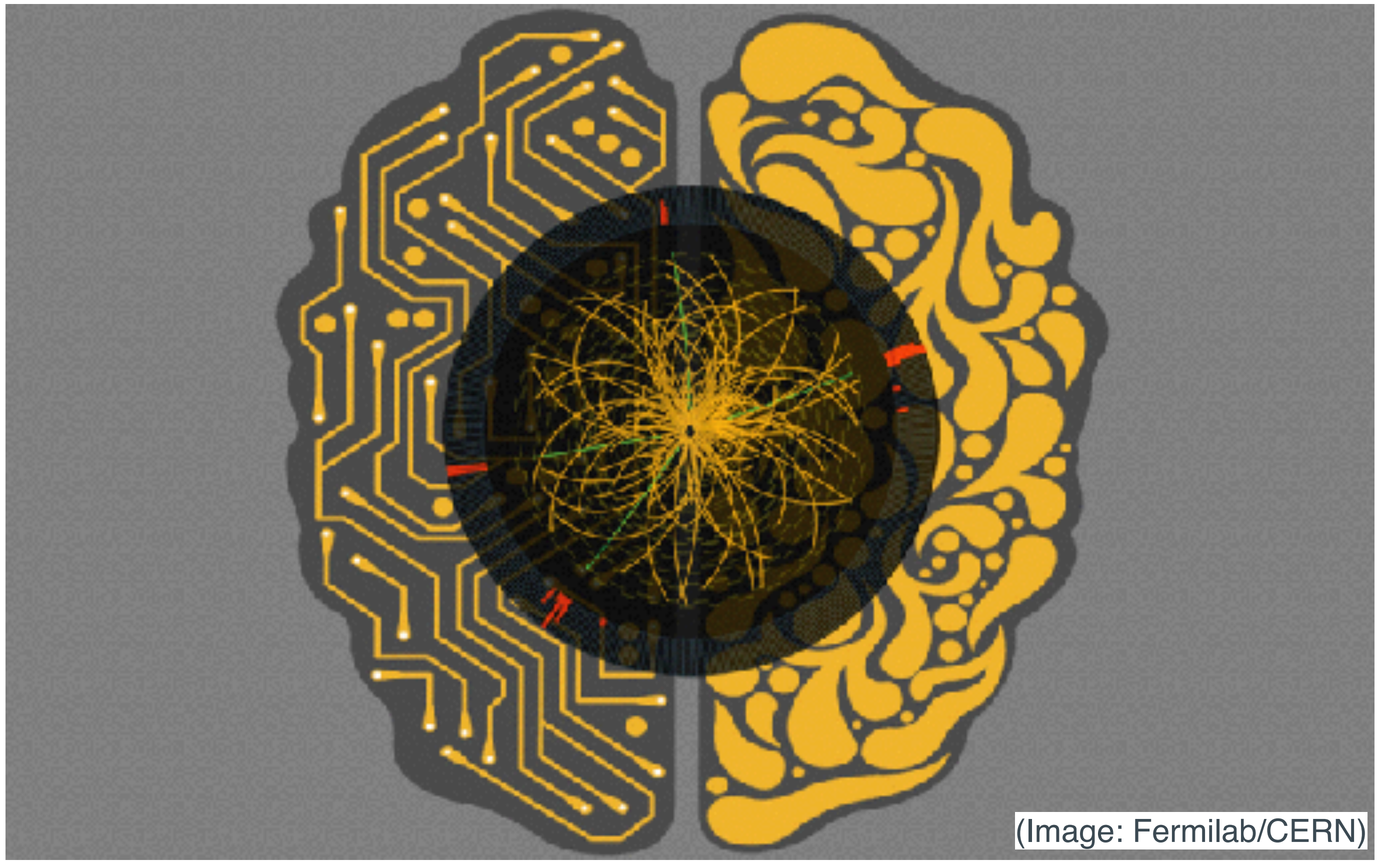
PHY 835: Collider Physics Phenomenology

Machine Learning in Fundamental Physics

Gary Shiu, UW-Madison



Lecture 3: Optimizers

Recap of Lecture 2

- Typical problem in ML.
- Splitting data into training set and test set.
- In-sample error E_{in} may not equal out-of-sample error E_{out}
- **Bias-variance trade-off**:

$$< E_{out} > = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

- Example: polynomial regression.

