

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

$$L(\theta)$$

◆ Gradient Descent:

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

◆ SGD:

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



◆ Problem Setup:

- Predictor: $f(x; \theta)$, θ – vector of all parameters θ
- Loss of making prediction $f(x)$ when the true state is y : $l(y, f(x; \theta))$
- Expected loss: $\mathbb{E}[l(y, f(x; \theta))]$, $(x, y) \sim \text{data}$
- 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: $\min_{\theta} L(\theta)$

◆ 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 $p(y = k|x) = \text{softmax}(f(x; \theta))_k$

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

◆ 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 :
- $\theta_{t+1} = \theta_t - \alpha_t g_t$
- Historically called “batch gradient descent”
- 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 $\tilde{g}_t = \frac{1}{M} \sum_{i \in I} \nabla l_i(\theta_t)$
- $\theta_{t+1} = \theta_t - \alpha_t \tilde{g}_t$
- $\{(x_i, y_i) \mid i \in I\}$ is called a **(mini)-batch**

◆ “Noisy” gradient \tilde{g}_t :

- $\mathbb{E}[\tilde{g}_t] = g_t$
- $\mathbb{V}[\tilde{g}_t] = \frac{1}{M} \mathbb{V}[\tilde{g}_t^1]$, where \tilde{g}_t^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

◆ Problem Setup:

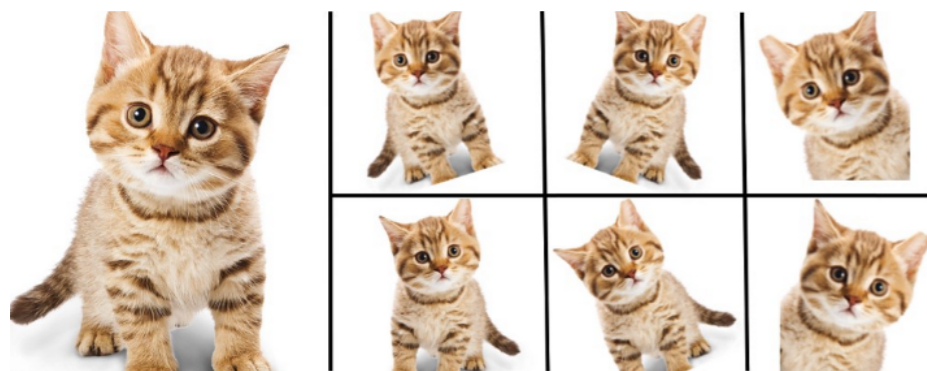
- Loss: $L(\theta) = \mathbb{E}_{(x,y) \sim p^*} [l(y, f(x; \theta))] + R(\theta)$
- Training set is given as a generator p^*
(fixed training set is a special case)
- $R(\theta)$ is a regularizer, not dependent on the data

◆ 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?

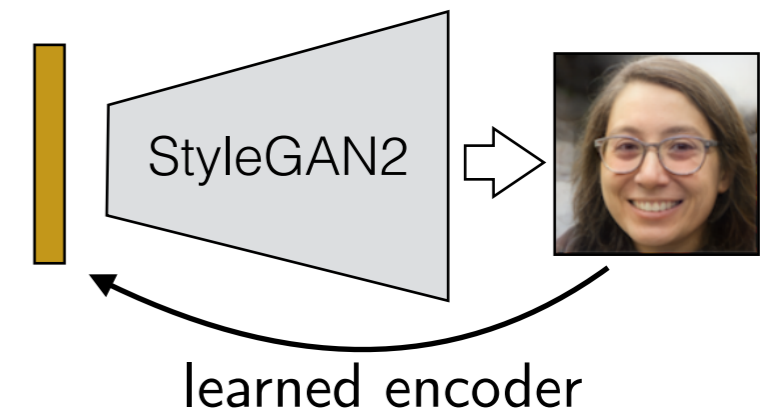
Randomized data augmentation



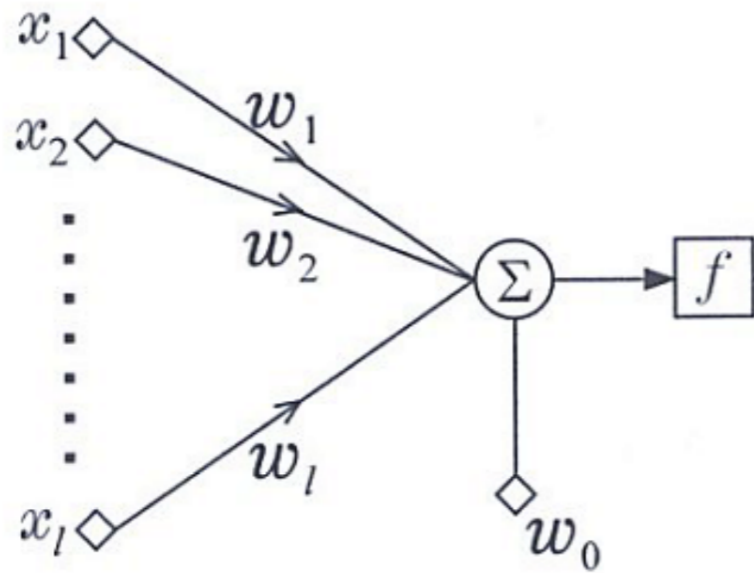
Simulation



Learning from a generative model



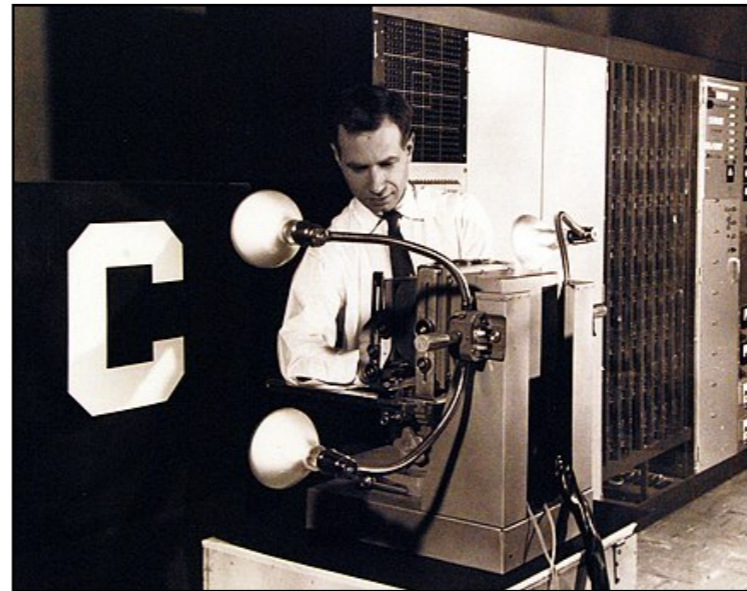
◆ Single Layer Perceptron (McCulloch-Pitts neuron):



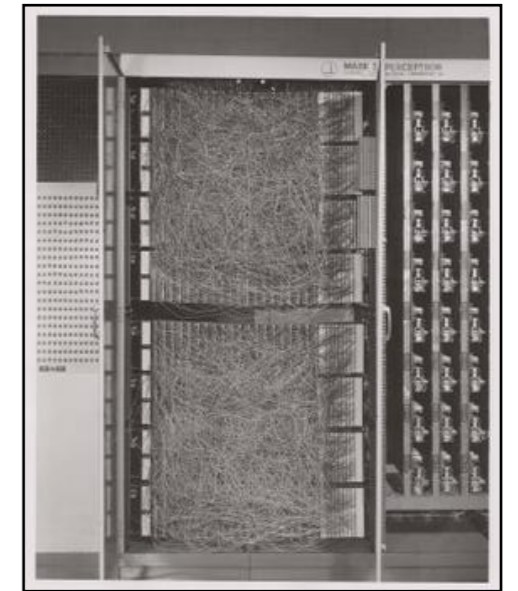
◆ Perceptron Algorithm as SGD:

- Two classes $y = \pm 1$
- Predictor: $f(x) = w^T x$, decide by sign
- Loss: $l(y, f(x)) = \max(-yw^T x, 0)$
- Draw a point (x, y) from the training data at random
- Stochastic gradient: $\tilde{g}_t = \begin{cases} -yx, & \text{if classified incorrectly} \\ 0, & \text{otherwise} \end{cases}$
- Make a step: $w_{t+1} = w_t + yx$
- No need of step size thanks to scale invariance

