CS60021: Scalable Data Mining

Large Scale Machine Learning

Sourangshu Bhattacharya

Much of ML is optimization

Linear Classification

$$\arg\min_{w} \sum_{i=1}^{n} ||w||^2 + C \sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } 1 - y_i x_i^T w \leq \xi_i$$
$$\xi_i \geq 0$$

Maximum Likelihood

$$\arg\max_{\theta} \sum_{i=1}^{n} \log\, p_\theta(x_i)$$

K-Means

$$\arg \min_{\mu_1, \mu_2, \ldots, \mu_k} J(\mu) = \sum_{j=1}^{k} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

Stochastic optimization

- Goal of machine learning :
  - Minimize expected loss

$$\min_h L(h) = \mathbf{E}\left[\text{loss}(h(x), y)\right]$$

given samples $(x_i, y_i)\ i = 1, 2...m$

- This is Stochastic Optimization
  - Assume loss function is convex

Batch (sub)gradient descent for ML

- Process all examples together in each step

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} \frac{\partial L(w, x_i, y_i)}{\partial w} \right)$$

where L is the regularized loss function

- Entire training set examined at each step
- Very slow when *n* is very large

Stochastic (sub)gradient descent

- "Optimize" one example at a time
- Choose examples randomly (or reorder and choose in order)
  - Learning representative of example distribution

for $i = 1$ to n:

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

where L is the regularized loss function

Stochastic (sub)gradient descent

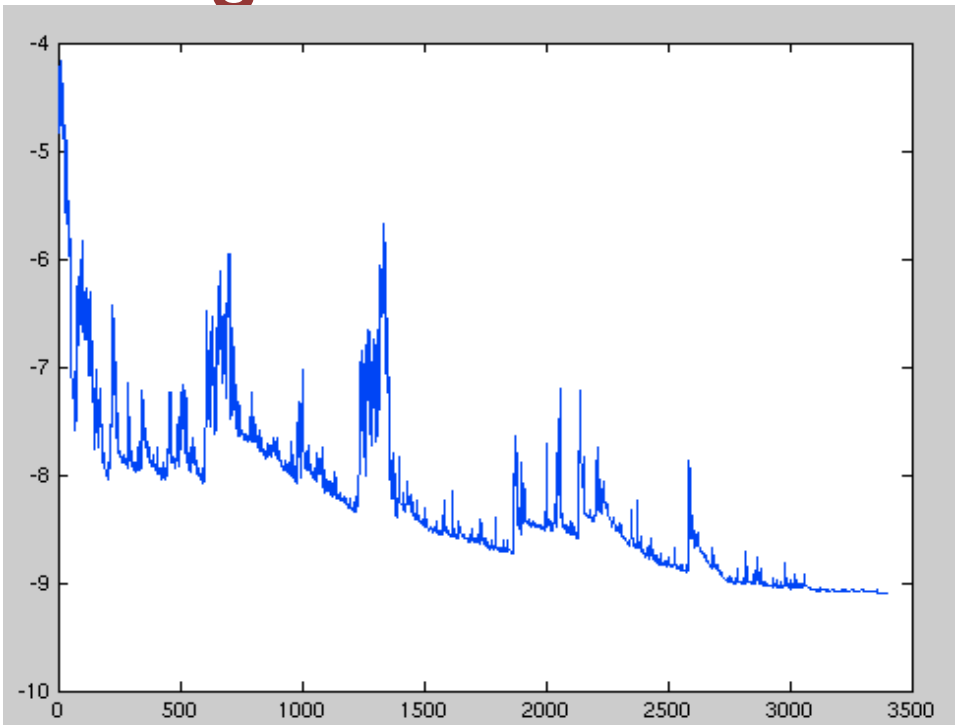
for $i = 1$ to n:

$$w^{(k+1)} \leftarrow w^{(k)} - \eta_t \frac{\partial L(w, x_i, y_i)}{\partial w}$$

where L is the regularized loss function

- Equivalent to online learning (the weight vector *w* changes with every example)
- Convergence guaranteed for convex functions (to local minimum)