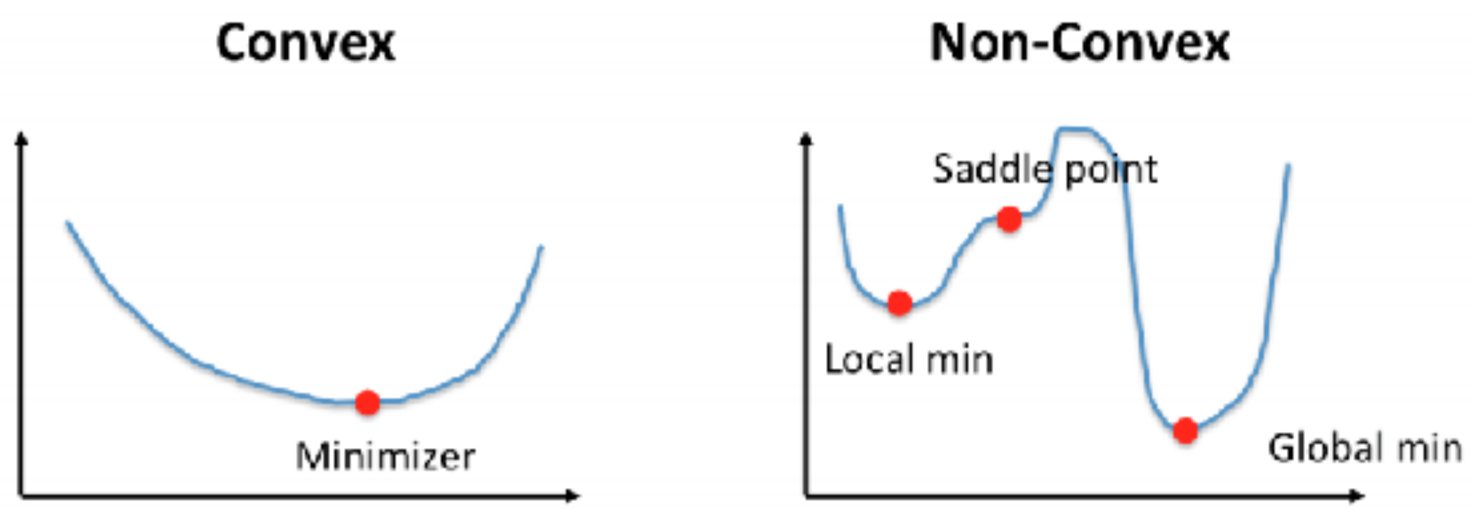
Outline for today

- Gradient Descent
- Gradient descent vs Newton's method
- Limitations of Gradient Descent
- Stochastic Gradient Descent
- Adding momentum

References: 1803.08823 (see also Goodfellow et al, Ch. 8)

Optimizers

- ML problems are mostly about minimizing a cost function. This can be a hard problem because:
  - The function depends on many parameters, say $\mathcal{O}(10^6)$ and hence the minimization is over a huge parameter space.
  - It becomes numerically expensive to evaluate the cost function, its gradient and higher derivatives.
  - Non-convex loss function → multiple minima