Frank Rosenblatt



Mark I Perceptron, 1958



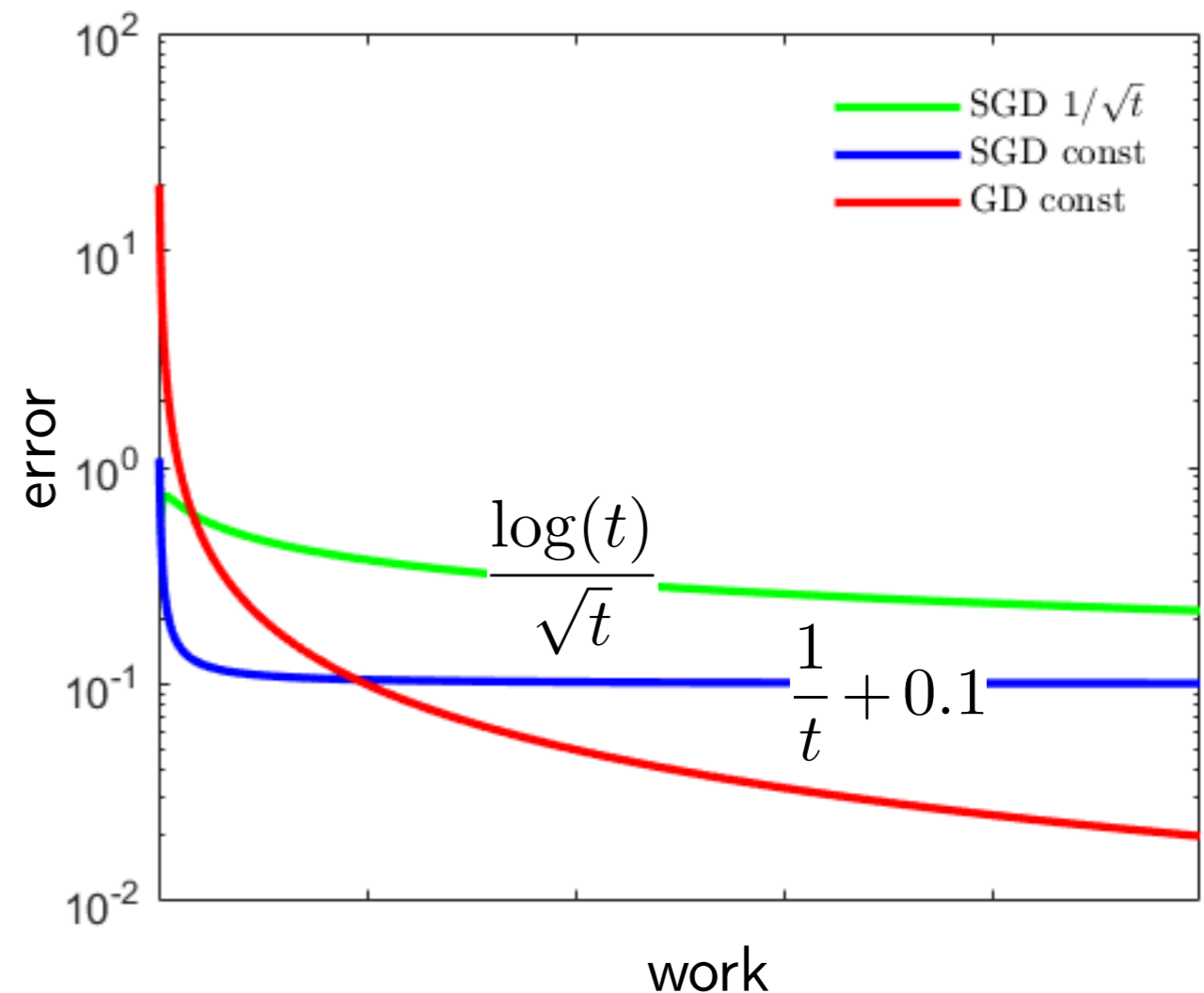
New York Times: “the embryo of an electronic computer that we expect will be able to walk, talk, see, write, reproduce itself and be conscious of its existence”

Understanding Convergence

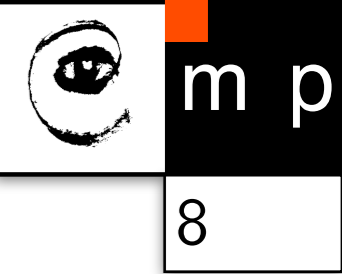


- ◆ Iteration cost:
 - GD: $O(n)$ – full data
 - SGD: $O(M)$ – mini-batch
- ◆ Guarantees on convergence rate **depend on assumptions**. Setup closest to NNs:
 - $L(\theta)$ is bounded from below
 - $\nabla L(\theta)$ is Lipschitz continuous with constant ρ
 - Bounded variance: $\mathbb{E}\|\nabla l_i(\theta) - \nabla L(\theta)\|^2 \leq \sigma^2$
or stronger condition $\mathbb{E}\|\nabla l_i(\theta)\|^2 \leq \sigma^2$ for some σ and all θ

- ◆ Convergence rates:
 - Error at iteration t : best over iterations
expected gradient norm,
 $\min_{k=1\dots t-1} \{\|\mathbb{E}[\nabla L(\theta_k)]\|\}$
 - 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: $O(\frac{1}{t}) + O(\alpha\rho\sigma^2)$



Understanding Convergence



◆ Convergence rates:

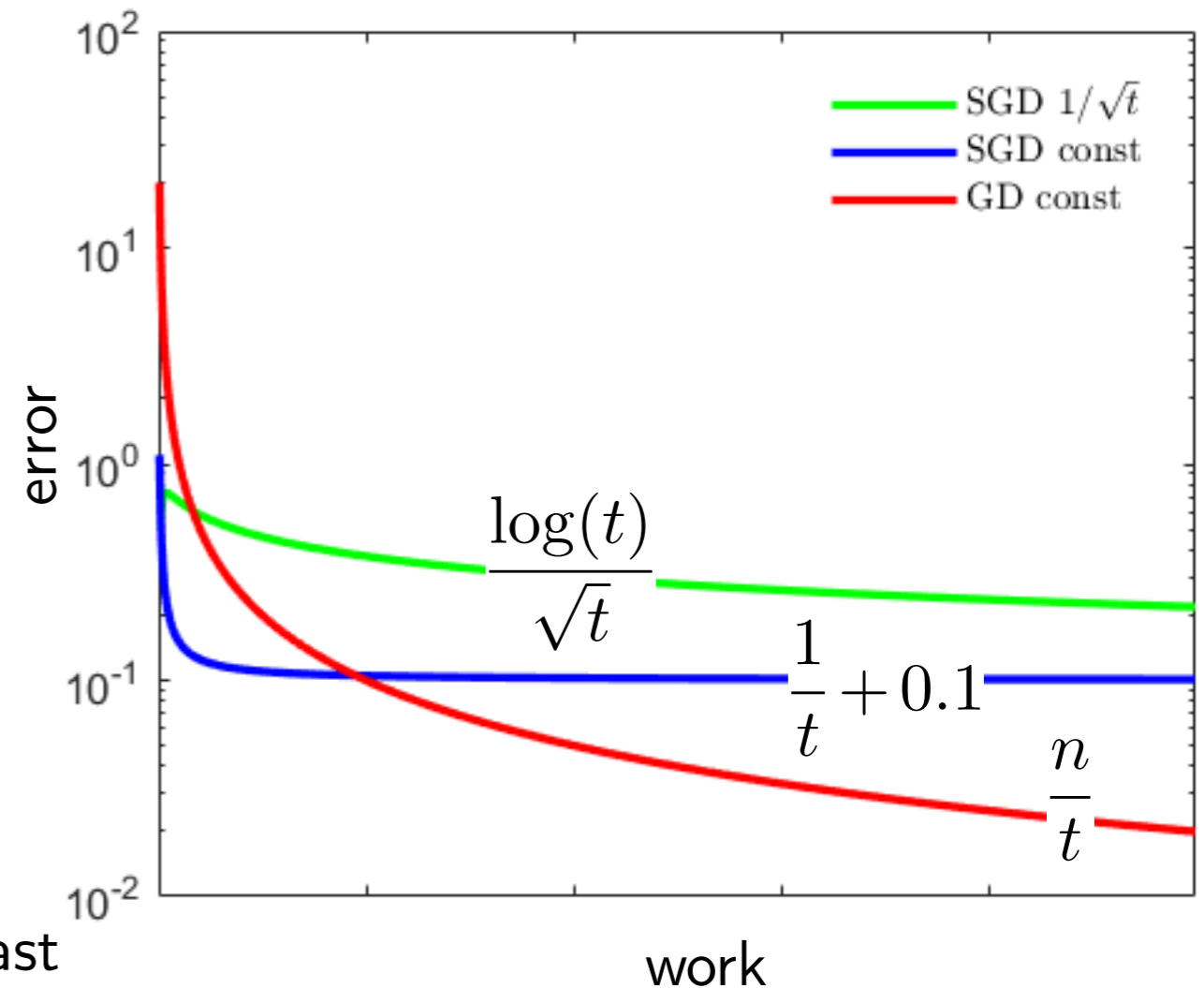
- GD with step size $\alpha_t = \alpha$
Error: $O(\frac{1}{t})$
- SGD with step size $\alpha_t = \alpha/\sqrt{t}$
Error: $O(\frac{\log(t)}{\sqrt{t}})$
- SGD with step size $\alpha_t = \alpha$
Error: $O(\frac{1}{t}) + O(\alpha\rho\sigma^2)$

