Deep Learning (BEV033DLE)
Lecture 4. SGD

Alexander Shekhovtsov

Czech Technical University in Prague

✦ Definitions and Main Properties

- Gradient Descent vs SGD
- Perceptron as SGD
- Understanding Convergence
- Variance Reduction: Running averages, Momentum
- Implicit regularization

Stochastic Gradient Descent

- Gradient Descent:
  - $g_t = \nabla_\theta L(\theta_t)$
  - $\theta_{t+1} = \theta_t - \alpha_t g_t$

- SGD:
  - Noisy gradient $\tilde{g}_t$
  - $\mathbb{E}[\tilde{g}_t] = g_t$
  - $\theta_{t+1} = \theta_t - \alpha_t \tilde{g}_t$

$L(\theta)$

Empirical Loss Function

- Problem Setup:
  - Predictor: $f(x;\theta)$, θ — vector of all parameters
  - $l(y, f(x;\theta))$ — loss of making prediction $f(x)$ when the true state is y
  - Expected loss: $\mathbb{E}[l(y, f(x;\theta))]$, $(x, y) \sim p^*$ – nature
  - Training set: $\mathcal{T} = (x_i, y_i)_{i=1}^n$ i.i.d.
  - Empirical loss: $L = \frac{1}{n}\sum_i l(y_i, f(x_i;\theta)) =: \frac{1}{n}\sum_i l_i(\theta)$
  - Learning problem: $\min\limits_{\theta} L(\theta)$

- Examples
  - Regression in $\mathbb{R}^m$:

    $f(x;\theta) \in \mathbb{R}^m$ – predicted values

    Squared error loss: $l_i = \|y_i - f(x_i;\theta)\|^2$
  - Classification with K classes:

    $f(x) \in \mathbb{R}^K$ – scores

    Predictive probabilities $p(y = k|x) = \mathrm{softmax}(f(x;\theta))_k$

    NLL loss: $l_i(\theta) = -(\log \mathrm{softmax}(f(x_i;\theta)))_{y_i}$

SGD for Empirical Loss

- Gradient Descent (**GD**):
  - Gradient at current point θ_t: $g_t = \nabla L(\theta_t) = \frac{1}{n}\sum_i \nabla l_i(\theta_t)$
  - Make a small step in the steepest descent direction of L:
  - $\theta_{t+1} = \theta_t - \alpha_t g_t$
  - If the dataset is very large, lots of computation to make a small step

- Stochastic Gradient Descent (**SGD**):
  - Pick M data points $I = \{i_1, \dots i_M\}$ at random
  - Estimate gradient as $\tilde{g}_t = \frac{1}{M}\sum_{i\in I} \nabla l_i(\theta_t)$
  - $\theta_{t+1} = \theta_t - \alpha_t \tilde{g}_t$
  - $\{(x_i, y_i) \,|\, i \in I\}$ is called a **(mini)-batch**

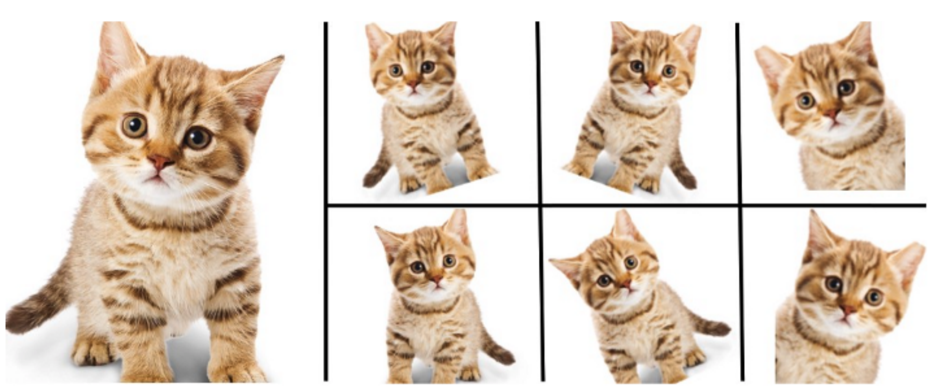
- "Noisy" gradient $\tilde{g}_t$:
  - $\mathbb{E}[\tilde{g}_t] = g_t$
  - $\mathbb{V}[\tilde{g}_t] = \frac{1}{M}\mathbb{V}[\tilde{g}^1_t]$, where $\tilde{g}^1$ is stochastic gradient with 1 sample
  - Diminishing gain in accuracy with larger batch size M
  - In the beginning a small subset of data suffices for a good direction

SGD for Generator

- Problem Setup:
  - Loss: $L(\theta) = \mathbb{E}_{(x,y)\sim p^*}[l(y, f(x;\theta))] + R(\theta)$
  - Training set is given as a generator p^*
    (fixed training set is a special case)
  - $R(\theta)$ is a regularizer, not dependent on the data
- SGD:
  - Draw a batch of data $(x_i, y_i)_{i=1}^M$ i.i.d. from p^*
  - $\tilde{g} = \frac{1}{M}\sum_i \nabla l(y_i, f(x_i, \theta)) + \nabla R(\theta)$

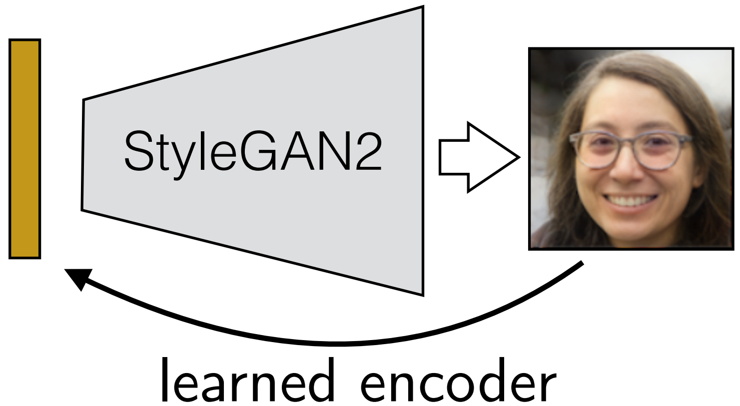
Why a generator?

Randomized data augmentation



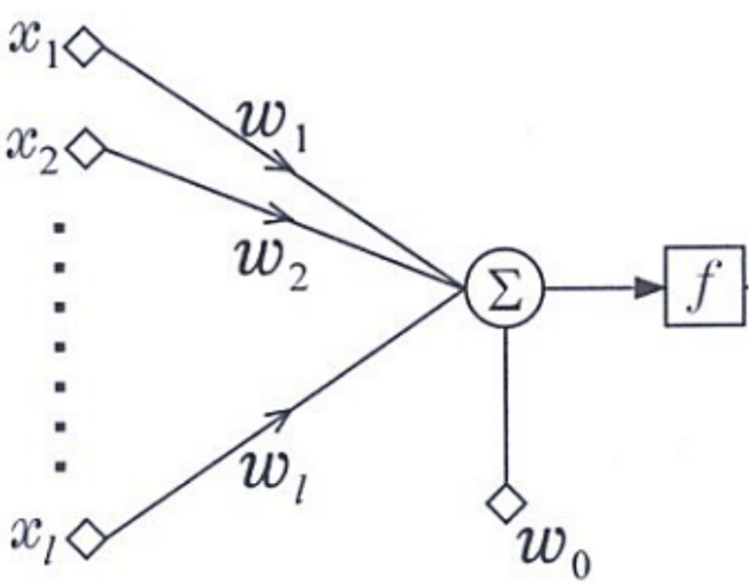
Simulation

Learning from a generative model

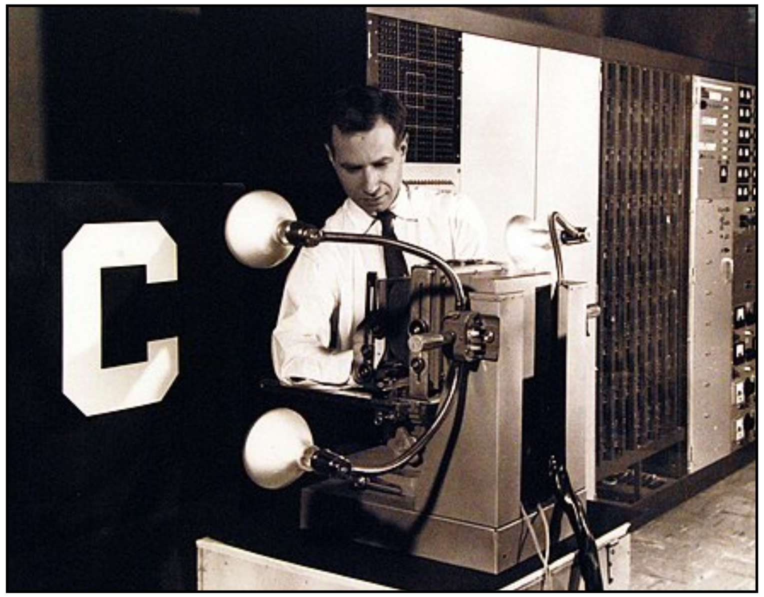


Perceptron

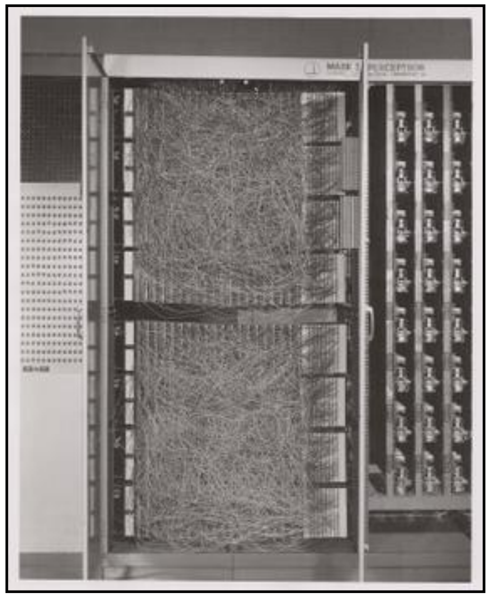
- Single Layer Perceptron (McCulloch-Pitts neuron 1943):