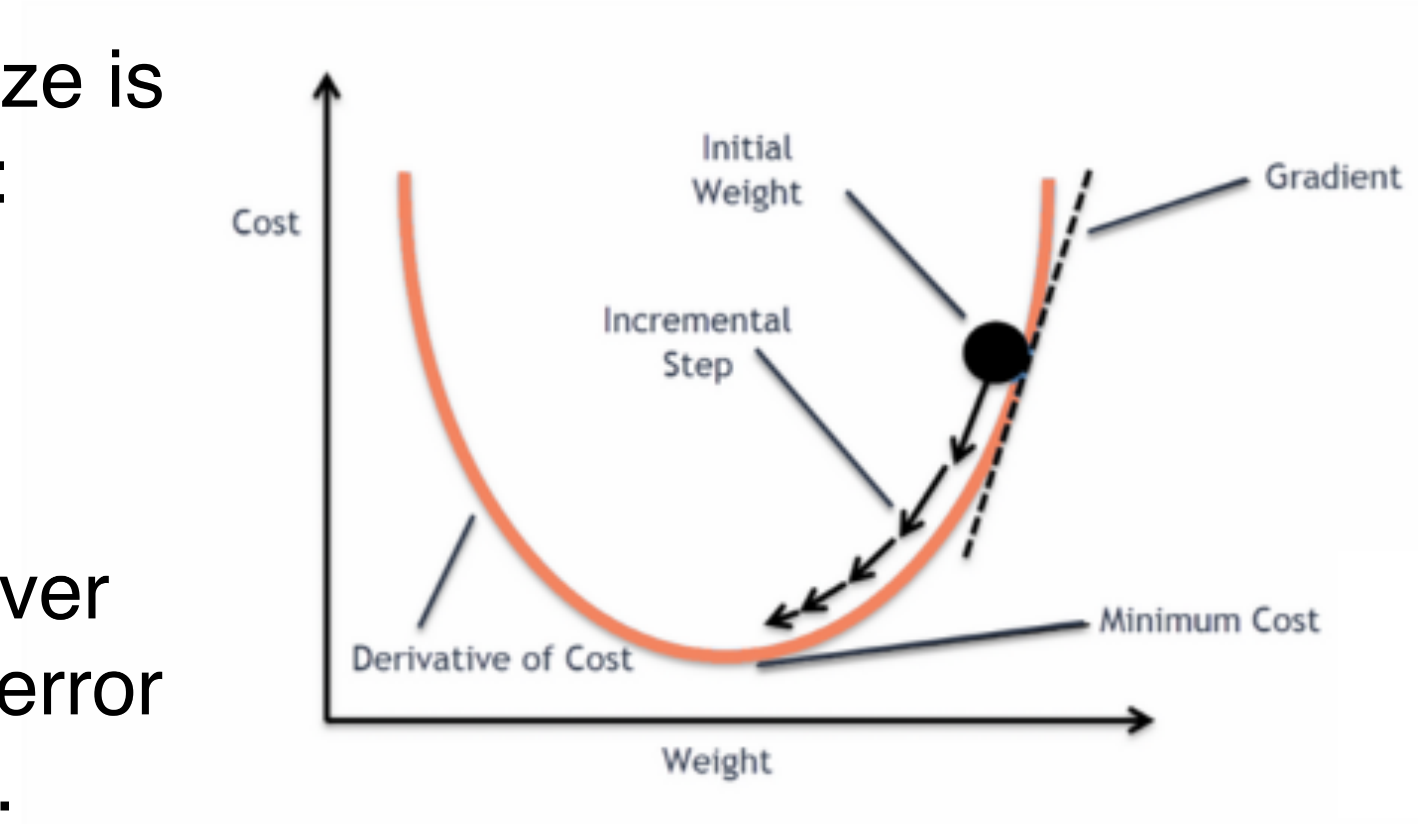
- Common method: **gradient descent** & variations.

Gradient Descent

- The "energy" we want to minimize is the cost function (loss function):

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} e_i(\mathbf{x}_i, \boldsymbol{\theta}).$$

can often be written as a sum over data points, e.g., mean-square error or cross-entropy (classification).



- **Idea:** adjust parameters in the direction where the gradient of $E(\theta)$ is large and negative. Gradually shifting towards a local minimum.

learning rate

$$\mathbf{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t),$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t$$

