Northeastern

CS 7140:
ADVANCED MACHINE LEARNING

Recap:Constraint-Based Structure Learning

Learn an I-Equivalent class and find minimal I-Map

Algorithm input $X_1, \cdots, X_n$, output $G^\star$

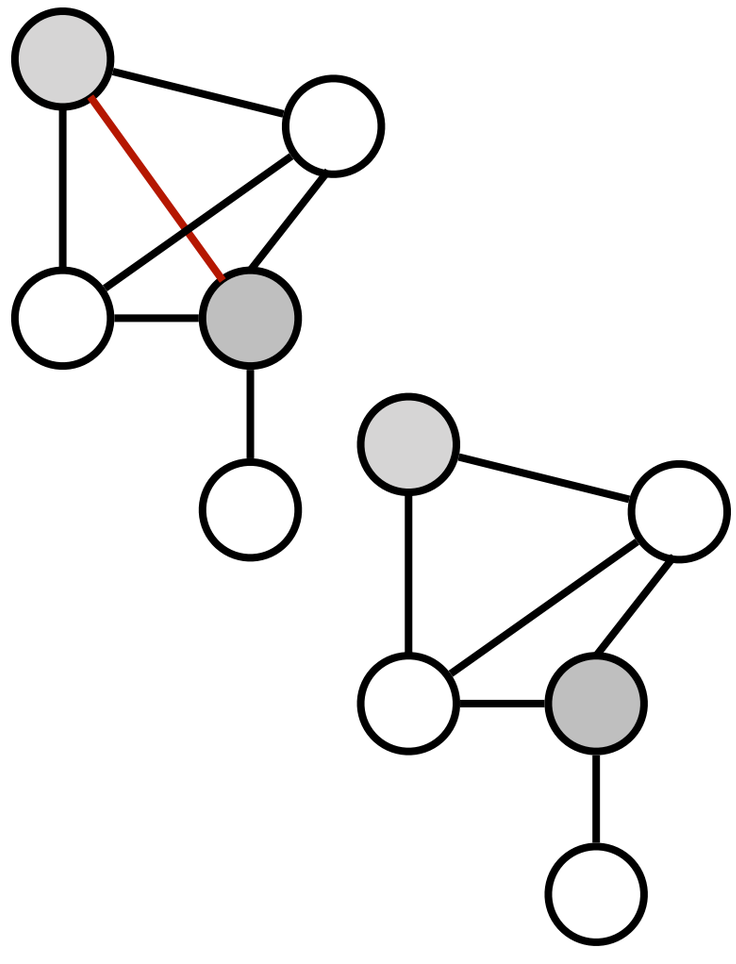
1. Build a complete graph for $X_1, \cdots, X_n$

2. For every pair X_i, X_j, find the witness set U such that $X_i \perp X_j \mid U$

3. Remove edge $X_i - X_j$ if U is not empty

4. Identify triplets that are *immoralities*

5. Orient edges by propagating a set of *constraints*



Recap:Score-Based Structure Learning

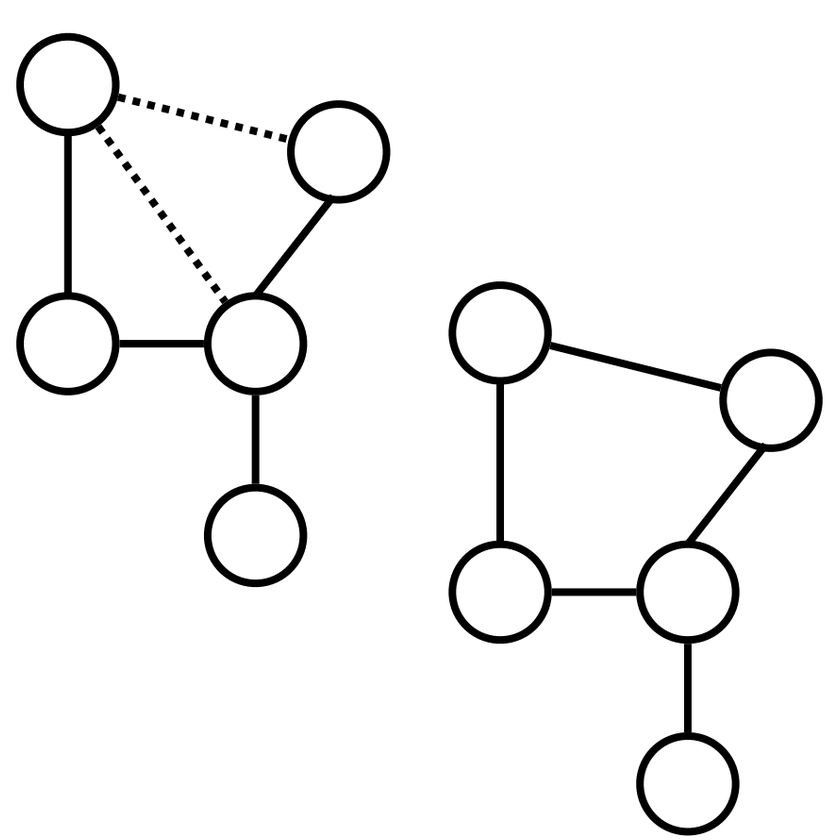
Optimize the network structure score that best fits the data

Algorithm input $X_1, \cdots, X_n$, output $G^\star$, a scoring function

1. Generate a set of possible network structures with bounded indegree

2. Search the space of graphs

3. Return the high-scoring one



Recap:Bayesian Model Averaging

Compute the average prediction over candidate network structures

$$\mathbb{E}_{P(G|D)}[f(G)] = \sum_G f(G)P(G \mid D)$$

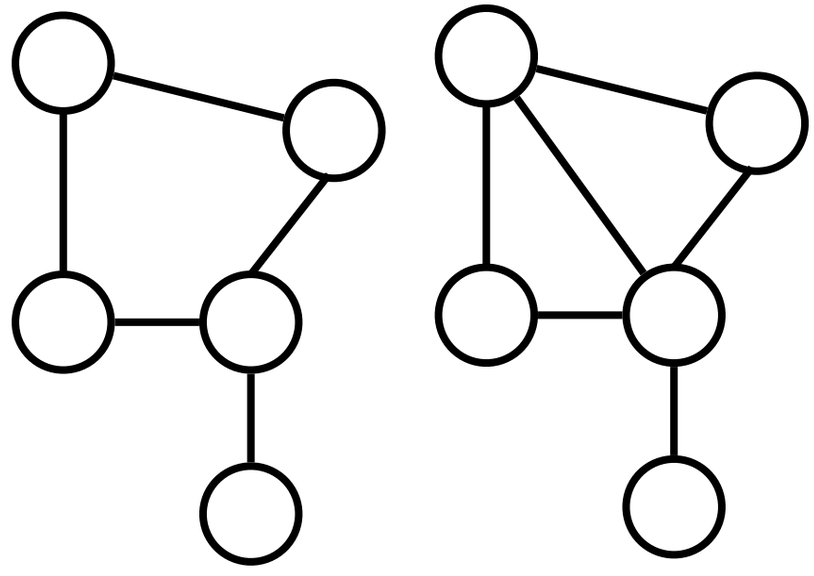
Algorithm input $X_1, \cdots, X_n$, output $G^\star$, a prediction function

1.Find a set $\mathscr{G}'$ of high scoring structures

2.Estimate the probability of the structure in $\mathscr{G}'$

3.Compute average prediction

$$P(f \mid D) \approx \frac{\sum_{G \in \mathscr{G}'} P(G \mid D) f(G)}{\sum_{G \in \mathscr{G}'} P(G \mid D)}$$



