Training neural networks (2)

EECE454 Intro. to Machine Learning Systems

Fall 2024

Recap

- **Last class.** Setting up the training
    - Gradients and activation functions
    - Data preprocessing
    - Normalization layers
    - Parameter Initialization

Recap

- **Last class.** Setting up the training
  - Gradients and activation functions
  - Data preprocessing
  - Normalization layers
  - Parameter Initialization
- **Today.** Tuning the training process
  - Learning rate & Batch size
  - Optimizers
  - Regularizers
  - Hyperparameter tuning

Learning rate & Batch size

SGD

- **SGD.** Recall that the can be written as

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \left(\sum_{i=1}^{B} \ell(y_i, f_\theta(\mathbf{x}_i)) \right)$$

  - There are two <u>key hyperparameters</u>
    - Learning rate η
    - Batch size B

SGD

- **SGD.** Recall that the can be written as

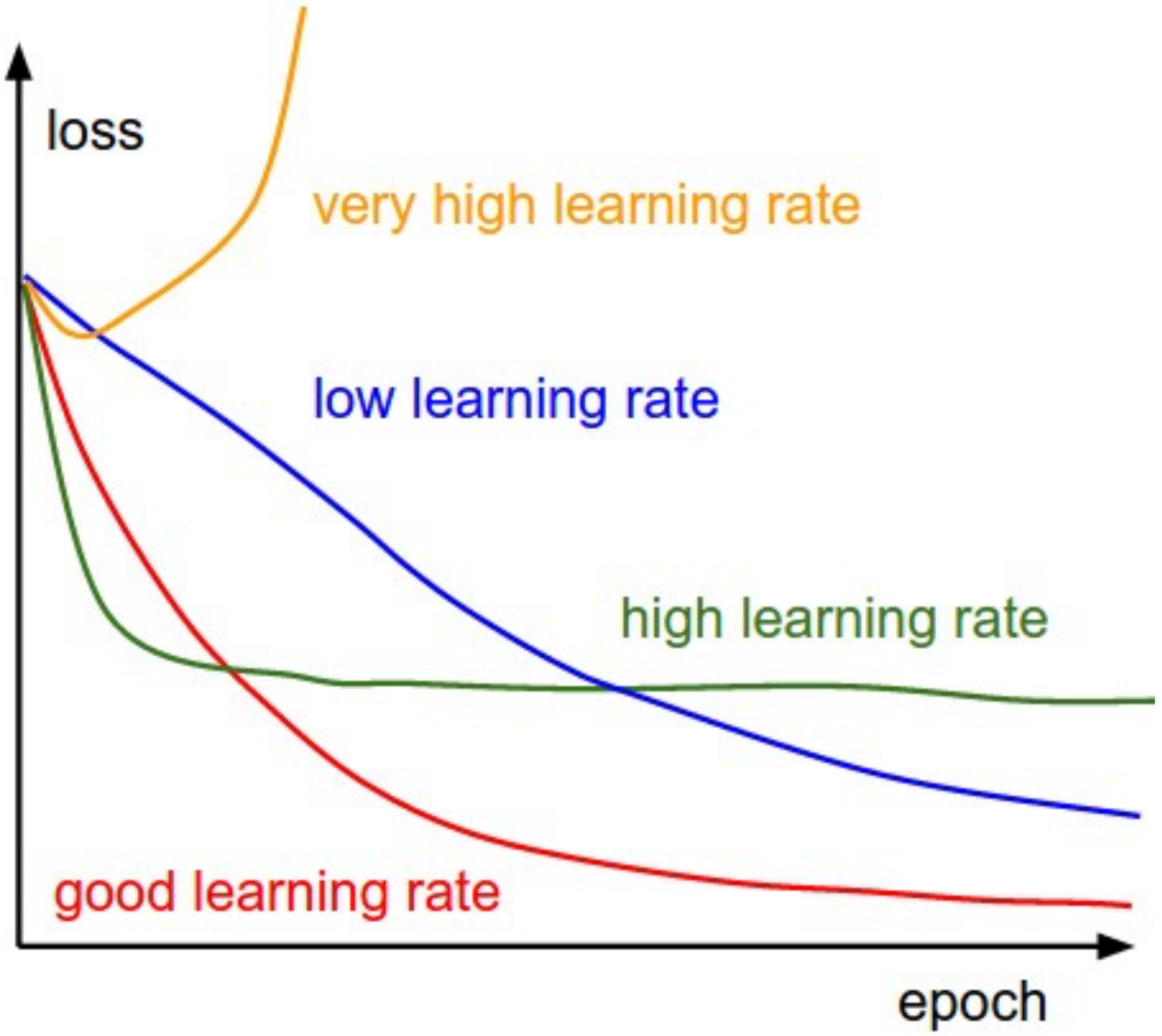
$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta \left(\sum_{i=1}^{B} \ell(y_i, f_\theta(\mathbf{x}_i)) \right)$$

  - There are two key hyperparameters
    - Learning rate η
    - Batch size B
- **Question.** How do we tune the hyperparameters?
  - Usually by trial and error, with validation sets (discussed soon)
  - Guideline. Choose the largest possible B, and then tune the η

Reason: Fast training
RAM constraints + Generalization

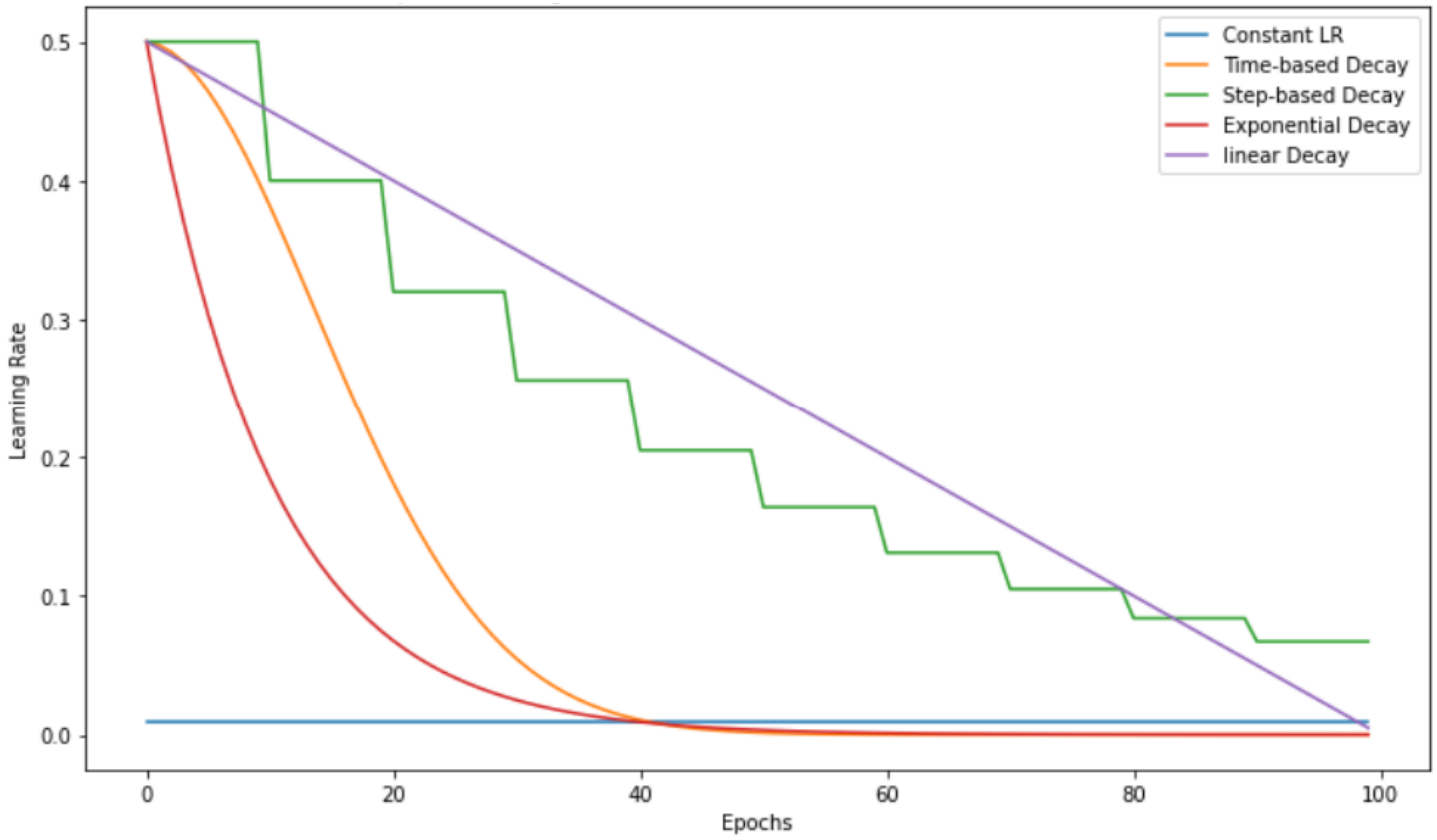
Learning rate vs. Loss

- **High LR.**
  - Faster convergence 👍
  - High final loss 👎
- **Low LR.**
  - Slower convergence 👎
  - Low final loss 👍
- **Question.** Can we get the best of both worlds?



Learning rate scheduling

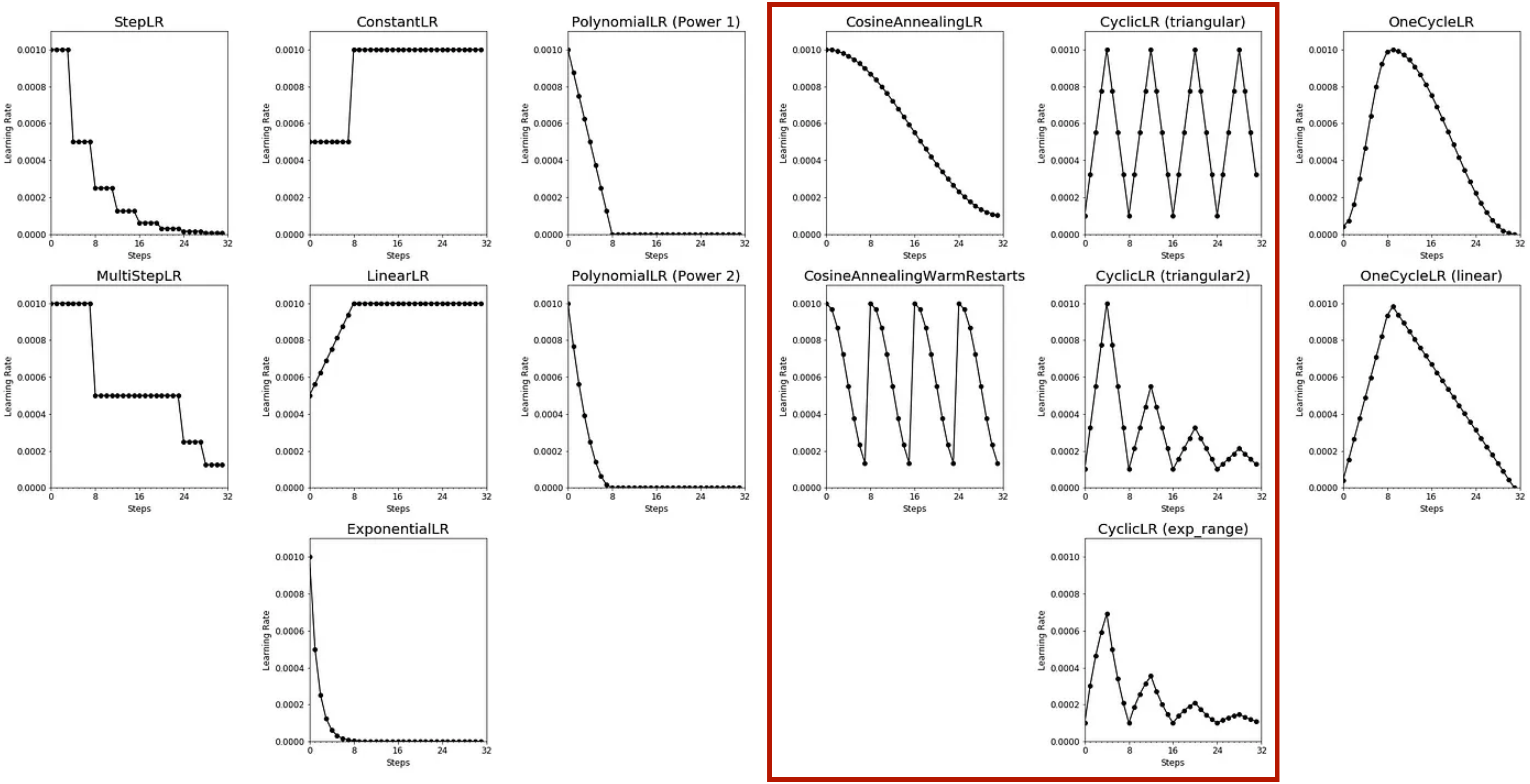
- **Idea.** Decay the learning rate.
  - This requires a careful scheduling of rates
    - Step decay
    - Linear decay
    - Exponential decay
    - ....



