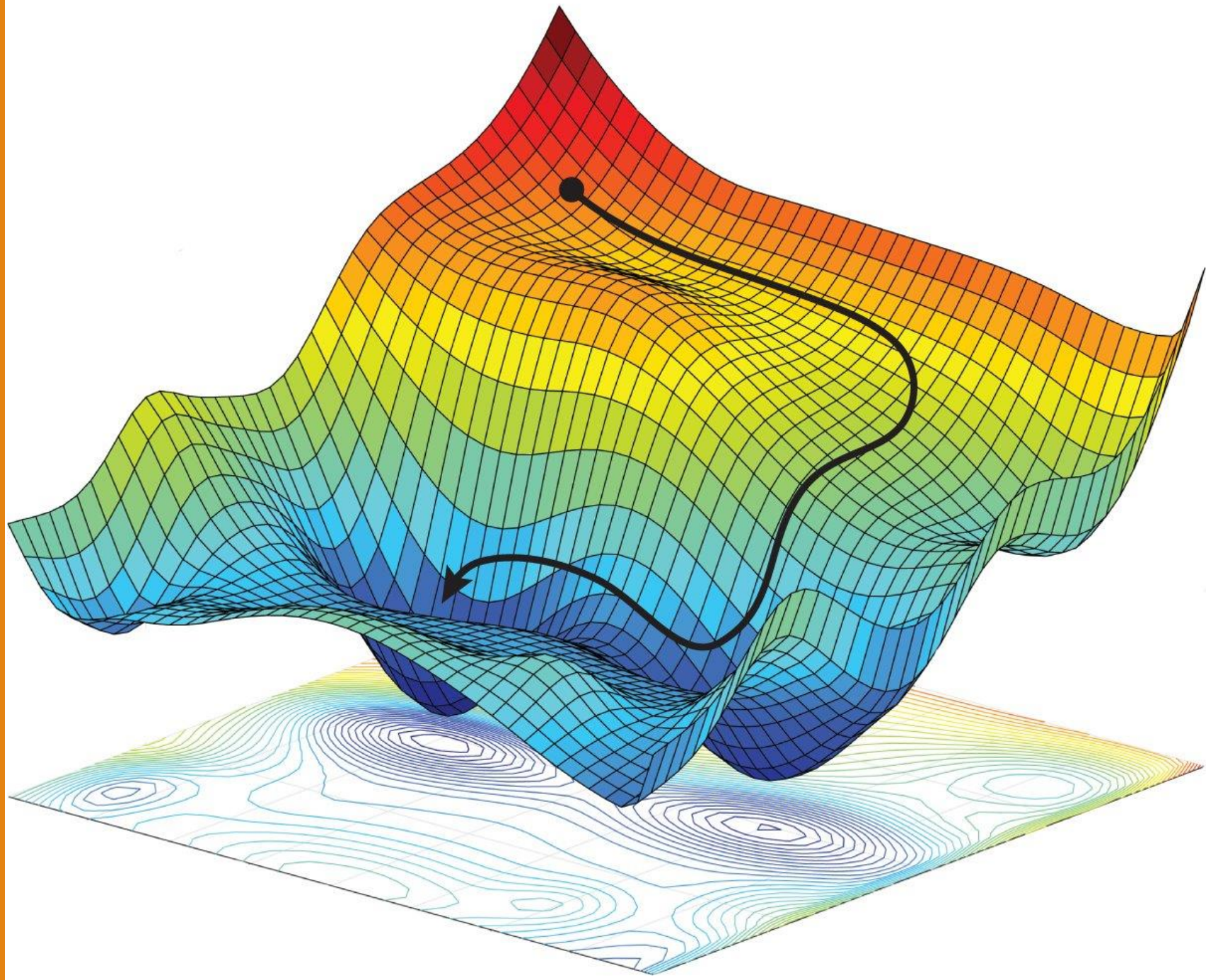
Gradient-based learning



Optimization vs Learning

- In optimization, we want to discover the optimal parameter solution
  - that minimize the cost of specific solution
  - given a parametric definition of model and a set of data
  - For instance, find the optimal train schedule given resources and population

- In learning, we have observed and unobserved data. We want to
  - not only make small errors on the observed data (training data)
  - but also generalize well to unseen data (validation & test data)

Learning optimal parameters

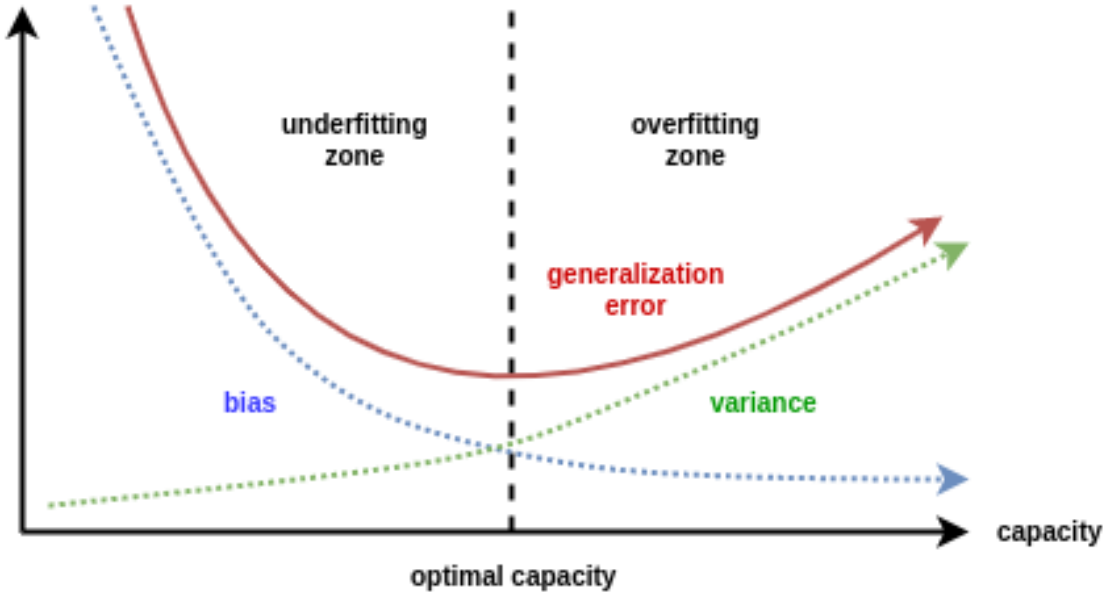
- That said, we still want to optimize on observed data
- Borrowing the optimization toolkit, with extra regularizations