SGD convergence



Stochastic gradient descent

- Given dataset $D = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
- Loss function: $L(\theta, D) = \frac{1}{N}\sum_{i=1}^{N} l(\theta; x_i, y_i)$
- For linear models: $l(\theta; x_i, y_i) = l(y_i, \theta^T \phi(x_i))$
- Assumption D is drawn IID from some distribution $\mathcal{P}$.
- Problem:

$$\min_{\theta} L(\theta, D)$$

Stochastic gradient descent

- Input: D
- Output: $\bar{\theta}$

Algorithm:

- Initialize θ^0
- For $t = 1, \ldots, T$

$$\theta^{t+1} = \theta^t - \eta_t \nabla_\theta l(y_t, \theta^T \phi(x_t))$$

- $\bar{\theta} = \frac{\sum_{t=1}^{T} \eta_t \theta^t}{\sum_{t=1}^{T} \eta_t}$.

SGD convergence

- Expected loss: $s(\theta) = E_{\mathcal{P}}[l(y, \theta^T \phi(x)]$
- Optimal Expected loss: $s^* = s(\theta^*) = \min_{\theta} s(\theta)$
- Convergence:

$$E_{\bar{\theta}}[s(\bar{\theta})] - s^* \leq \frac{R^2 + L^2 \sum_{t=1}^{T} \eta_t^2}{2 \sum_{t=1}^{T} \eta_t}$$

- Where: $R = \|\theta^0 - \theta^*\|$
- $L = \max \nabla l(y, \theta^T \phi(x))$

SGD convergence proof

- Define $r_t = \|\theta^t - \theta^*\|$ and $g_t = \nabla_\theta l\left(y_t, \theta^T \phi(x_t)\right)$
- $r_{t+1}^2 = r_t^2 + \eta_t^2 \|g_t\|^2 - 2\eta_t(\theta^t - \theta^*)^T g_t$
- Taking expectation w.r.t $\mathcal{P}, \bar{\theta}$ and using $s^* - s(\theta^t) \geq g_t^T(\theta^* - \theta^t)$, we get:
  $E_{\bar{\theta}}[r_{t+1}^2 - r_t^2] \leq \eta_t^2 L^2 + 2\eta_t(s^* - E_{\bar{\theta}}[s(\theta^t)])$

- Taking sum over $t = 1, \ldots, T$ and using

$$E_{\bar{\theta}}[r_{t+1}^2 - r_0^2] \leq L^2 \sum_{t=0}^{T-1} \eta_t^2 + 2 \sum_{t=0}^{T-1} \eta_t (s^* - E_{\bar{\theta}}[s(\theta^t)])$$

SGD convergence proof

- Using convexity of s:

$$\left(\sum_{t=0}^{T-1} \eta_t\right) E_{\bar{\theta}}[s(\bar{\theta})] \leq E_{\bar{\theta}}[\sum_{t=0}^{T-1} \eta_t s(\theta^t)]$$

- Substituting in the expression from previous slide:

$$E_{\bar{\theta}}[r_{t+1}^2 - r_0^2] \leq L^2 \sum_{t=0}^{T-1} \eta_t^2 + 2 \sum_{t=0}^{T-1} \eta_t (s^* - E_{\bar{\theta}}[s(\bar{\theta})])$$

- Rearranging the terms proves the result.

SGD - Issues

- Convergence very sensitive to learning rate (η_t) (oscillations near solution due to probabilistic nature of sampling)
  - Might need to decrease with time to ensure the algorithm converges eventually
- Basically – SGD good for machine learning with large data sets!