- Note. Optimizers have different sensitivities to LR decay
  - e.g., less critical issues in Adam than SGD + Momentum

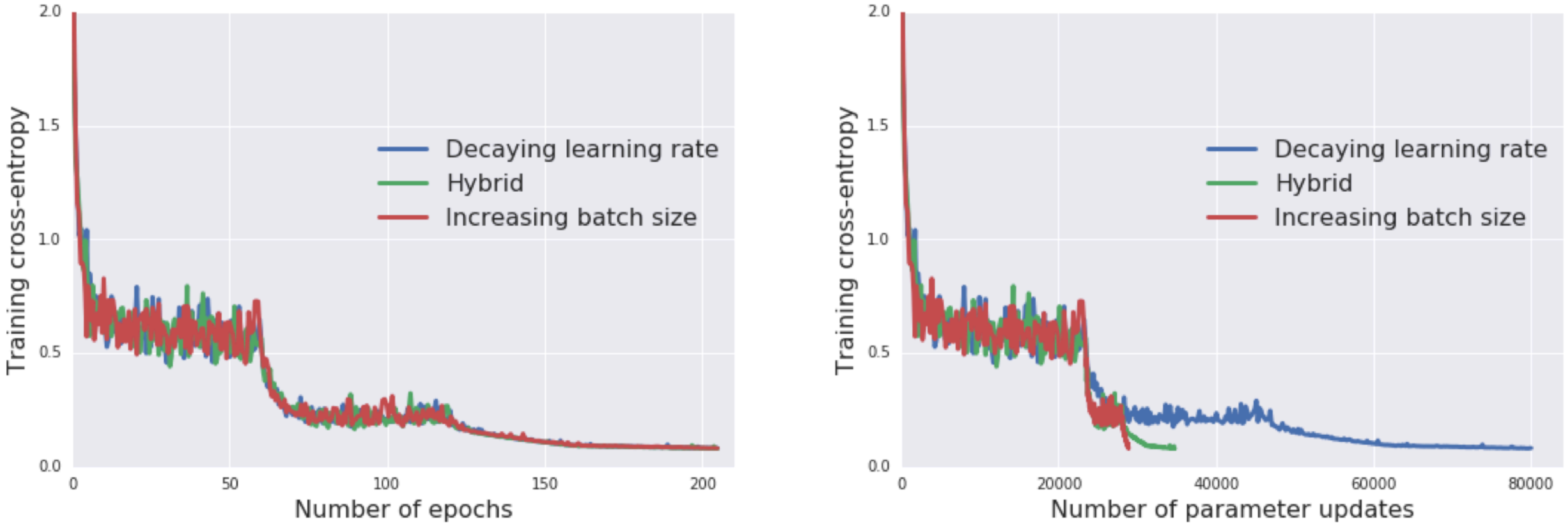
Learning rate scheduling

- **Popular.** Quite common to use cosine annealing / cyclic LR (Optional: Warm restart, Warmup)



Learning rate vs. Batch size

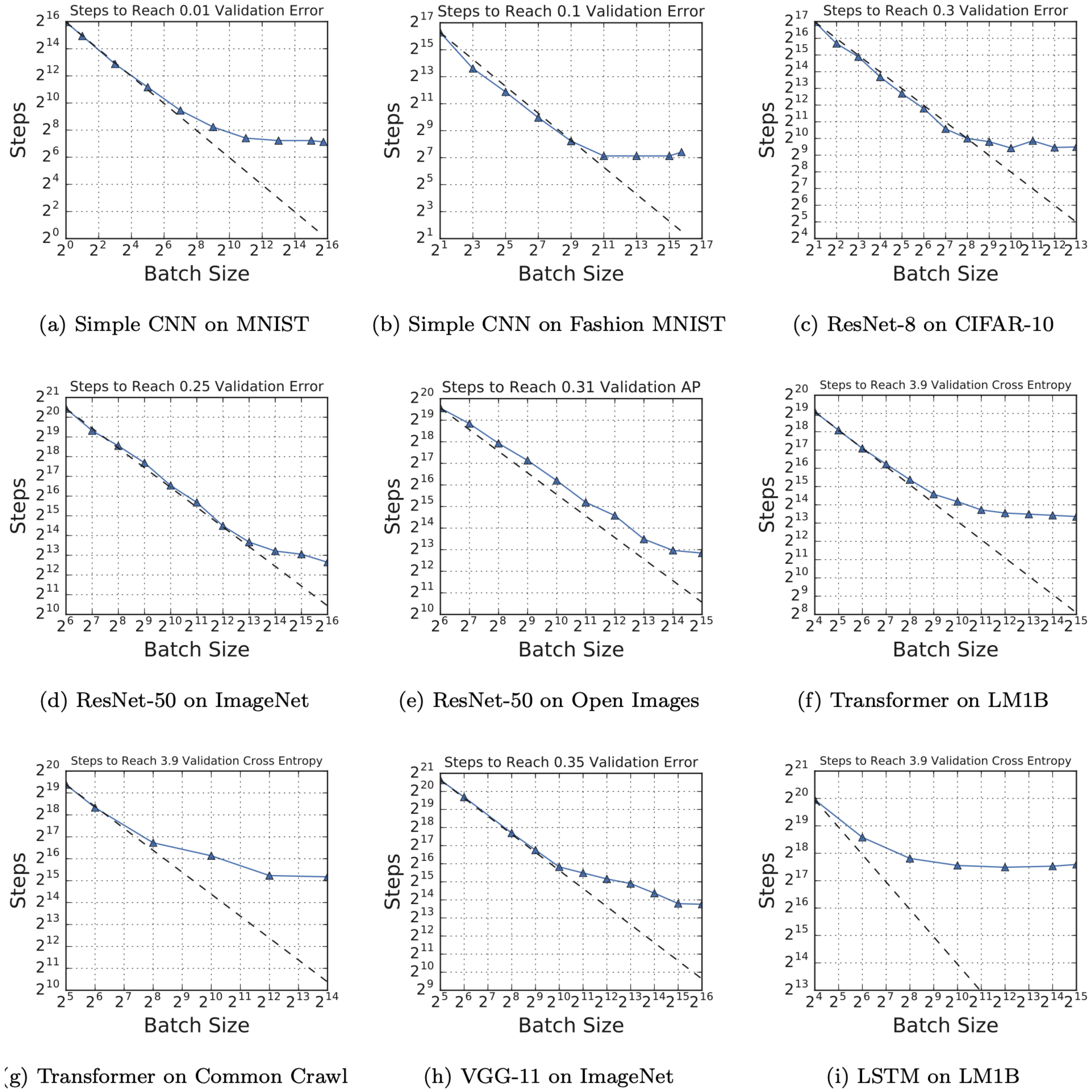
- Empirically, increasing the batch size has a similar effect to decreasing the learning rate



Smith et al., “Don’t decay the learning rate, increase the batch size,” ICLR 2018

Learning rate vs. Batch size

- With a **larger batch size**, we expect
  - Reduced #SGD steps needed to achieve the similar test performance
    - Optimal LR scales linearly with batch size
    - Eventually the benefit saturates



(a) Simple CNN on MNIST (b) Simple CNN on Fashion MNIST (c) ResNet-8 on CIFAR-10

(d) ResNet-50 on ImageNet (e) ResNet-50 on Open Images (f) Transformer on LM1B

(g) Transformer on Common Crawl (h) VGG-11 on ImageNet (i) LSTM on LM1B

Shallue et al., "Measuring the effects of data parallelism of neural network training," JMLR 2019

Optimizers

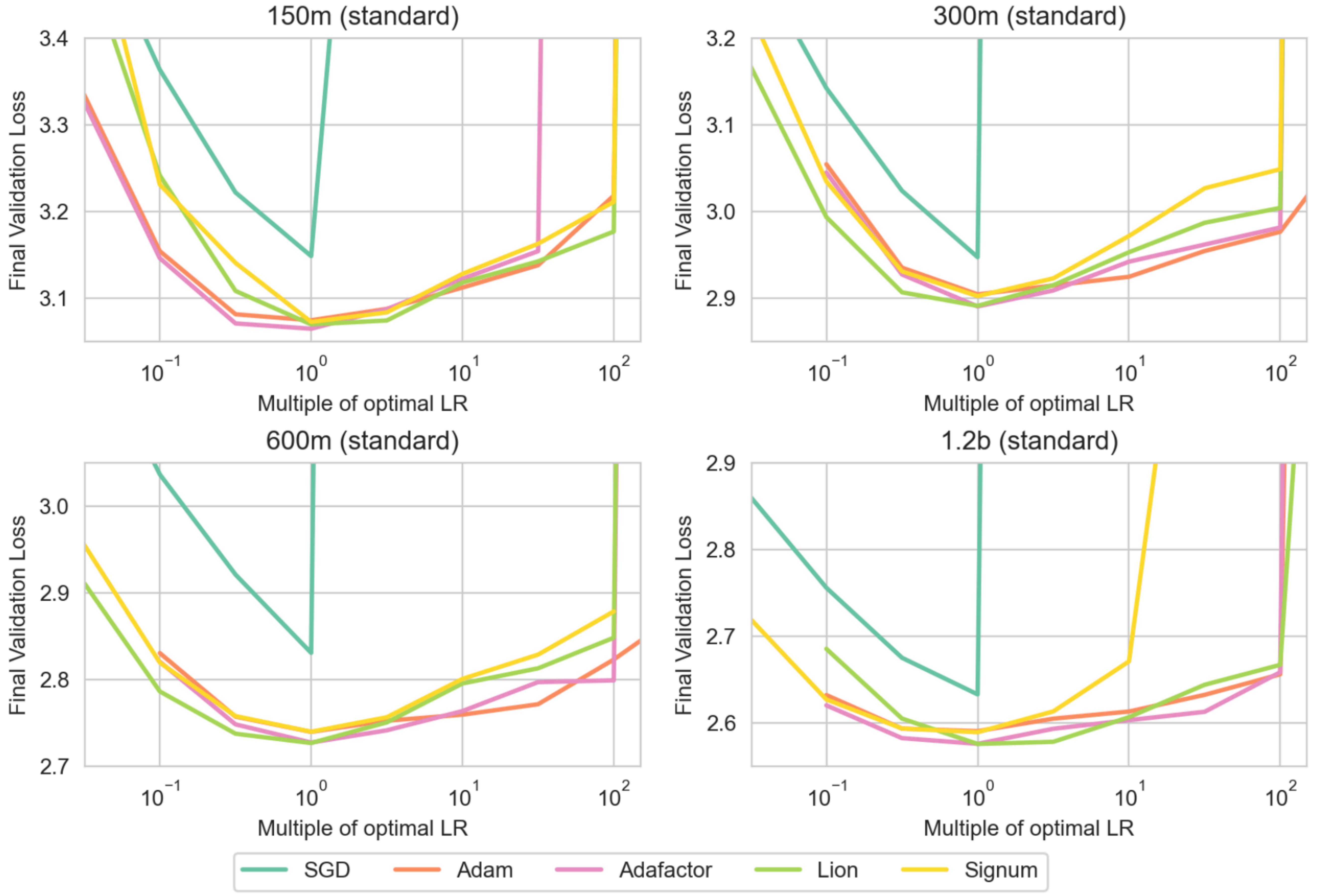
Zhao et al., "Deconstructing What Makes a Good Optimizer for Language Models," arXiv 2024