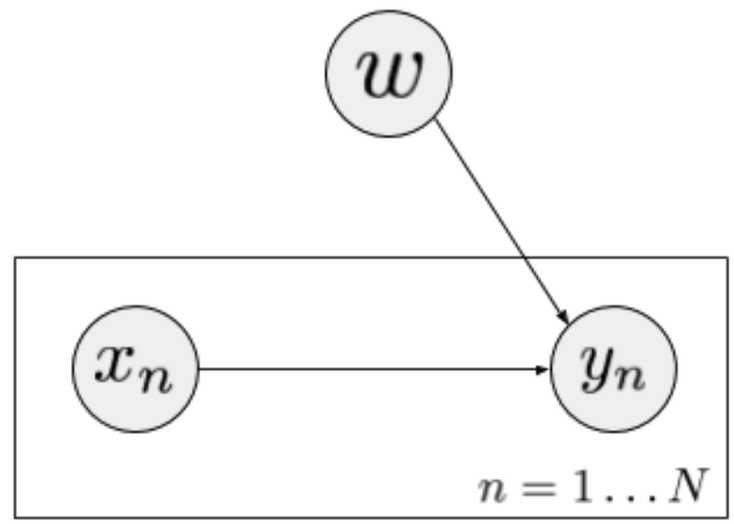
$f(G_1)$ + $f(G_2)$

LEARNING AS INFERENCE

Motivating Example: Logistic Regression

- Two variables x, y, the probability

$$P(y = 1 \mid x, w) = \frac{1}{1 + \exp(-w^\top x)}$$

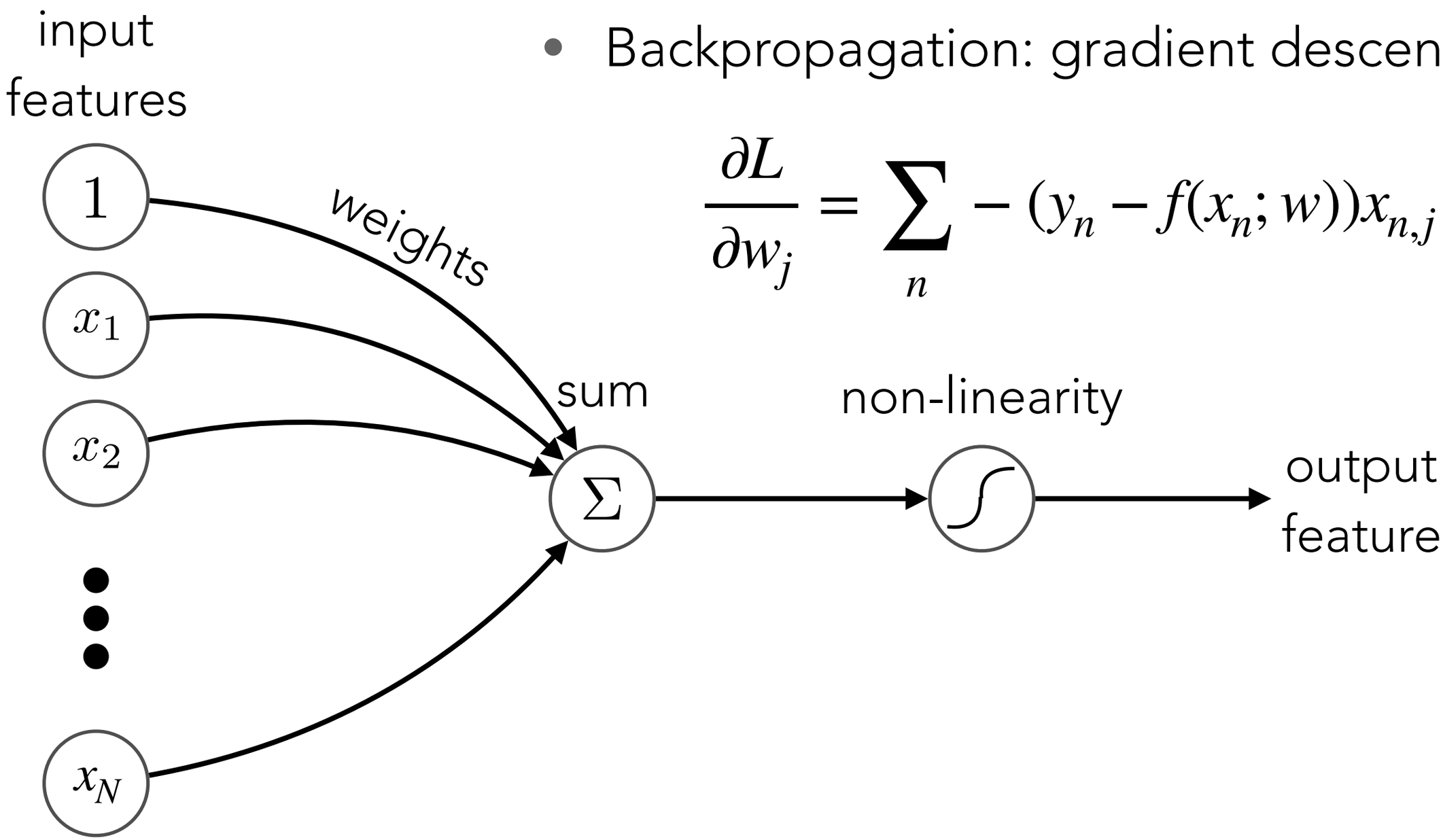


- A neural network implements a nonlinear function $f(x; w)$ with inputs x, parameterized by weights w.

- Cross entropy loss:

$$L(w) = -\sum_n \left[y_n \log f(x_n; w) + (1 - y_n) \log(1 - f(x_n; w)) \right]$$

Artificial Neuron



- Backpropagation: gradient descent

$$\frac{\partial L}{\partial w_j} = \sum_n -(y_n - f(x_n; w))x_{n,j}$$

Probabilistic Interpretation

- Cross entropy with regularization

  $J(w) = L(w) + \alpha R(w)$

  $L(w) = -\sum_n \left[y_n \log f(x_n; w) + (1 - y_n) \log(1 - f(x_n; w)) \right] \quad R(W) = \frac{1}{2} \sum_i w_i^2$

- Activation of a single neuron $f(x; w) \equiv P(y = 1 \mid x, w)$ defines the probability input x belongs to positive class

- Regularizer can be interpreted as a log prior probability over the parameters $P(w \mid \alpha) = \frac{1}{Z} \exp(-\alpha R(w))$