Frank Rosenblatt



Mark I Perceptron, 1958



- Perceptron Algorithm:
  - Training data (x_i, y_i), $y_i \in \{-1, 1\}$
  - If x_i is classified incorrectly by w_t:

$$w_{t+1} = w_t + y_i x_i$$

Exercise ($\star$): instance of SGD

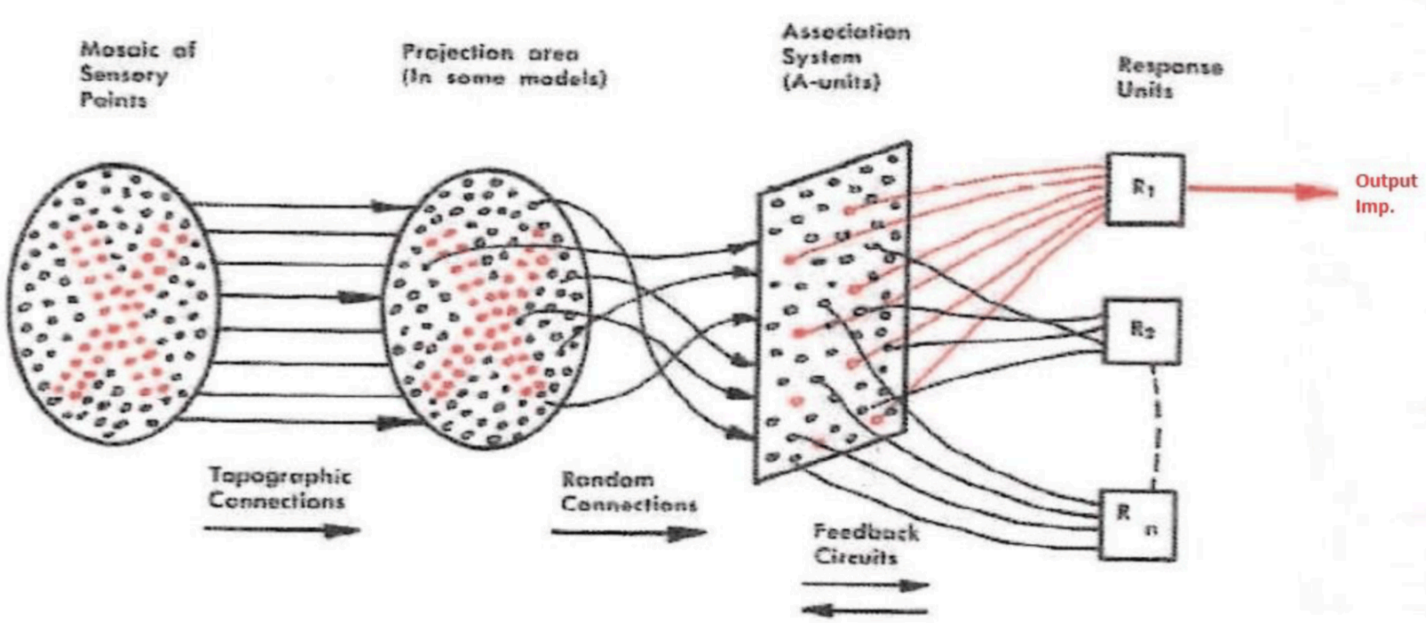


FIG. 2 — Organization of a perceptron.

NewYork Times: "the embryo of an electronic computer that we expect will be able to walk, talk, see, write, reproduce itself and be conscious of its existence"

Understanding Convergence

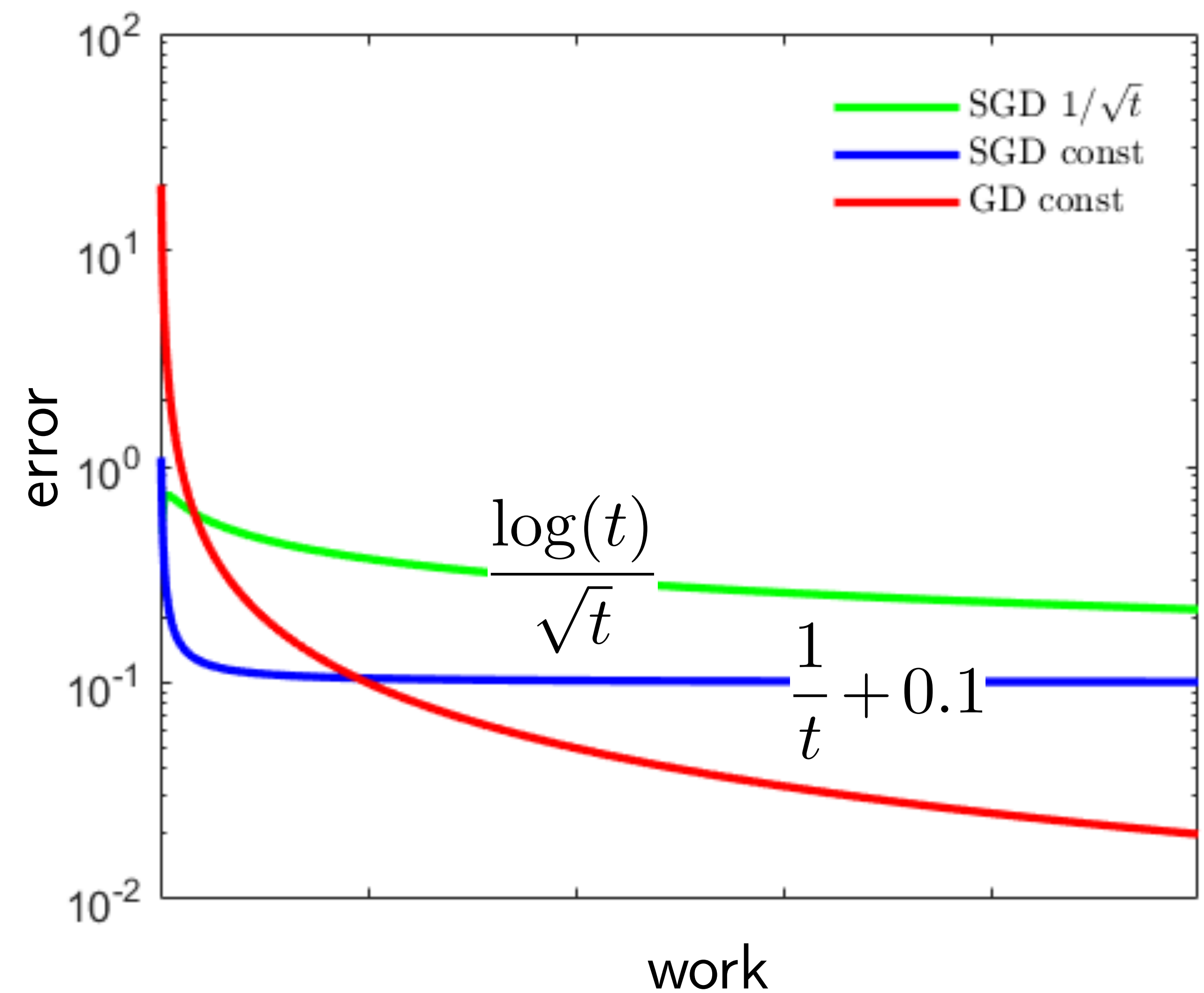
- Iteration cost:
  - GD: $O(n)$ – full data
  - SGD: $O(M)$ – mini-batch
- Guarantees on convergence rate **depend on assumptions**. Setup closest to NNs:
  - $L(\theta)$ is bounded from below
  - $\nabla L(\theta)$ is Lipschitz continuous with constant ρ
  - Bounded variance: $\mathbb{E}\|\tilde{g}(\theta) - \nabla\mathcal{L}(\theta)\|^2 \leq \sigma^2$

    (or a slightly stronger but simpler condition $\mathbb{E}\|\tilde{g}(\theta)\|^2 \leq \sigma^2$)
- Convergence rates:
  - Error at iteration t: best over iterations expected gradient norm, $\min_{k=1\ldots t-1}\{\|\mathbb{E}[\nabla L(\theta_k)]\|\}$
  - GD with step size $\alpha_t = \alpha$