◆ Insights:

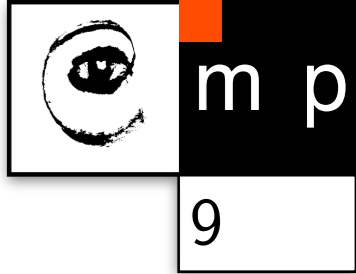
- SGD wins when there is a lot of data
- Convergence with a constant step size is fast but to within a “region” around optimum

◆ 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



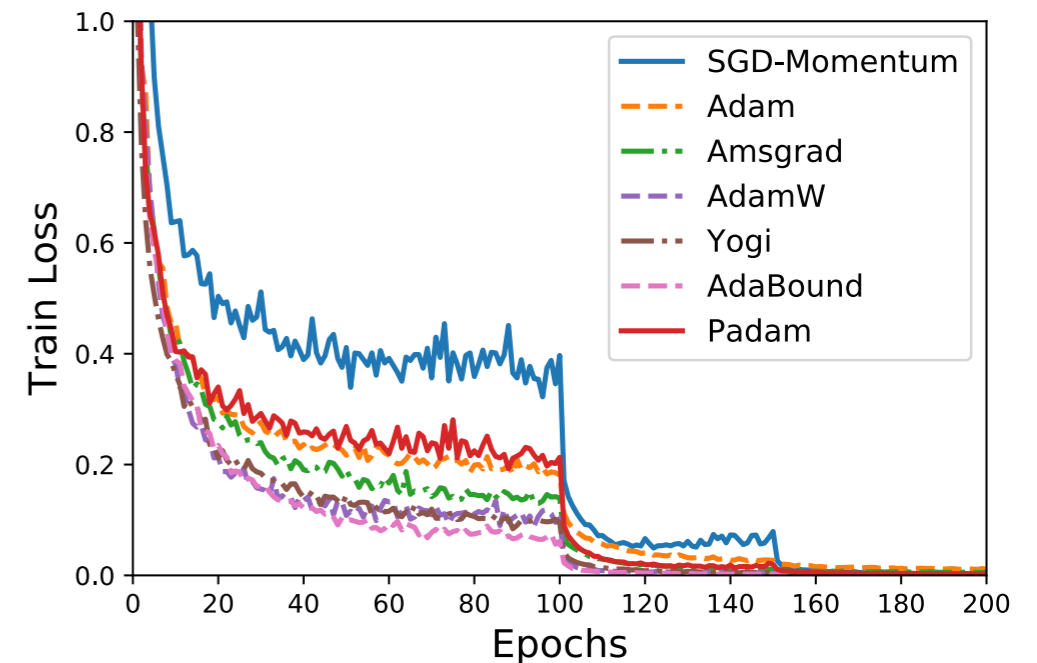
Learning Rate Schedule



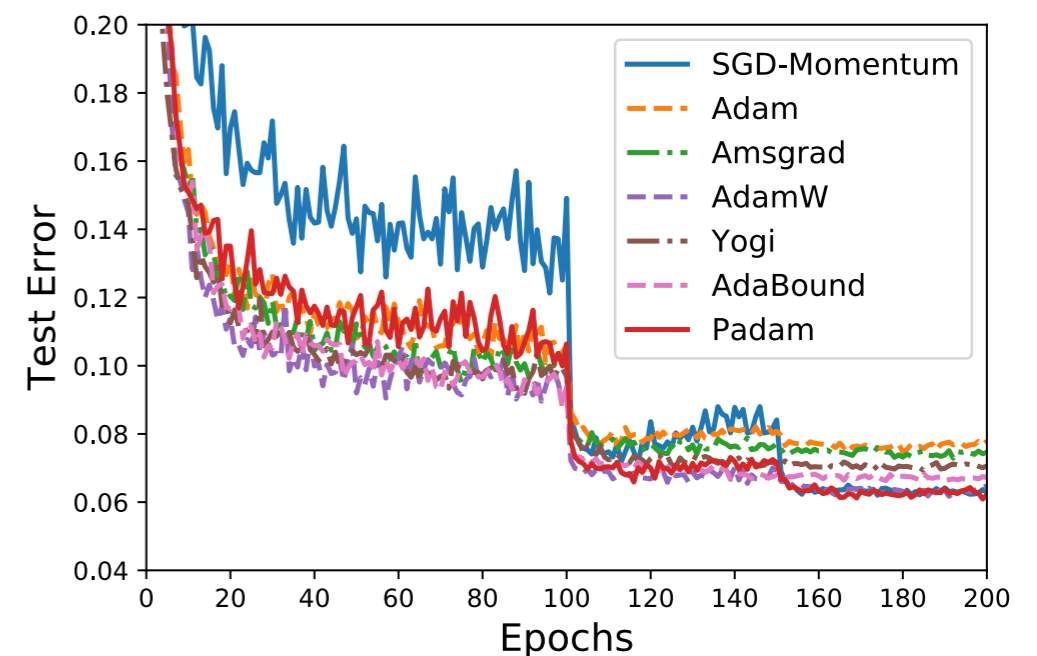
- ◆ Common practice: decrease learning rate in steps
 - Example: start with $\alpha = 0.1$ then decrease by factor of 10 at epochs 100 and 150

◆ 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

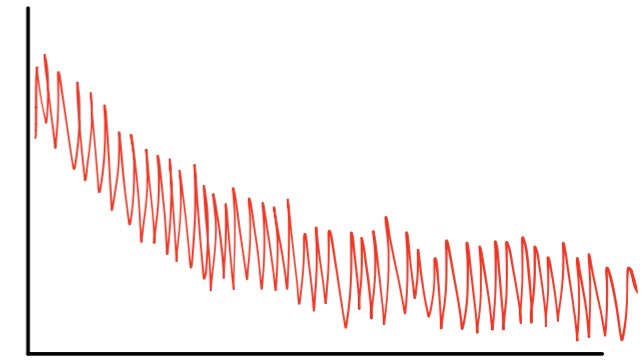
Courtesy: [Chen et al. “Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks”]

How to Measure the Progress?