Mini-batch SGD

- Stochastic – 1 example per iteration
- Batch – All the examples!
- Mini-batch SGD:
  - Sample *m* examples at each step and perform SGD on them
- Allows for parallelization, but choice of *m* based on heuristics

Example: Text categorization

- **Example by Leon Bottou:**
  - **Reuters RCV1** document corpus
    - Predict a category of a document
      - One **vs.** the rest classification
  - ***n* = 781,000** training examples (documents)
  - 23,000 test examples
  - ***d* = 50,000** features
    - One feature per word
    - Remove stop-words
    - Remove low frequency words

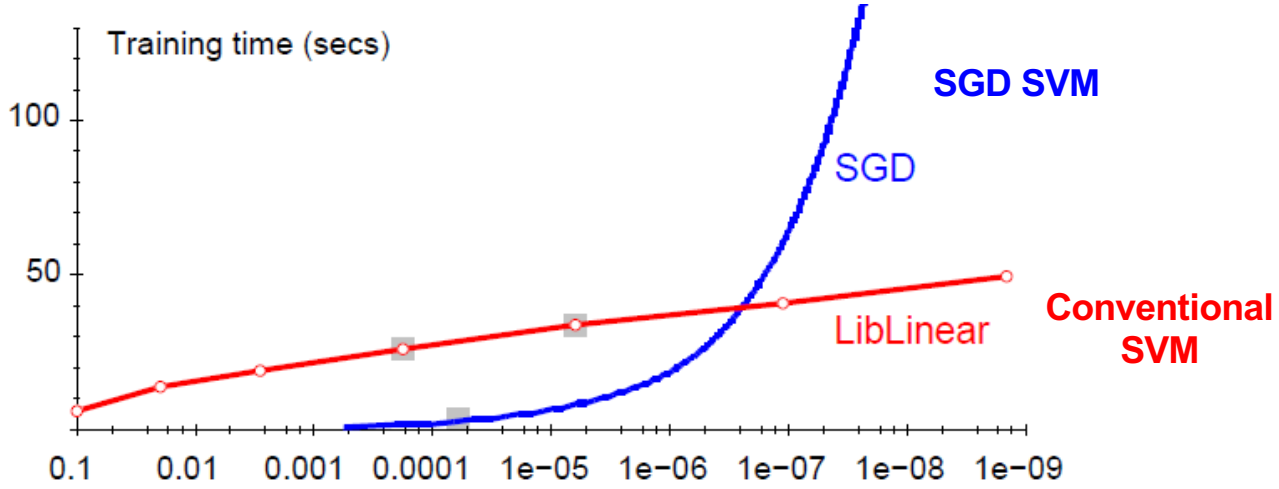
Example: Text categorization

- **Questions:**
  - **(1)** Is **SGD** successful at minimizing ***f(w,b)***?
  - **(2)** How quickly does **SGD** find the min of ***f(w,b)***?
  - **(3)** What is the error on a test set?

	Training time	*Value of f(w,b)*	*Test error*
Standard SVM	23,642 secs	*0.2275*	*6.02%*
"Fast SVM"	66 secs	0.2278	6.03%
SGD SVM	1.4 secs	*0.2275*	*6.02%*

(1) SGD-SVM is successful at minimizing the value of ***f(w,b)***
(2) SGD-SVM is super fast
(3) SGD-SVM test set error is comparable

Optimization "Accuracy"