    Error: $O(\frac{1}{t})$
  - SGD with step size $\alpha_t = \alpha/\sqrt{t}$

    Error: $O(\frac{\log(t)}{\sqrt{t}})$
  - SGD with step size $\alpha_t = \alpha$

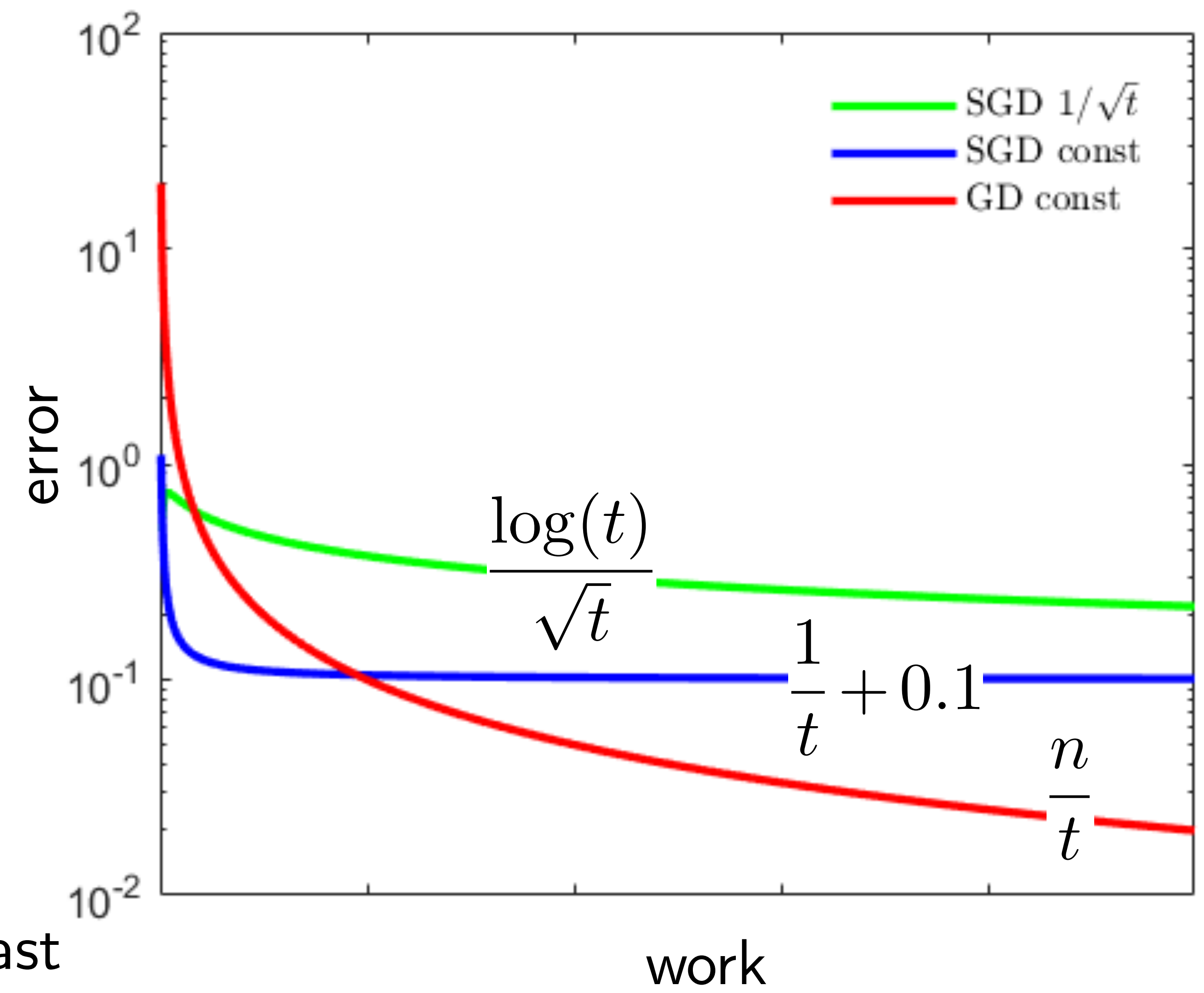
    Error: $O(\frac{1}{t}) + O(\alpha\rho\sigma^2)$



[Mark Smidt CPSC 540 Lecture 11]

Understanding Convergence

- Convergence rates:
  - GD with step size $\alpha_t = \alpha$
    Error: $O(\frac{1}{t})$
  - SGD with step size $\alpha_t = \alpha/\sqrt{t}$
    Error: $O(\frac{\log(t)}{\sqrt{t}})$
  - SGD with step size $\alpha_t = \alpha$
    Error: $O(\frac{1}{t}) + O(\alpha\rho\sigma^2)$



- Insights:
  - SGD wins when there is a lot of data
  - Convergence with a constant step size is fast but to within a "region" around optimum

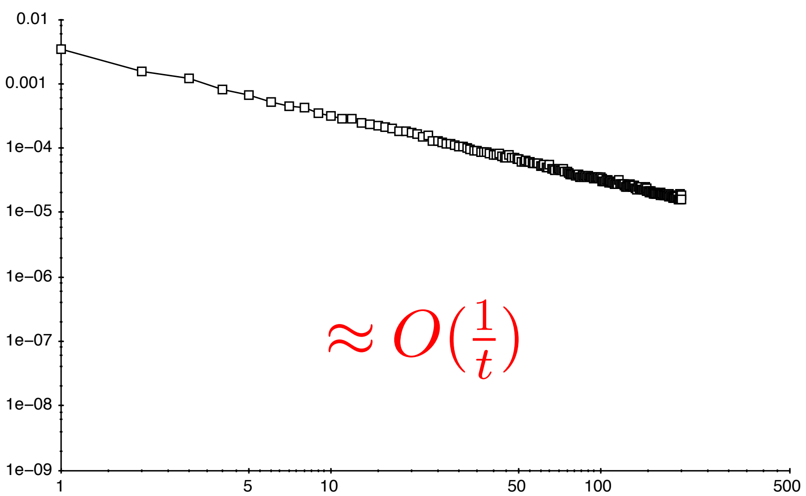
- Remarks:
  - To have guarantees need to use conservative estimates with very small step sizes, etc.
  - Different other setups possible: convex / strongly convex, smooth/non-smooth
  - The rate is often faster in practice, but the general picture stays

How to Draw Data Points?

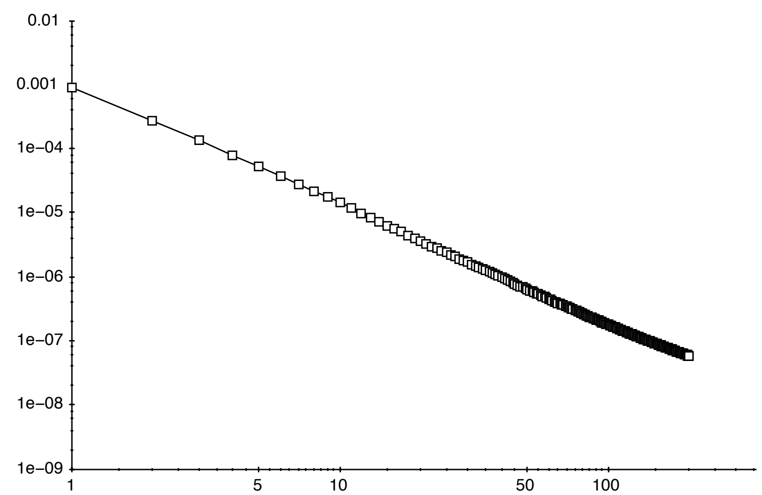
- How should we draw data points for SGD:
  - every time select randomly with replacement
  - shuffle the data once
  - shuffle at each epoch but draw without replacement
- Empirical evidence:

Bottou (2009): "Curiously Fast Convergence of some Stochastic Gradient Descent Algorithms"

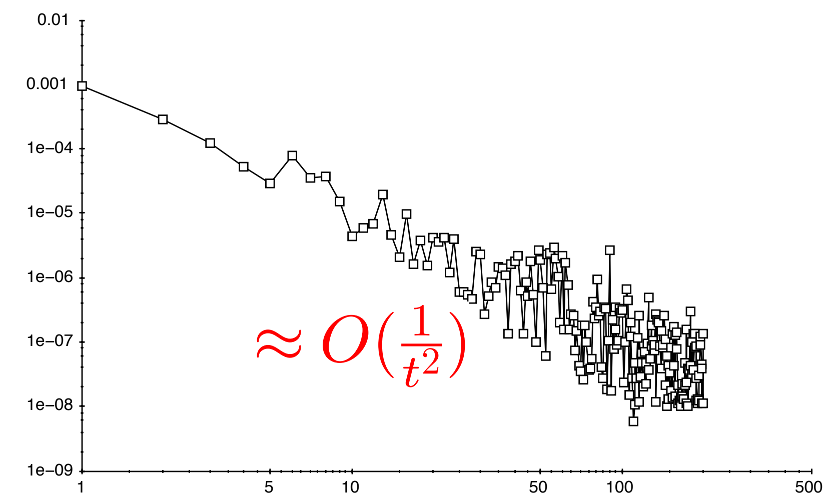
logistic regression $d = 47{,}152$, $n = 781{,}256$



Random selection: slope=−1.0003



Cycling the same random shuffle: slope=−1.8393



Random shuffle at each epoch: slope=−2.0103

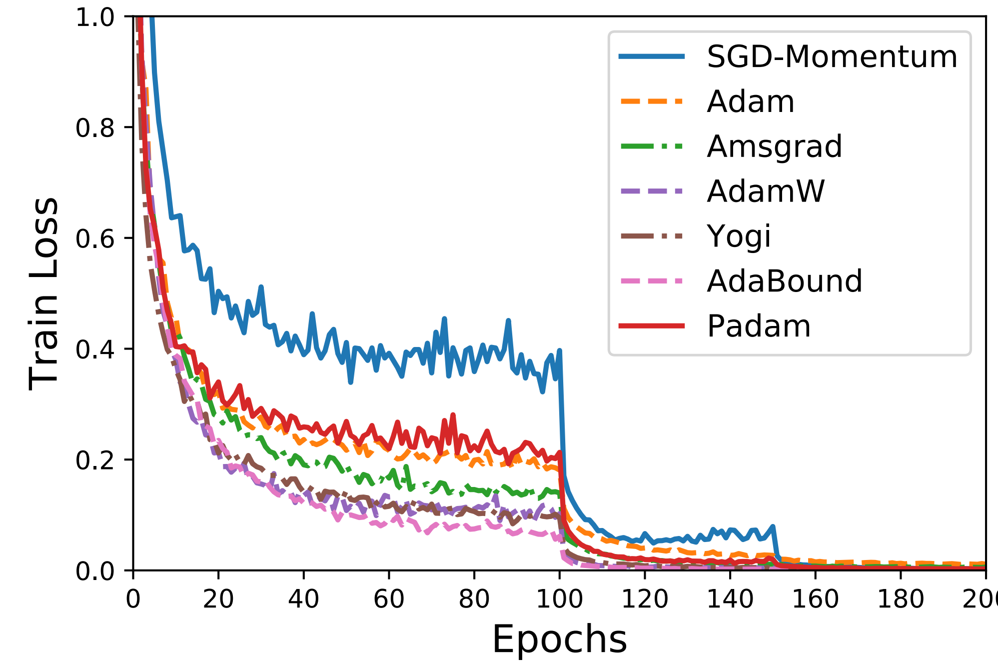
- A simple consideration:

Drawing n times with replacement from the dataset of size n some points may not be selected – efficiently using a subset of data per epoch.