$$\min_{\mathbf{w}} \mathbb{E}_{\boldsymbol{x},l\sim p(\boldsymbol{x},l)}[\mathcal{L}(\mathrm{y}, l)] + \lambda\Omega(\boldsymbol{w})$$

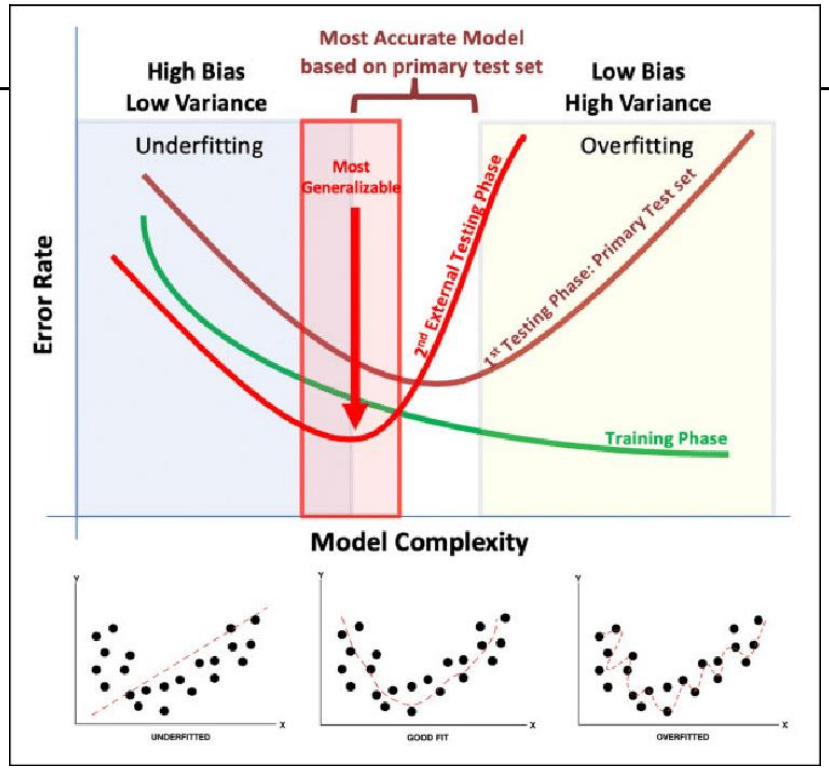
where $y = h_L \circ h_{L-1} \circ \cdots \circ h_1\ (\boldsymbol{x})$ and each module h_l comes with parameters $\boldsymbol{w}_l$

- In simple words, (1) make sure the model predictions are not too wrong
- While (2) not being "too geared/optimized" towards the observed data

Bias-variance tradeoff



Link



Link

- No need for the "optimal" parameters, as they can never be truly optimal
  - We will not see the training data ever again
  - And, the learning objective is often a relaxed proxy
    - we cannot optimize for {0, 1} hard and discrete predictions, we relax it to a continuous proxy (0, 1)

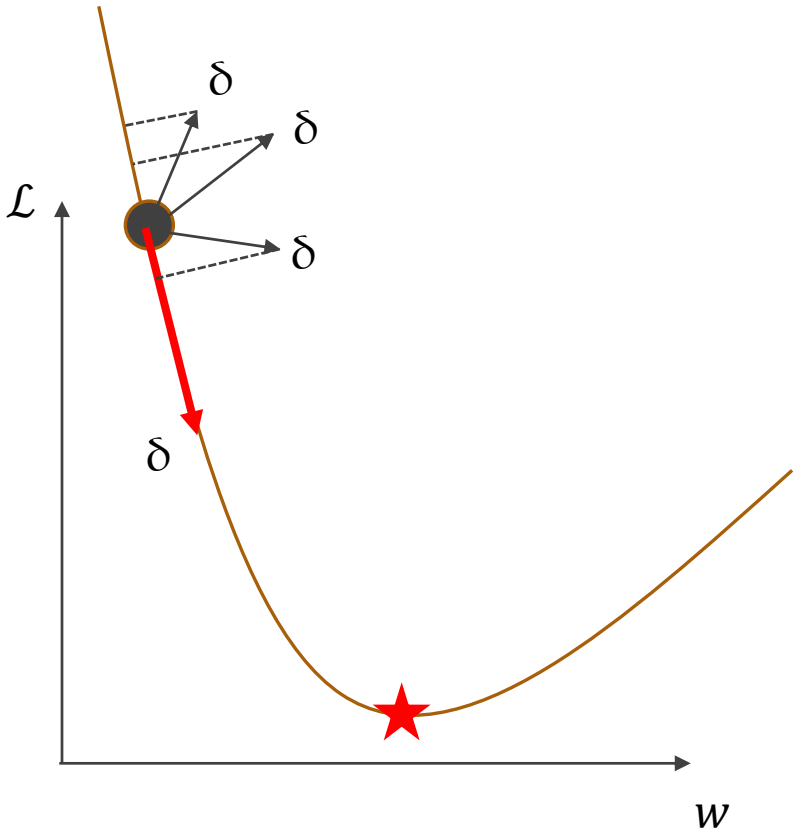
Minimizing the empirical risk

- The expectation of a random variable (RV) should technically be computed on all possible combinations $\boldsymbol{x}, l$
- Clearly, that is not possible
  - Instead stick on observed in training
  - Source of stochasticity