Optimizers

- We rarely use the "vanilla" version of the SGD

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta L(\theta^{(t)})$$

- There are many alternatives:
  - **PyTorch native.** AdaDelta, AdaFactor, AdaGrad, Adam AdamW, SparseAdam, AdaMax, ASGD, LBFGS, NAdam, RAdam, RMSProp, RProp, ...
  - **More recent.** Shampoo, Lion, Signum, ...
- **Now.** Understand key concepts



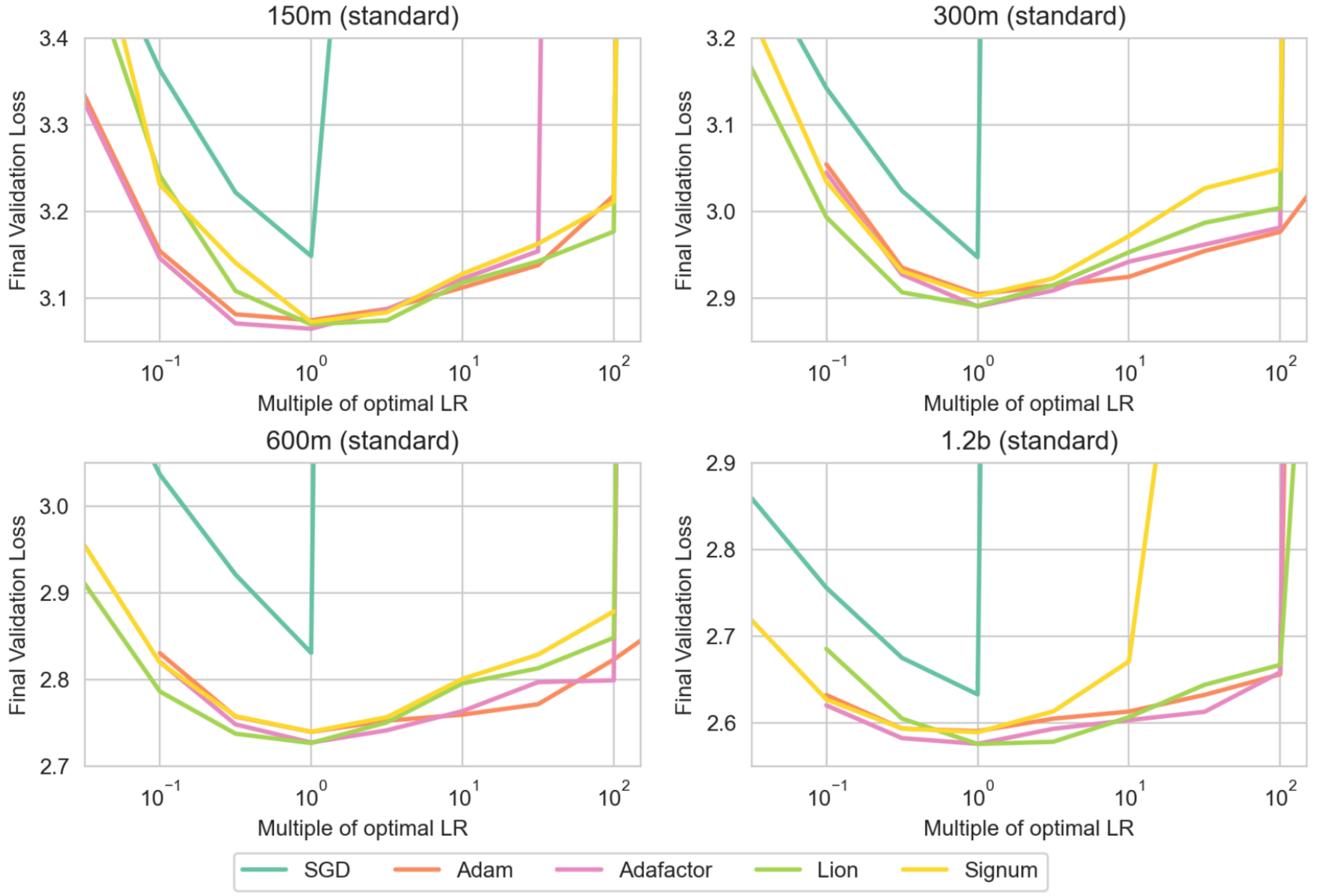
Zhao et al., "Deconstructing What Makes a Good Optimizer for Language Models," arXiv 2024

Optimizers

- We rarely use the "vanilla" version of the SGD

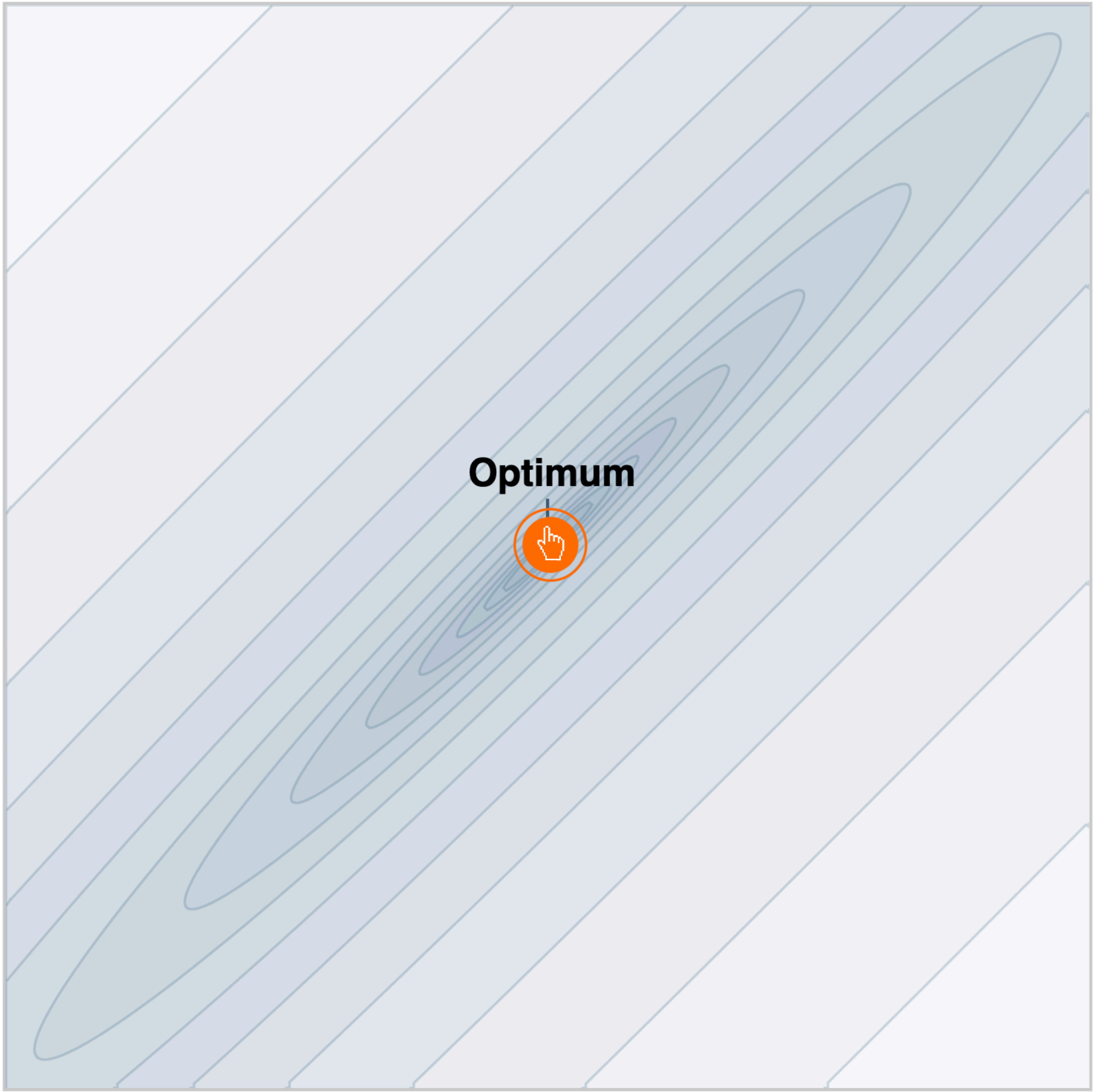
$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta L(\theta^{(t)})$$

- There are many alternatives:
  - **PyTorch native.** AdaDelta, AdaFactor, AdaGrad, Adam AdamW, SparseAdam, AdaMax, ASGD, LBFGS, NAdam, RAdam, RMSProp, RProp, ...
  - **More recent.** Shampoo, Lion, Signum, ...
- **Now.** Understand a key concept; momentum



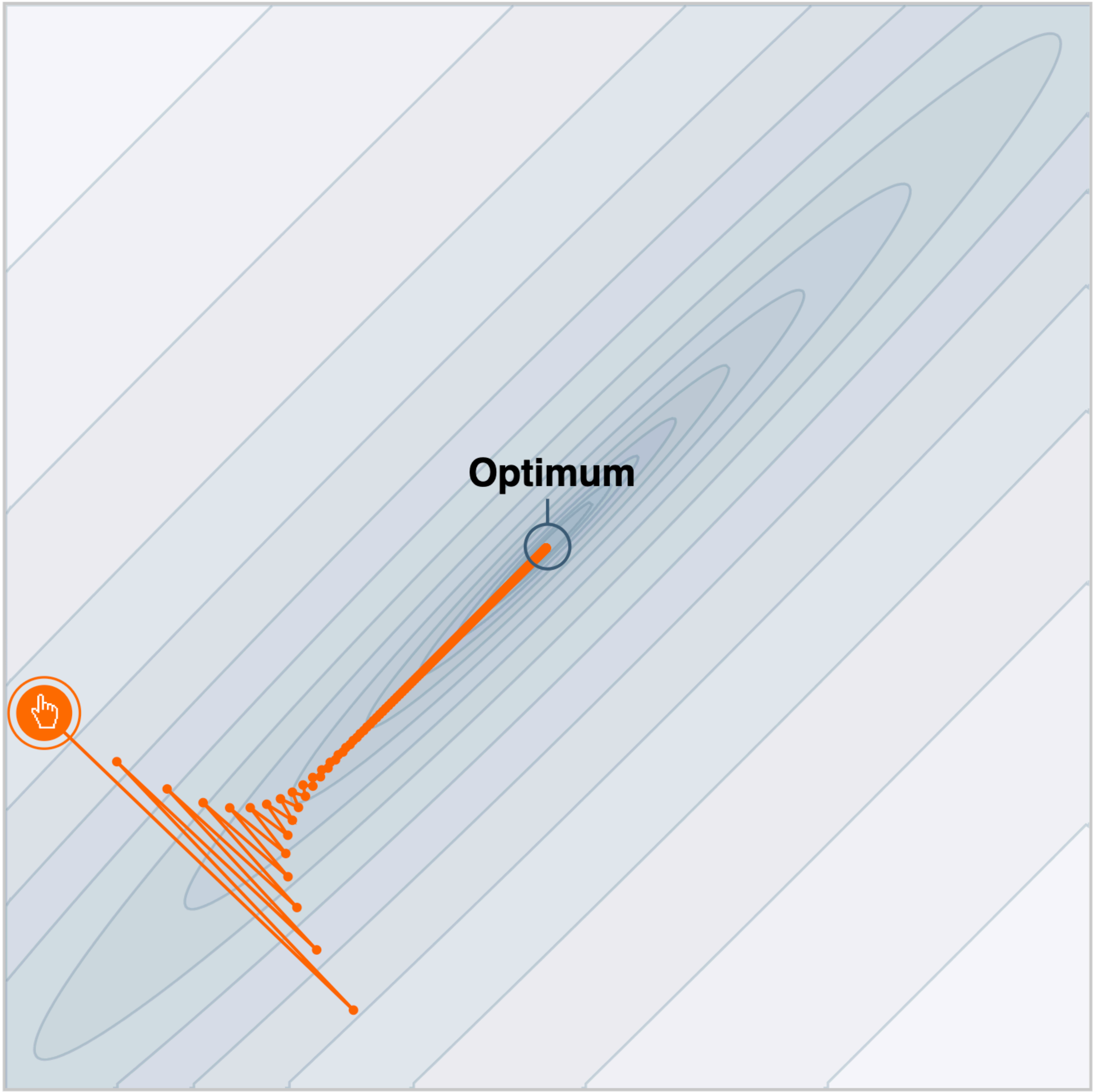
Momentum

- **Motivation.** Suppose that the risk
  - Changes fast in one direction
  - Changes slowly in another direction
- **Question.** What would happen?