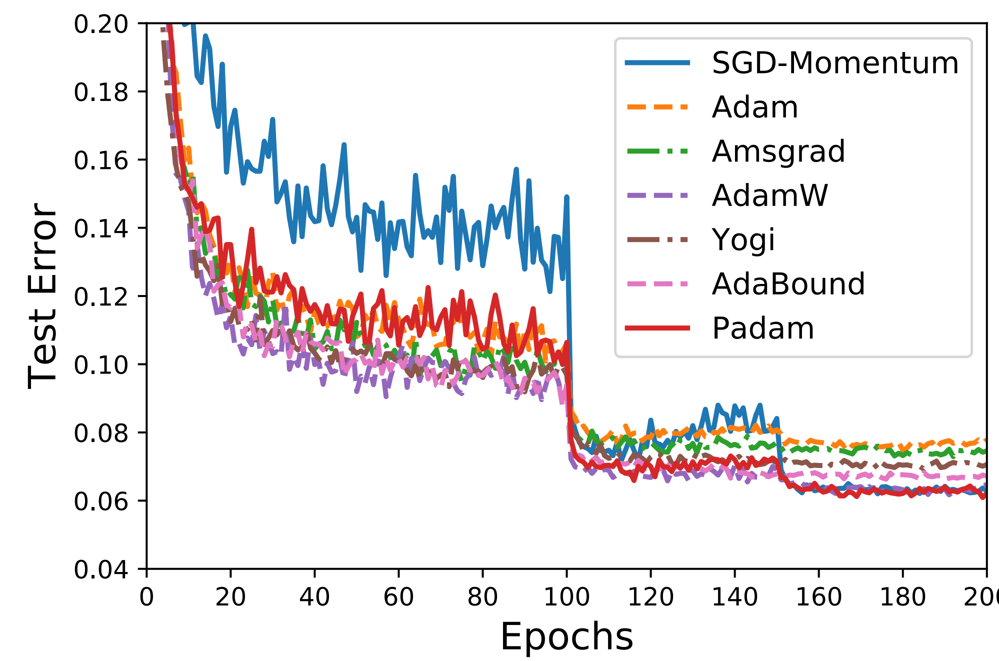
Learning Rate Schedule

- (Basic) common practice: decrease learning rate in steps
  - Example: start with $\alpha = 0.1$ then decrease by factor of 10 at epochs 100 and 150

- Comments
  - Consistent with the idea of fast convergence to a region
  - After the sep size decrease, "1/n" rate replays
  - Many other empirically proposed schedules



(a) Train Loss for VGGNet

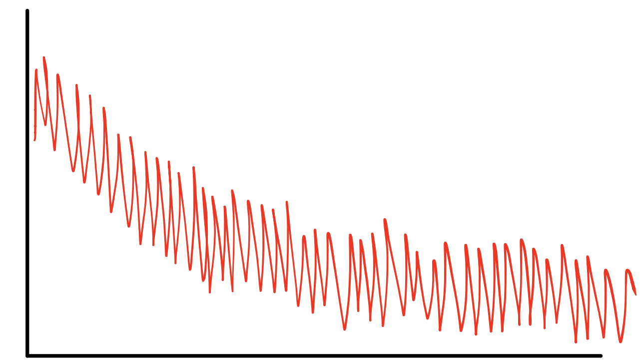


(d) Test Error for VGGNet

Courtesy: [Chen et al. "Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks"]

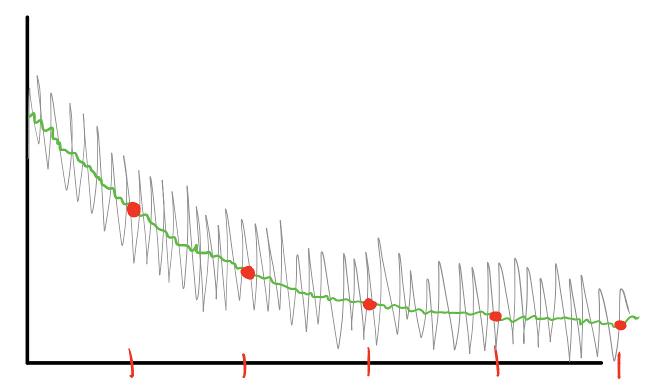
How to Measure the Progress?

✦ Batch Estimate



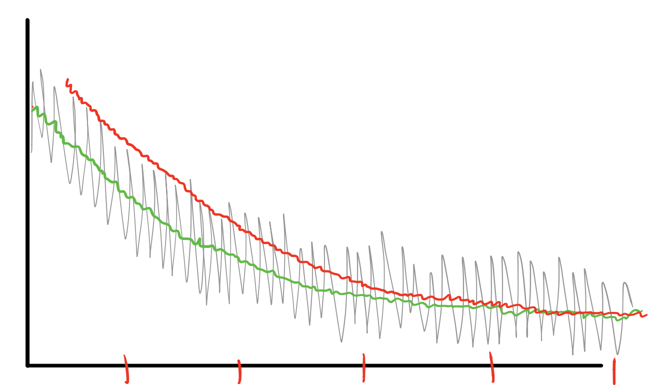
- Batch mean: $\tilde{L} = \frac{1}{M}\sum_{i \in I} l_i$
- Unbiased, but high variance

✦ Training data mean



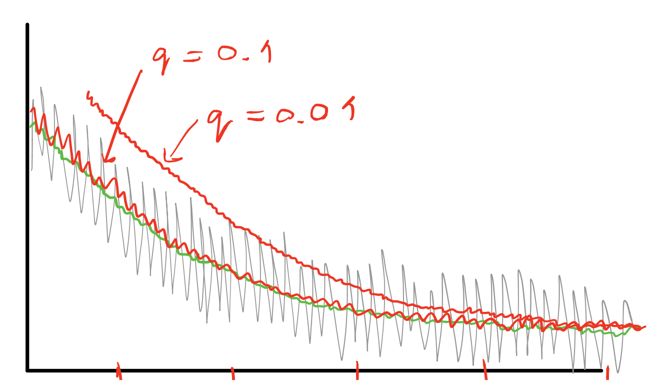
- $L = \frac{1}{n}\sum_{i=1}^{n} l_i$
- Unbiased, zero variance, but may be too costly

✦ Average using all last known loss values



- $\hat{L} := \frac{1}{n}\Big(\sum_{i \in I} l_i^{\text{new}} + \sum_{i \notin I} l_i^{\text{old}}\Big)$
- Low variance, hysteresis 1 epoch
- Need to remember losses for full dataset

✦ Running Averaging

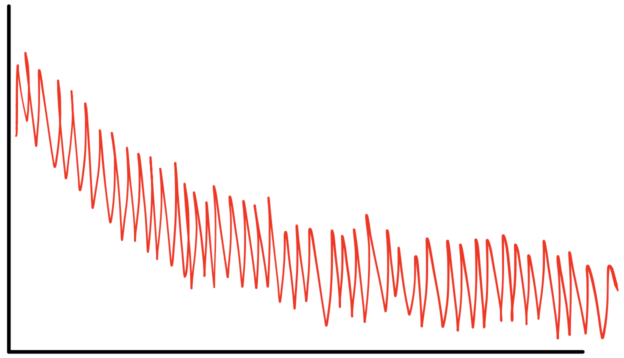


- $\hat{L}^{t+1} := (1-q)\hat{L}^{t} + q\tilde{L}$
- Variance-hysteresis tradeoff controlled by q
- Need to remember only the running average loss

Same Applied to Gradient — Variance Reduction

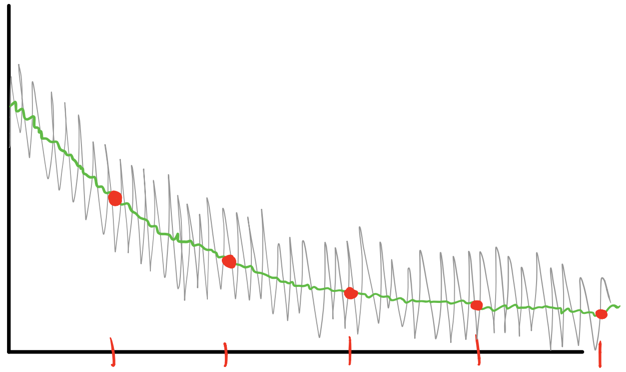
✦ SGD

- Batch mean: $\tilde{g} = \frac{1}{M}\sum_{i \in I} \nabla l_i$
- Need a small step size



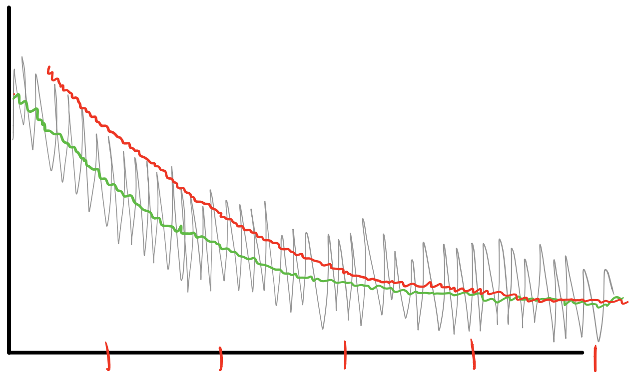
✦ GD

- Full gradient: $g = \frac{1}{n}\sum_{i=1}^{n} \nabla l_i$
- Too costly



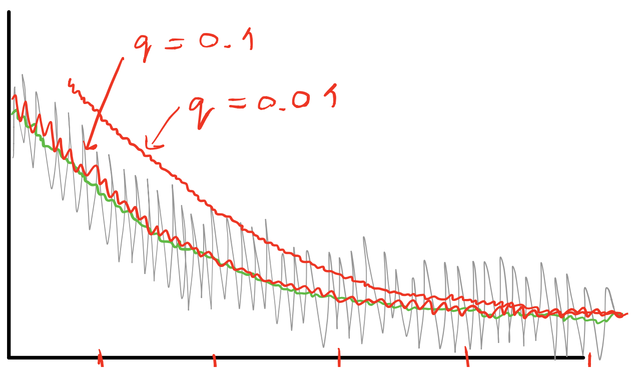
✦ Stochastic Average Gradient (**SAG**)