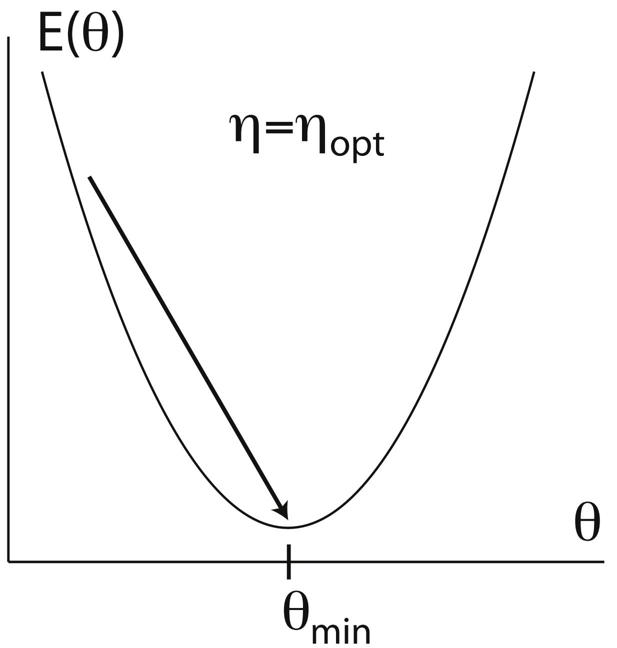
Newton's Method

- Inspiration for many widely used optimization methods.

- Choose the step $\mathbf{v}$ for the parameter θ to minimize a 2nd order Taylor expansion:

$$E(\boldsymbol{\theta}+\mathbf{v}) \approx E(\boldsymbol{\theta}) + \nabla_{\theta}E(\boldsymbol{\theta})\mathbf{v} + \frac{1}{2}\mathbf{v}^T H(\boldsymbol{\theta})\mathbf{v},$$

where $H(\theta)$ is the Hessian. Differentiate w.r.t. $\mathbf{v}$, noting that for the optimal value $\mathbf{v}_{opt}$, $\nabla_{\theta}E(\theta+\mathbf{v}_{opt}) = 0$:



$$\mathbf{v}_t = H^{-1}(\boldsymbol{\theta}_t)\nabla_{\theta}E(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t.$$

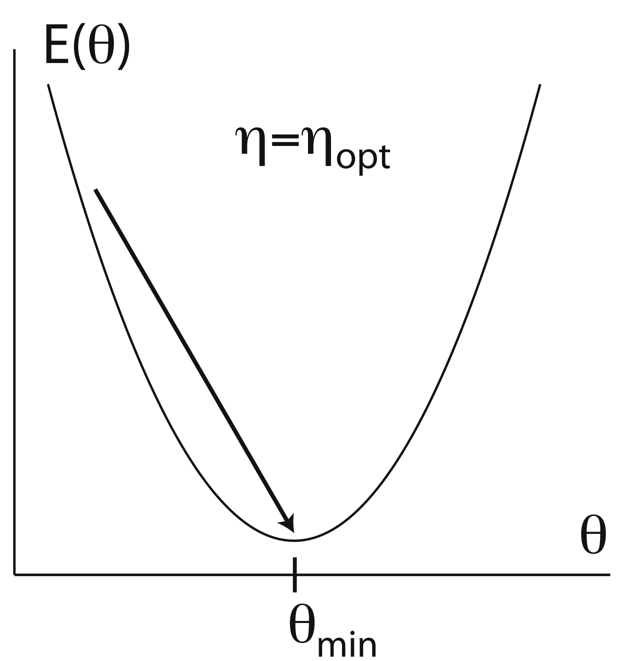
Newton's Method

- Inspiration for many widely used optimization methods.

- Choose the step $\mathbf{v}$ for the parameter θ to minimize a 2nd order Taylor expansion:

$$E(\boldsymbol{\theta} + \mathbf{v}) \approx E(\boldsymbol{\theta}) + \nabla_\theta E(\boldsymbol{\theta})\mathbf{v} + \frac{1}{2}\mathbf{v}^T H(\boldsymbol{\theta})\mathbf{v},$$

where $H(\theta)$ is the Hessian. Differentiate w.r.t. $\mathbf{v}$, noting that for the optimal value $\mathbf{v}_{opt}$, $\nabla_\theta E(\theta + \mathbf{v}_{opt}) = 0$:



$[H(\theta_t) + \epsilon I]^{-1}$

$$\mathbf{v}_t = H^{-1}(\boldsymbol{\theta}_t)\nabla_\theta E(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t.$$

Gradient Descent vs Newton's Method

- Newton's method requires knowledge of 2nd derivatives (n^2 component Hessian) which is computationally expensive.