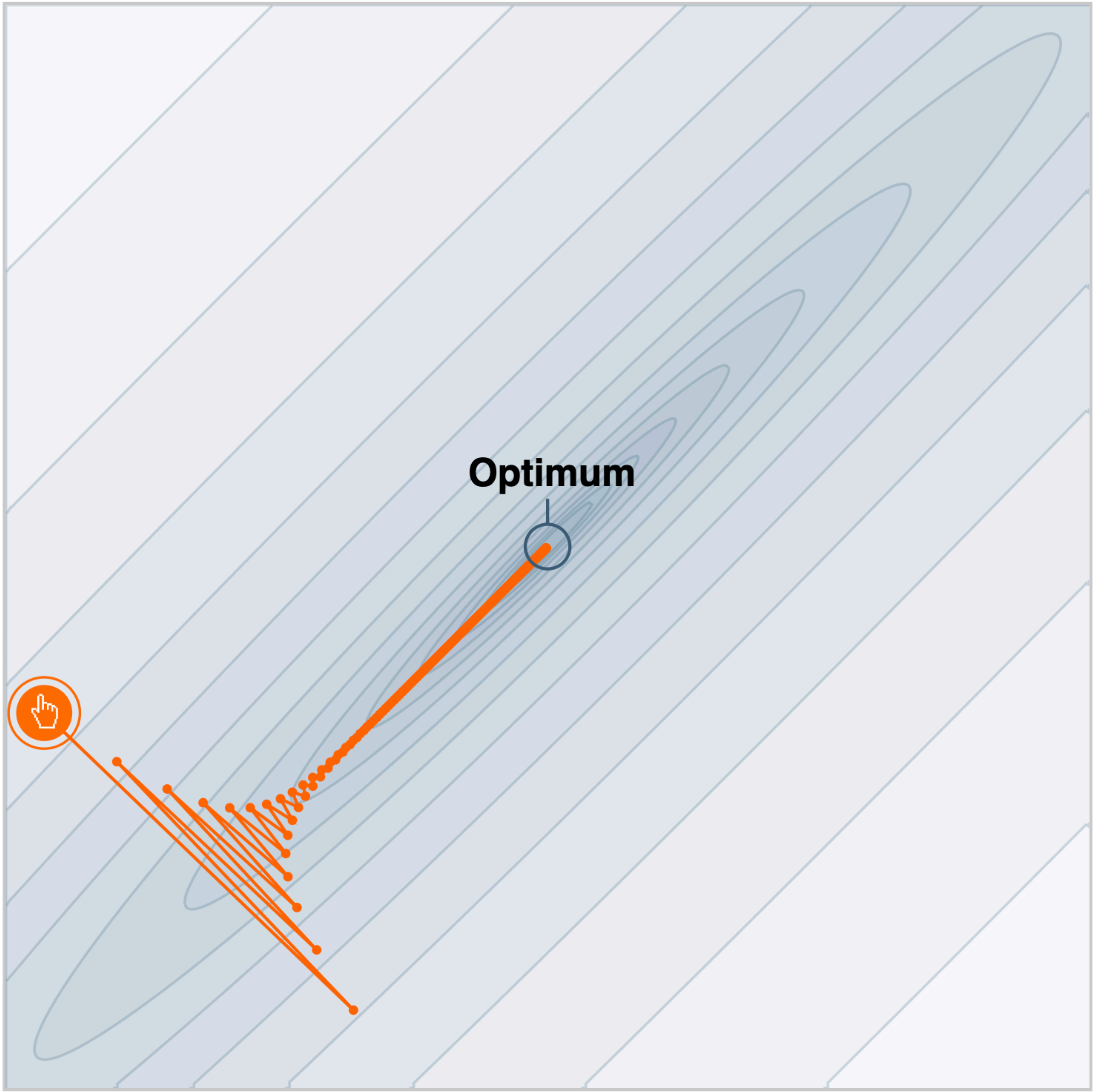
Momentum

- **Motivation.** Suppose that the risk
  - Changes fast in one direction
  - Changes slowly in another direction
- **Question.** What would happen?
- **Observation.** GD evolves as...
  - Slow progress in shallow direction
  - High jitter in steep direction
- Note. The loss has a large "condition number"



Momentum

- **Idea.** Let our GD have an inertia
  - If we were moving to one direction consistently, move more faster in that direction



Momentum

- **Idea.** Let our GD have an inertia
  - If we were moving to one direction consistently, move more faster in that direction

- Original GD

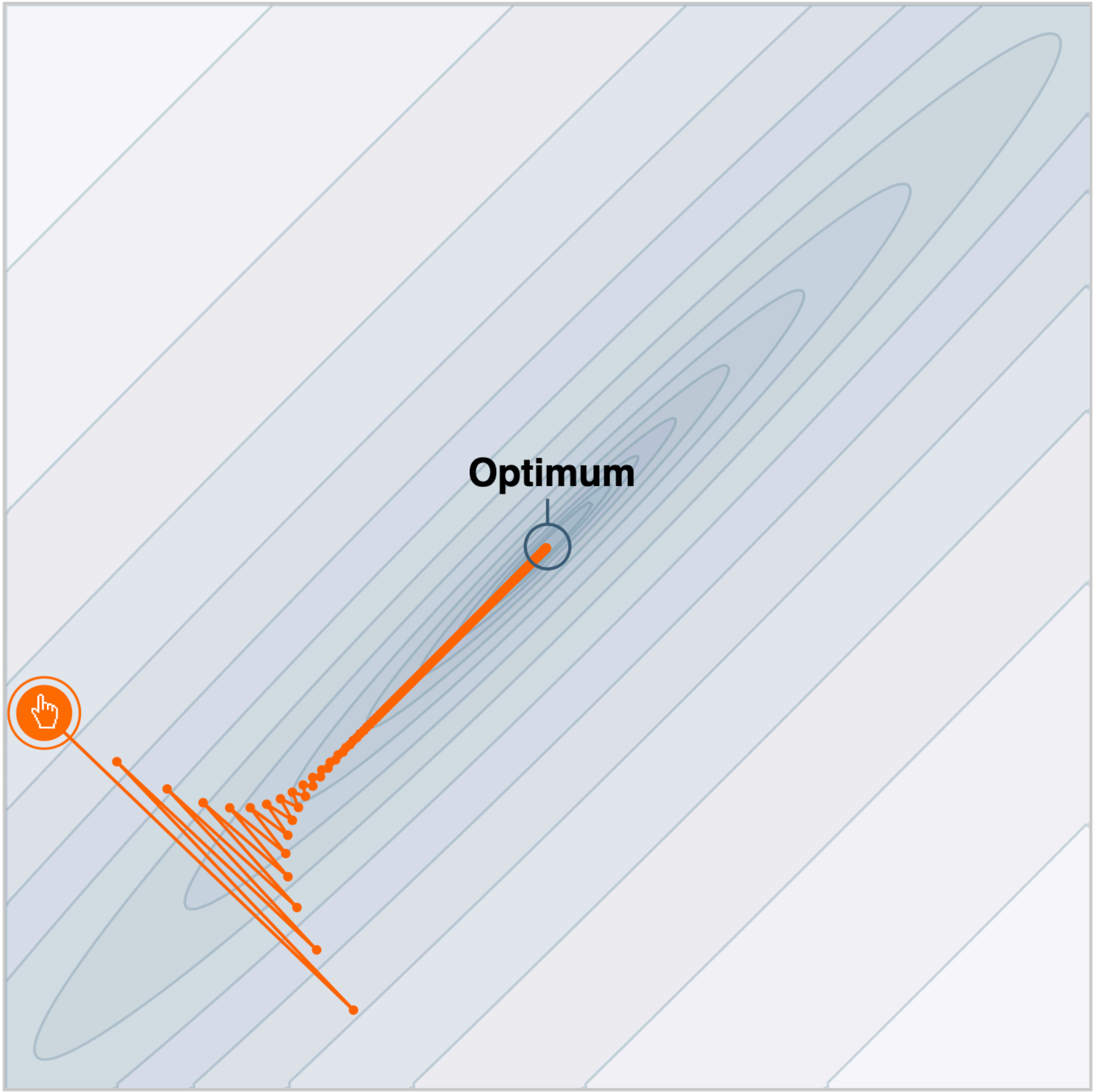
$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta L(\theta^{(t)})$$

- GD + Momentum

Accumulated gradients

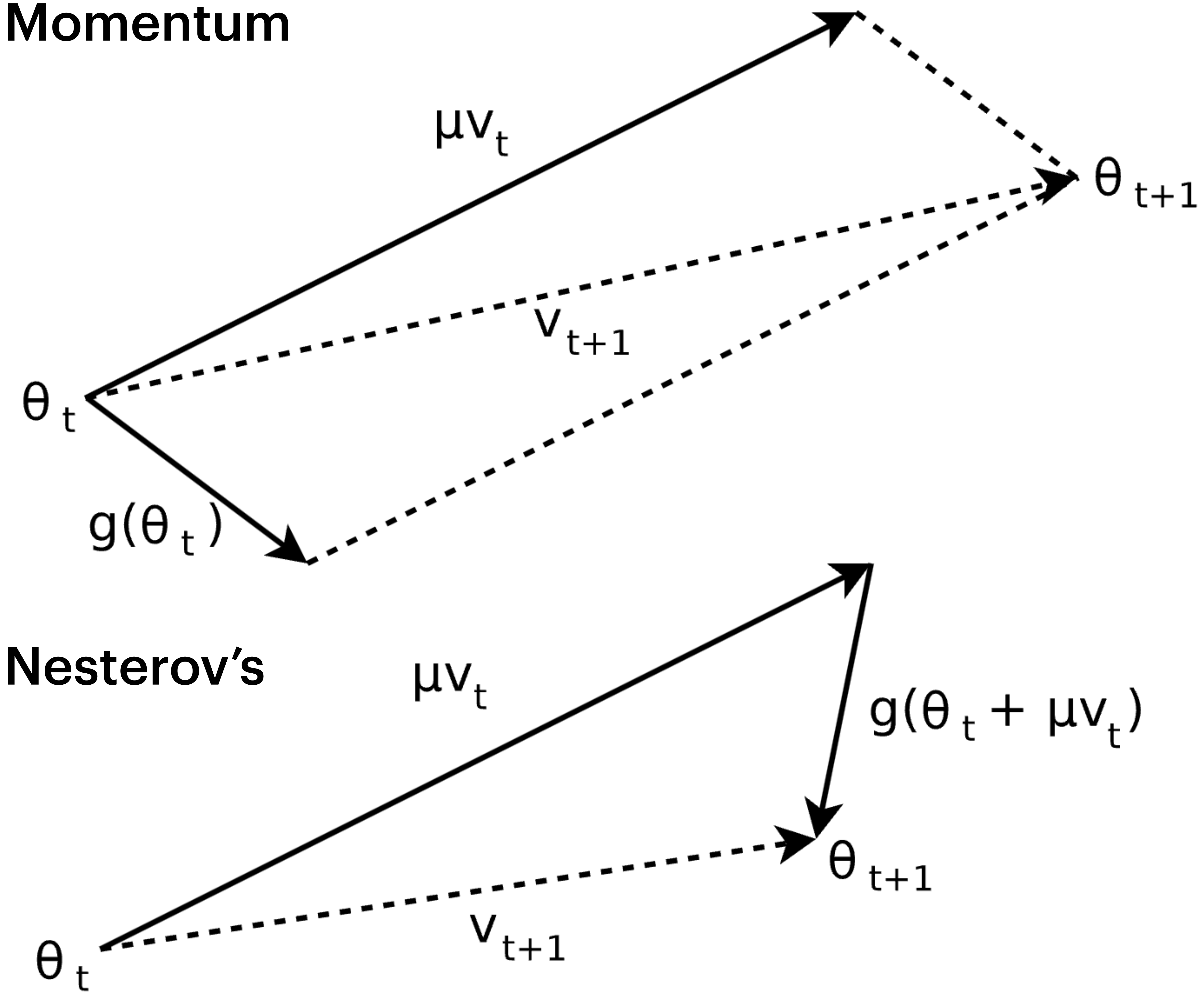
$$v^{(t+1)} = \beta \cdot v^{(t)} + \nabla_\theta L(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot v^{(t+1)}$$