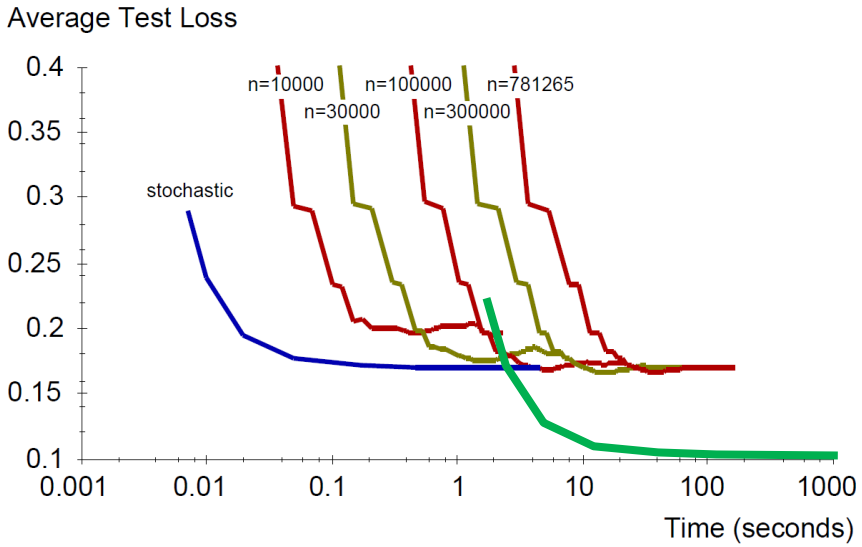


Optimization quality: $| f(w,b) - f(w^{opt},b^{opt}) |$

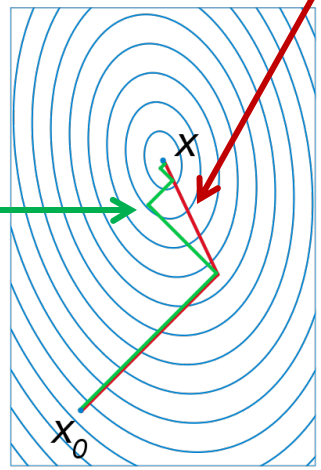
For optimizing *f(w,b) within reasonable* quality
SGD-SVM is super fast

SGD vs. Batch Conjugate Gradient

- **SGD** on full dataset vs. **Conjugate Gradient** on a sample of ***n*** training examples



Theory says: Gradient descent converges in linear time k. **Conjugate gradient** converges in $\sqrt{k}$.
k… condition number



Bottom line: Doing a simple (but fast) SGD update many times is better than doing a complicated (but slow) CG update a few times