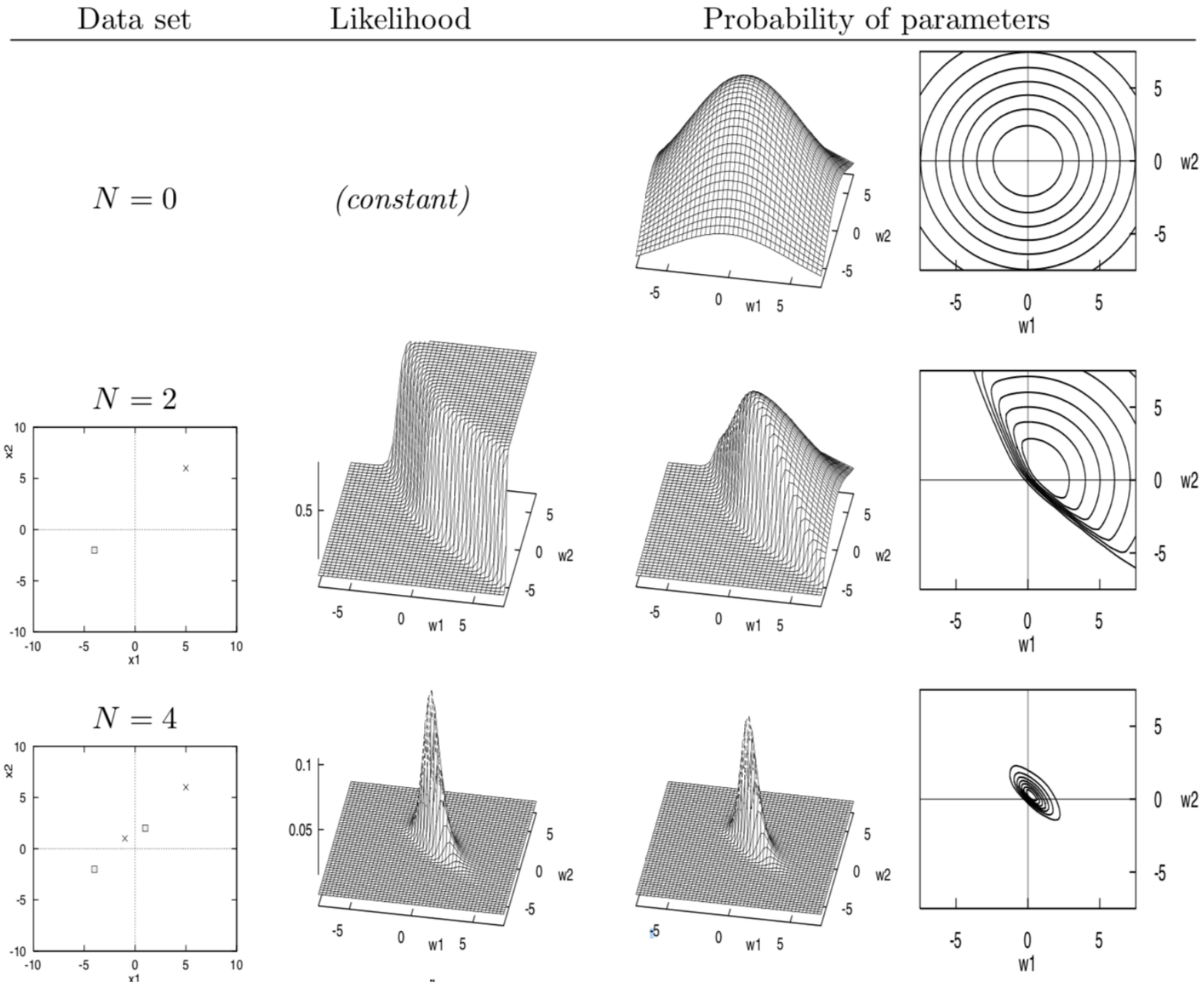
- Objective function $J(w)$ has a log probability interpretation

Example

- Consider a neuron with two inputs and no bias

$$f(y; w) = \frac{1}{1 + \exp^{-(w_1 x_1 + w_2 x_2)}}$$

- The estimator maximizes the posterior probability density



Making Predictions

- The posterior probability of w, $P(w \mid D, \alpha)$

$$P(w \mid D, \alpha) = \frac{1}{Z} \exp(-J(w))$$

- Bayesian prediction

$$P(y_{N+1} = 1 \mid x_{N+1}, D, \alpha) = \int d^K w f(x; w) \frac{1}{Z} \exp(-J(w))$$

- How to compute the integral?
  - **General**: exact inference, approximate inference
  - **NN**: Monte Carlo, Gaussian approximation

Langevin Monte Carlo

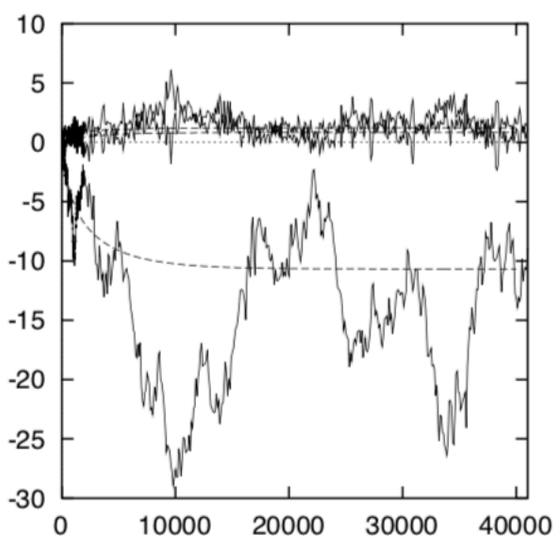
- Monte Carlo: evaluate the integral of a distribution

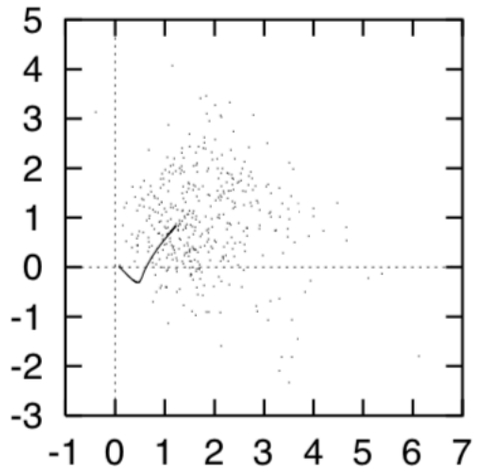
$$\mathbb{E}[f(w)] \approx \frac{1}{R}\sum_{r} f(w^{(r)})$$

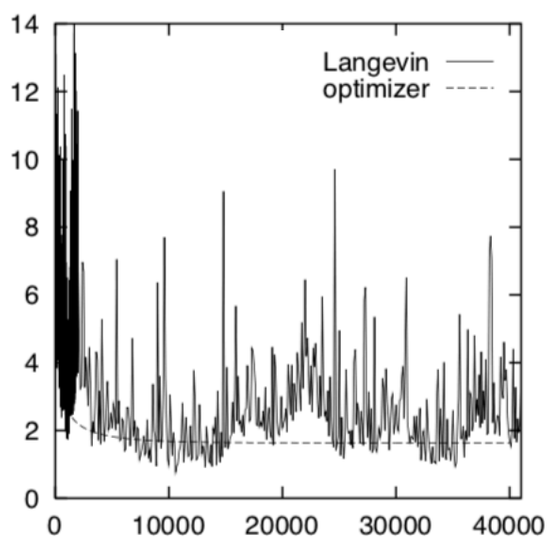
- Langevin: gradient descent with added noise $p \sim \mathcal{N}(0,1)$

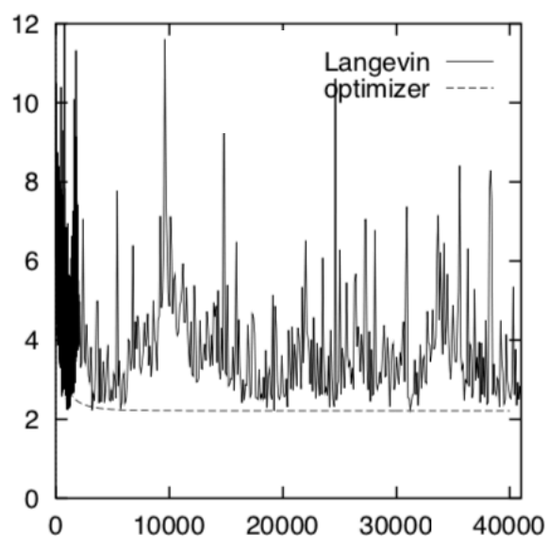
$\Delta w = \frac{1}{2}\epsilon^2 g + \epsilon p$ accept or reject depending on $J(w)$, g is the gradient