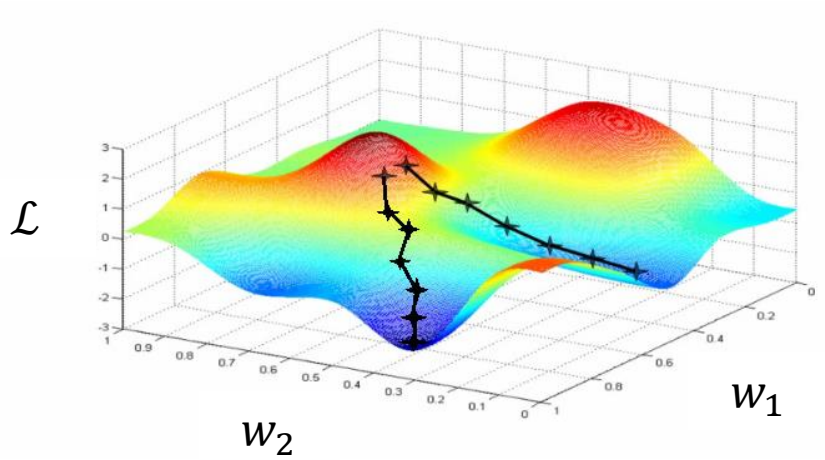
$$\mathbb{E}_{\boldsymbol{x},l\sim p(\boldsymbol{x},l)}[\mathcal{L}(\mathrm{y}, l)] \approx \sum_{x,y^*\in(X,L)_{train}} \mathcal{L}(\mathrm{y}, l)$$

- To minimize any function take a step δ
  - Our best bet: the (negative) gradient

$$-\sum \frac{d}{dw}\mathcal{L}(\mathrm{y}, l)$$



Gradient descent



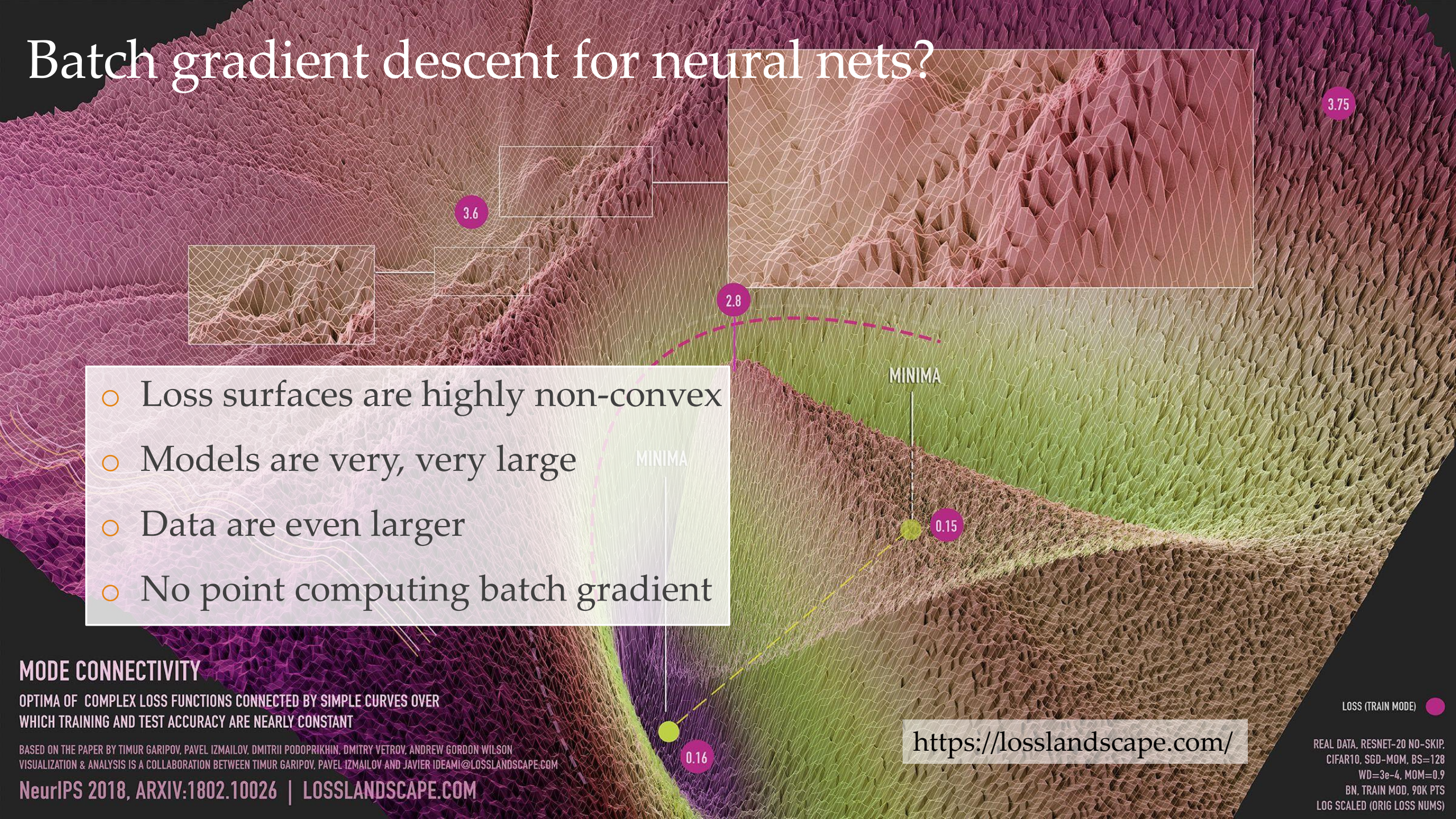
- Gradient descent is the go-to approach