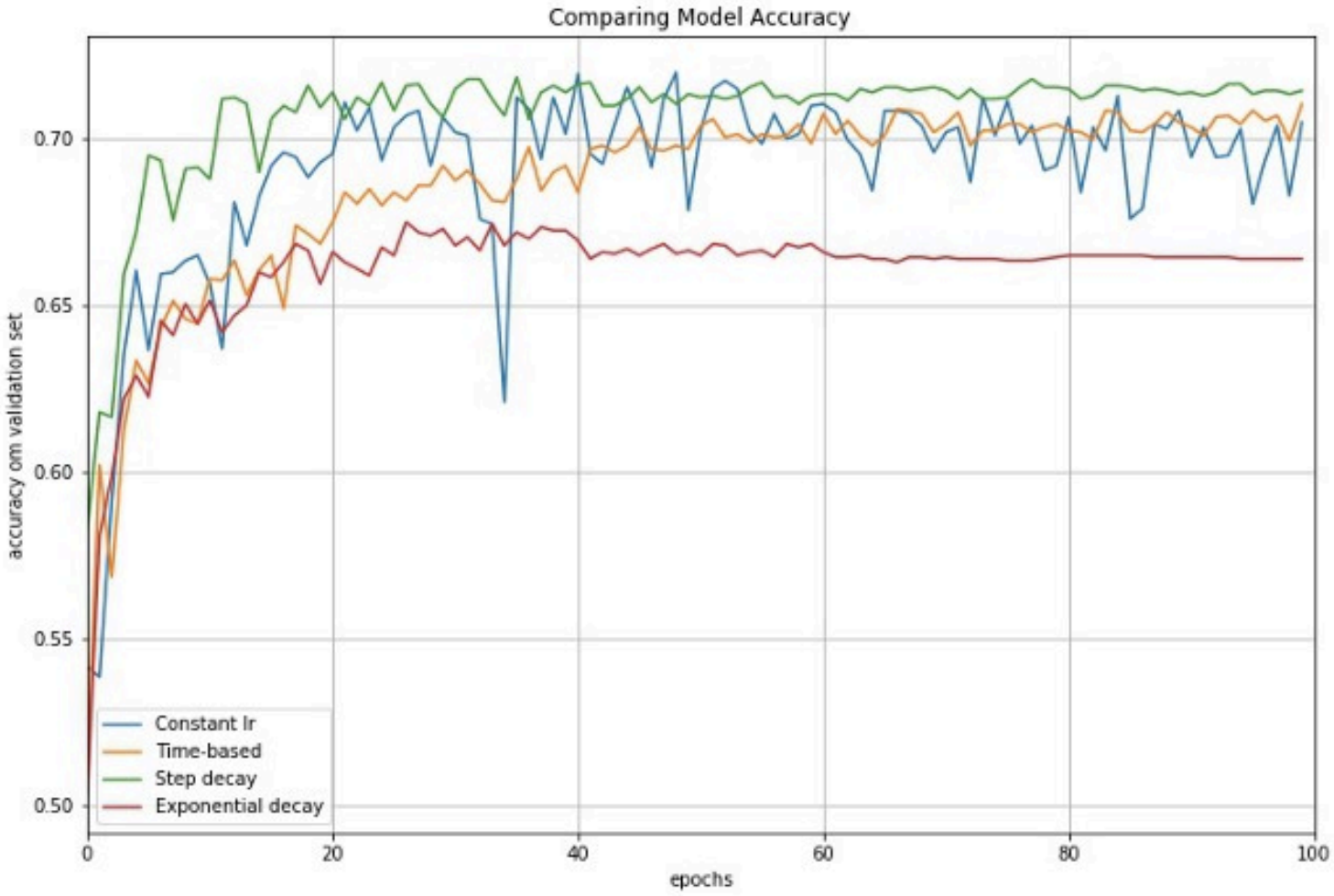
Practical Considerations

- **Need to choose learning rate η and t_0**

$$w_{t+1} \leftarrow w_t - \frac{\eta_t}{t + t_0}\left(w_t + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

- **Leon suggests:**
  - Choose $\mathbf{t_0}$ so that the expected initial updates are comparable with the expected size of the weights
  - Choose η:
    - Select a **small subsample**
    - Try various rates η (e.g., 10, 1, 0.1, 0.01, …)
    - Pick the one that most reduces the cost
    - Use η for next 100k iterations on the full dataset

Learning rate comparison



Practical Considerations

- **Sparse Linear SVM:**
  - **Feature vector x_i is sparse (contains many zeros)**
    - Do not do: $\boldsymbol{x_i}$ = [0,0,0,1,0,0,0,0,5,0,0,0,0,0,0,…]
    - But represent $\mathbf{x_i}$ as a sparse vector $\mathbf{x_i}$=[*(4,1), (9,5),* …]
  - **Can we do the SGD update more efficiently?**

$$w \leftarrow w - \eta\left(w + C\frac{\partial L(x_i, y_i)}{\partial w}\right)$$

  - **Approximated in 2 steps:**

$$w \leftarrow w - \eta C\frac{\partial L(x_i, y_i)}{\partial w}$$

cheap: $\boldsymbol{x_i}$ is sparse and so few coordinates ***j*** of ***w*** will be updated

$$w \leftarrow w(1-\eta)$$

expensive: ***w*** is not sparse, all coordinates need to be updated

Practical Considerations

- **Solution 1:** $\boldsymbol{w = s \cdot v}$
  - Represent vector $\boldsymbol{w}$ as the product of scalar $\boldsymbol{s}$ and vector $\boldsymbol{v}$
  - Then the update procedure is:
    - $\boldsymbol{(1)\ v = v - \eta C \frac{\partial L(x_i, y_i)}{\partial w}}$
    - $\boldsymbol{(2)\ s = s(1 - \eta)}$

Two step update procedure:

$$(1) w \leftarrow w - \eta C \frac{\partial L(x_i, y_i)}{\partial w}$$

$$(2)\ w \leftarrow w(1 - \eta)$$

- **Solution 2:**
  - Perform only step **(1)** for each training example
  - Perform step **(2)** with lower frequency and higher η