- ϵ controls the step size, too *large*, moves may be rejected; too *small*, progress around the state space is slow









Gaussian Approximation

- Taylor expansion $J(w) \approx J(w_0) + \frac{1}{2}(w - w_0)^\top A(w - w_0) + \cdots$

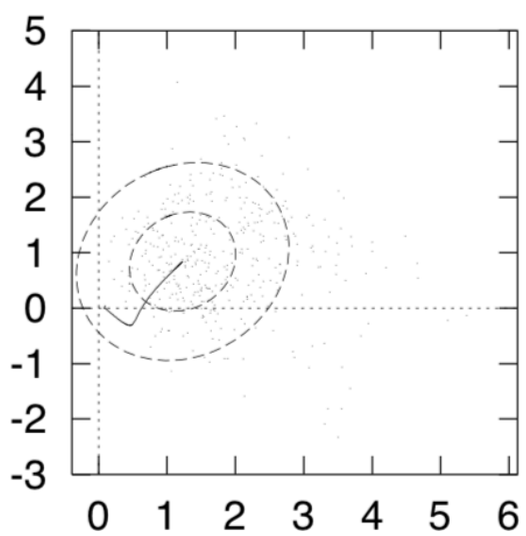
  where $A_{ij} \equiv \frac{\partial^2}{\partial w_i w_j} J(w)$ is the *Hessian* matrix

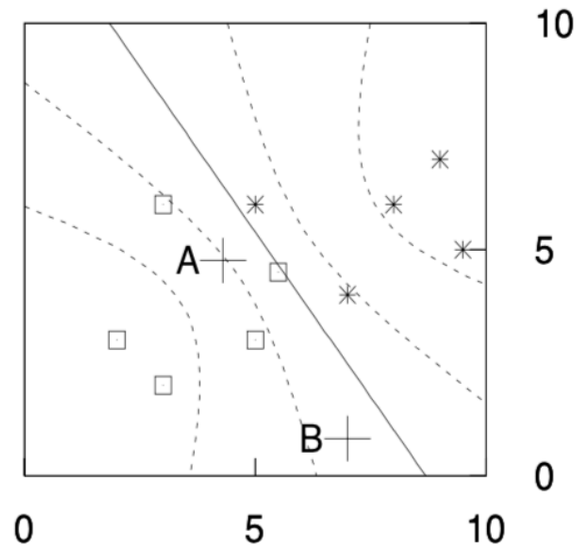
- Approximate the posterior with a Gaussian

  $$Q(w; w_0, A) = [\det A/2\pi]^{1/2} \exp\Big[-\frac{1}{2}(w - w_0)^\top A(w - w_0)\Big]$$

- Compute the integral using Gaussian posterior

  $$\Delta w = w - w_0 \qquad P(w \mid D, \alpha) \approx (1/Z_Q)\exp(-\frac{1}{2}\Delta w^\top A \Delta w)$$





OPTIMIZATION

Stochastic Gradient Descent

- large scale (n) datasets leads to memory bottleneck

- stochastic gradient descent (SGD): $w = w - \alpha \tilde{\nabla}_w \mathcal{L}$

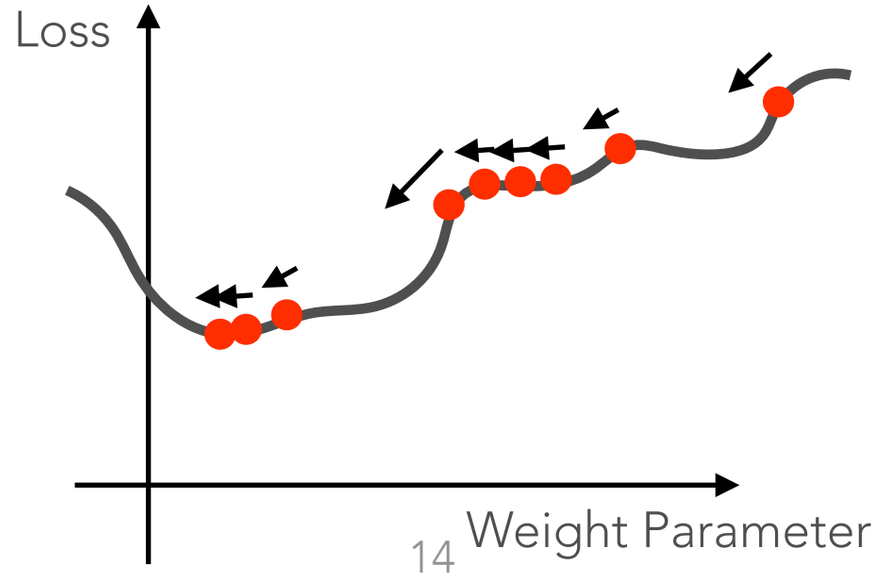
use *stochastic gradient* estimate to *descend* the surface of the loss function

batch gradient

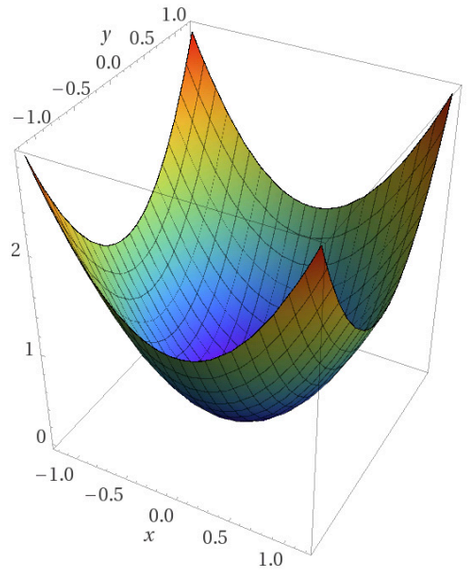
$$\nabla L = \sum_{i=1}^{n} \nabla_w L(f(\mathbf{x}^{(i)}; w), y^{(i)})$$

stochastic gradient

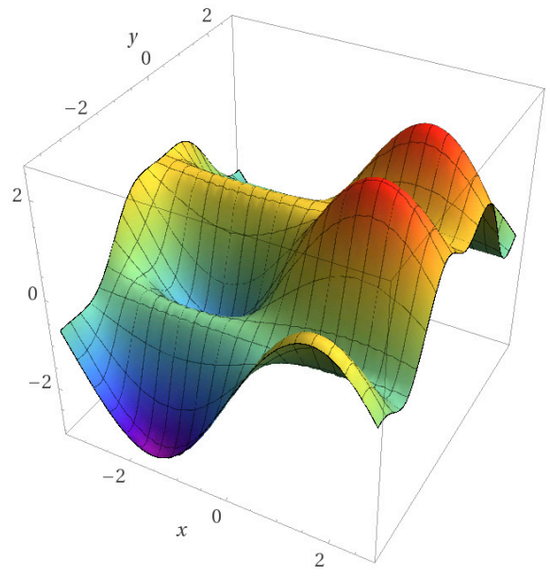
$$\tilde{\nabla}_w L = \nabla_w L(f(\mathbf{x}^{(i)}; w), y^{(i)})$$