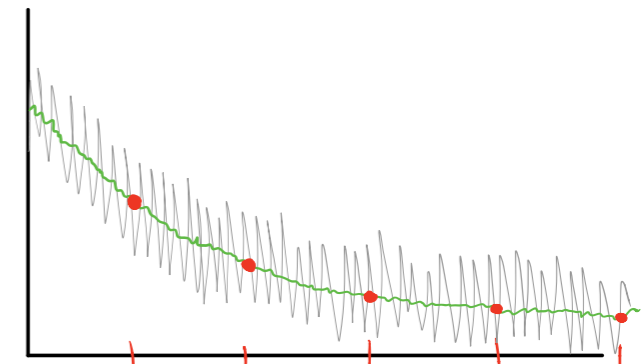
◆ Batch Estimate

- Batch mean: $\tilde{L} = \frac{1}{M} \sum_{i \in I} l_i$
- Not good idea, too high variance



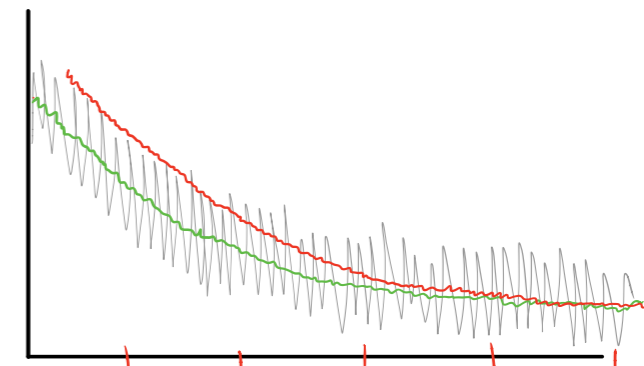
◆ Training data mean

- $L = \frac{1}{n} \sum_i l_i$
- Accurate, good if the dataset not too large



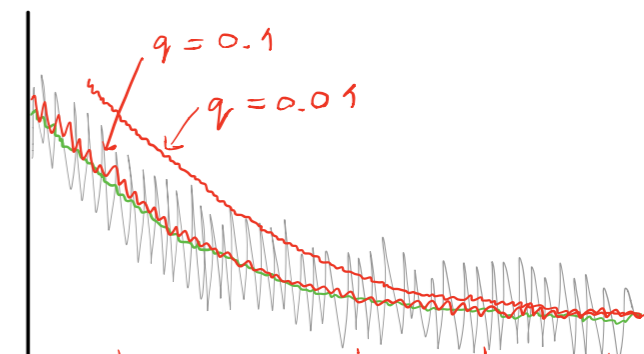
◆ Average using all last known loss values

- $\hat{L} := \frac{1}{n} \sum_i l_i^{\text{last}}$
- Low variance, hysteresis 1 epochs
- need to remember losses for full dataset



◆ Running Exponentially Weighted Average (**EWA**)

- $\hat{L} := (1 - q)\hat{L} + q\tilde{L}$
- Higher variance/ larger hysteresis
- Need to remember only the running average loss

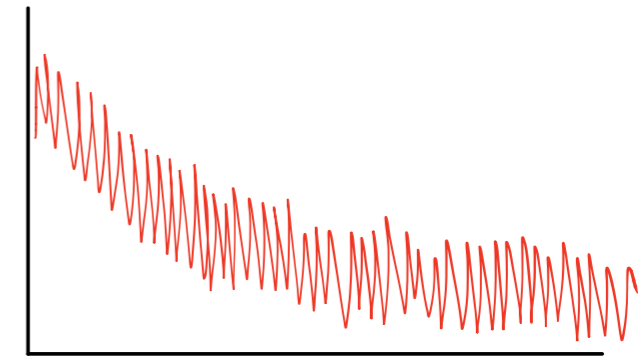


Same Applied to Gradient — Variance Reduction



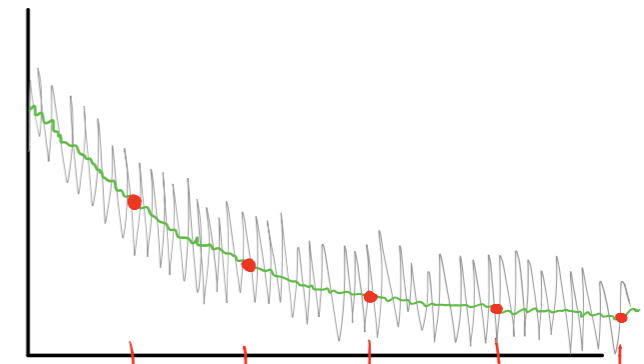
◆ SGD

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



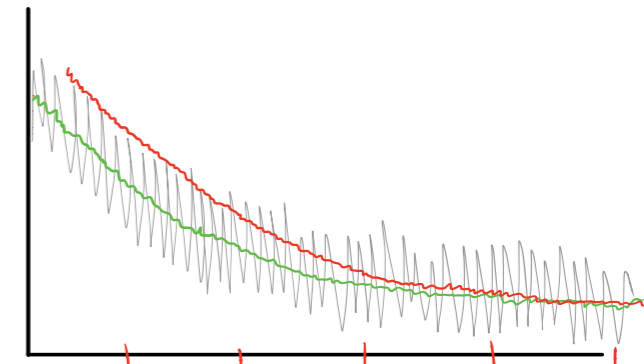
◆ GD

- Full gradient: $g = \frac{1}{n} \sum_i \nabla l_i$
- too costly



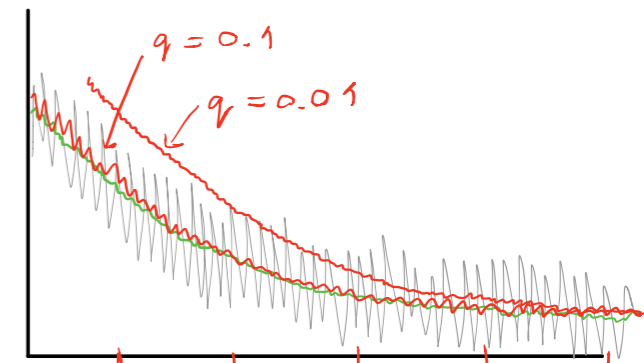
◆ Stochastic Average Gradient (**SAG**)

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



◆ SGD with **momentum**

- $g := (1 - q)g + q\tilde{g}$
- practical variance reduction
- remember only the running average gradient



Running Averages

◆ 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 $q_t = q$
- $\mu_1 = (1 - q)\mu_0 + qX_1$
- $\mu_2 = (1 - q)^2\mu_0 + (1 - q)qX_1 + qX_2$
- ...
- $\mu_t = (1 - q)^t\mu_0 + \sum_{1 \leq k \leq t} (1 - q)^{t-k}qX_k$

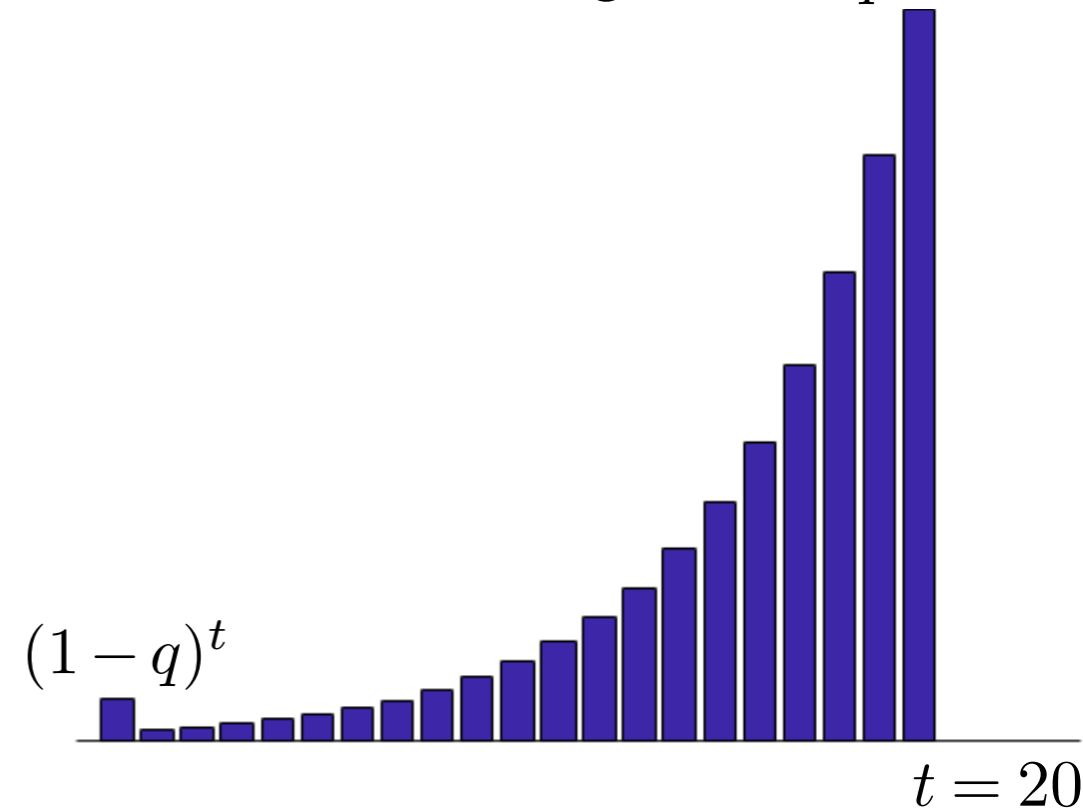
$$= w_0\mu_0 + \sum_{1 \leq k \leq t} w_k X_k$$