- Calculating inverse of the Hessian is expensive especially for millions of parameters (common in neural network applications).

  ⇒ Newton's method unfeasible for typical ML systems.

- However, useful to get intuition how to choose the **learning rate:**

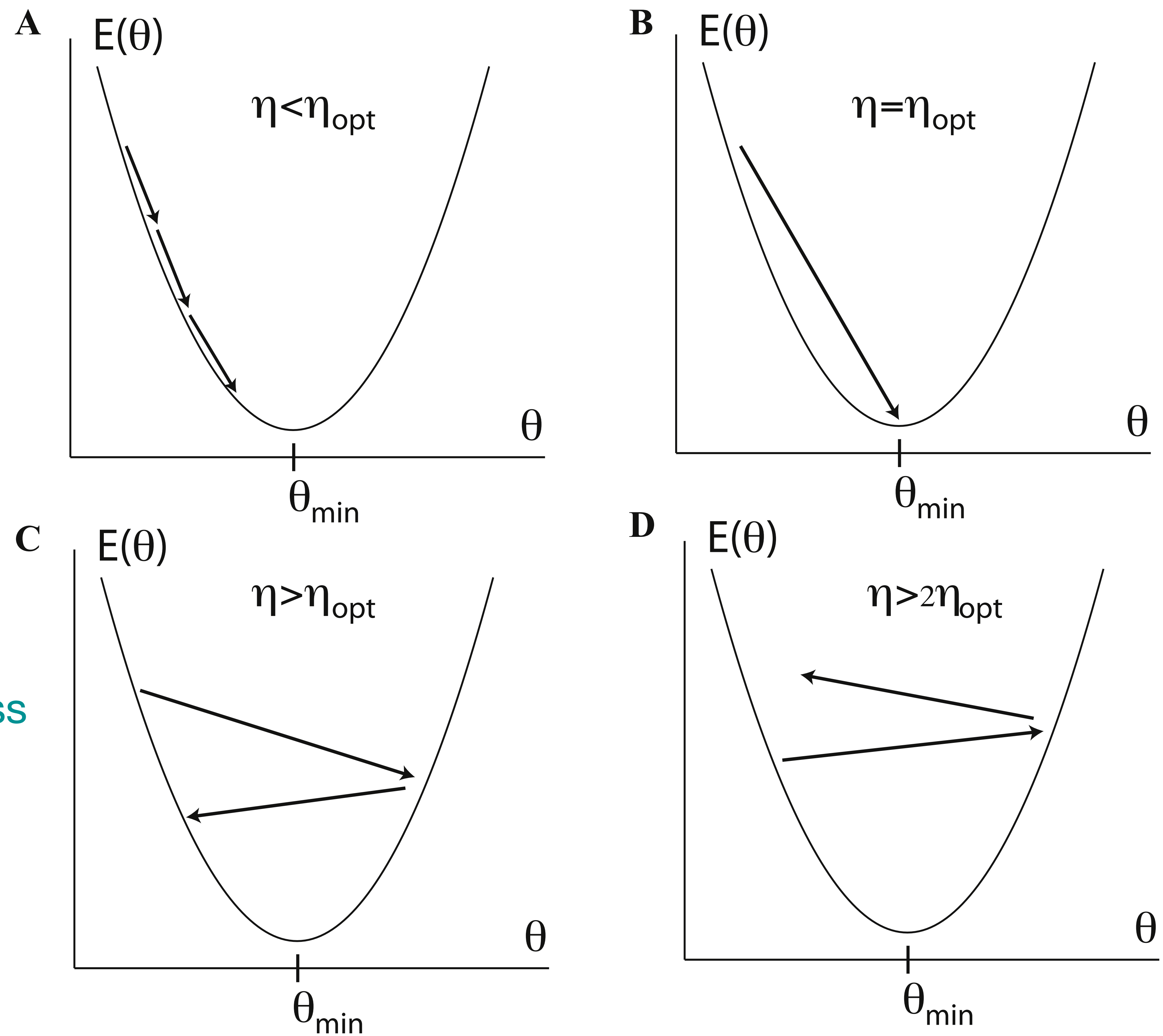
$$\eta_{\text{opt}} = [\partial_\theta^2 E(\theta)]^{-1}$$ **(1-dim)**

- Newton's method automatically **adjusts the learning rate**: takes larger steps in flat directions and smaller steps in steep directions.

