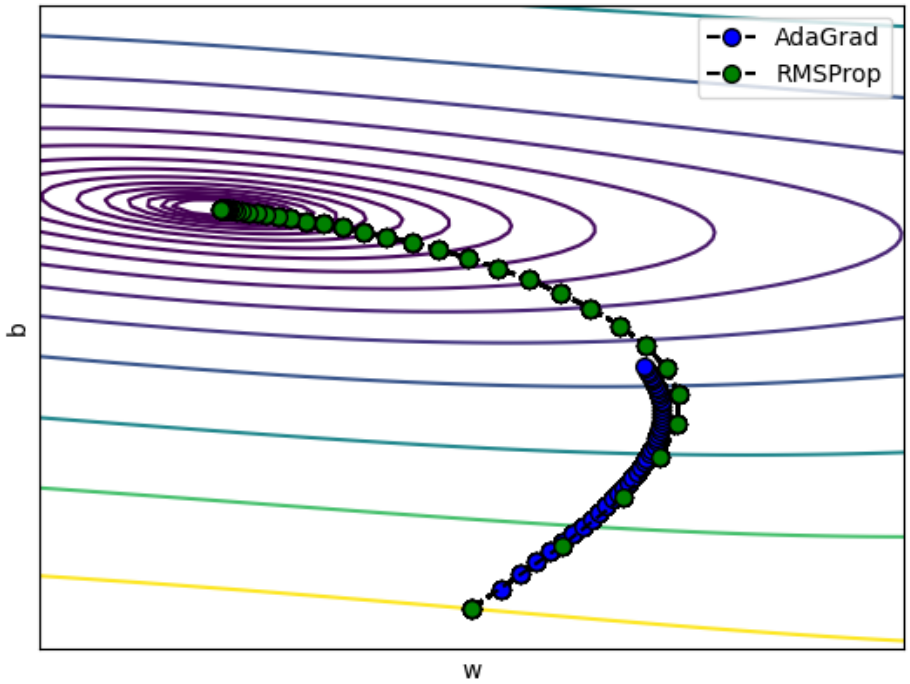


Figure: AdaGrad & RMSProp Epoch: 50
Total Gradient Evaluations: 50000

Adam (Adaptive Moment Estimation)

- Adam in addition to taking care of denominator decay problem as RMSProp, uses a cumulative history of gradients.
- Update rule:

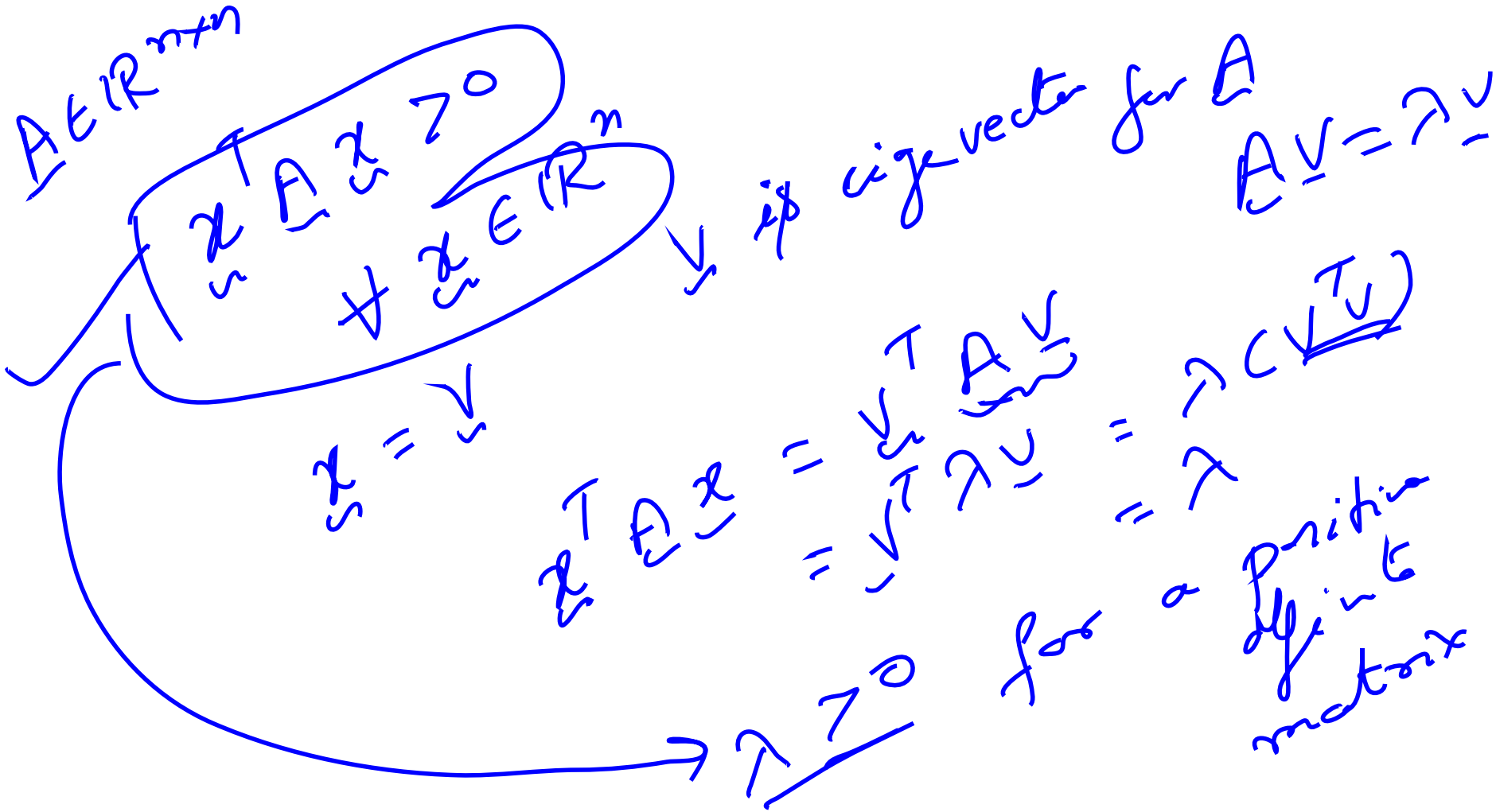
$$\begin{bmatrix} m_w \\ m_b \end{bmatrix} := \beta_1 \begin{bmatrix} m_w \\ m_b \end{bmatrix} + (1-\beta_1) \begin{bmatrix} \frac{\partial f(w,b)}{\partial w} \\ \frac{\partial f(w,b)}{\partial b} \end{bmatrix}$$