◆ Running mean:

- $q_t = \frac{1}{t}$
- $\mu_1 = 0\mu_0 + X_1$
- $\mu_t = \frac{t-1}{t}\mu_{t-1} + \frac{1}{t}X_t$
- $\mu_{t+1} = \frac{t}{t+1}\mu_t + \frac{1}{t+1}X_{t+1} = \frac{t-1}{t+1}\mu_{t-1} + \frac{1}{t+1}(X_t + X_{t+1})$

EWA weights

$q = 0.2$



Running mean weights



(★) Smooth transition from running mean to EWA, Unbiased Filter

◆ Algorithm

- Stochastic gradient: $\tilde{g} = \frac{1}{M} \sum_{i \in I_t} \nabla l_i$
- EWA filtered gradient: $g_t = (1 - q)g_{t-1} + q\tilde{g}$
- Step: $\theta_t = \theta_{t-1} - \alpha g_t$

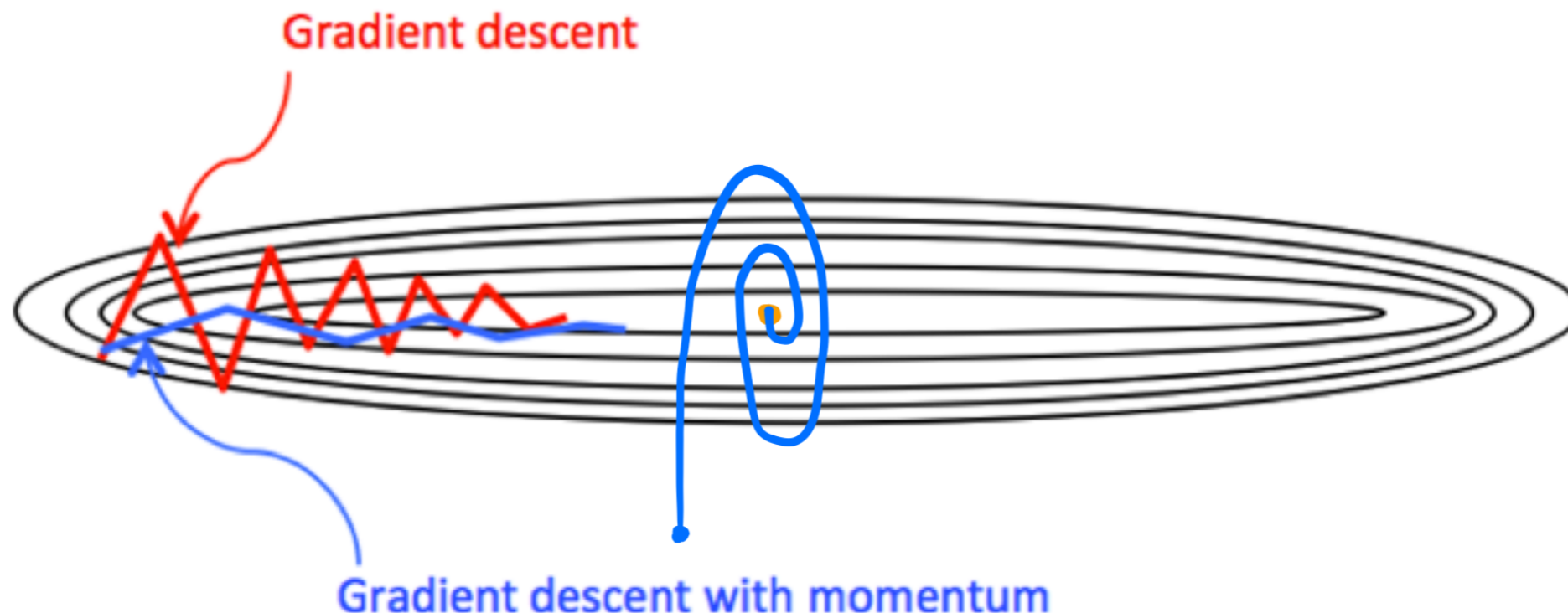
◆ In Pytorch:

- Velocity: $v_t = \mu v_{t-1} + \tilde{g}$
- Step: $\theta_t = \theta_{t-1} - \varepsilon v_t$

(★) Show equivalence by relating v_t and g_t and μ and q

- When changing momentum μ often need to adjust the learning rate as well

- ◆ With variance sufficiently low \rightarrow GD with momentum, *i.e.* consider \tilde{g} is noise-free
 - Velocity: $v_t := \mu v_{t-1} + \tilde{g}$
 - Step: $\theta_t = \theta_{t-1} - \varepsilon v_t$
- ◆ Even exact gradient may not be a good direction
- ◆ Cancels “noise” in the incorrect prediction of the function change



- ◆ The "**heavy ball**" method
 - Friction ($\mu < 1$) and slope forces build up velocity
 - Recall the hysteresis effect from using estimates from the past
 - The inertia may lead to oscillatory behavior (not good)
 - Sometimes helpful to overcome plateaus

“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: $x_{t+1} = y_t - \varepsilon \tilde{g}(y_t)$

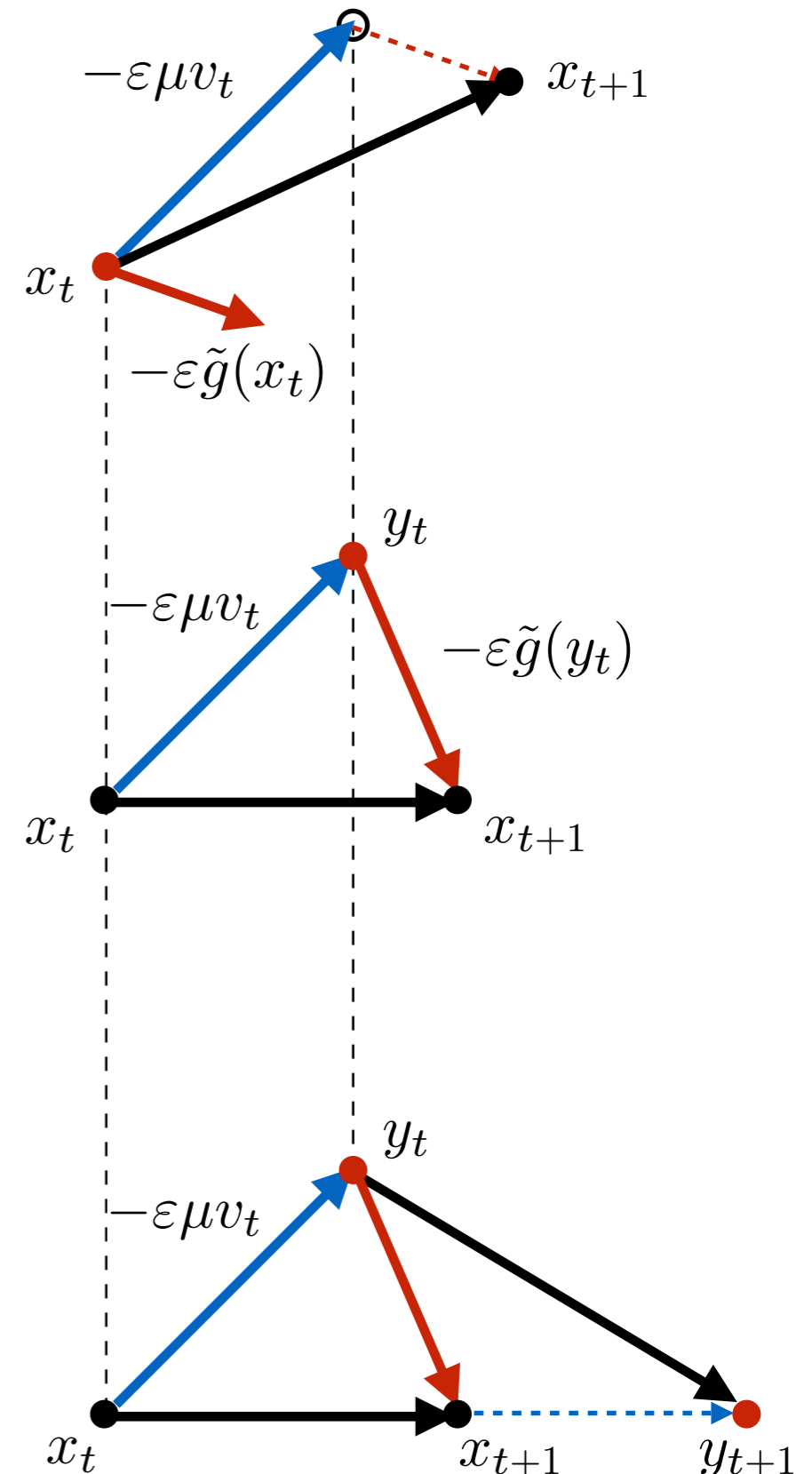
Takes advantage of the known part of the step