- $\tilde{g} := \frac{1}{n}\Big(\sum_{i \in I} (\nabla l_i)^{\text{new}} + \sum_{i \notin I} (\nabla l_i)^{\text{old}}\Big)$
- Improved convergence rates (convex analysis)
- Need to remember gradients



✦ **SGD** with filtered gradient (SGD with **momentum)**

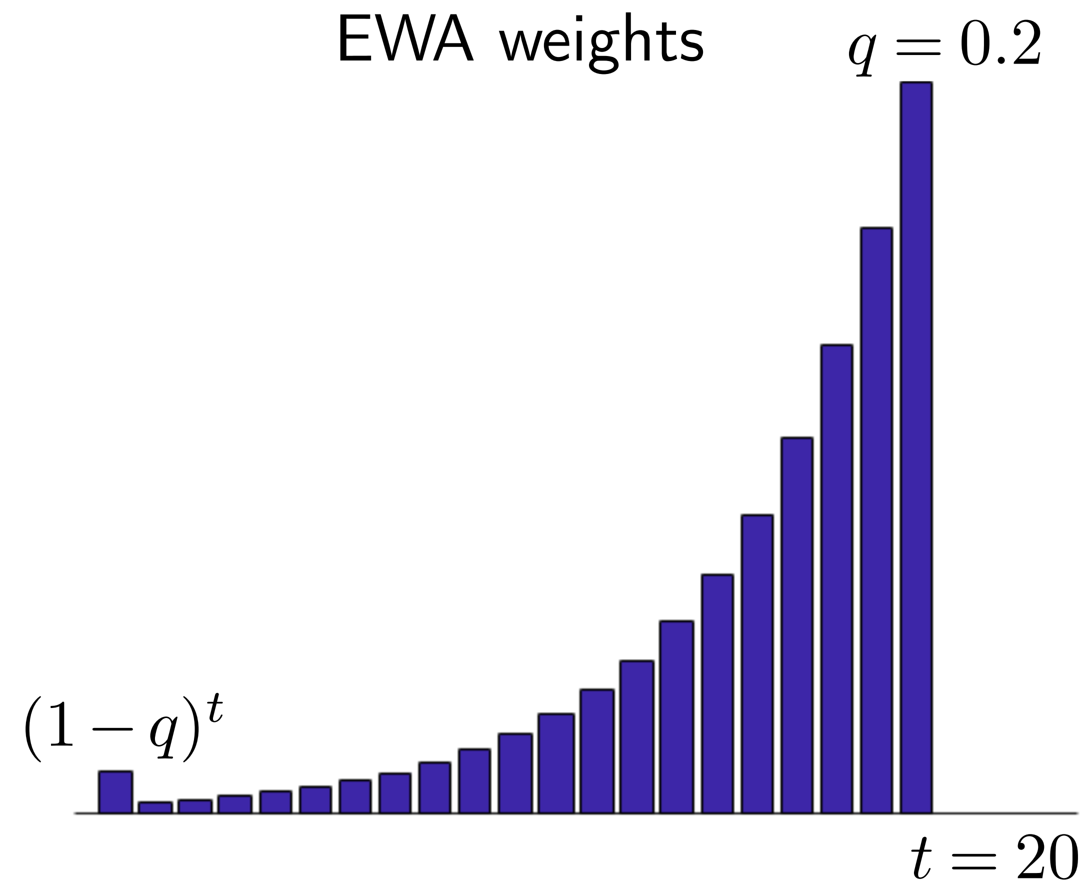
- $g := (1-q)g + q\tilde{g}$
- Variance-hysteresis tradeoff controlled by q
- Remember only the running average gradient

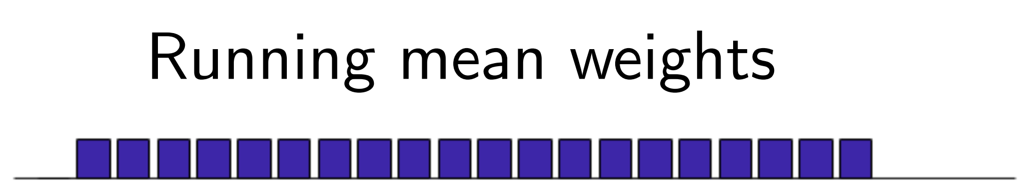


First Order Filter

- General setup:
  - $X_k,\ k = 1, \dots, t$ – independent random variables
  - $q_t \in (0, 1]$
  - First order filter: $\mu_t = (1 - q_t)\mu_{t-1} + q_t X_t$
- Exponentially Weighted Average (**EWA**):
  - Constant $q_t = q$
  - $\mu_1 = (1 - q)\mu_0 + qX_1$
  - $\mu_2 = (1 - q)^2\mu_0 + (1 - q)qX_1 + qX_2$
  - $\dots$
  - $\mu_t = (1 - q)^t\mu_0 + \sum_{1 \le k \le t} (1 - q)^{t-k} qX_k$

    $= w_0\mu_0 + \sum_{1 \le k \le t} w_k X_k$





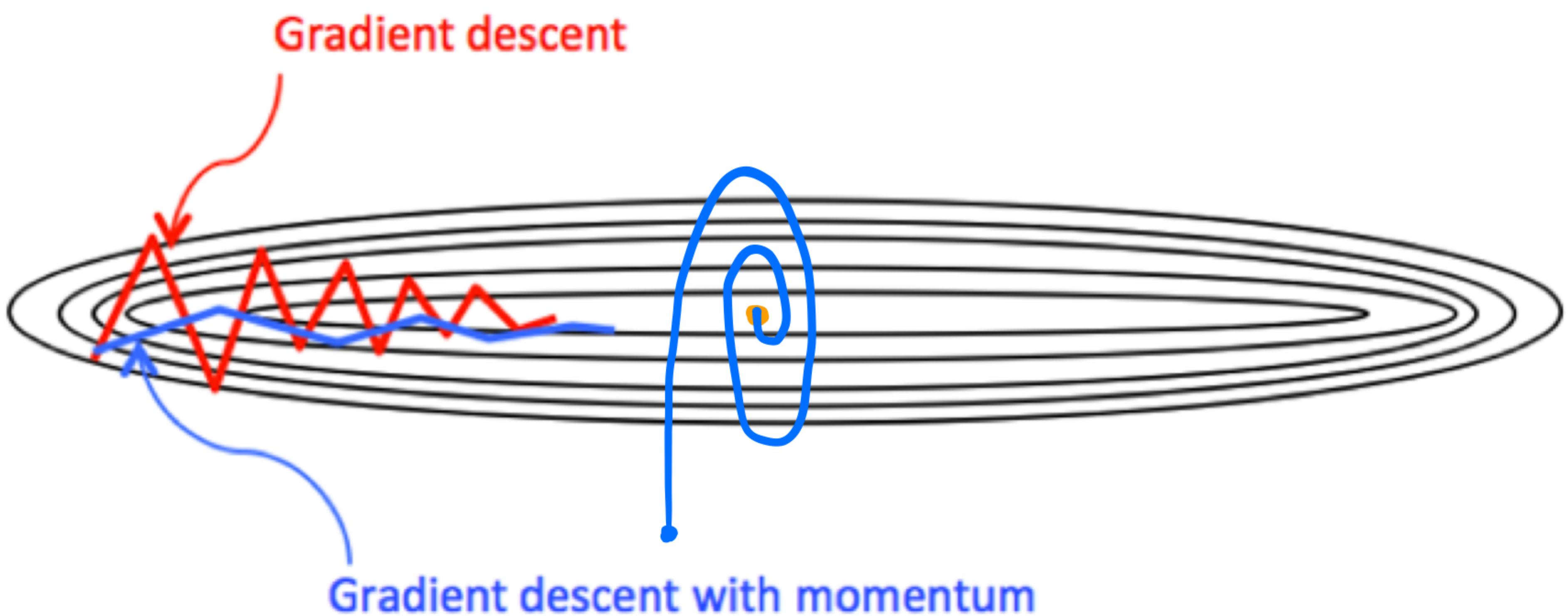
- **Running mean**:
  - $q_t = \frac{1}{t}$
  - $\mu_1 = 0\mu_0 + X_1$
  - $\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}X_t$
  - $\mu_{t+1} = \frac{t}{t+1}\mu_t + \frac{1}{t+1}X_{t+1} = \frac{t-1}{t+1}\mu_{t-1} + \frac{1}{t+1}\left(X_t + X_{t+1}\right)$

- Averaging over past gradients reduces variance, but introduces a hysteresis bias

Hysteresis Bias

- With variance sufficiently low → GD with momentum. Consider $\tilde{g}$ is noise-free

  Equivalent form of SGD with EWA gradient ($\star$):
  - Velocity: $v_t := \mu v_{t-1} + \tilde{g}$
  - Step: $\theta_t = \theta_{t-1} - \varepsilon v_t$



- The **"heavy ball"** method
  - Friction ($\mu < 1$) and slope forces build up velocity
  - Cancels "noise" in the incorrect prediction of the function change, helpful to overcome plateaus
  - The inertia may lead to oscillatory behavior (not good)

"Nesterov" Momentum

- Common Momentum
  - Velocity: $v_{t+1} = \mu v_t + \tilde{g}(x_t)$
  - Step: $x_{t+1} = x_t - \varepsilon v_{t+1}$

The step consists of momentum and current gradient

The momentum part of the step is **known in advance**

Can make it before computing the gradient:

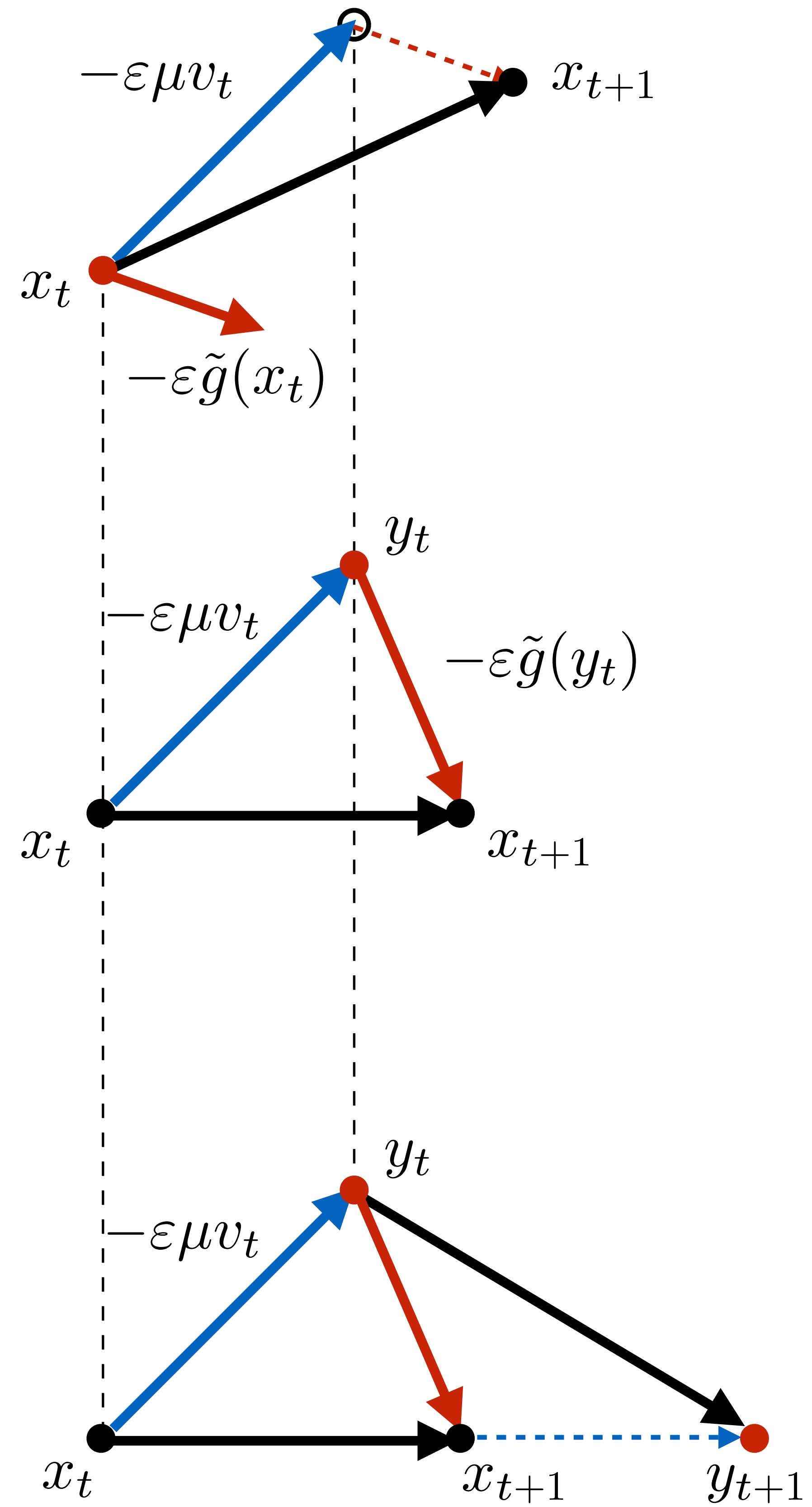
- Nesterov Momentum
  - Leading sequence: $y_t = x_t - \varepsilon \mu v_t$
  - Velocity: $v_{t+1} = \mu v_t + \tilde{g}(y_t)$
  - Step: $x_{t+1} = y_t - \varepsilon \tilde{g}(y_t)$

Takes advantage of the known part of the step

Less overshooting

- ($\star$) Can express as steps on the leading sequence alone:
  - Velocity: $v_{t+1} = \mu v_t + \tilde{g}(y_t)$
  - Step: $y_{t+1} = y_t - \varepsilon\big(\tilde{g}(y_t) + \mu v_{t+1}\big)$