$$\begin{bmatrix} v_w \\ v_b \end{bmatrix} := \beta_2 \begin{bmatrix} v_w \\ v_b \end{bmatrix} + (1-\beta_2) \begin{bmatrix} \left(\frac{\partial f(w,b)}{\partial w}\right)^2 \\ \left(\frac{\partial f(w,b)}{\partial b}\right)^2 \end{bmatrix}$$

$$\begin{bmatrix} \hat{m}_w \\ \hat{m}_b \end{bmatrix} := \frac{1}{1-\beta_1^i} \begin{bmatrix} m_w \\ m_b \end{bmatrix} \qquad \begin{bmatrix} \hat{v}_w \\ \hat{v}_b \end{bmatrix} := \frac{1}{1-\beta_2^i} \begin{bmatrix} v_w \\ v_b \end{bmatrix} \qquad \text{for } i^{th} \text{ iteration}$$

$$\begin{bmatrix} w \\ b \end{bmatrix} := \begin{bmatrix} w \\ b \end{bmatrix} - \begin{bmatrix} \frac{\eta}{\sqrt{\hat{v}_w+\epsilon}} * \hat{m}_w \\ \frac{\eta}{\sqrt{\hat{v}_b+\epsilon}} * \hat{m}_b \end{bmatrix}$$

- Adam takes a cumulative history of gradients which speeds it up and converges faster than AdaGrad and RMSProp.


$A \in \mathbb{R}^{n \times n}$
$x^T A x > 0$
$\forall x \in \mathbb{R}^n$
$x \neq 0$
$x^T A x = v^T A v$
$= v^T \lambda v = \lambda c$
$= \lambda$
$\lambda > 0$
v is eigenvector for A
$Av = \lambda v$
for a positive definite matrix

Adam

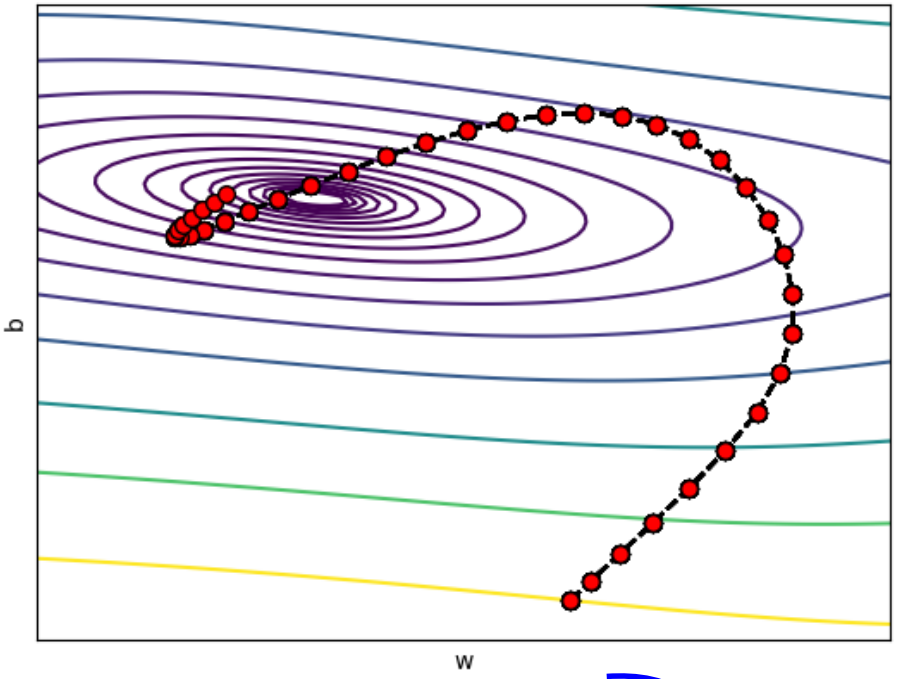


Figure: Epoch: 40
Total Gradient Evaluations: 40000