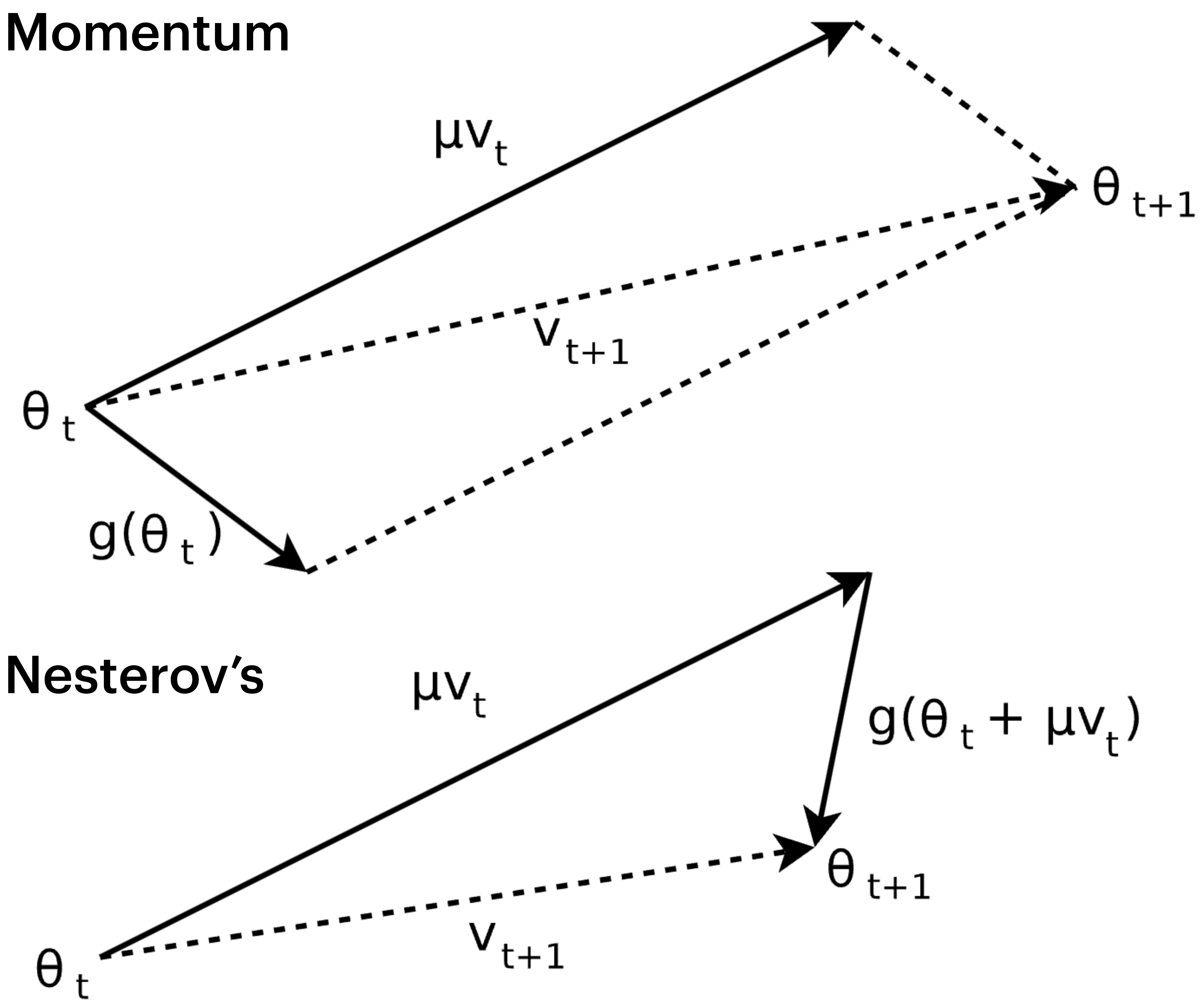
Nesterov's momentum

- A small fix to the momentum
- **Idea.** Evaluate the gradients at the parameter + momentum, not the current parameter
  - <u>Interpretation</u>. Looking ahead any hardship that will come next



Nesterov's momentum

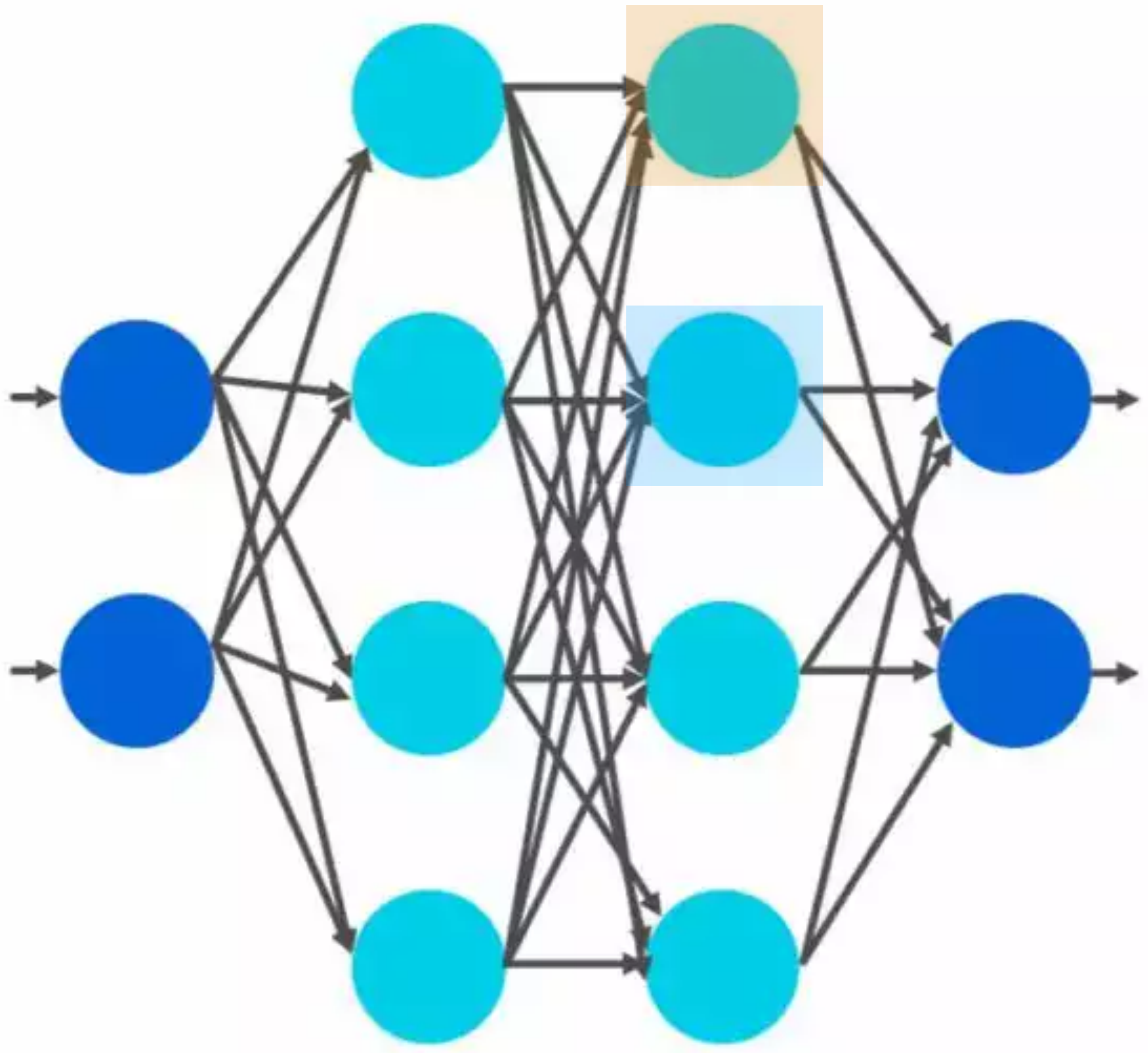
- A small fix to the momentum
- **Idea.** Evaluate the gradients at the parameter + momentum, not the current parameter
  - Interpretation. Looking ahead any hardship that will come next
- **Empirically.** Neat idea, but does not always guarantee a better convergence rate 😅
  - Thus, try both



Adaptive learning rate

- **Motivation.** Single learning rate may not work well for all parameters.
  - Example. Suppose that we have a "cat neuron" and a "dog neuron."

    If we see much less "dogs" than "cats,"
    then maybe using higher LR for "dogs" will help run faster.



Adaptive learning rate

- **Motivation.** Single learning rate may not work well for all parameters.
    - Example. Suppose that we have a "cat neuron" and a "dog neuron."

      If we see much less "dogs" than "cats,"
      then maybe using higher LR for "dogs" will help run faster.

- **RMSProp.** Keep the moving average of gradient^2, and divide the LR by it (do this elementwise)

$$g^{(t+1)} = \gamma \cdot g^{(t)} + (1-\gamma) \cdot \left(\nabla_\theta L(\theta^{(t)})\right)^2$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{g^{(t+1)}} + \boxed{\varepsilon}} \cdot \nabla_\theta L(\theta^{(t)})$$

tiny value, for avoiding division by zero

Adaptive learning rate

- **Motivation.** Single learning rate may not work well for all parameters.
  - Example. Suppose that we have a "cat neuron" and a "dog neuron."

    If we see much less "dogs" than "cats,"
    then maybe using higher LR for "dogs" will help run faster.
- **RMSProp.** Keep the moving average of gradient^2, and divide the LR by it.

$$g^{(t+1)} = \gamma \cdot g^{(t)} + (1-\gamma) \cdot \left(\nabla_\theta L(\theta^{(t)})\right)^2$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\eta}{\sqrt{g^{(t+1)} + \varepsilon}} \cdot \nabla_\theta L(\theta^{(t)})$$

- **Adam.** RMSProp + Momentum (most cited paper in last 10 years)

Remarks

- **Memory.** Optimizer states should also be stored on memory!
  - For each parameter (32bits), we keep …
    - Gradient (32bits)
    - Momentum (32bits)
    - Adaptive LR (32bits)
- **Tuning.** Advanced optimizers introduce additional hyperparameters to tune
  - Much training computation needed for the best performance