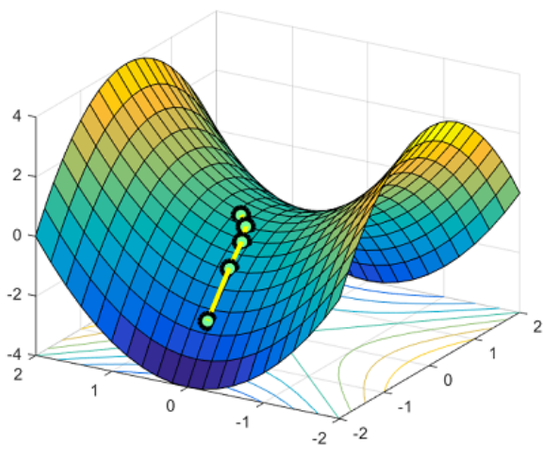
Optimizing Deep Neural Nets



convex

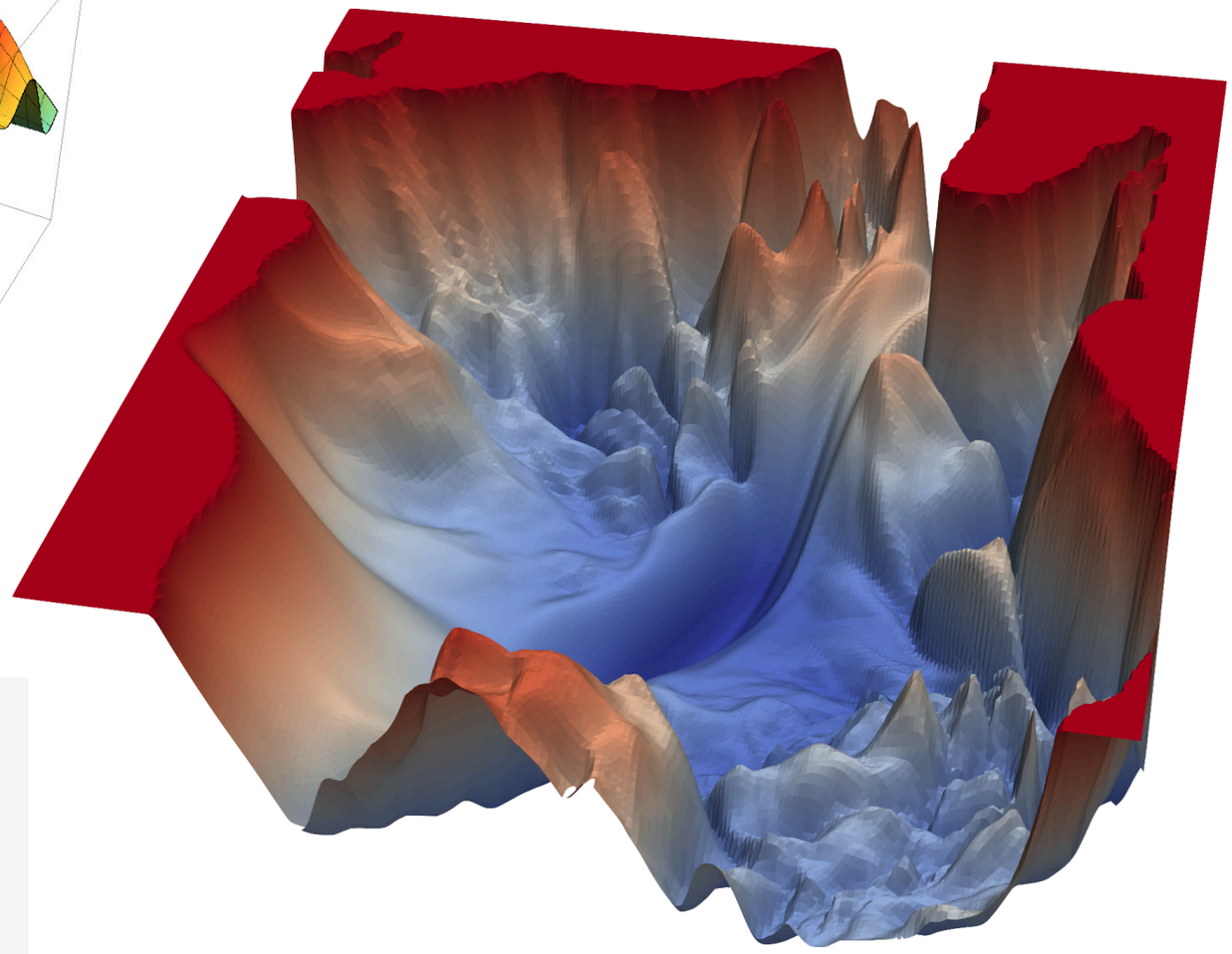


non-convex



saddle point

VGG-56

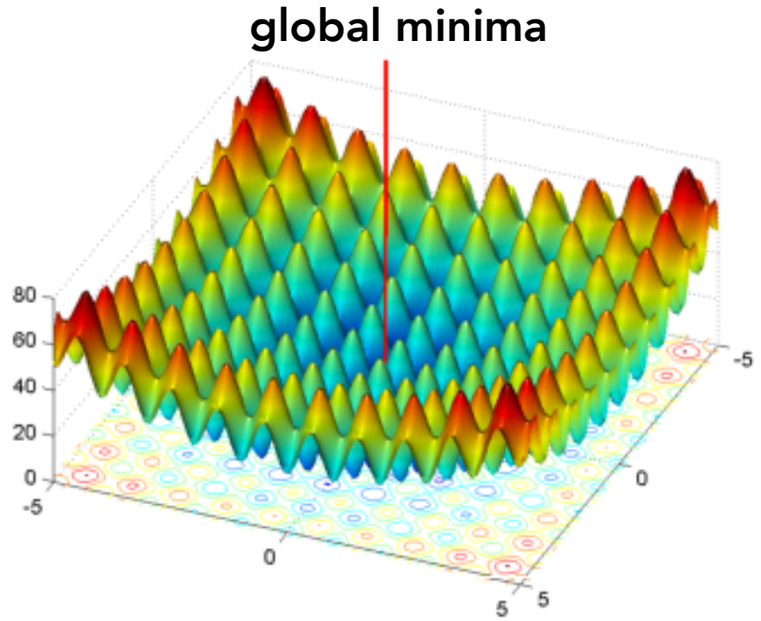


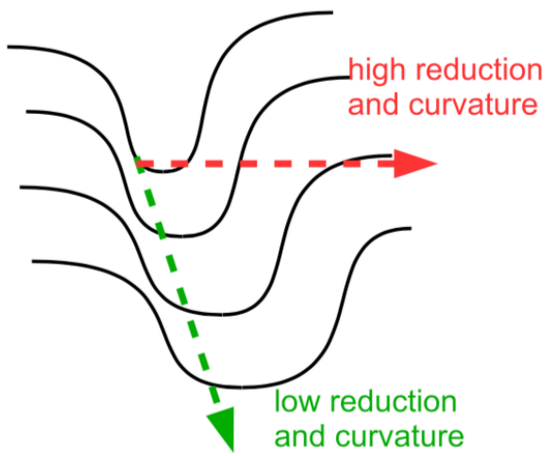
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *Advances in Neural Information Processing Systems*. 2018.