Regimes of Learning Rate

multiple
small steps

single
step



oscillate across
both sides

diverges

LeCun et al, 1998

Convergence in Higher Dimensions

- Natural generalization of $\partial_\theta^2 E(\theta)$ is the Hessian.
- Perform a singular value decomposition of the Hessian matrix:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^{\mathrm{T}},$$

where U and V are orthogonal matrices and D is diagonal with eigenvalues $\{\lambda_{min}, \ldots, \lambda_{max}\}$.

- Convergence of gradient descent requires:

$$\eta < \frac{2}{\lambda_{\max}},$$

- If $\lambda_{min} \ll \lambda_{max}$, convergence is slow in the λ_{min} direction. Convergence time scale scales with $\kappa = \lambda_{max}/\lambda_{min}$.

Gradient Descent — Limitations

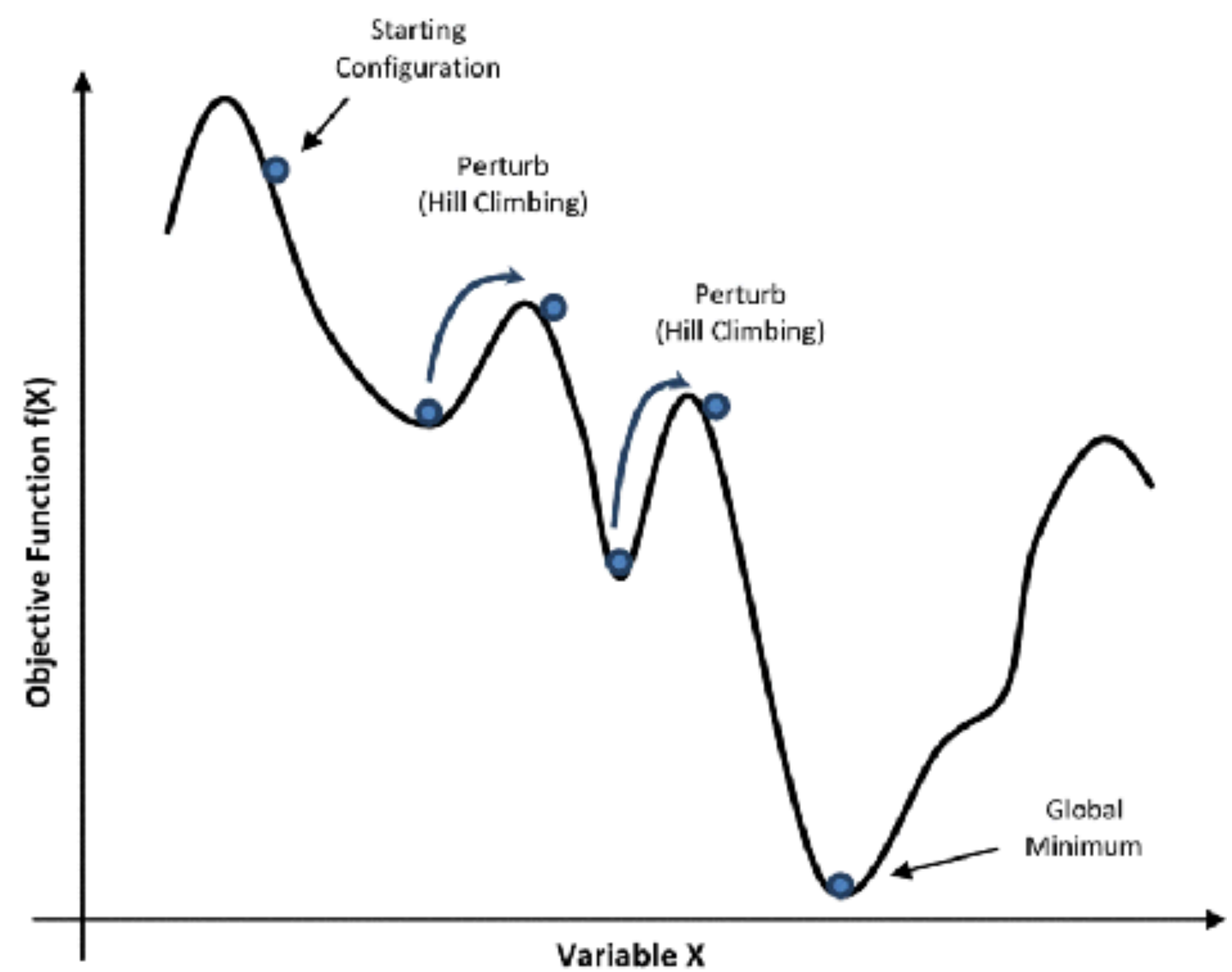
- Finds **local minima**: simulated annealing introduces a "temperature" (stochasticity) to tunnel over energy barriers.