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{d\mathcal{L}}{d\boldsymbol{w}}$$

  - $\boldsymbol{w}^{(t)}$ are the parameters of our neural network at epoch (t)
  - $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is the gradient of the loss w.r.t. the parameters

- If $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is computed from whole dataset → Batch gradient descent
- If $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}}$ is computed from random mini-batch → Stochastic gradient descent
- On non-convex landscapes no guarantee for global optimum

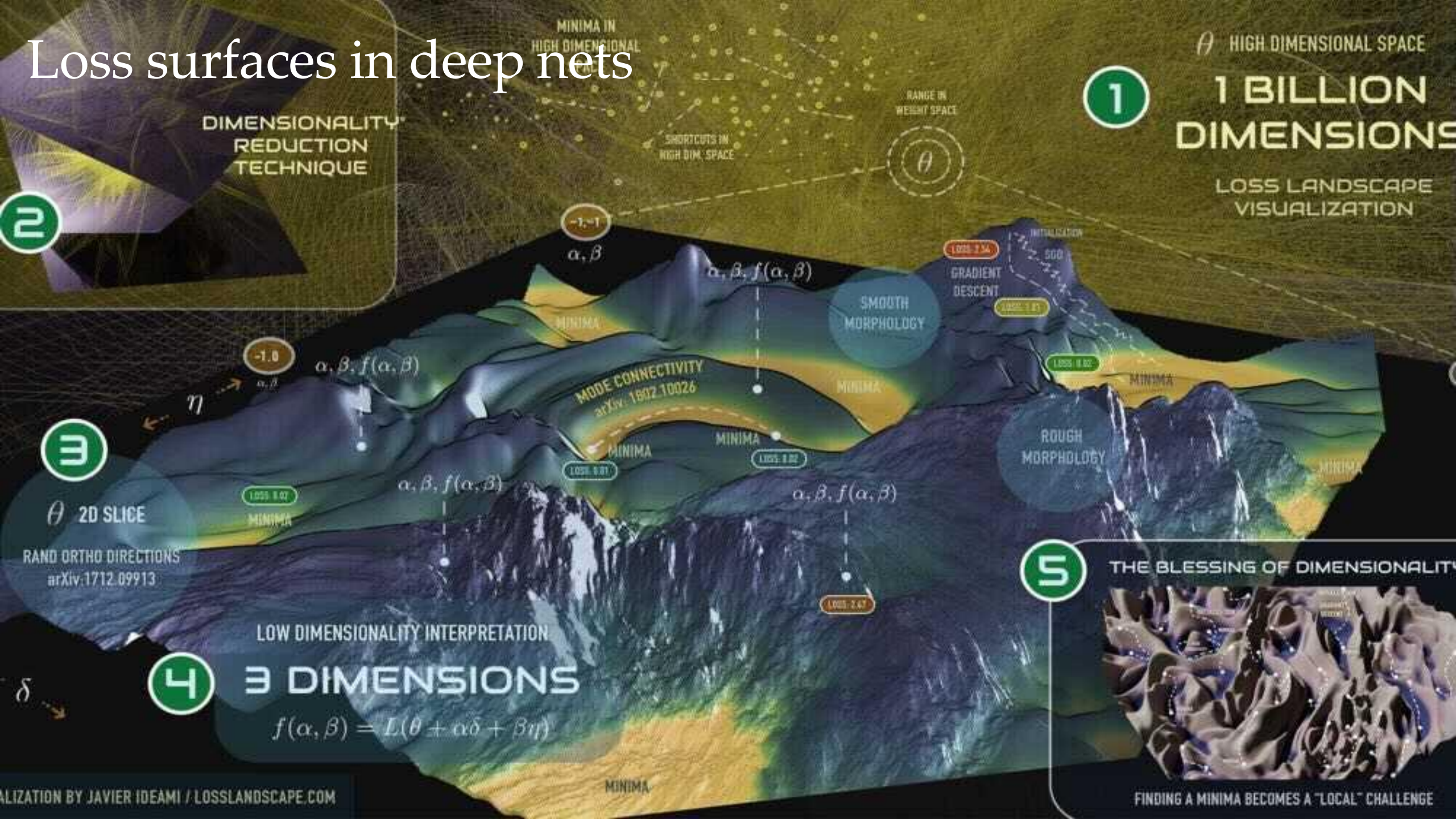
Batch gradient descent for neural nets?



