Lecture 5: Optimization

COMPSCI/DATA 182: Deep Learning

2024/09/12

Today .....

- So far we know
  - The simple neural network model
  - Negative log likelihood (cross-entropy) loss function
  - Computing *gradients* of the loss function with respect to the model parameters: backprop and autodiff
- Backpropagation
- Today, is all about Gradient based Optimization

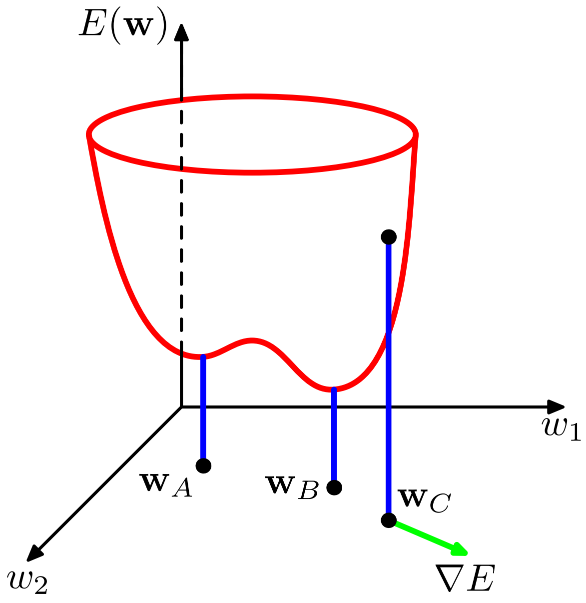
Today’s Material

- Chapter 7 on Gradient Optimization (Bishop Book) is an excellent reference

Error Surfaces

$$\delta E \simeq \delta \mathbf{w}^{\mathrm{T}} \nabla E(\mathbf{w})$$

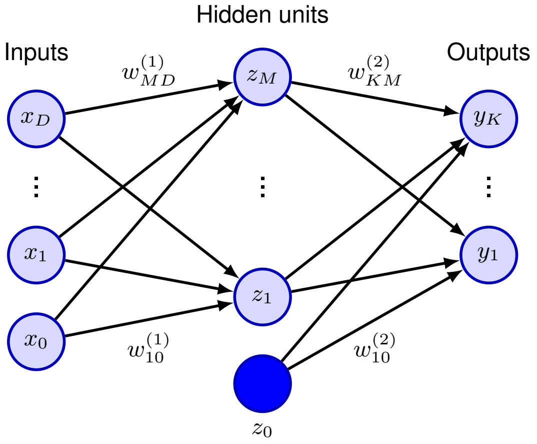
$$\nabla E(\mathbf{w}) = 0$$

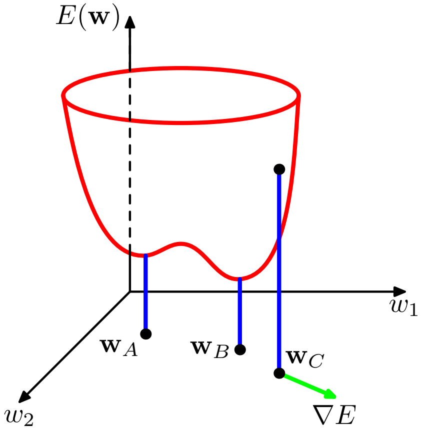


- Minima, Maxima, Saddle point (aka Local minimum)

LEGEND
KIDS
FAMILY ZONE
TERRAIN PARKS
GMC RACE CENTRE
COCA-COLA TUBE PARK
BEGINNERS
BASE II
BEGINNERS
BLACKCOMB DAYLODGE
VILLAGE
CREEKSIDE

Complexity of this surface





- $M!2^M$ points

Optimization techniques: Local quadratic optimization

- Taylor expansion
- Hessian: matrix of second-order derivatives
  - $O(W^3)$
- Merit of using gradient descent : $O(W^2)$

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}})$$

Gradient Descent

- Little hope of finding an analytical solution to delta $\Delta E(w) = 0$
- Iterative optimization for complex continuous nonlinear functions
  - Well studied
- Initial weights: w^0
- Gradients, and complexity
- **Batch** gradient descent

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \Delta\mathbf{w}^{(\tau-1)}$$

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

Stochastic Gradient Descent

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}).$$

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \nabla E_n(\mathbf{w}^{(\tau-1)}).$$

Algorithm 7.1: Stochastic gradient descent

Input: Training set of data points indexed by $n \in \{1, \dots, N\}$
Error function per data point $E_n(\mathbf{w})$
Learning rate parameter η
Initial weight vector $\mathbf{w}$
Output: Final weight vector $\mathbf{w}$

$n \leftarrow 1$
repeat
  $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_n(\mathbf{w})$ `// update weight vector`
  $n \leftarrow n + 1 (\mathrm{mod}\ N)$ `// iterate over data`
until convergence
return $\mathbf{w}$

- *All* data points
- An epoch

Stochastic Gradient Descent

- *Mini* batch
- Parameter initialization
  - *"He initialization"* (Gaussian)