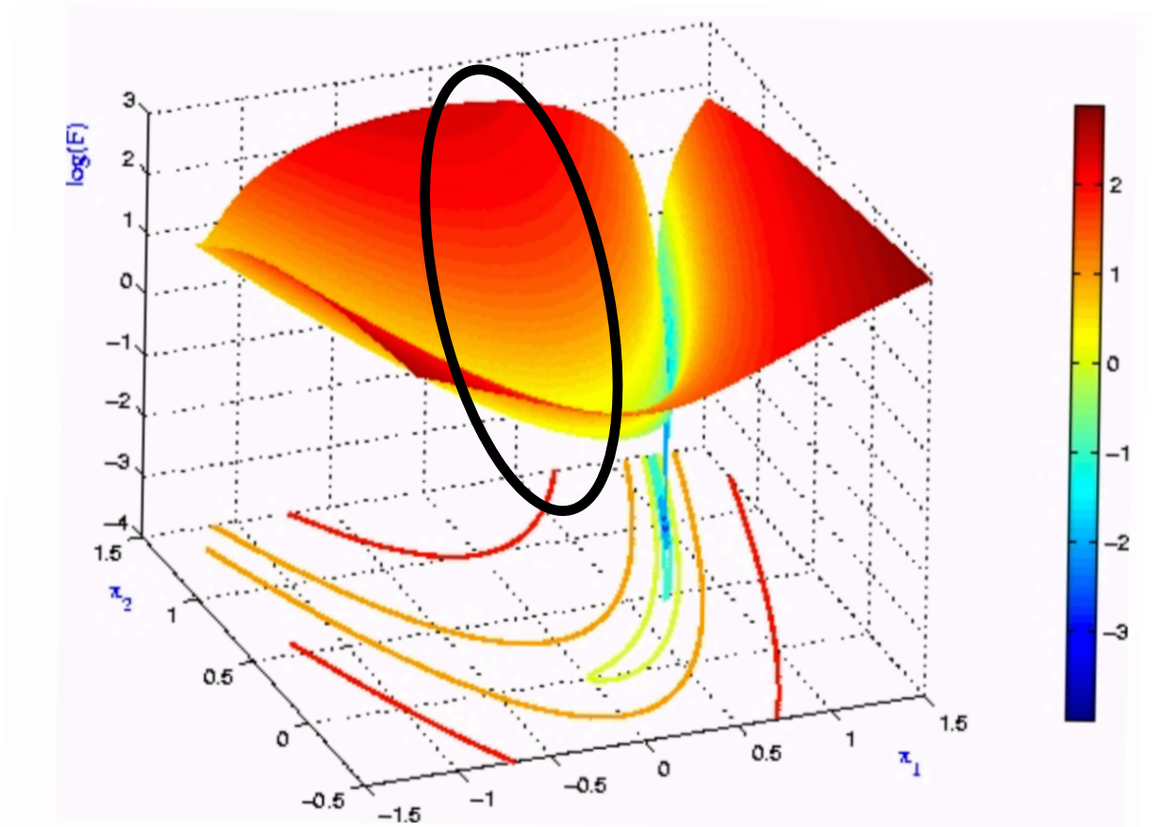
SGD is Not Enough

- Increased frequency and severity of bad local minima

- pathological curvature, like the type seen in the well-known Rosenbrock function







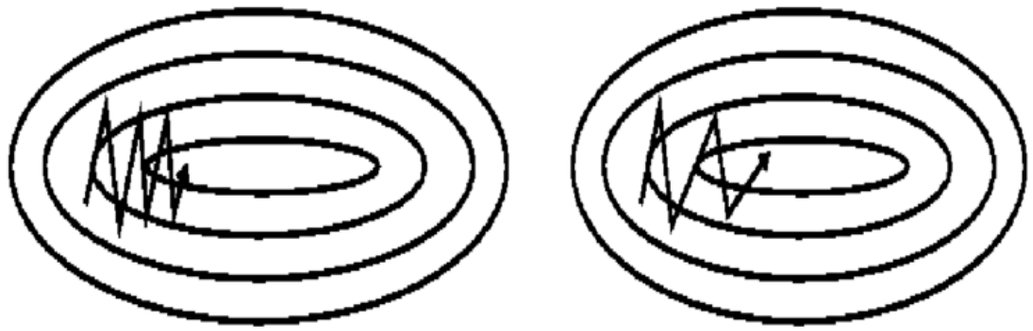
Accelerating SGD

- Momentum

physical interpretation: update velocity

$$w = w - \alpha \tilde{\nabla}_w \mathcal{L}$$

$$v_t = \alpha v_{t-1} - \epsilon \, \tilde{\nabla}_w \, L(w_t)$$
$$w_{t+1} = w_t + v_t$$



SGD *without* momentum SGD *with* momentum

- Nesterov Momentum

stronger theoretical guarantee

$$v_t = \alpha v_{t-1} - \epsilon \, \tilde{\nabla}_w \, L(w_t + \alpha v_{t-1})$$
$$w_{t+1} = w_t + v_t$$