- Loss surfaces are highly non-convex
- Models are very, very large
- Data are even larger
- No point computing batch gradient

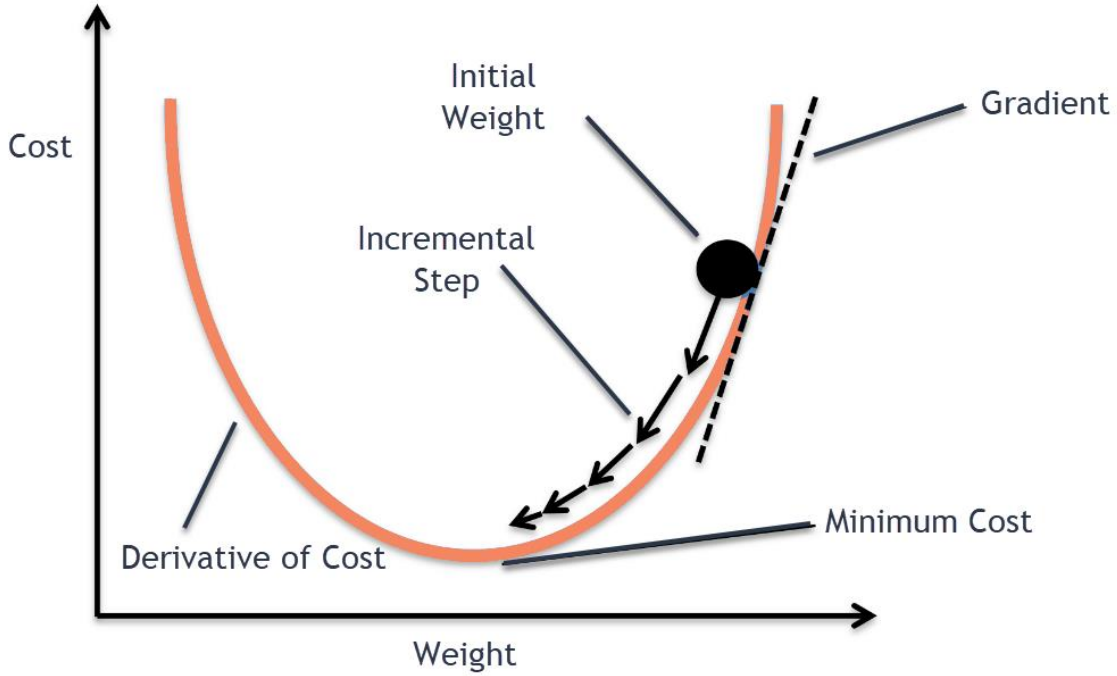
https://losslandscape.com/

Loss surfaces in deep nets



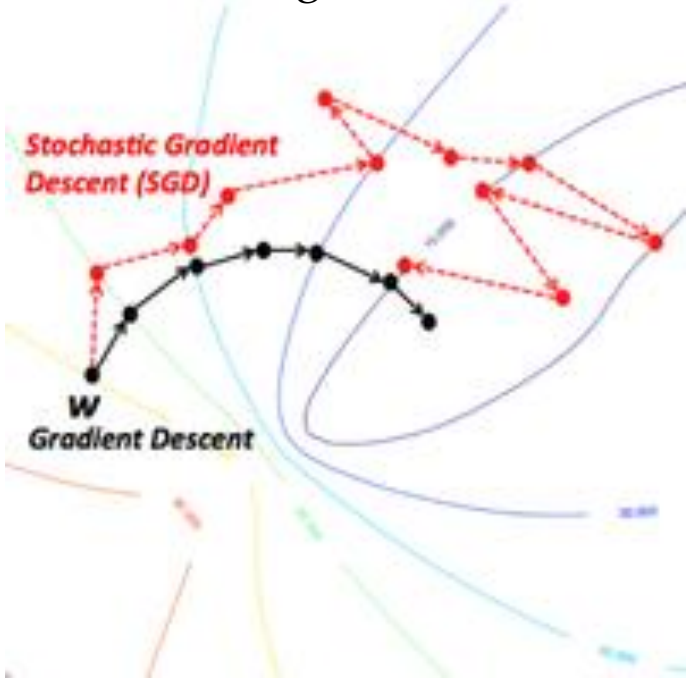
Gradient descent vs. Stochastic Gradient Descent