- **Sensitive to initial conditions** (which local minimum depends on starting point)

→ important to consider sensible **initialization** of training process.

- Gradients computationally expensive for large datasets

→ calculate gradient using small subset of data: **"mini-batches"** (gives stochasticity)



Stochastic Gradient Descent (SGD)

Gradient Descent — Limitations

- **Sensitive to choice of learning rates** (too small would take a long time to train, too large would diverge from minima).

  → Furthermore need to adaptively choose learning rate.

- **Treats all directions uniformly**

  → ideally large steps in flat directions, small steps in steep directions

  → second derivatives needed to account for “curvature effects”.

- Takes exponential amount of time to **escape a saddle point**.

You are encouraged to experiment with gradient descent and its variants using the Juypter notebook on:

https://physics.bu.edu/%7Epankajm/MLnotebooks.html

SGD with Mini-batches

- **Stochasticity** by approximating gradient on subset of data, so-called **mini-batches**, denoted as B_k (size varies ~10-100):

$$D \to B_1, B_2, \ldots, B_n$$

- **Speed up gradient computation:**

$$\nabla_\theta E(\boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla_\theta e_i(\mathbf{x}_i, \boldsymbol{\theta}) \longrightarrow \sum_{i \in B_k} \nabla_\theta e_i(\mathbf{x}_i, \boldsymbol{\theta})$$

- Perform gradient descent:

$$\begin{aligned} \mathbf{v}_t &= \eta_t \nabla_\theta E^{MB}(\boldsymbol{\theta}), \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \mathbf{v}_t. \end{aligned}$$

- Cycle through mini-batches. One entire cycle is known as an **epoch.**

- Bonus: works effectively as a natural regularizer that **prevents overfitting** in deep, isolated minima **(Bishop 1995)**.

GD with Momentum (GDM)

- **Idea:** add memory of the direction we move in parameter space

$$\mathbf{v}_t = \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta E(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{v}_t,$$

  by introducing a **momentum parameter** γ, with $0 \leq \gamma \leq 1$

- The step taken $\mathbf{v}$ is a running average of recently encountered gradients with the characteristic time scale for the memory set by γ.

- To get some physics intuition, consider a massive particle in viscous medium with viscous damping coefficient μ, and potential $E(\theta)$:

$$m \frac{d^2\mathbf{w}}{dt^2} + \mu \frac{d\mathbf{w}}{dt} = -\nabla_w E(\mathbf{w}).$$

GD with Momentum (GDM)

- Discrete version of this EOM:

$$m\frac{\mathbf{w}_{t+\Delta t} - 2\mathbf{w}_t + \mathbf{w}_{t-\Delta t}}{(\Delta t)^2} + \mu\frac{\mathbf{w}_{t+\Delta t} - \mathbf{w}_t}{\Delta t} = -\nabla_w E(\mathbf{w}).$$

- Bringing it to a form of a GDM:

$$\Delta\mathbf{w}_{t+\Delta t} = -\frac{(\Delta t)^2}{m + \mu\Delta t}\nabla_w E(\mathbf{w}) + \frac{m}{m + \mu\Delta t}\Delta\mathbf{w}_t.$$

- The momentum parameter and the learning rate are then identified:

$$\gamma = \frac{m}{m + \mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m + \mu\Delta t}.$$

GD with Momentum (GDM)

- **Gain speed** in directions with persistent but small gradient, while **suppressing oscillations** in high curvature directions.