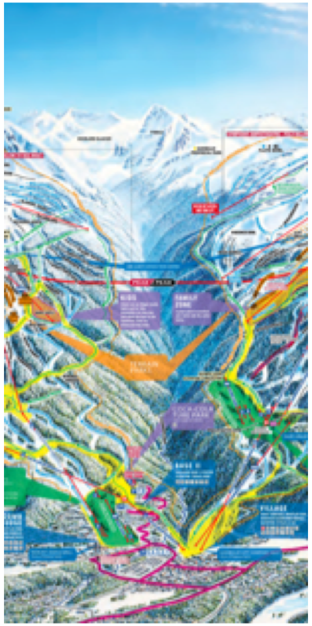
$$a_i^{(l)} = \sum_{j=1}^{M} w_{ij} z_j^{(l-1)}$$

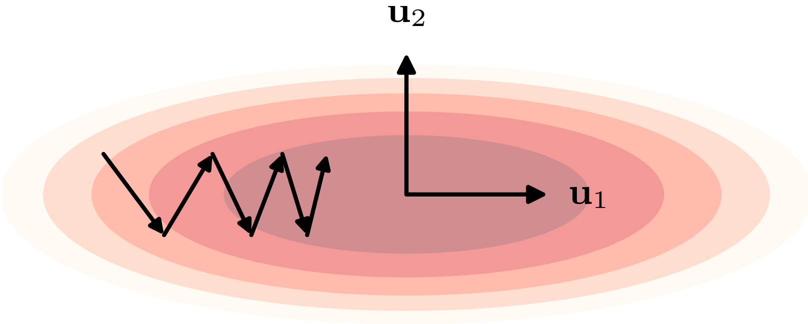
$$z_i^{(l)} = \mathrm{ReLU}(a_i^{(l)})$$

$$\mathbb{E}[a_i^{(l)}] = 0$$

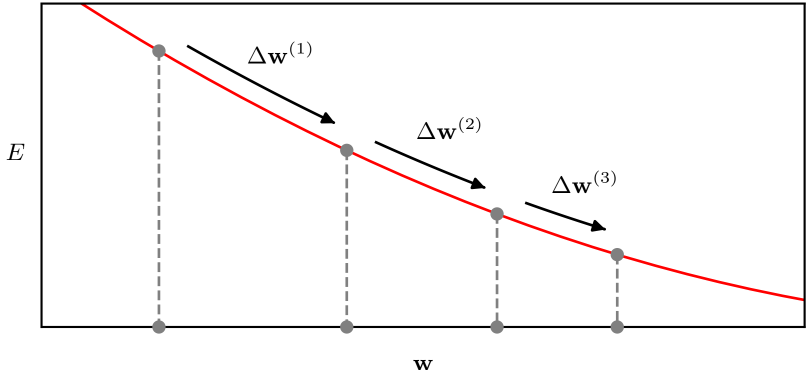
$$\mathrm{var}[z_j^{(l)}] = \frac{M}{2}\epsilon^2\lambda^2$$

Convergence: Problem with fixed step gradient



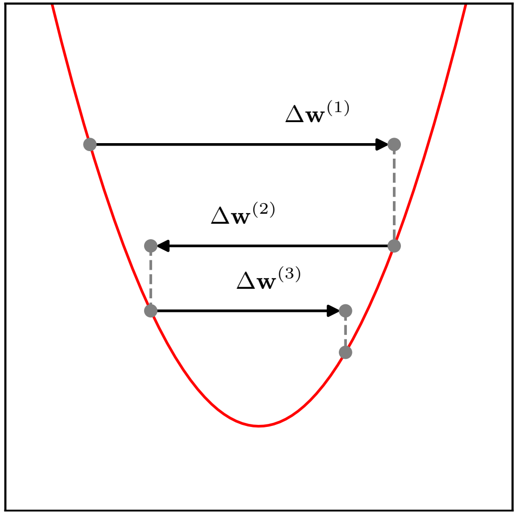


Momentum



$$\Delta\mathbf{w}^{(\tau-1)} = -\eta\nabla E\left(\mathbf{w}^{(\tau-1)}\right) + \mu\Delta\mathbf{w}^{(\tau-2)}$$