Gradient descent



Link

Stochastic gradient descent



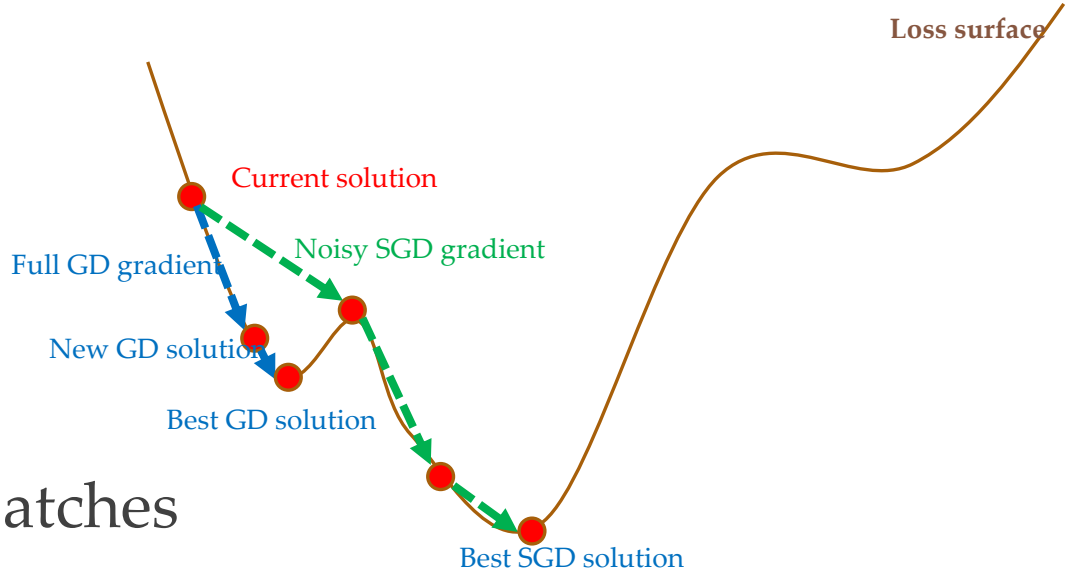
Link

SGD properties

- SGD is doubly stochastic
  - (1) Replacing expectation with sum: $\mathbb{E}_{\boldsymbol{p}(\boldsymbol{x},l)}[\mathcal{L}] \approx \sum_{x,l} \mathcal{L}$ (like Batch GD)
  - (2) Replacing whole training set with mini-batches
- Standard error: $\sigma/\sqrt{n_b}$ where σ is the variance in $\boldsymbol{p}(\boldsymbol{x}, y^*)$, n_b the batch size
  - To compute 2x more accurate gradients, we need 4x data points
  - $n_b = 32$–64 is usually ok
- No need to run through whole dataset to compute a single gradient
- As gradients are good enough and faster, accuracy often better
- Good for streaming data

SGD properties

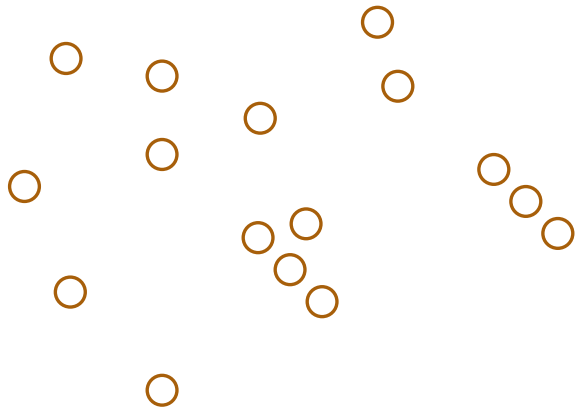
- Randomness reduces overfitting
  - No guarantees though
  - Good shuffling is important
  - Better different mini-batches per epoch
- Must make sure of class/data balance in batches
- Could select samples with max information content
  - But make sure the selection is still "random enough"

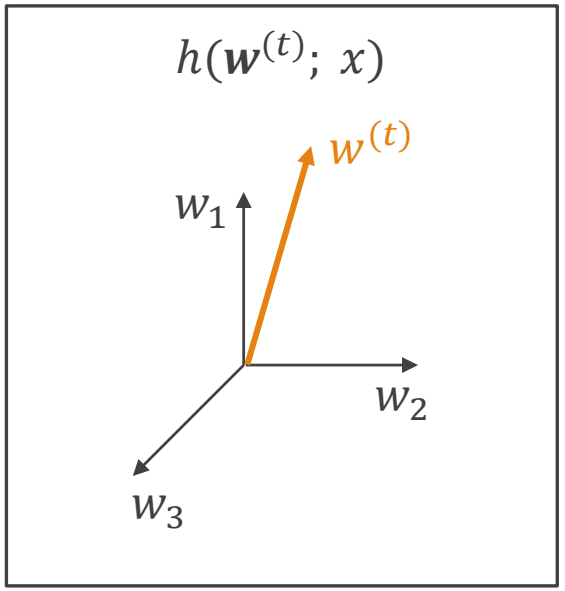


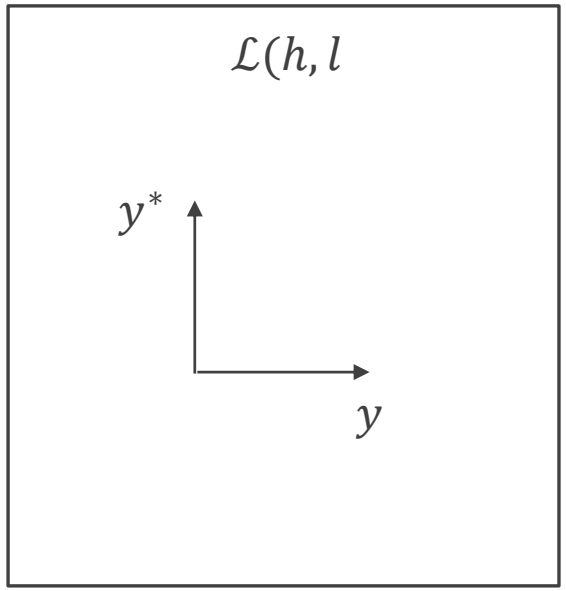
- Preferred because batch GD anyways optimizes what is not the main objective
  - The training data

Stochastic gradient-based optimization

- Stochastic gradient-based optimization is the go-to approach
- It is local, so prone to local peculiarities and minima
- But efficient and effective, so it gets the job done