Practical Considerations

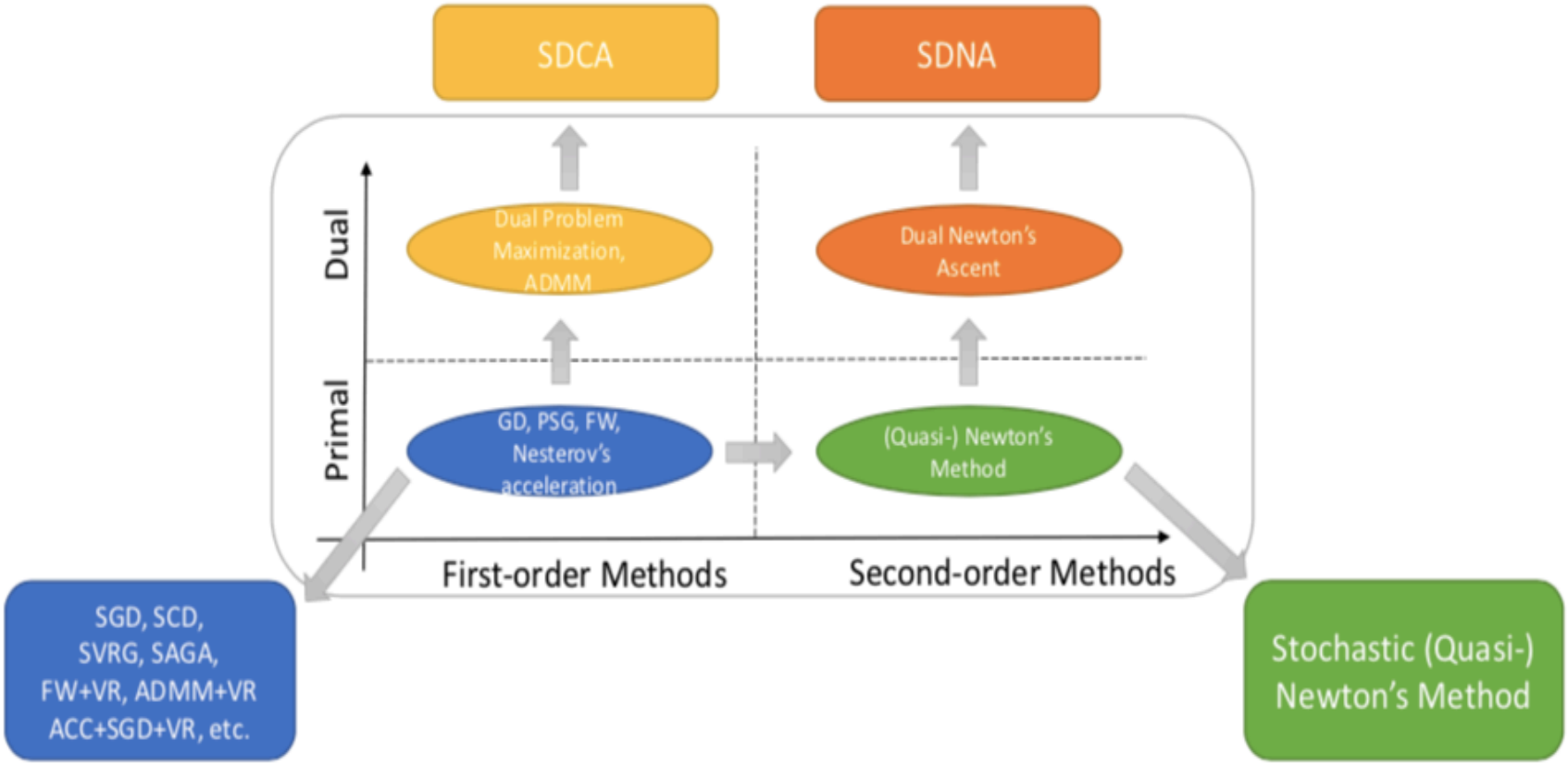
- **Stopping criteria:**

  How many iterations of SGD?