Adaptive Learning Rate

Cost is sensitive to learning rate only in some directions in parameter space

Use a separate learning rate for each parameter

- AdaGrad

gradient

approximate Hessian

update

$$g = \tilde{\nabla}_w L(w_t)$$

$$r_t = r_{t-1} + g \odot g$$

$$w_{t+1} = w_t - \frac{\epsilon}{\delta + \sqrt{r}} \odot g$$

- RMSProp

gradient

approximate Hessian

update

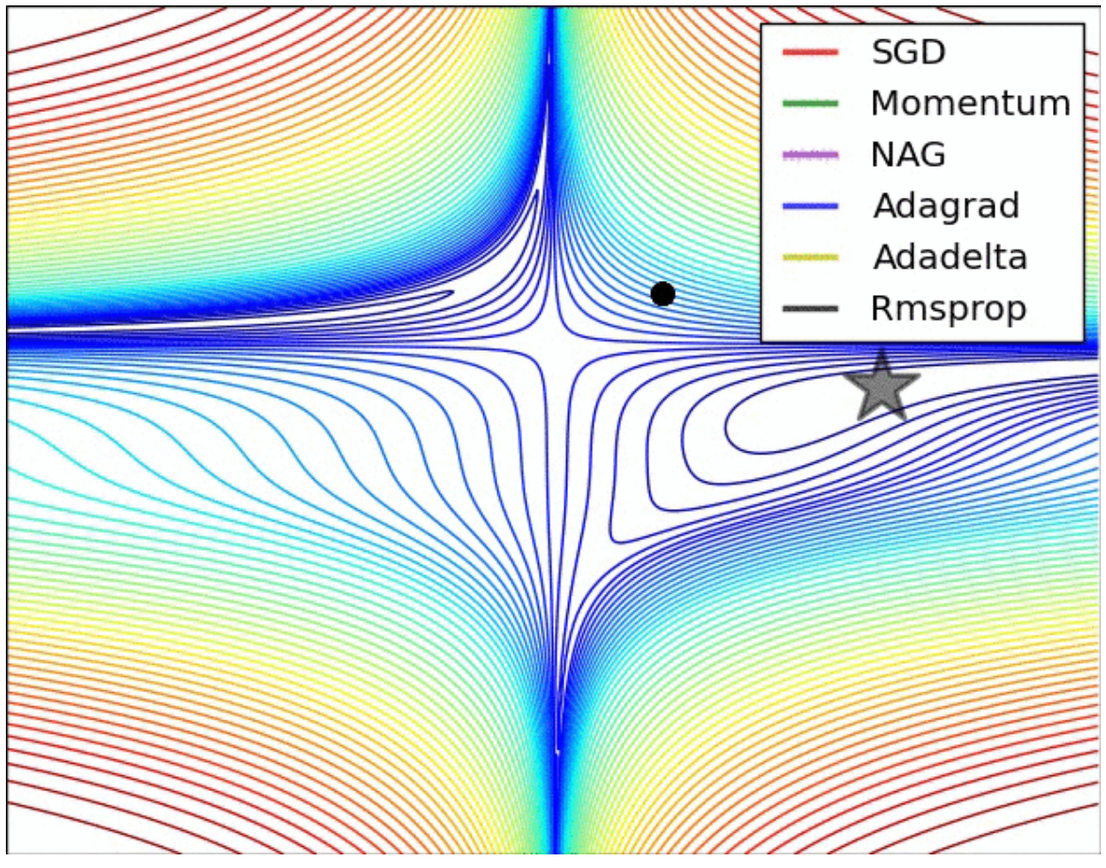
$$g = \tilde{\nabla}_w L(w_t)$$

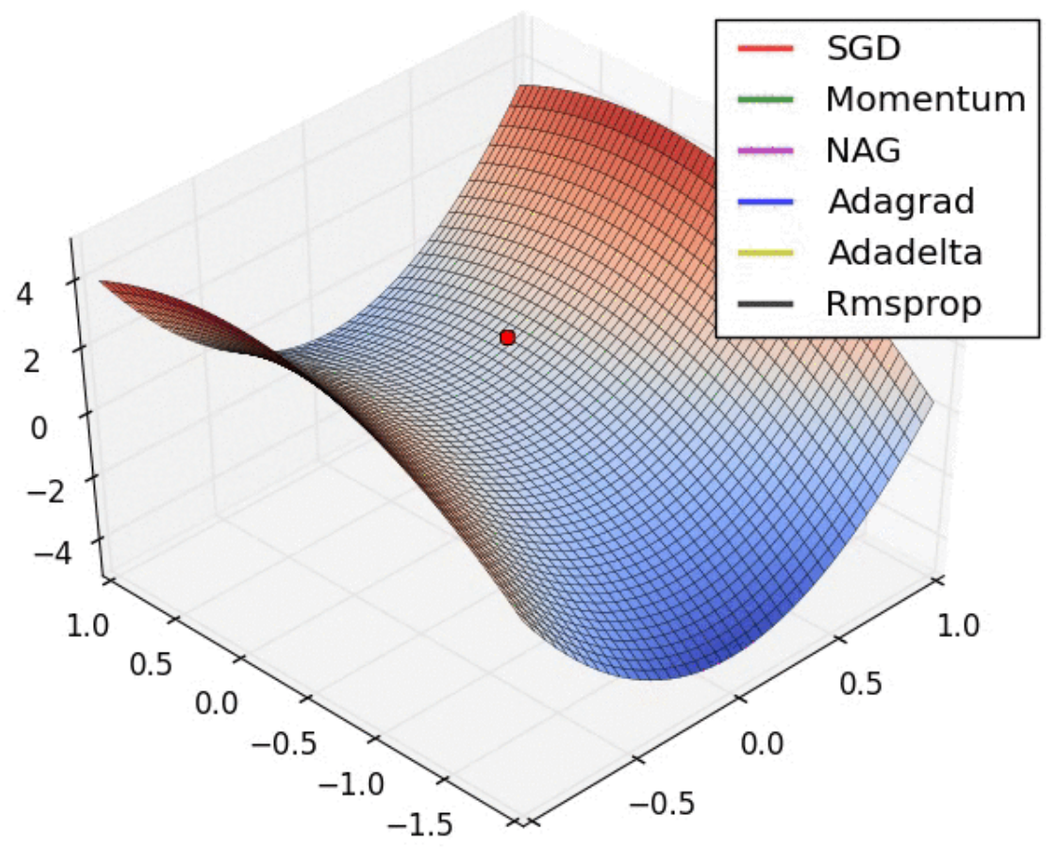
$$r_t = \rho r_{t-1} + (1 - \rho) g \odot g$$

$$w_{t+1} = w_t - \frac{\epsilon}{\sqrt{\delta + r}} \odot g$$

Comparison of Optimization Algorithms

local minima and saddle points are largely not an issue
in many dimensions, can move in exponentially more directions





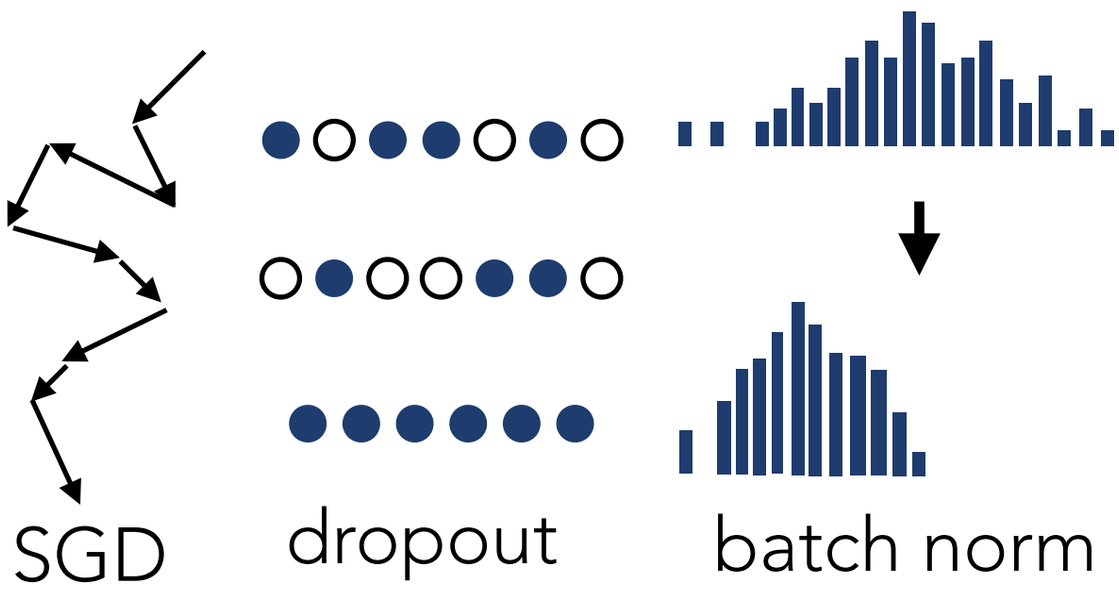
http://sebastianruder.com/optimizing-gradient-descent/index.html

Regularization

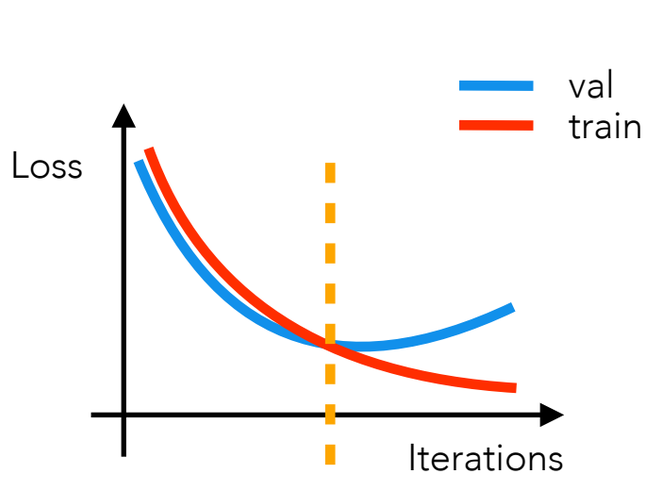
neural networks are amazingly flexible…
given enough parameters, they can perfectly fit random noise

regularization combats **overfitting**

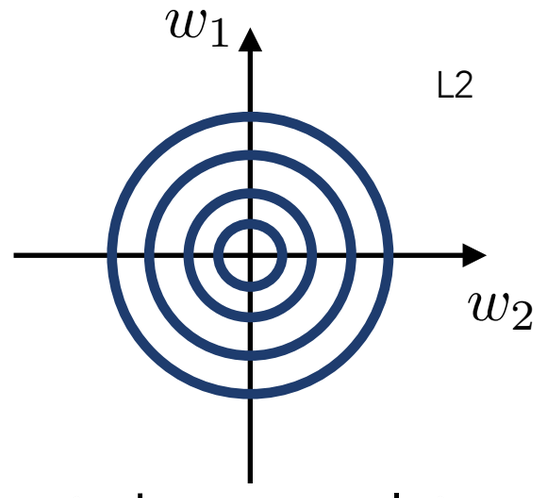
stochasticity (uncertainty)