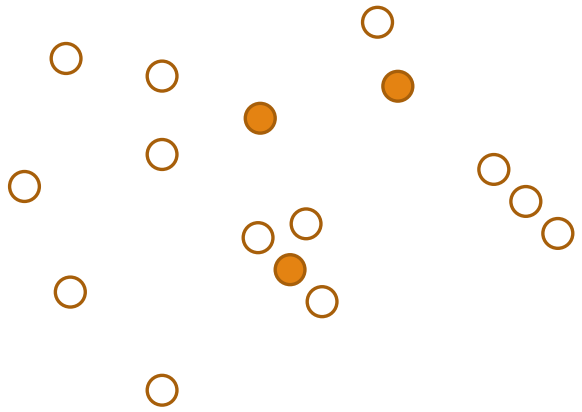




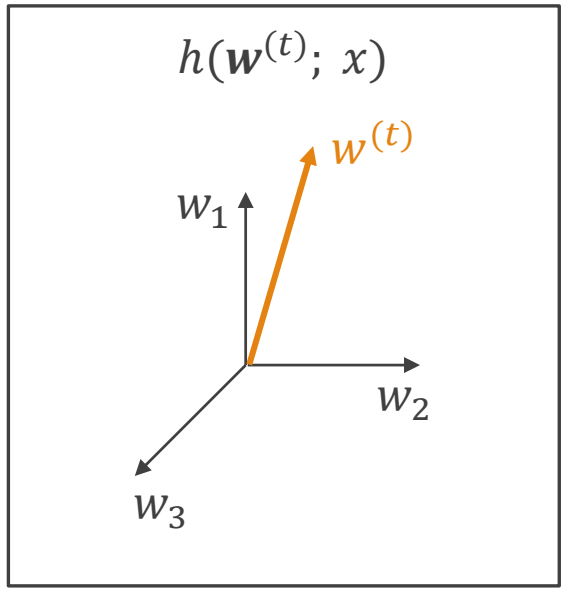


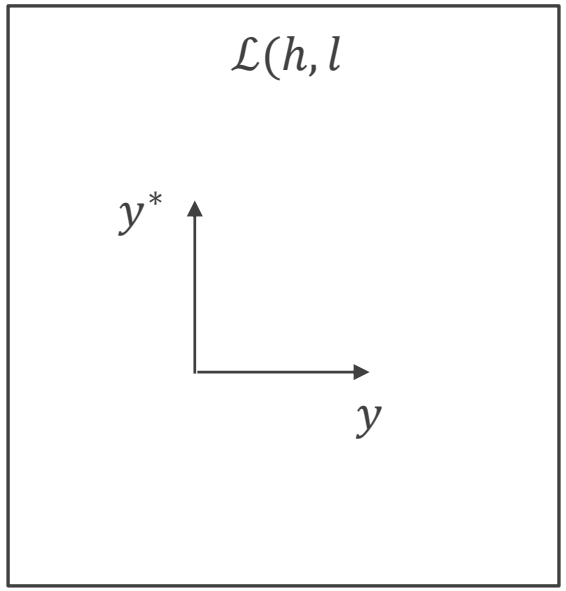
Stochastic gradient-based optimization

- Stochastic gradient-based optimization is the go-to approach
- It is local, so prone to local peculiarities and minima
- But efficient and effective, so it gets the job done