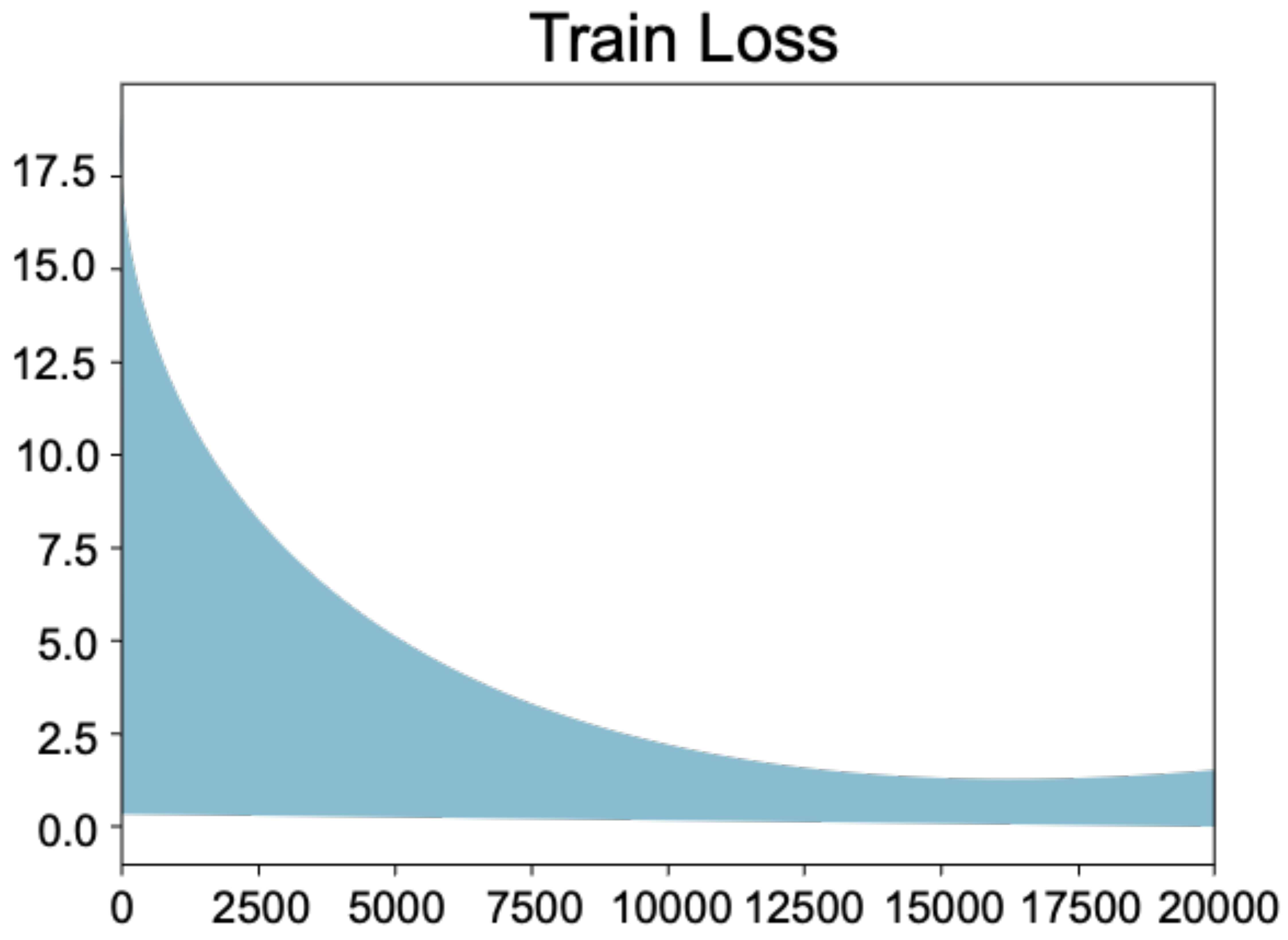
References

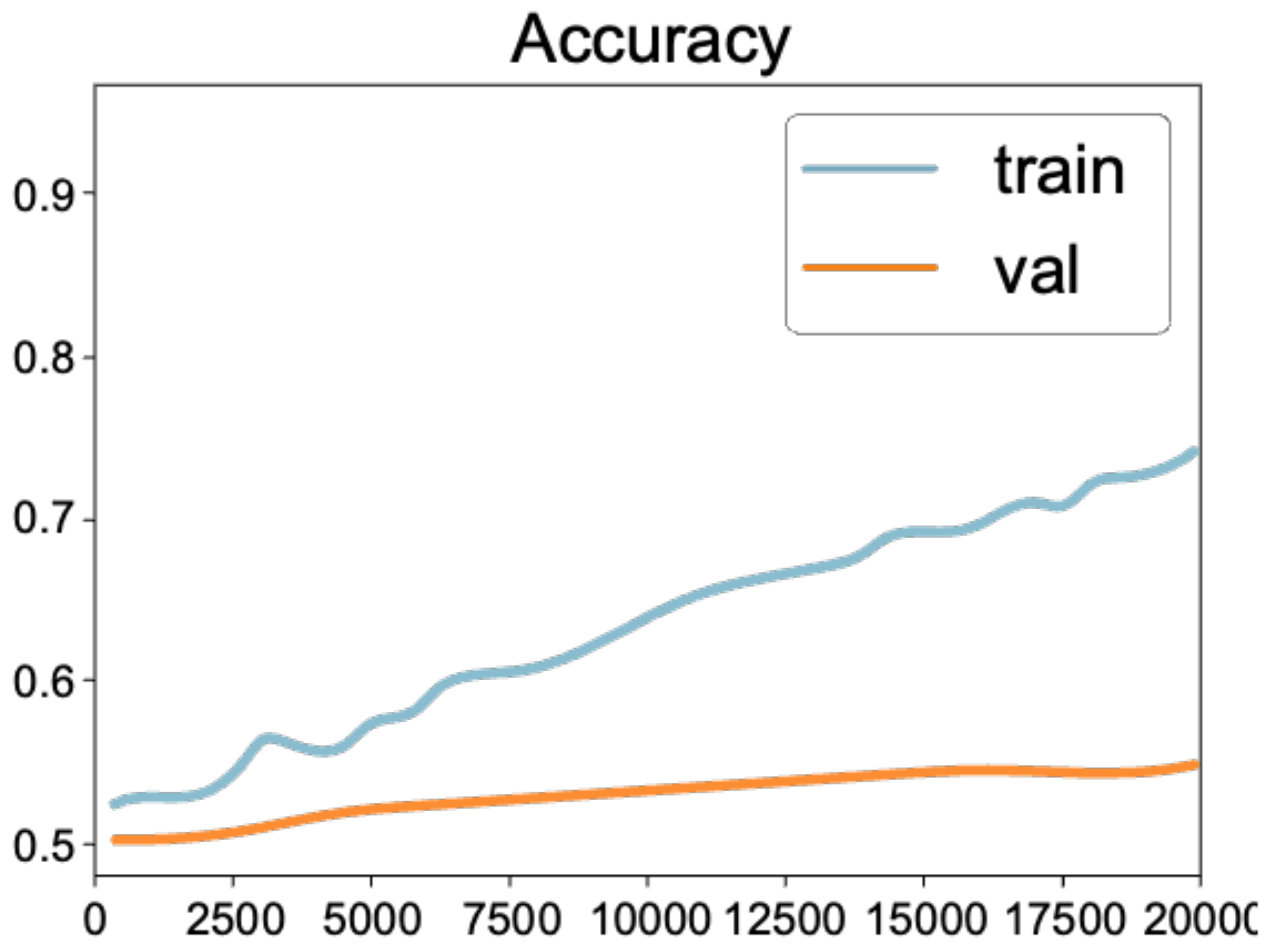
- **Momentum.** https://distill.pub/2017/momentum/
- **Adam.** https://optimization.cbe.cornell.edu/index.php?title=Adam
- **Others.** https://cs231n.github.io/neural-networks-3/

Regularization

Beyond training error

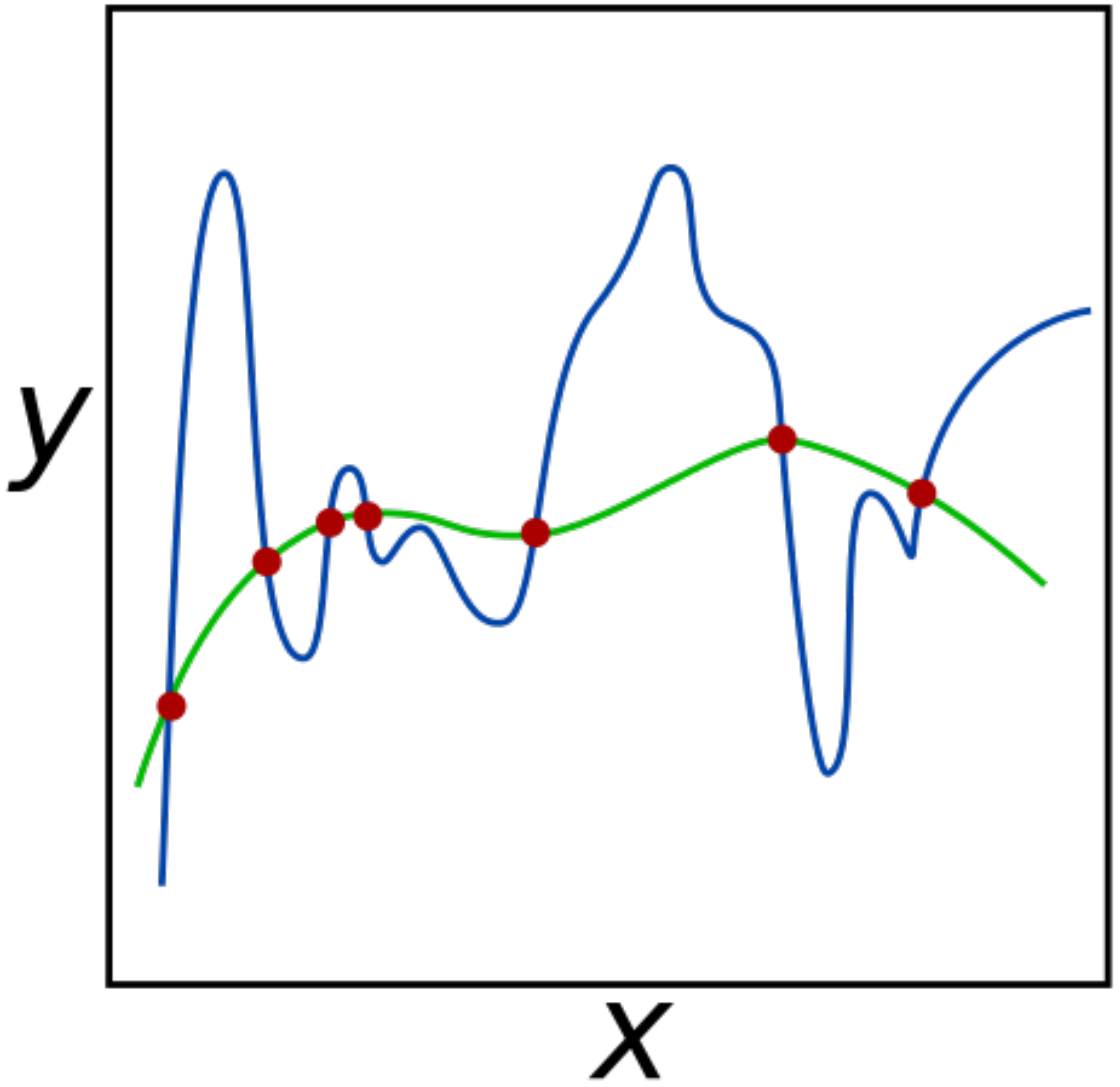
- Better optimization algorithms help reduce the training loss
  - But we actually care about the test performance — how can we reduce the gap?