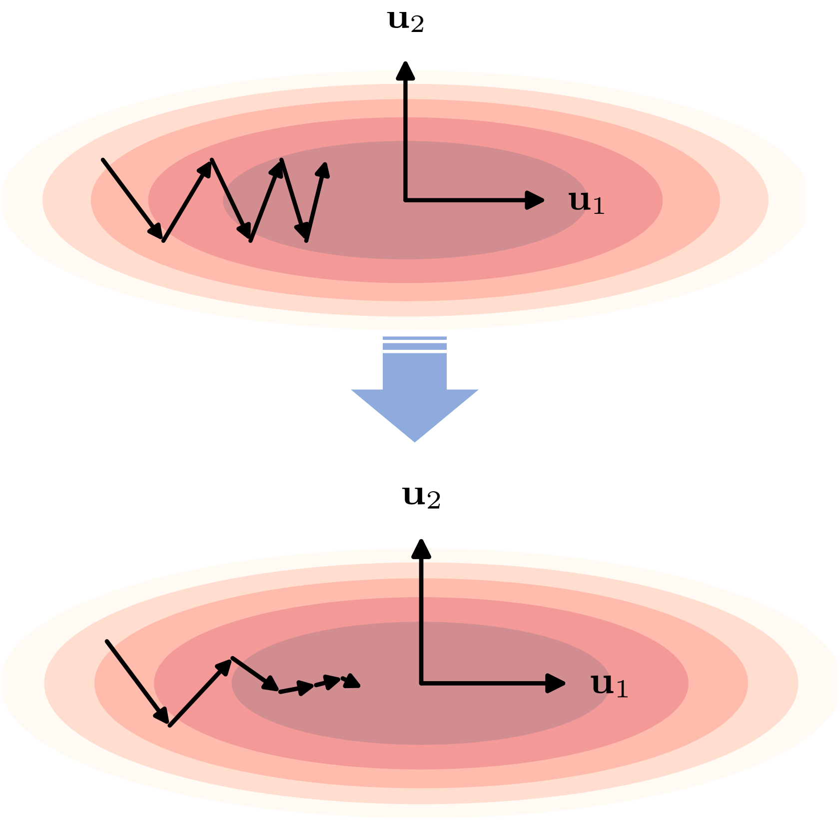
$$\begin{aligned}\Delta\mathbf{w} &= -\eta\nabla E\{1+\mu+\mu^2+\ldots\} \\ &= -\frac{\eta}{1-\mu}\nabla E\end{aligned}$$



Momentum

$$\Delta \mathbf{w}^{(\tau-1)} = -\eta \nabla E\left(\mathbf{w}^{(\tau-1)}\right) + \mu \Delta \mathbf{w}^{(\tau-2)}$$

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



Nesterov Momentum

$$\Delta \mathbf{w}^{(\tau-1)} = -\eta \nabla E\left(\mathbf{w}^{(\tau-1)} + \mu \Delta \mathbf{w}^{(\tau-2)}\right) + \mu \Delta \mathbf{w}^{(\tau-2)}.$$

Learning Rate Schedule

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta^{(\tau-1)} \nabla E_n(\mathbf{w}^{(\tau-1)}).$$

AdaGrad, RMSProp, Adam

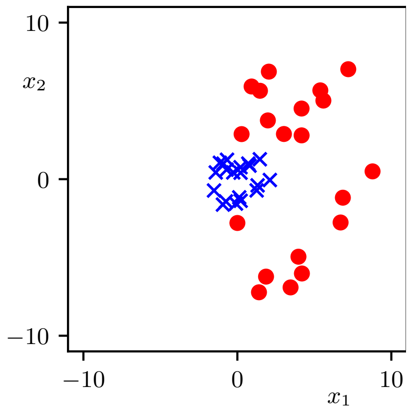
$$r_i^{(\tau)} = r_i^{(\tau-1)} + \left(\frac{\partial E(\mathbf{w})}{\partial w_i}\right)^2$$

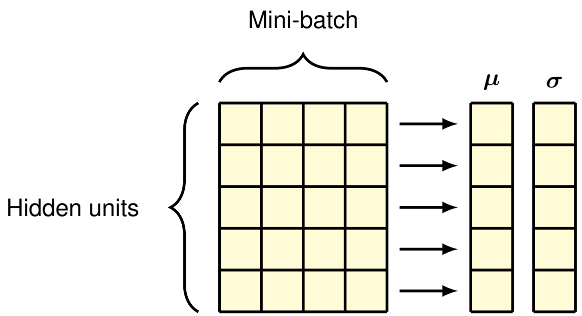
$$w_i^{(\tau)} = w_i^{(\tau-1)} - \frac{\eta}{\sqrt{r_i^{\tau}} + \delta}\left(\frac{\partial E(\mathbf{w})}{\partial w_i}\right)$$

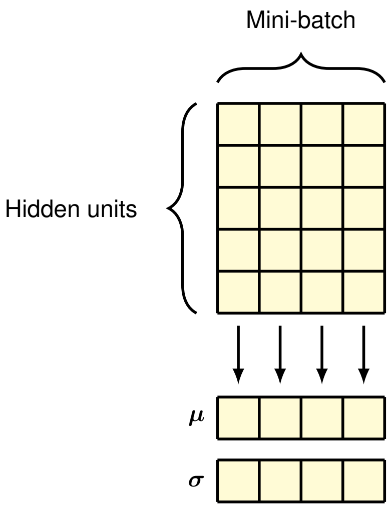
- Adam: *Combine* RMSProp and Momentum

Normalization

- Data
- Batch
- Layer







What’s so great about Adam?

- Empirically, Adam seems to work well “out of the box” for many neural networks
- It combines momentum with a cheap approximation of second order information — actual second order methods like *Newton’s method* are far too expensive
  - There’s also some relationship to methods which “adapt” the learning rate separately for each parameter — *AdaGrad* and *RMSProp*
- The important takeaway: when tackling a new deep learning problem, most people will try both stochastic gradients with momentum and Adam
  - Hopefully at least one of them does well…