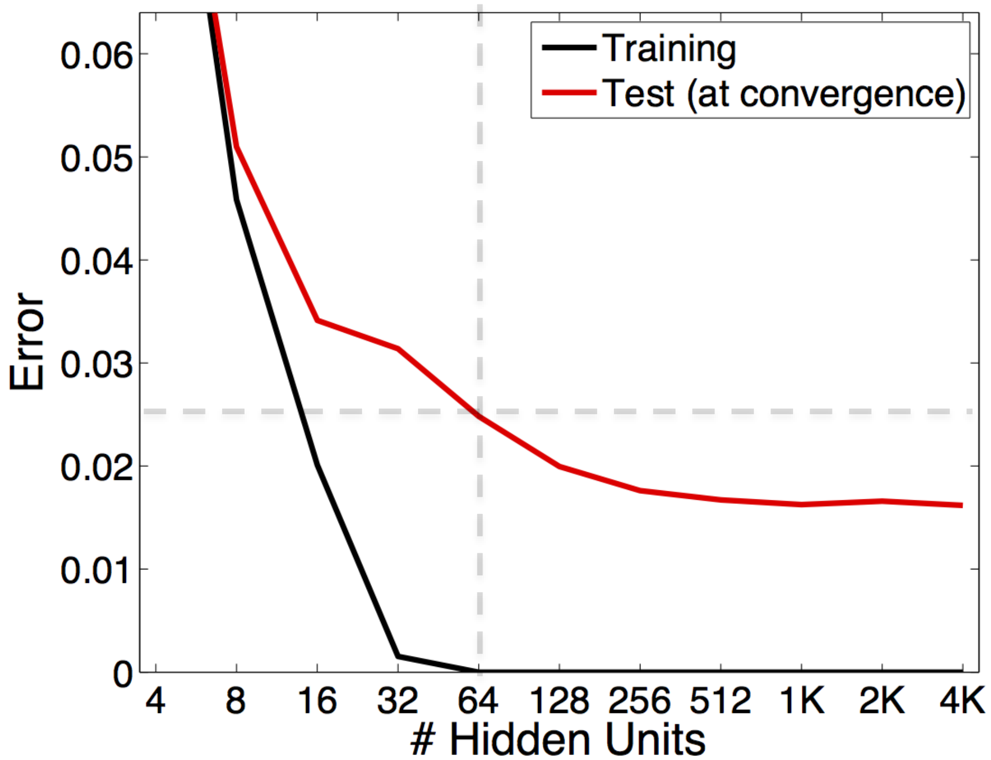
The two sequences eventually converge



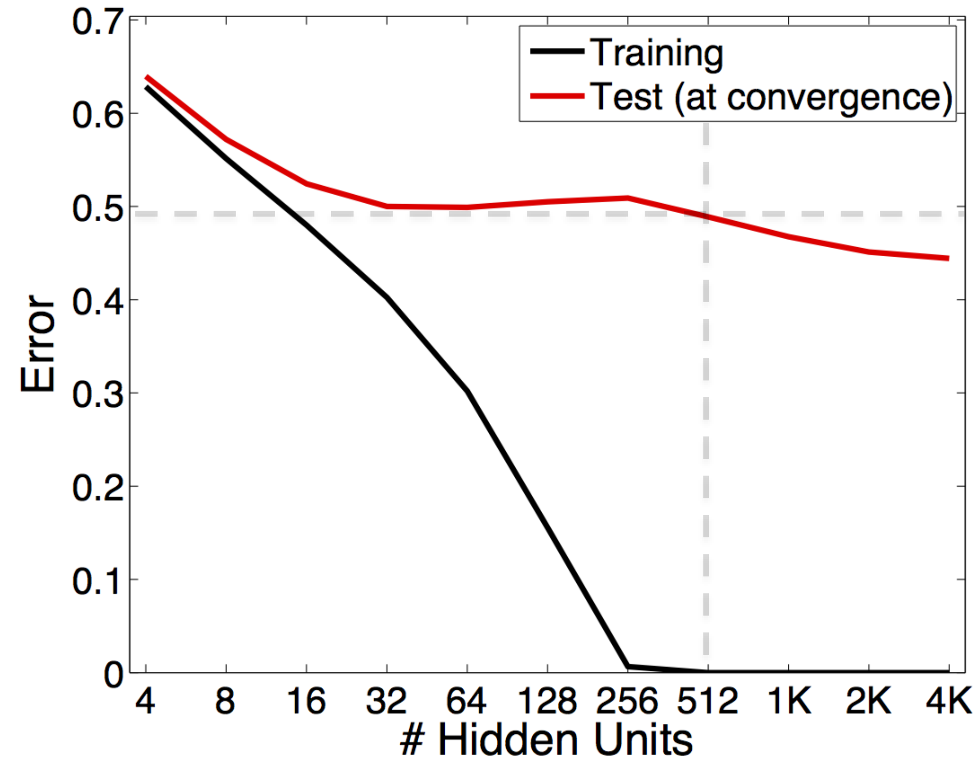
Implicit Regularization

Implicit Regularization

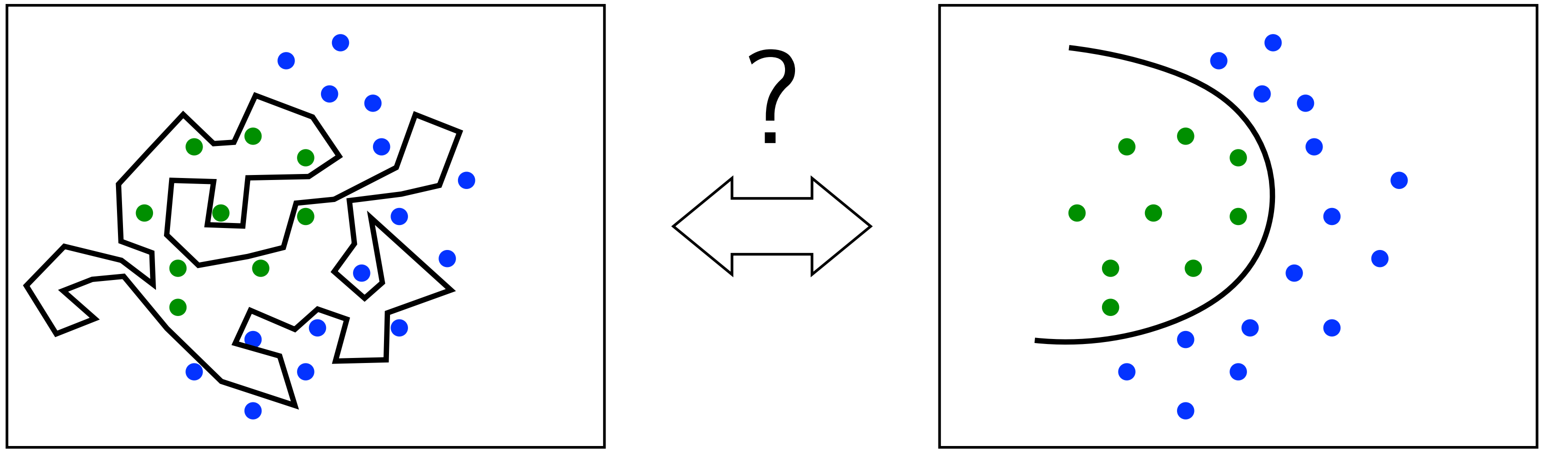
MNIST



CIFAR-10



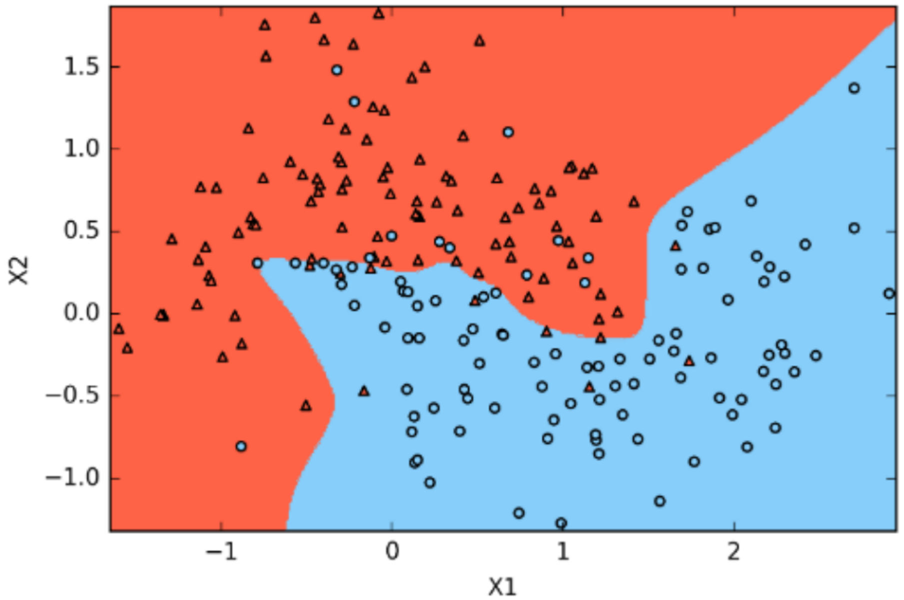
- We increase the network capacity but generalization improves, why?
  - There exist global minima that generalize poorly
  - SGD somehow finds a good global minimum



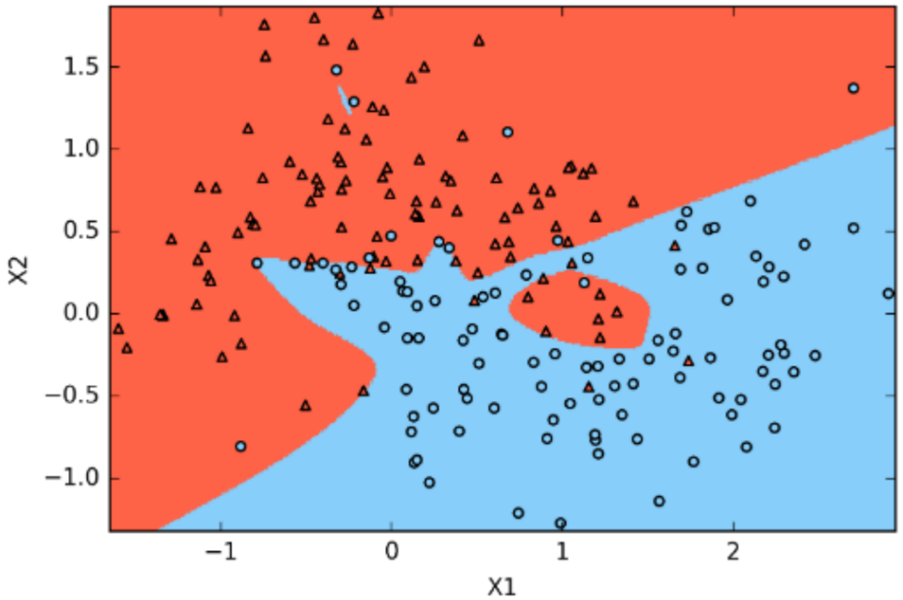
Smaller Batch Size -> More Regularization

- Typically choose batch size to fully utilize parallel throughput (in GPUs means ~10^4 independent arithmetic computations in parallel)
- Limited by memory
- Smaller batch -> noisier gradient -> implicit regularization

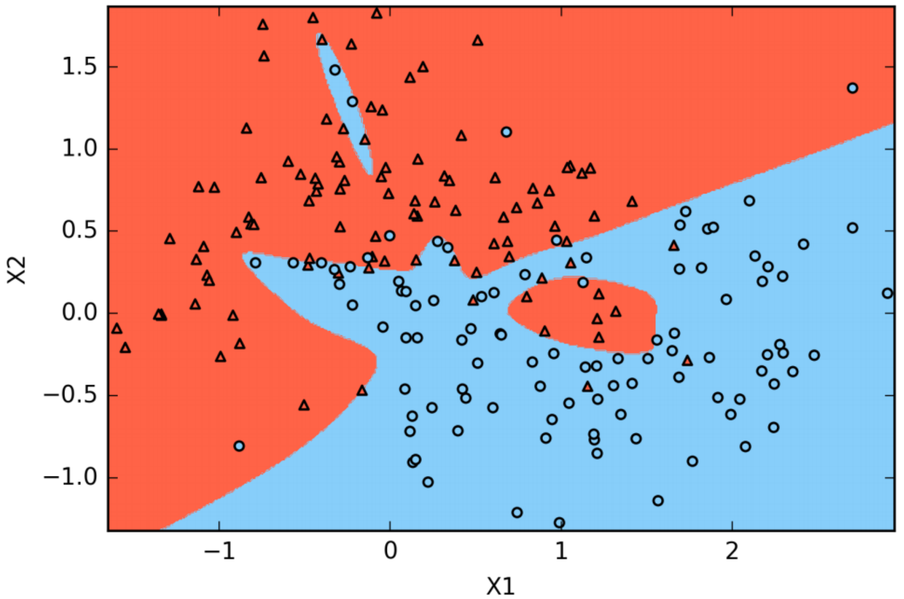
Synthetic data



Decision boundary of batch size 1

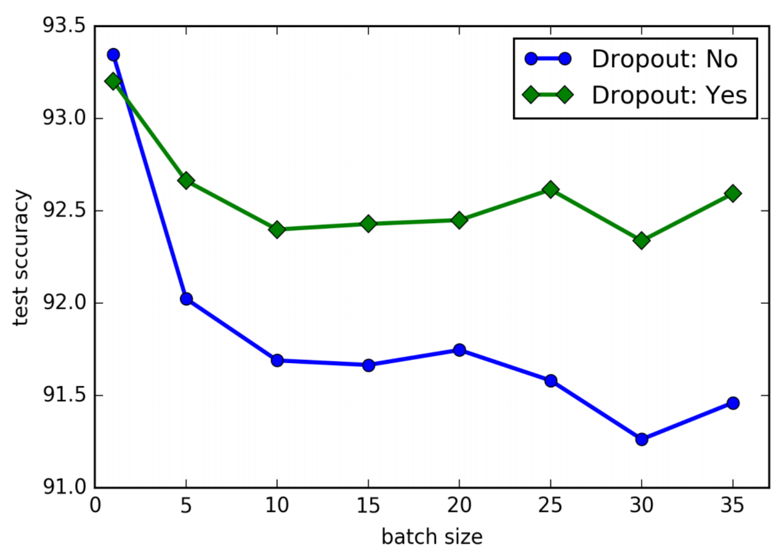


Decision boundary of batch size 5



Decision boundary of batch size 30

NLP data



Lei et al. (2018) "Implicit Regularization of Stochastic Gradient Descent in Natural Language Processing: Observations and Implications"

Implicit Regularization

- Logistic (or multinomial) regression:

$$\operatorname{argmin}_w \mathcal{L}(w) + \lambda \|w\|_p^p \xrightarrow{\lambda \to 0} w \to \text{max margin w.r.t. } \|\cdot\|_p \qquad [1]$$

GD for $\min_w \mathcal{L}(w)$ iteration can be written as:

$$w^{t+1} = w^t + \underset{\Delta w}{\operatorname{argmin}} \left(\langle \Delta w, \nabla \mathcal{L}(w^t) \rangle + \frac{1}{2\varepsilon} \|\Delta w\|_2^2 \right)$$

$$t \to \infty \qquad \frac{w^t}{\|w^t\|} \to \text{max margin w.r.t. } \|\cdot\|_2 \qquad [2]$$

- Linear model with any loss:

$$\min_w \mathcal{L}(w) := \sum_{n=1}^{N} \ell(\langle w, x_n \rangle, y_n). \qquad \mathcal{W} \text{ – set of optimal solutions}$$

SGD iteration, generalizing the norm:

$$w^{t+1} = w^t + \underset{\Delta w}{\operatorname{argmin}} \left(\langle \Delta w, \tilde{\nabla} \mathcal{L}(w^t) \rangle + \frac{1}{2\varepsilon} \|\Delta w\|_p^p \right)$$

$$t \to \infty \qquad w^t \to \text{point in } \mathcal{W}\text{, nearest to } w^0 \text{ in } \|\cdot\|_p \qquad [3]$$