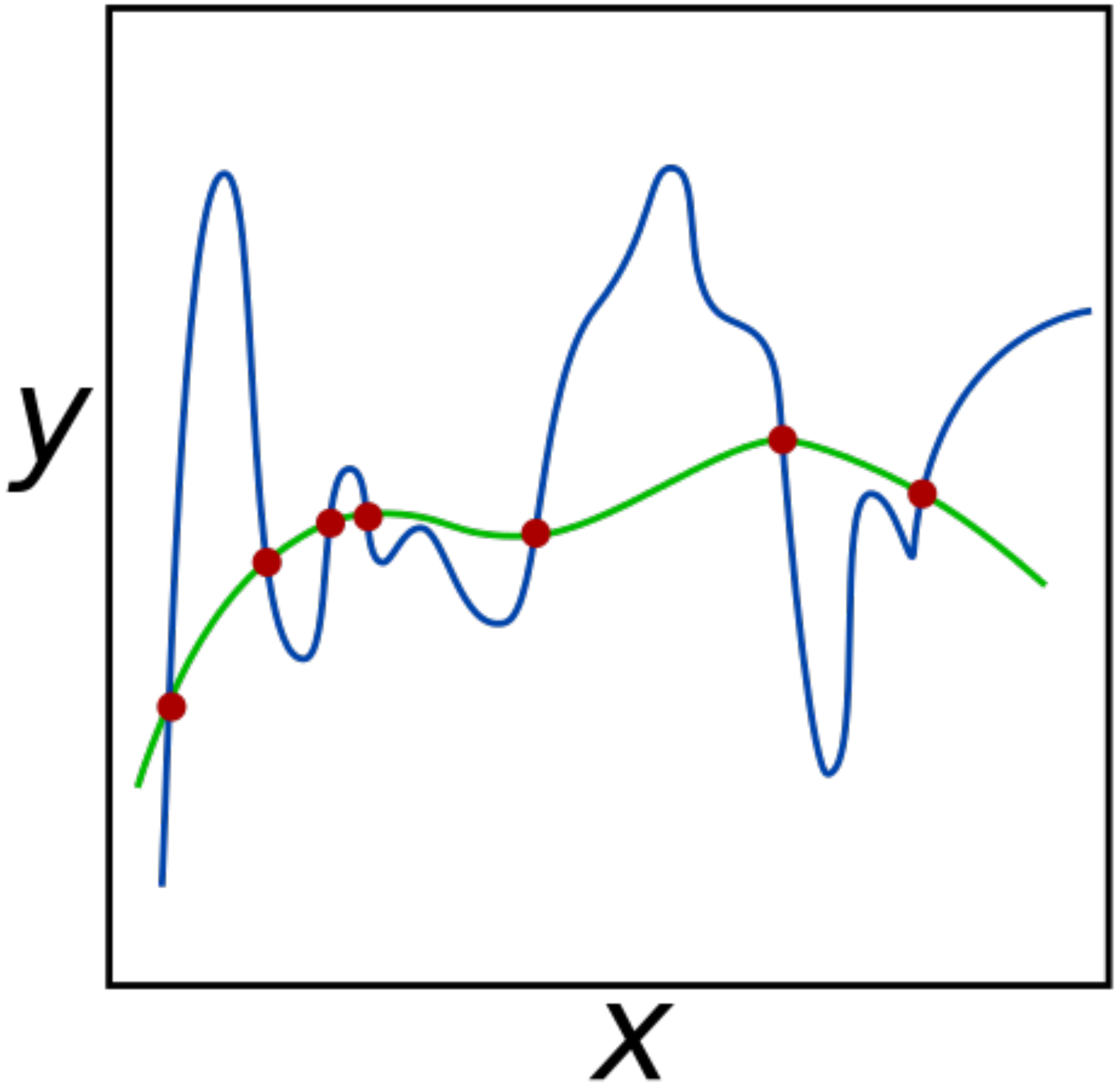
Beyond training error

- Better optimization algorithms help reduce the training loss
  - But we actually care about the test performance — how can we reduce the gap?
- **Core philosophy.** Most regularization methods follow the principle of Occam's razor
  - "Whenever possible, use simpler models"



Beyond training error

- Better optimization algorithms help reduce the training loss
  - But we actually care about the test performance — how can we reduce the gap?
- **Core philosophy.** Most regularization methods follow the principle of Occam's razor
  - "Whenever possible, use simpler models"
  - Simplicity. Many definitions, including
    - Number of weight parameters
    - Norm of the weight parameters
    - Prediction confidence …



Regularization (through loss)

- **Idea.** Add complexity to the loss function; doable for differentiable complexity measures

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \quad + \quad \text{complexity}(\theta)$$

Regularization (through loss)

- **Idea.** Add complexity to the loss function; doable for differentiable complexity measures

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i)) \quad + \quad \mathrm{complexity}(\theta)$$

- **Example.** L2 regularization; use smaller ℓ_2 norm solution, whenever possible.

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta (L(\theta) + \lambda \cdot \|\theta\|_2)$$

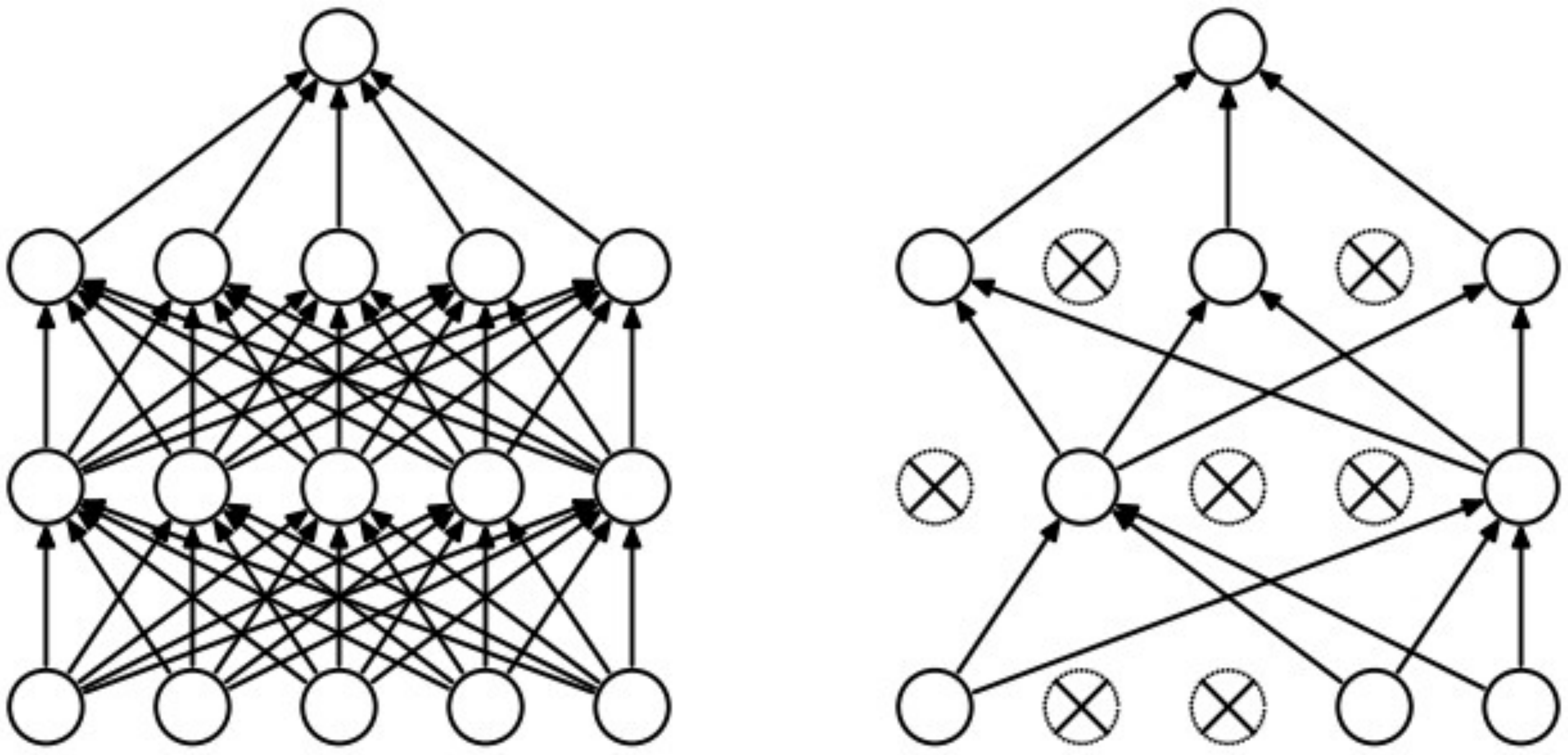
  - This is equivalent to a simpler-to-implement form:

$$\theta^{(t+1)} = (1 - \eta\lambda)\theta^{(t)} - \eta \cdot \nabla_\theta L(\theta)$$

(thus often called "weight decay")

Nondifferentiable complexities

- For non-differentiable complexities, design a customized algorithm.
- **Example.** Whenever possible, use a smaller number of parameters
    - Dropout. During the training, randomly remove each neuron, w.p. p
        - For the inference, rescale the weights back to $1/p$.



(a) Standard Neural Net (b) After applying dropout.

Nondifferentiable complexities

- For non-differentiable complexities, design a customized algorithm.
- **Example.** Whenever possible, use a smaller number of parameters
  - Dropout. During the training, randomly remove each neuron, w.p. p
    - For the inference, rescale the weights back to $1/p$.
- **Example.** Whenever possible, use a parameter that can be discovered within a shorter time
  - Early stopping. Pause training when validation error does not drop anymore.

Remarks

- **Optimization.** Sometimes, regularization make the optimization easier.
  - Example. Consider solving a least-square problem $\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$
    - Ordinary solution:

$$\mathbf{w} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}, \qquad \text{(no guarantee that } \mathbf{X}^\top\mathbf{X} \text{ is invertible)}$$

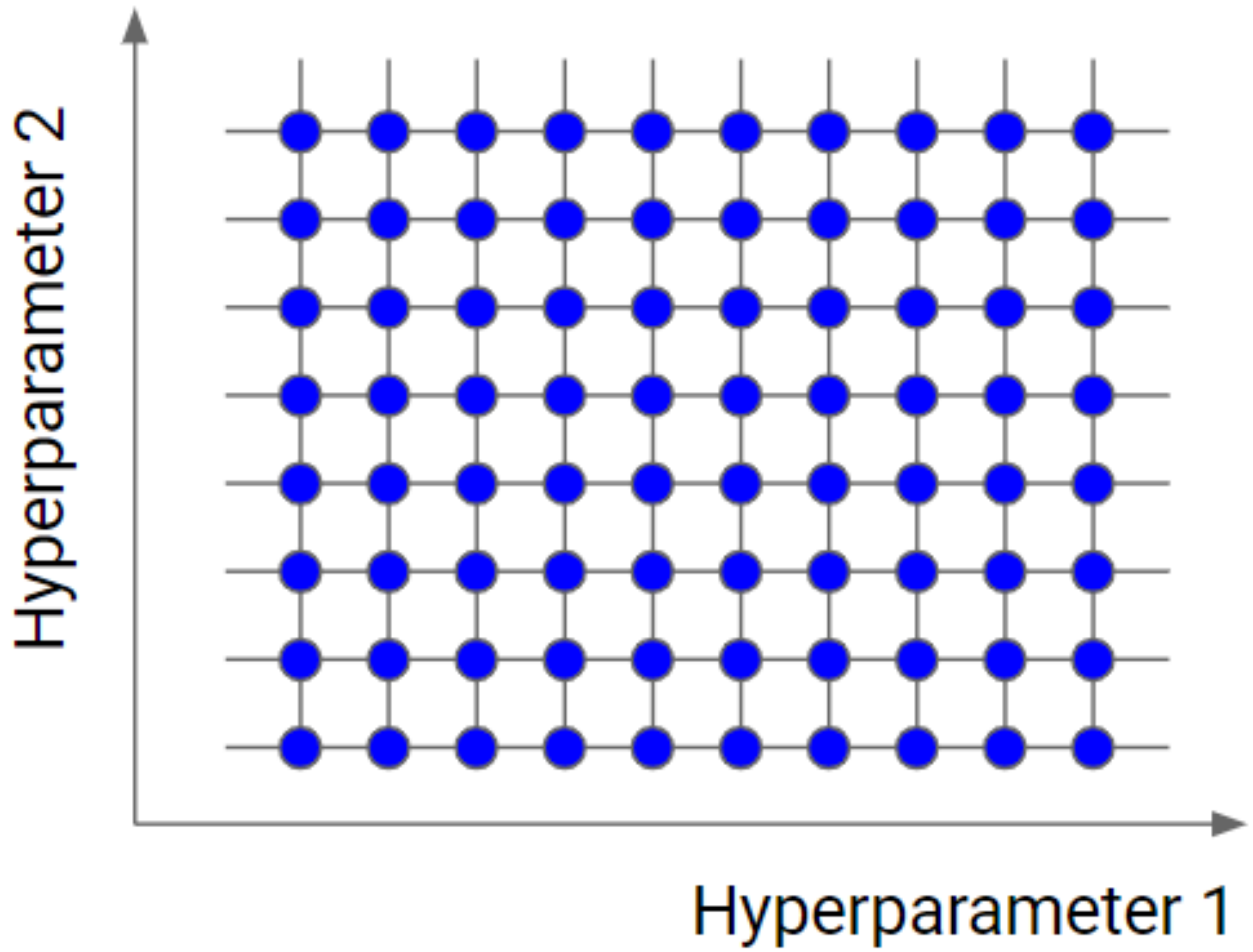
    - Added ℓ_2 penalty: $\min_{\mathbf{w}} \left(\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\|\mathbf{w}\|^2\right)$

$$\mathbf{w} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^\top\mathbf{y} \qquad \text{(invertible, if } \lambda \text{ is nonzero)}$$