- Especially useful when $E(\theta)$ is shallow and flat in some directions, and narrow and steep in others.

- More useful during the **transient phase** than the fine-tuning phase.

- Slight modification: Nesterov accelerated gradient (NAG) descent (update at expected value of parameters with current momentum):

$$\begin{aligned} \mathbf{v}_t &= \gamma \mathbf{v}_{t-1} + \eta_t \nabla_\theta E(\boldsymbol{\theta}_t + \gamma \mathbf{v}_{t-1}) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \mathbf{v}_t . \end{aligned}$$

Using the 2nd Moment

- Ideally calculate/approximate Hessian and normalize learning rates accordingly.

- In addition to keeping a running average of the first moment of the gradient (momentum), we also keep track of the **second moment**:

$$\mathbf{S}_t = \mathbb{E}[\mathbf{g}_t^2]$$

- Methods include: AdaGrad (2011), AdaDelta (2012), **RMS-Prop (2012), ADAM (2014).**

- **RMS-Prop** update rules:

$$\begin{aligned} \mathbf{g}_t &= \nabla_\theta E(\boldsymbol{\theta}) \\ \mathbf{s}_t &= \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2 \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t + \epsilon}}, \end{aligned}$$

RMS-Prop

- RMS-Prop update rules:

$\beta \approx 0.9$ controls the averaging time of the 2nd moment

$$
\begin{aligned}
\mathbf{g}_t &= \nabla_\theta E(\boldsymbol{\theta}) \\
\mathbf{s}_t &= \beta \mathbf{s}_{t-1} + (1-\beta)\mathbf{g}_t^2 \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{g}_t}{\sqrt{\mathbf{s}_t + \epsilon}},
\end{aligned}
$$

$\epsilon \approx 10^{-8}$ regularizes divergences

- Learning rate is reduced in directions where the norm of the gradient is consistently large.

- Speeds up convergence by allowing us to use a larger learning rate for flat directions.

ADAM

- Using a running average of both the 1st and 2nd moments:

$$
\begin{aligned}
\mathbf{g}_t &= \nabla_\theta E(\boldsymbol{\theta}) \\
\mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \\
\mathbf{s}_t &= \beta_2 \mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \\
\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1-(\beta_1)^t} \\
\hat{\mathbf{s}}_t &= \frac{\mathbf{s}_t}{1-(\beta_2)^t} \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon},
\end{aligned}
$$

- Memory lifetimes of the 1st and 2nd moment are typically:

$$\beta_1 = 0.9,\ \beta_2 = 0.99$$

ADAM

- Understanding the update rule a bit further. Consider limits of

$$\Delta\theta_{t+1} = -\eta_t \frac{\hat{m}_t}{\sqrt{\sigma_t^2 + \hat{m}_t^2} + \epsilon}. \qquad \boldsymbol{\sigma}_t^2 = \hat{\mathbf{s}}_t - (\hat{\mathbf{m}}_t)^2$$

variance

Case 1: $\sigma_t^2 \ll m_t^2$

$$\Delta\theta_{t+1} = -\eta_t$$

Cutting off large persistent gradients at 1 (limiting step size)

→ prevents oscillations and divergences

Case 2: $\sigma_t^2 \gg m_t^2$

$$\Delta\theta_{t+1} = -\eta_t \frac{m_t}{\sigma_t}$$

Learning rate adapted to signal-to-noise (natural unit)

Practical Tips

- There is no absolute superior optimizer; one should experiment which optimizer and which hyperparameters are suitable for the problem at hand.

- Standard tools: mini-batches, momentum, randomize your batches, transform input to get uniform loss landscape

- Use your physical understanding to find a good method. Analyze the performance difference, find out why something is not working, adapt your method (examples: variants of gradient descent)...

Summary

- What is Gradient Descent?
- Comparing gradient descent vs Newton’s method
- Limitations of Gradient Descent
- Stochastic Gradient Descent
- How can it be modified? E.g. adding momentum
- Second order methods (RMSProp and ADAM)