Less overshooting

◆ Can express as steps on the leading sequence alone (★):

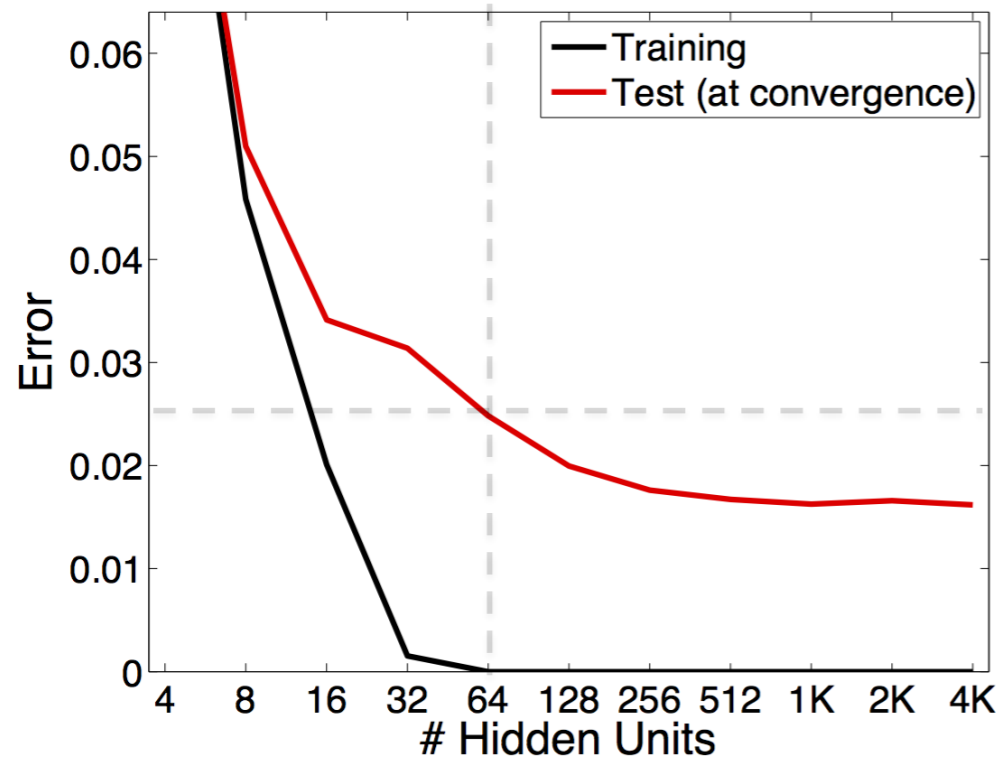
- Velocity: $v_{t+1} = \mu v_t + \tilde{g}(y_t)$
- Step: $y_{t+1} = y_t - \varepsilon (\tilde{g}(y_t) + \mu v_{t+1})$

The two sequences eventually converge

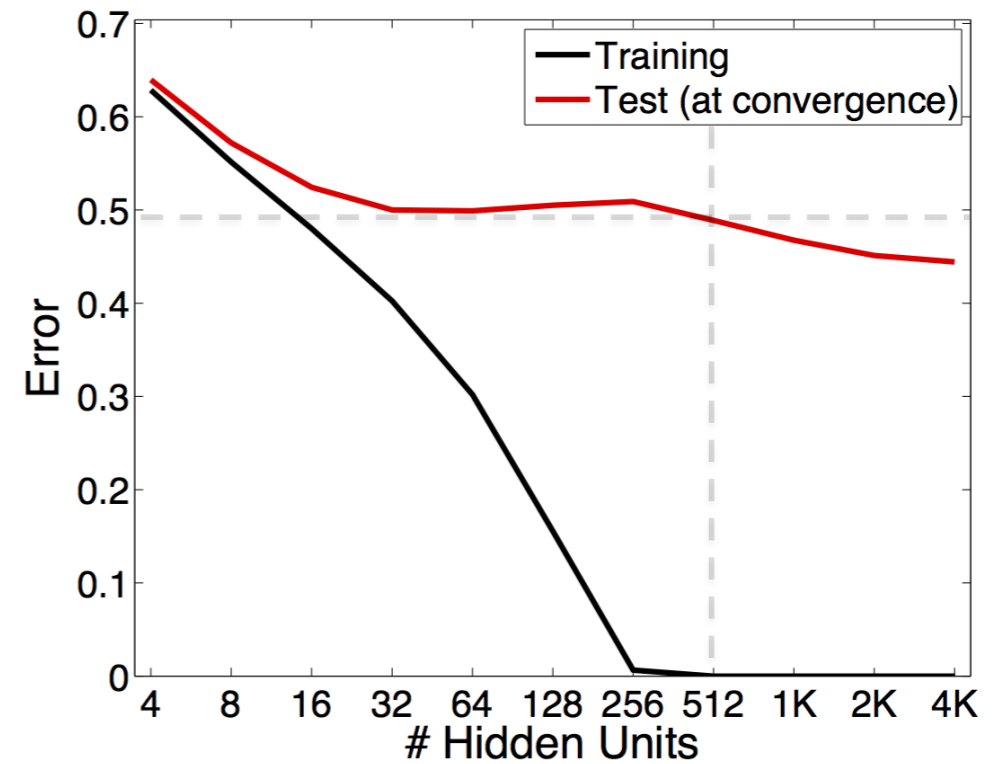


Implicit Regularization

MNIST

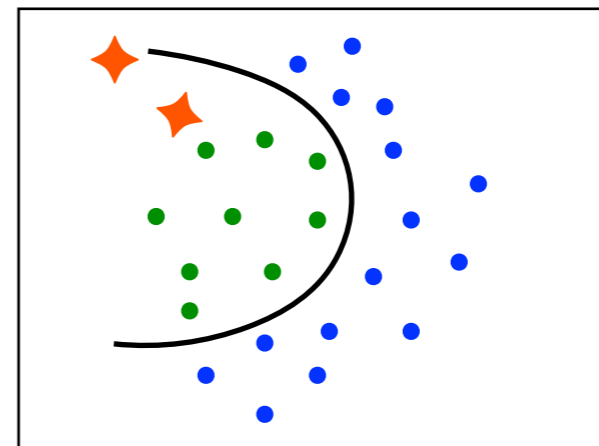
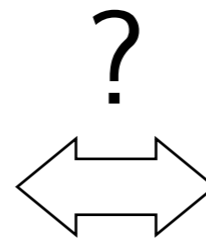
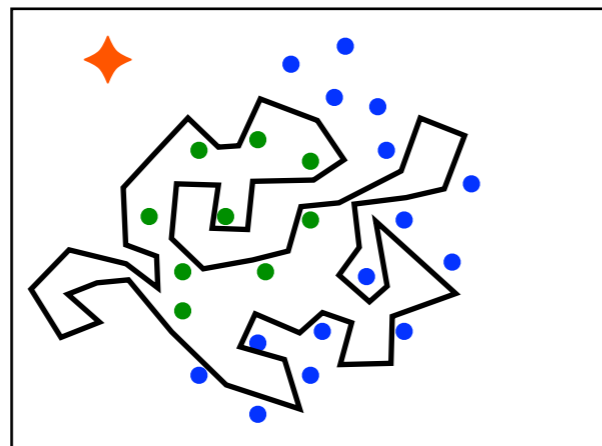


CIFAR-10



◆ We increase the network capacity but generalization improves, why?