Hyperparameter tuning

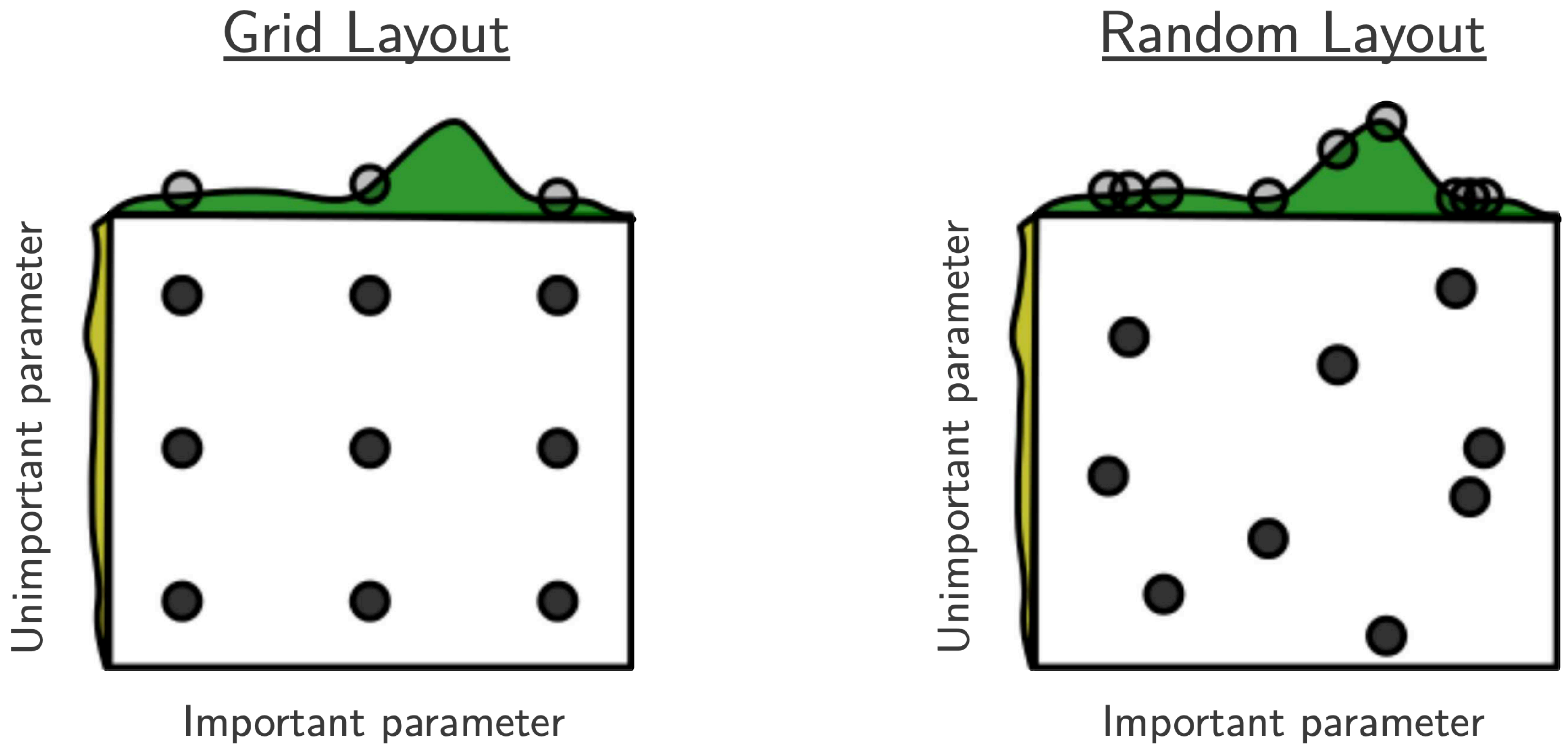
Strategy

- **Question.** How do we select the hyperparameter?
  - Grid search. Use coarse-to-fine grids, to reduce #trials.
    - Sometimes, use log-scales
      (e.g., search LR from $\{10^{-2}, 10^{-3}, 10^{-4}, \cdots\}$)



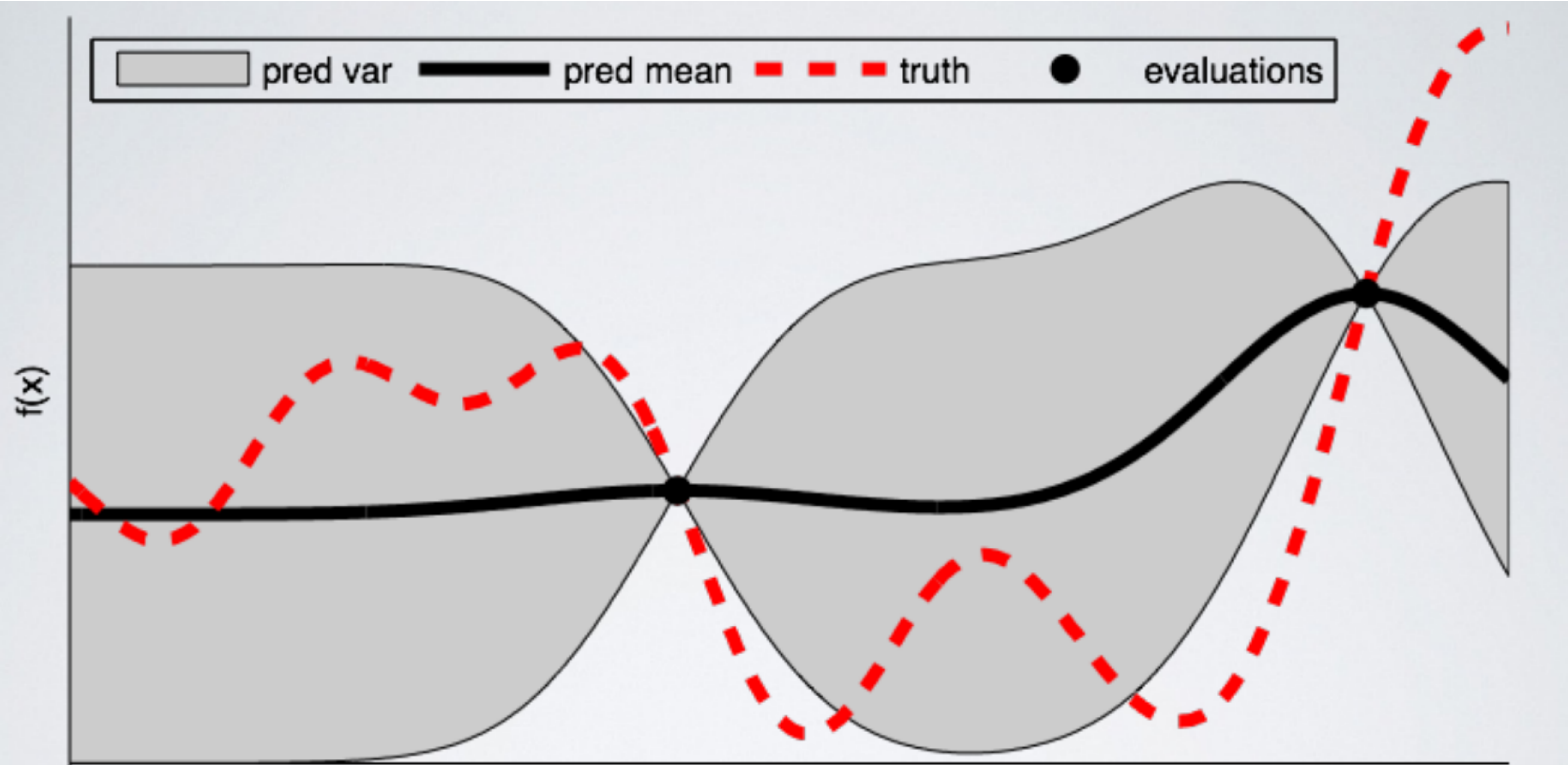
Strategy

- **Question.** How do we select the hyperparameter?
  - <u>Grid search</u>. Use coarse-to-fine grids, to reduce #trials.
    - Sometimes, use log-scales
      (e.g., search LR from $\{10^{-2}, 10^{-3}, 10^{-4}, \cdots\}$)
  - <u>Random search</u>. Use randomly sampled HPs
    - Has a larger "effective sample size"



Strategy

- **Sophisticated.** In some cases, we use Bayesian HP optimization techniques...
  - Idea. The performance-HP relationship may be a smooth function as well.
    - Predict the performance with Gaussian processes



Strategy

- Even more sophisticated. For LLMs, we transfer the hyperparameters.
  - Tune HPs on a small model, and use them on larger models
    - Requires a special parameterization (called μ-parameterization)

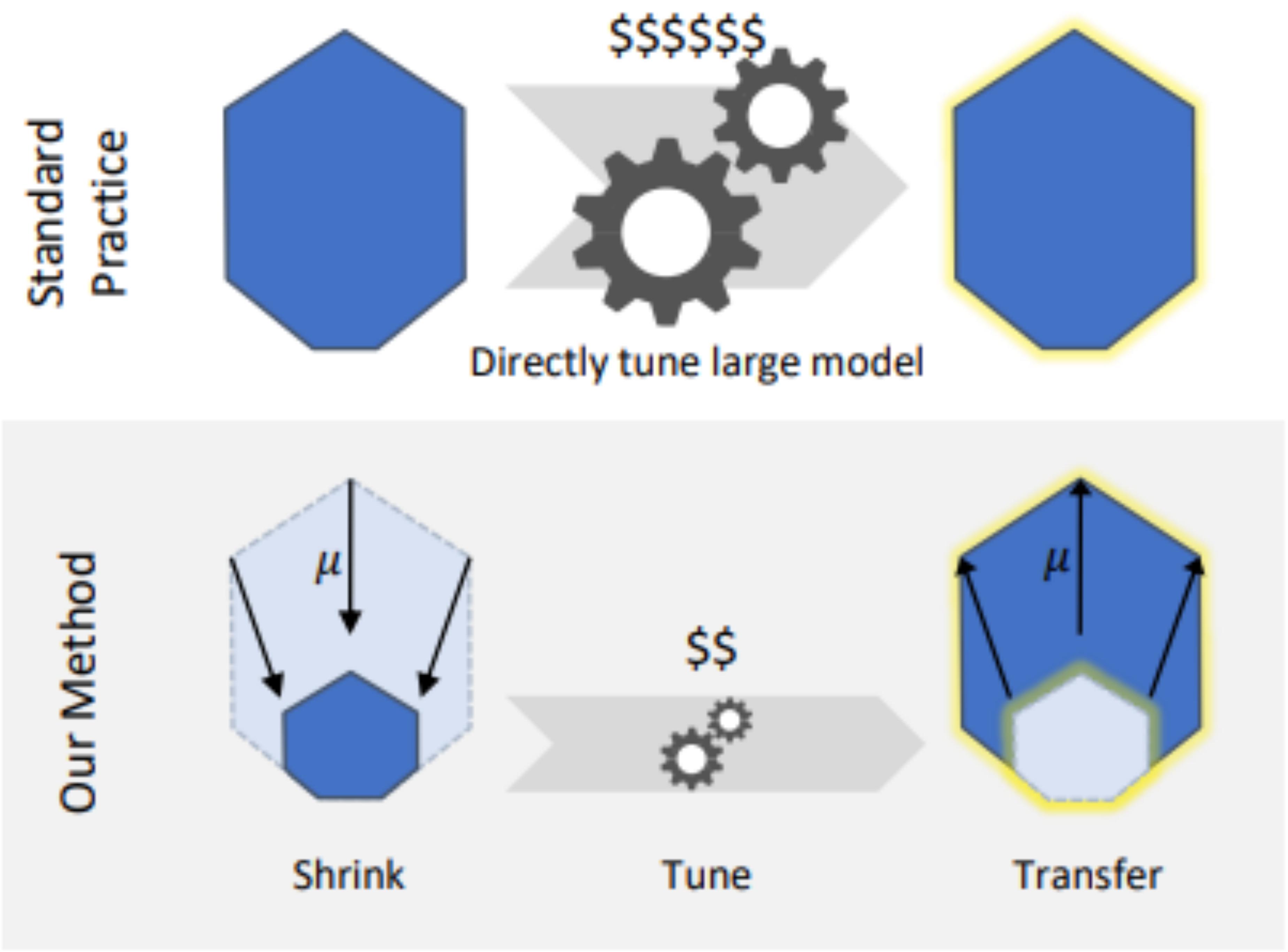


Figure 2: Illustration of μTransfer

Cheers