- SGD induces implicit p-norm regularization, helping to improve p-norm margin

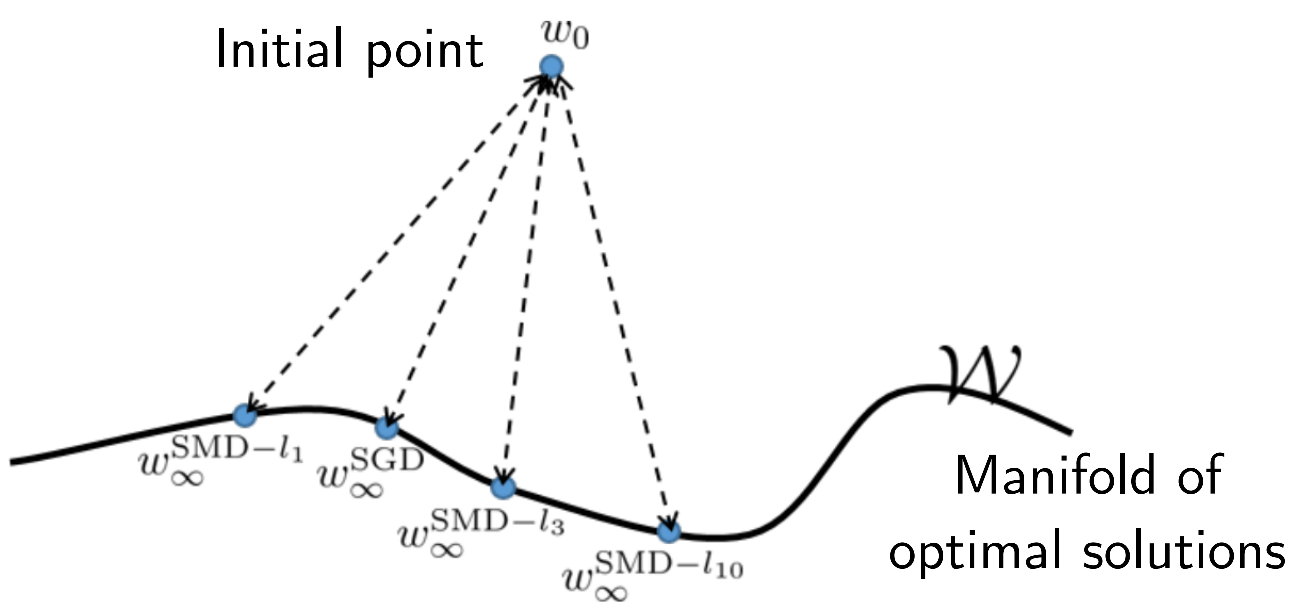
[1] Rosset et al. (2004) Margin Maximizing Loss Functions

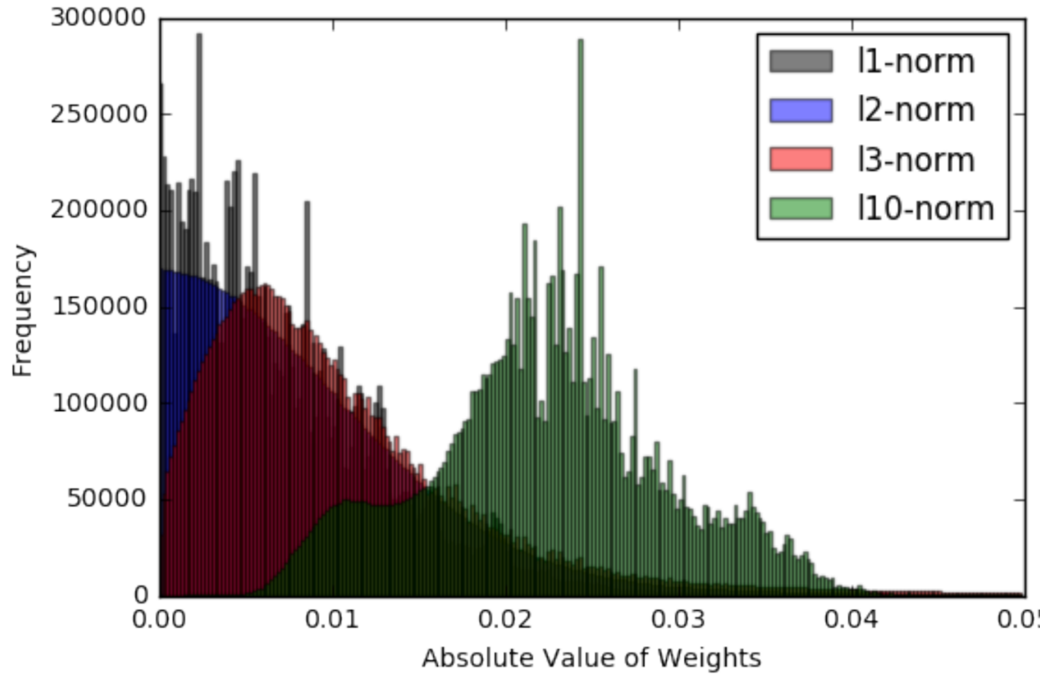
[2] Soudry et al. (2018) "The Implicit Bias of Gradient Descent on Separable Data"

[3] Gunasekar et al. (2018) "Characterizing Implicit Bias in Terms of Optimization Geometry"

Implicit Regularization by SGD / SMD

- Consider step proximal problem: $\min_x \langle \nabla f(x_0), x - x_0 \rangle + \lambda \|x - x_0\|_p^p$
  - i.e., p-norm stochastic mirror descent
- Using different p leads to solutions with different properties

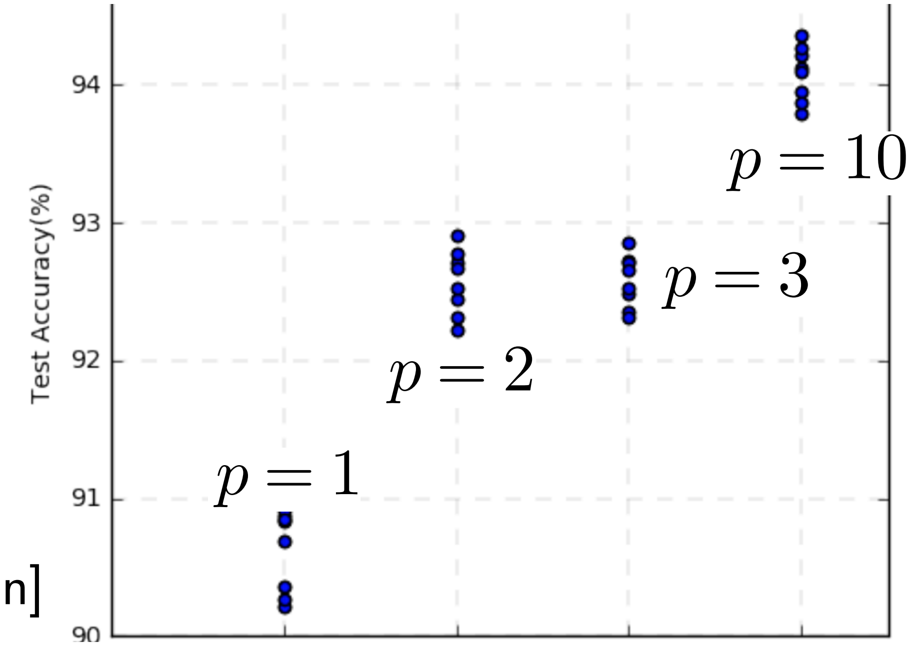




- Iterates tend to $\mathrm{argmin}_{w \in \mathcal{W}} \|w - w_0\|_p^p$, the closest point in the respective norm

	SMD 1-norm	SMD 2-norm (SGD)	SMD 3-norm	SMD 10-norm
1-norm BD	141	9.19×10^3	4.1×10^4	2.34×10^5
2-norm BD	3.15×10^3	562	1.24×10^3	6.89×10^3
3-norm BD	4.31×10^4	107	53.5	1.85×10^2
10-norm BD	6.83×10^{13}	972	7.91×10^{-5}	2.72×10^{-8}

- Different sparsity and generalization



[Azizan et al. (2019) Stochastic Mirror Descent on Overparameterized Nonlinear Models: Convergence, Implicit Regularization, and Generalization]

(*) EWA: How Much Variance Reduction?

- General setup
  - X_t – independent random variables
  - $q_t \in (0,1]$
  - Running mean: $\mu_t = (1-q_t)\mu_{t-1} + q_t X_t$ is a r.v.
- Expectation:
  - $\mathbb{E}[\mu_t] = (1-q_t)\mathbb{E}[\mu_{t-1}] + q_t\mathbb{E}[X_t]$ – running average of expectations
  - $\mathbb{E}[\mu_t] = w_0\mathbb{E}[\mu_0] + \sum\limits_{k=1}^{t} w_k\mathbb{E}[X_k]$
  - In context of SGD with learning rate $\varepsilon \to 0$, all $E[X_k]$ are the same and μ_t is an unbiased estimate
- Variance:
  - $\mathbb{V}[\mu_t] = (1-q_t)^2\mathbb{V}[\mu_{t-1}] + q_t^2\mathbb{V}[X_t]$
  - $\mathbb{V}[\mu_t] = w_0^2\mathbb{V}_0 + \sum\limits_{k=1}^{t} w_k^2\mathbb{V}[X_k]$
  - Variance reduction of running mean: $\sum_{k=0}^{t} w_k^2 = \sum_{k=1}^{t}\frac{1}{t^2} = \frac{1}{t}$
  - Variance reduction of EWA: $\sum_{k=0}^{t} w_k^2 = \frac{q^2}{1-(1-q)^2}$ – in the limit of large t
  - ($\star$) Equivalent window size of EWA: $n = \frac{2}{q} - 1$. E.g. $q = 0.1 \leftrightarrow n = 19$

- Can use EWA with a decreasing q series for a progressive smoothing