- There exist global minima that generalize poorly
- SGD somehow finds a good global minimum



Implicit Regularization



- ◆ Logistic (or multinomial) regression:

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

GD for $\min_w \mathcal{L}(w)$

GD iteration (\star):

$$w^{t+1} = w^t + \operatorname{argmin}_{\Delta w} \left(\langle \Delta w, \nabla \mathcal{L}(w^t) \rangle + \frac{1}{2\varepsilon} \|\Delta w\|_2^2 \right)$$
$$t \rightarrow \infty \quad \frac{w^t}{\|w^t\|} \rightarrow \max \text{ margin w.r.t. } \|\cdot\|_2 \quad [2]$$

- ◆ Linear model, loss with a unique finite root, \mathcal{W} – set of optimal solutions

SGD (stepsteepest in $\|\cdot\|_p$) iteration:

$$w^{t+1} = w^t + \operatorname{argmin}_{\Delta w} \left(\langle \Delta w, \tilde{\nabla} \mathcal{L}(w^t) \rangle + \frac{1}{2\varepsilon} \|\Delta w\|_p^p \right)$$
$$t \rightarrow \infty \quad w^t \rightarrow \text{nearest (in } \|\cdot\|_p) \text{ optimal point to } w^0 \quad [3]$$

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

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

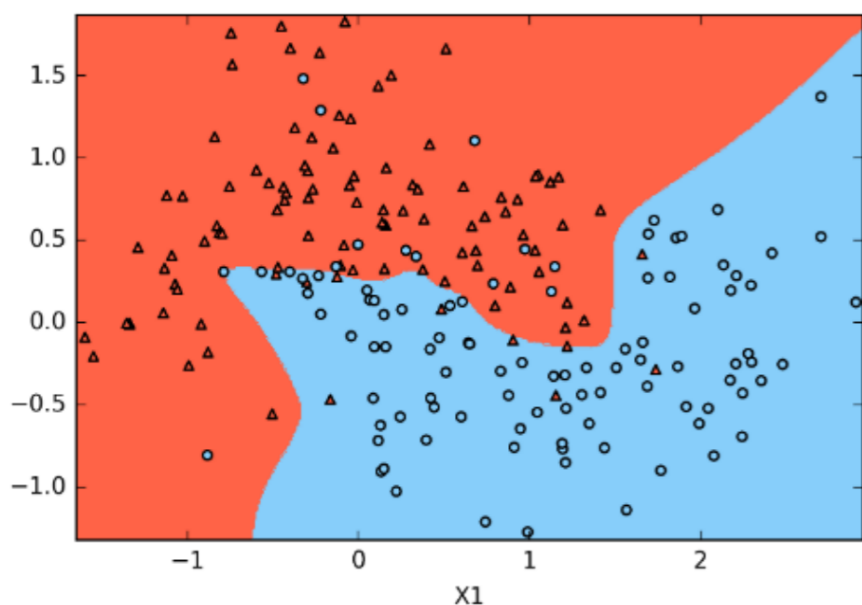
[2] Soudry et al. (2018) "The Implicit Bias of Gradient Descent on Separable Data"

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

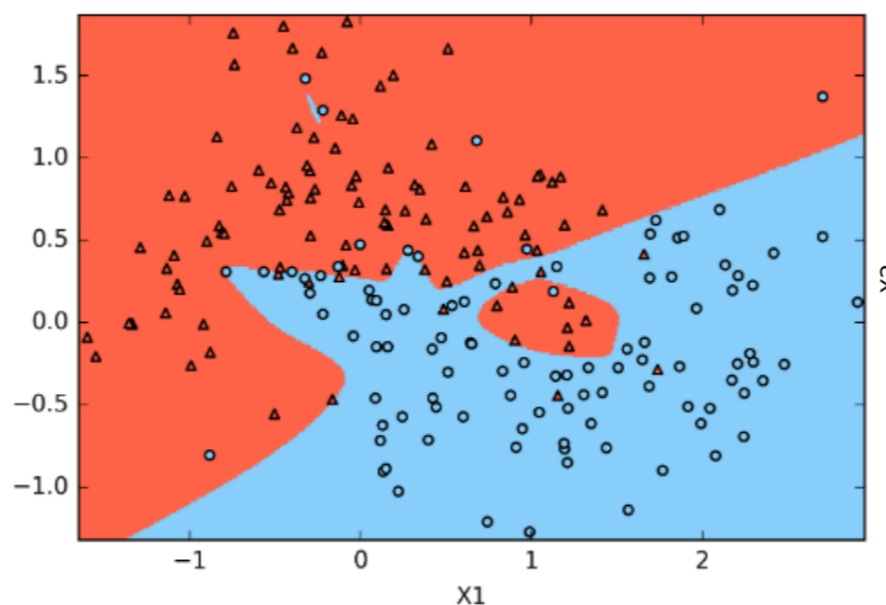
Smaller Batch Size -> More Regularization

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

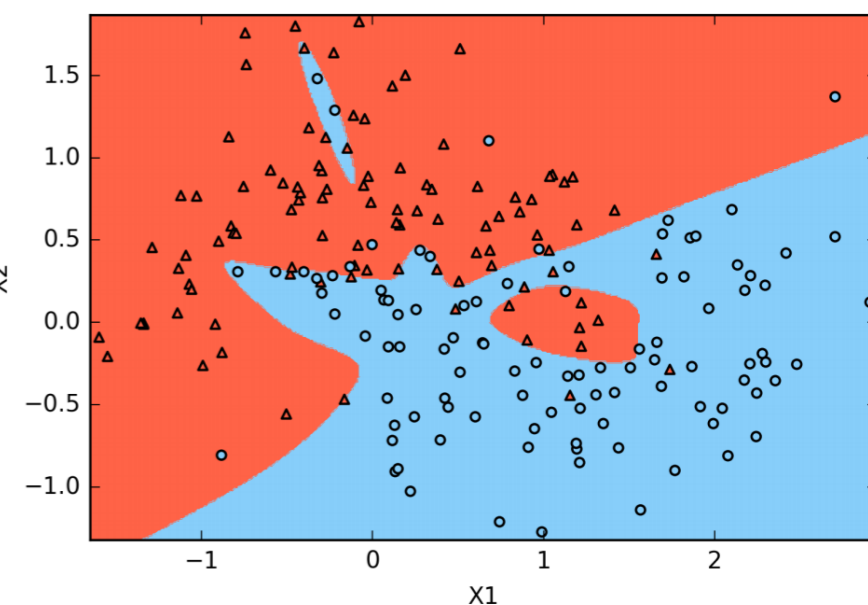
Synthetic data



Decision boundary of batch size 1

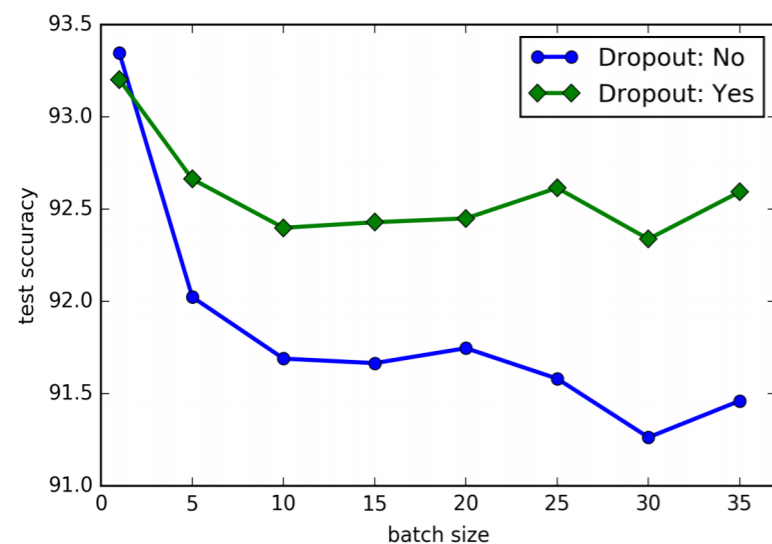


Decision boundary of batch size 5



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"

More about SGD

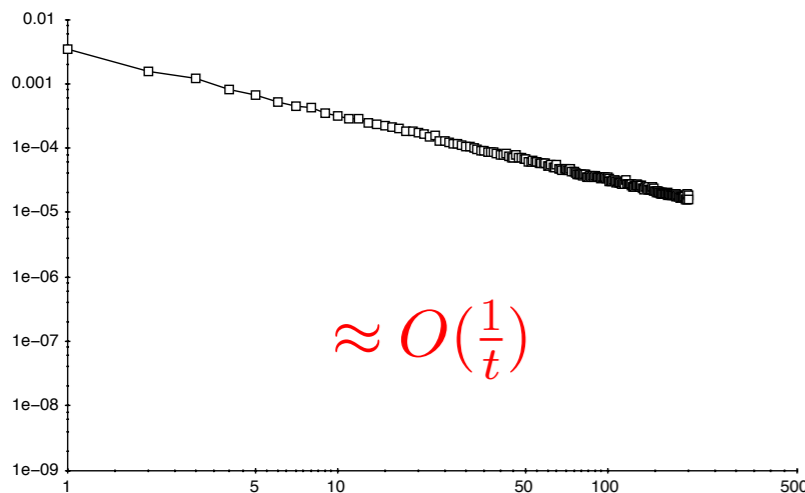
How to Draw Data Points?