constraints



early stopping

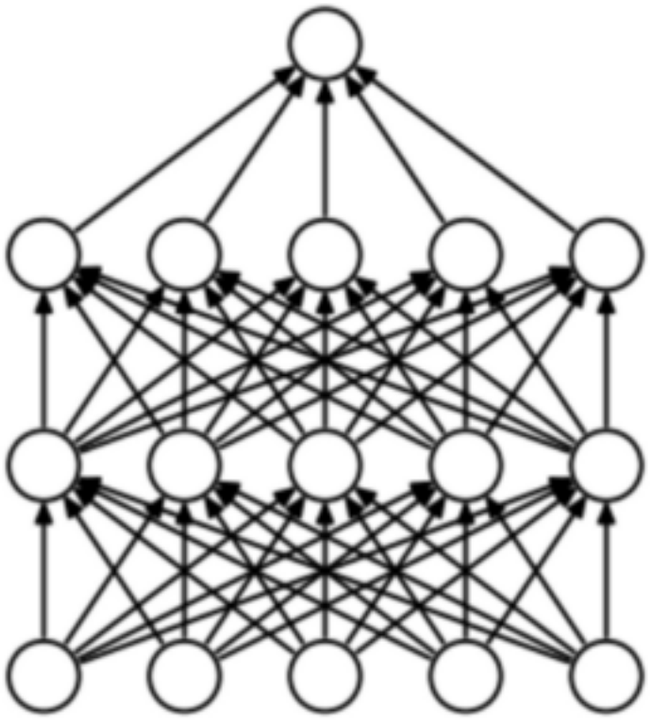


weight penalties

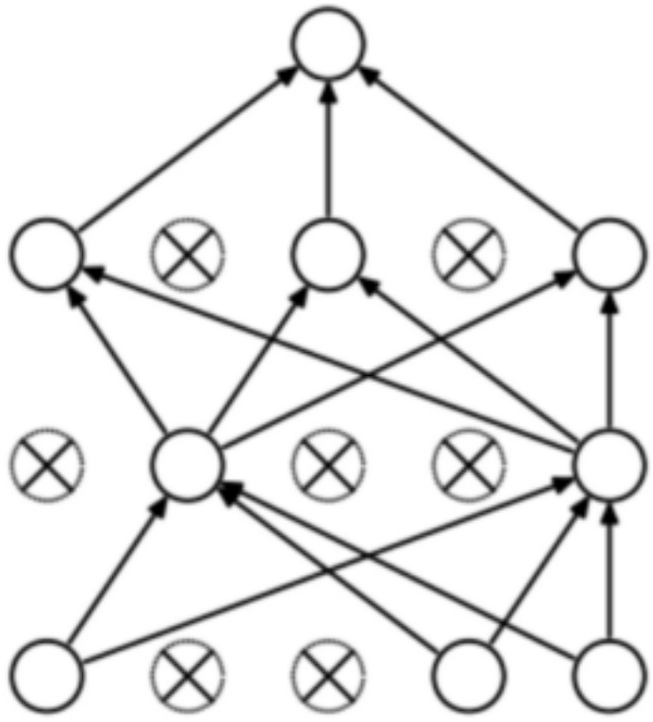
Dropout

Dropping out units (both hidden and observed) in a neural network

Keep with probability p, drop out with probability 1-p



standard neural net



neural net with dropout

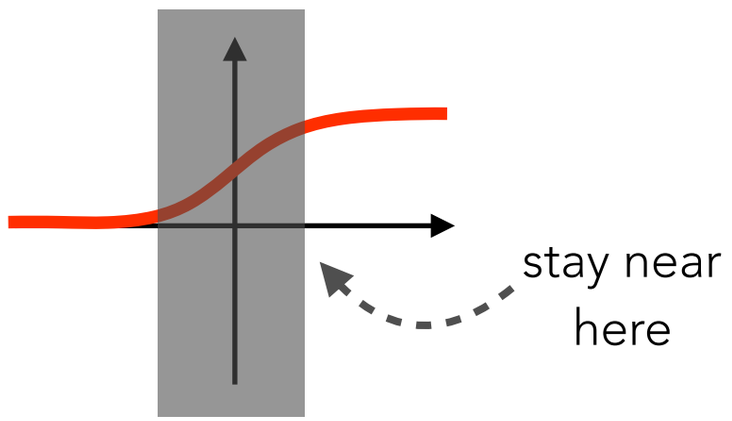
Batch Normalization

batch norm. normalizes each layer's activations according to the statistics of the batch

$$\mathbf{s}^{(\ell)} \leftarrow \gamma \frac{\mathbf{s}^{(\ell)} - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} + \beta$$

$\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}$ are the batch mean and std. deviation

γ, β are additional parameters (affine transformation)

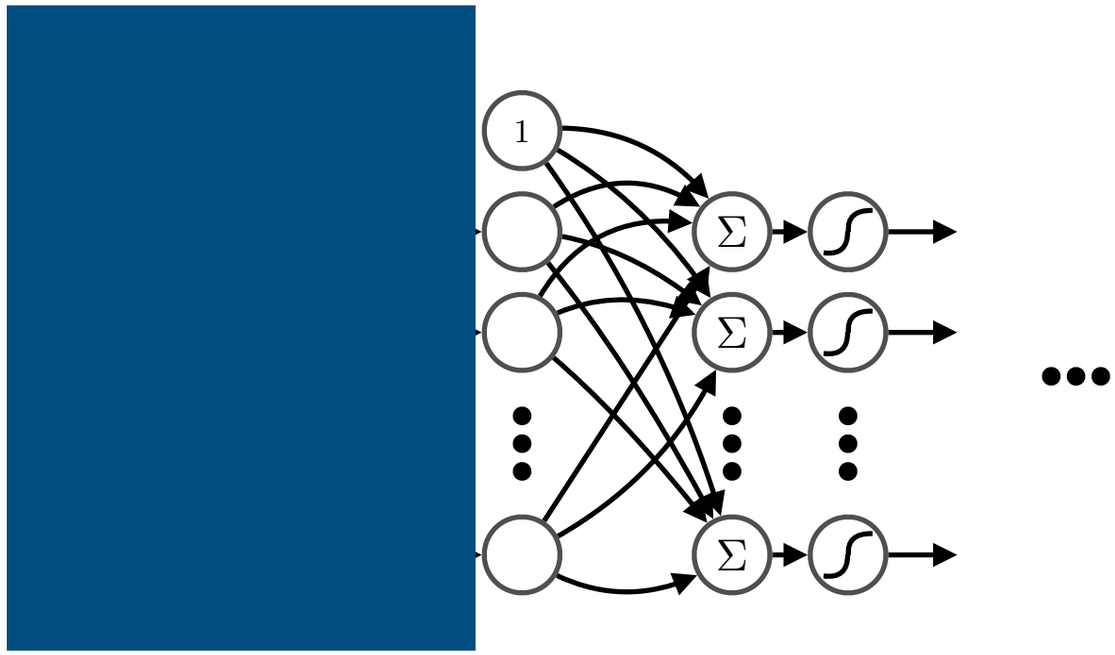


keeps internal activations in similar range, speeding up training

Batch Normalization, Szegedy & Ioffe, 2015

Why Does it Work?

covariate shift



input to the next layer changes, making it difficult to learn

→ *internal covariate shift*

histogram of unit activations