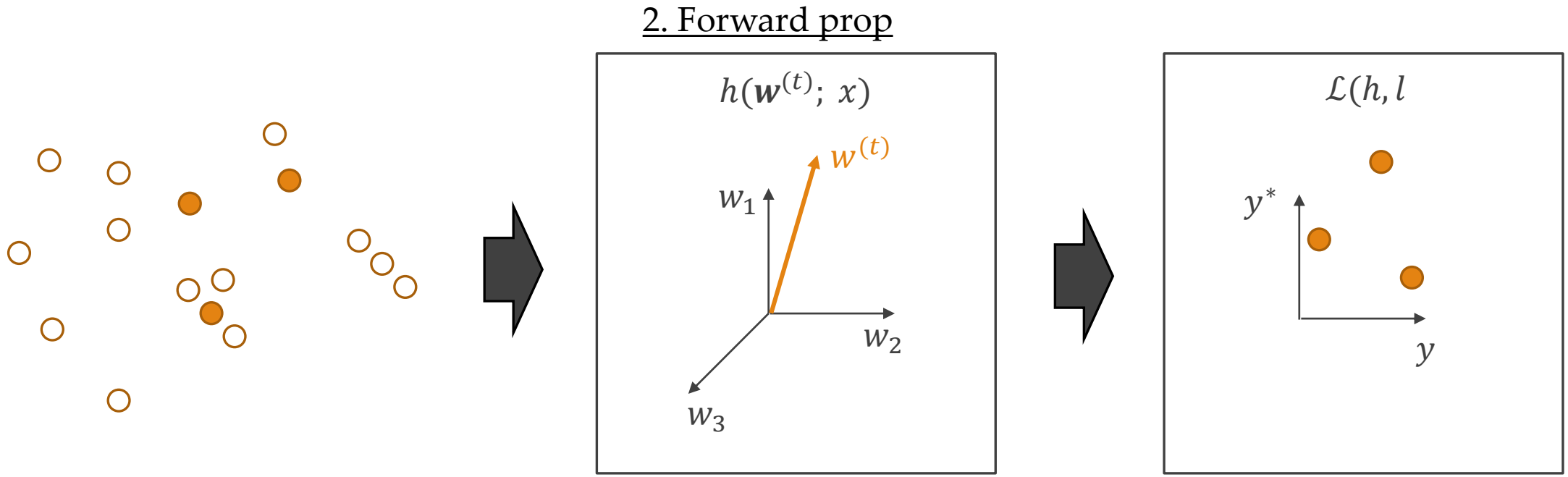
1. Sample mini-batch





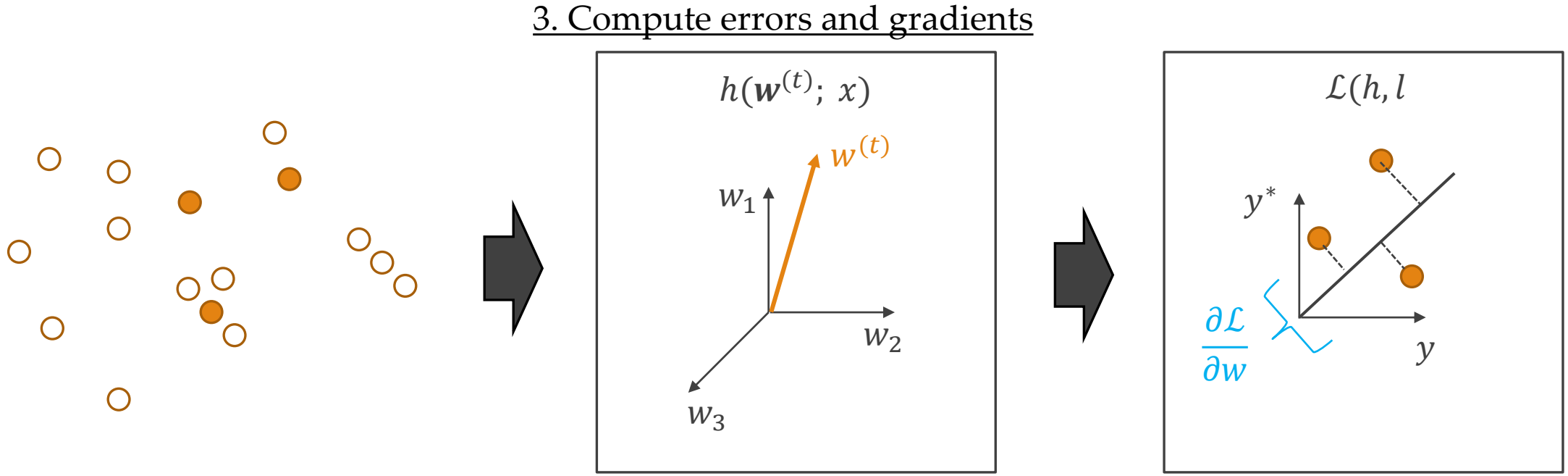
Stochastic gradient-based optimization

- Stochastic gradient-based optimization is the go-to approach
- It is local, so prone to local peculiarities and minima
- But efficient and effective, so it gets the job done



Stochastic gradient-based optimization

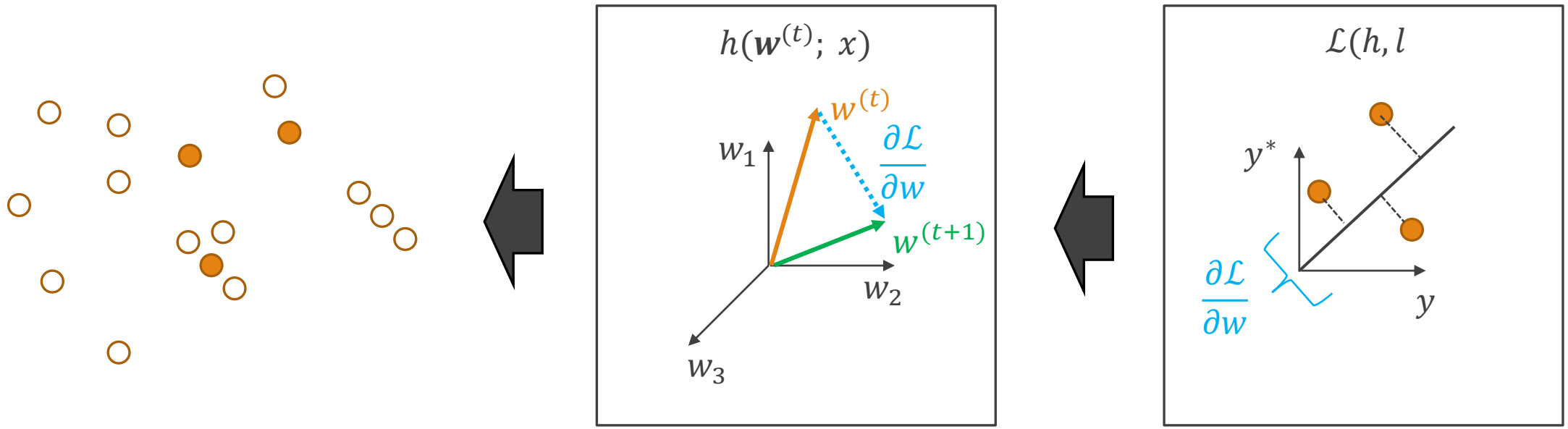
- Stochastic gradient-based optimization is the go-to approach
- It is local, so prone to local peculiarities and minima
- But efficient and effective, so it gets the job done



Stochastic gradient-based optimization

- Stochastic gradient-based optimization is the go-to approach
- It is local, so prone to local peculiarities and minima
- But efficient and effective, so it gets the job done

4. Update model parameters and repeat



Batch gradient descent *vs* stochastic gradient descent

	Batch GD	Stochastic GD
Conditions of convergence	Yes	No
Apply Hessians & accelerations on curvatures	Yes	No
Theoretical analysis	Yes	No
Scales up/efficient	No	Yes
Guaranteed theoretical fast convergence	No	No
Overfitting	Easier	Harder
Global minimum	No	No
In practice, good results	No	Yes

- For deep neural nets: use stochastic gradient descent
  - Virtually nobody uses batch gradient descent with deep nets
  - Doesn't mean you should not explore

In a nutshell

- First, define your neural net

$$y = h_L \circ h_{L-1} \circ \cdots \circ h_1 \, (\boldsymbol{x})$$

where each module h_l comes with parameters $\boldsymbol{w}_l$

- Finding an "optimal" neural network means optimizing the empirical risk

$$\boldsymbol{w}^* = \underset{\mathrm{w}}{\operatorname{argmin}} \sum_{(\boldsymbol{x},y^*)\subseteq(X,L)} \mathcal{L}(y, l)$$

- To optimize the empirical risk, rely on stochastic gradient descent methods

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{d\mathcal{L}}{d\boldsymbol{w}}$$