- ◆ 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

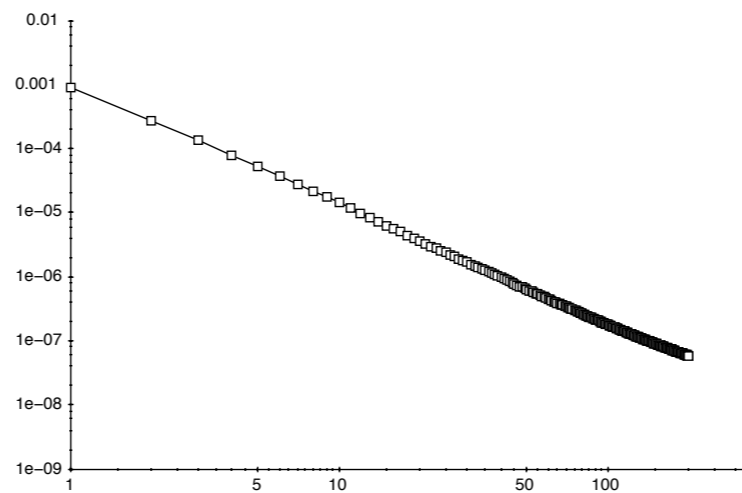
◆ Empirical evidence:

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

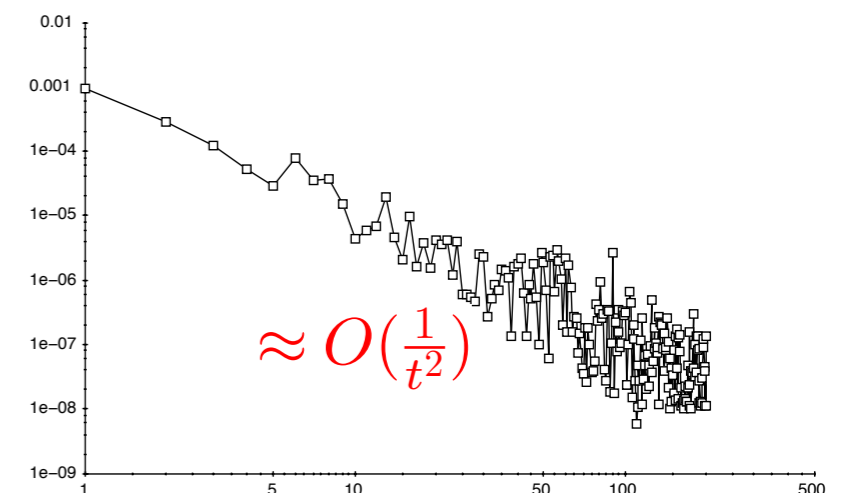
logistic regression $d = 47,152, n = 781,256$



Random selection:
slope = -1.0003



Cycling the same random
shuffle: slope = -1.8393



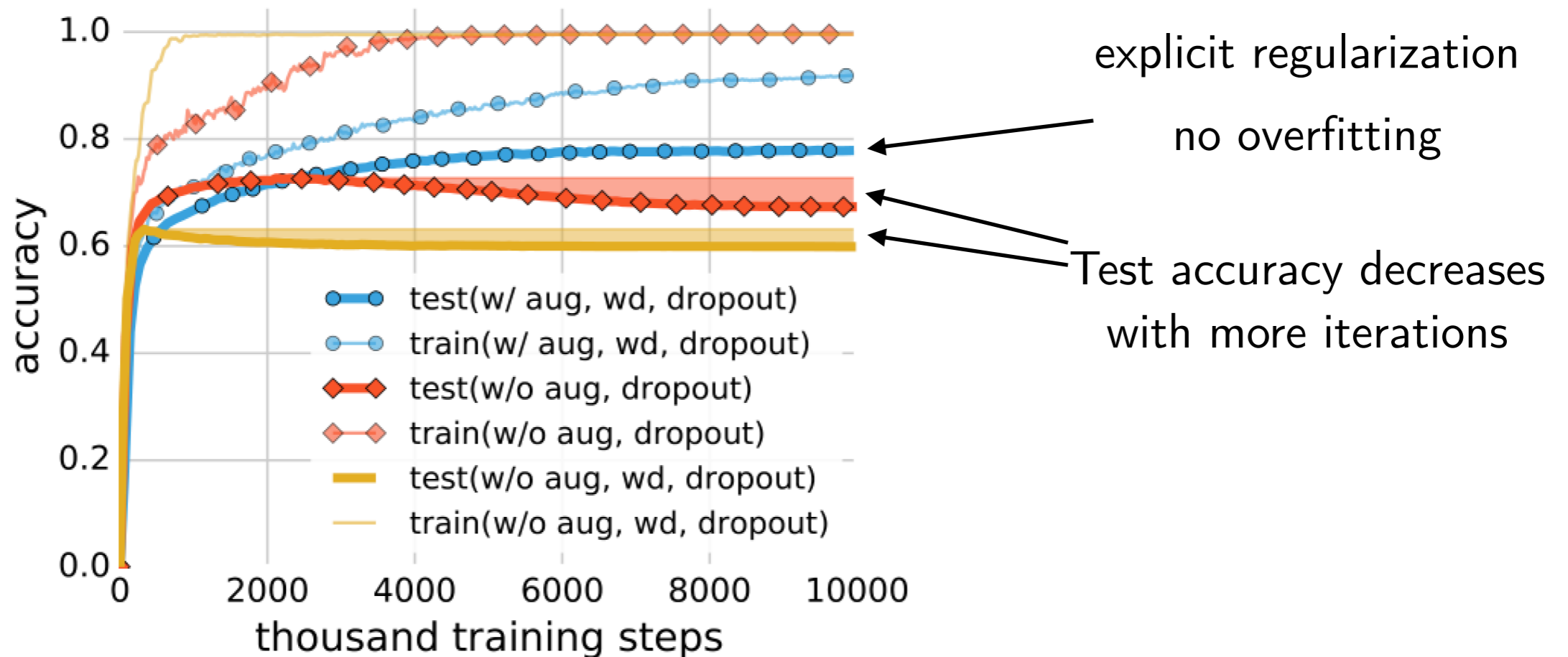
Random shuffle at each
epoch: slope = -2.0103

◆ A simple consideration:

Drawing n times with replacement from the dataset of size n some points may not be selected. On average each point is selected with probability ≈ 0.63 for large n . Takes long time to even out (★) – associated exercise

Implicit Regularization: Early Stopping

- ◆ We expect the learning to overfit, often it does not
- ◆ Example when it does:



(a) Inception on ImageNet

[Zhang et al. (2017) “Understanding Deep Learning Requires ReThinking Generalization”]

- ◆ Early stopping could potentially improve generalization when other regularizers are absent
- ◆ Need a validation set

Running Averages: How Much Smoothing?



◆ 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
- $\mathbb{E}[\mu_t] = w_0\mathbb{E}[\mu_0] + \sum_{k=1}^t w_k\mathbb{E}[X_k]$
- In context of SGD with learning rate $\varepsilon \rightarrow 0$, all $E[X_k]$ are the same and μ_t is an unbiased estimate

◆ Variance:

- $\mathbb{V}[\mu_t] = (1 - q_t)^2\mathbb{V}[\mu_{t-1}] + q_t^2\mathbb{V}[X_t]$
 - $\mathbb{V}[\mu_t] = w_0^2\mathbb{V}_0 + \sum_{k=1}^t w_k^2\mathbb{V}[X_k]$
 - 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
- (★) 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

More in Lecture 8



- ◆ Loss Landscape of NNs
 - Permutation invariance and overcomplete parameterizations
 - Local minima and saddle points in high dimensions
 - Empirical evidence of many good local minima
 - Redundancy helps optimization
- ◆ SGD sensitivity to change of variables
- ◆ Adaptive methods