  - **Early stopping with cross validation**
    - Create a validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - **Early stopping**
    - Extract two disjoint subsamples **A** and **B** of training data
    - Train on **A**, stop by validating on **B**
    - Number of epochs is an estimate of ***k***
    - Train for ***k*** epochs on the full dataset

THEORETICAL GUARRANTEES

Stochastic Optimization



Convergence Rate and Computational Complexity

Overall Complexity (ϵ) = Convergence Rate^{-1}(ϵ) $*$ Complexity of each iteration

	Strongly Convex + Smooth			Convex + Smooth		
	Convergence Rate	Complexity of each iteration	Overall Complexity	Convergence Rate	Complexity of each iteration	Overall Complexity
GD	$O\left(\exp\left(-\frac{t}{Q}\right)\right)$	$O(n \cdot d)$	$O\left(nd \cdot Q \cdot \log\left(\frac{1}{\epsilon}\right)\right)$	$O\left(\frac{\beta}{t}\right)$	$O(n \cdot d)$	$O\left(nd \cdot \beta \cdot \left(\frac{1}{\epsilon}\right)\right)$
SGD	$O\left(\frac{1}{t}\right)$	$O(d)$	$O\left(\frac{d}{\epsilon}\right)$	$O\left(\frac{1}{\sqrt{t}}\right)$	$O(d)$	$O\left(\frac{d}{\epsilon^2}\right)$
SCD	$O\left(\exp\left(-\frac{t}{d \cdot \max_j Q_j}\right)\right)$	$O(n)$ (For separable cases)	$O\left(nd \cdot \max_j Q_j \cdot \log\left(\frac{1}{\epsilon}\right)\right)$	$O\left(\frac{\max_j \beta_j}{d \cdot t}\right)$	$O(n)$ (For separable cases)	$O\left(nd \cdot \max_j \beta_j \cdot \frac{1}{\epsilon}\right)$

1. When data size n is very large, SGD is faster than GD.
2. SCD is faster than GD for separable cases. ($\max_j \beta_j \le \beta \le d\beta_j$, $\max_j Q_j \le Q \le dQ_j$)

SGD Analysis

THEOREM 14.8 *Let $B, \rho > 0$. Let f be a convex function and let $\mathbf{w}^\star \in \operatorname{argmin}_{\mathbf{w}:\|\mathbf{w}\|\leq B} f(\mathbf{w})$. Assume that SGD is run for T iterations with $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$. Assume also that for all t, $\|\mathbf{v}_t\| \leq \rho$ with probability 1. Then,*

$$\mathbb{E}\left[f(\bar{\mathbf{w}})\right] - f(\mathbf{w}^\star) \leq \frac{B\rho}{\sqrt{T}}.$$

Therefore, for any $\epsilon > 0$, to achieve $\mathbb{E}[f(\bar{\mathbf{w}})] - f(\mathbf{w}^\star) \leq \epsilon$, it suffices to run the SGD algorithm for a number of iterations that satisfies

$$T \geq \frac{B^2\rho^2}{\epsilon^2}.$$

SGD Analysis

LEMMA 14.1 *Let $\mathbf{v}_1, \ldots, \mathbf{v}_T$ be an arbitrary sequence of vectors. Any algorithm with an initialization $\mathbf{w}^{(1)} = 0$ and an update rule of the form*

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t \tag{14.4}$$

satisfies

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{\|\mathbf{w}^\star\|^2}{2\eta} + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2. \tag{14.5}$$

In particular, for every $B, \rho > 0$, if for all t we have that $\|\mathbf{v}_t\| \leq \rho$ and if we set $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$, then for every $\mathbf{w}^\star$ with $\|\mathbf{w}^\star\| \leq B$ we have

$$\frac{1}{T} \sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle \leq \frac{B\rho}{\sqrt{T}}.$$

SGD Analysis

Proof Using algebraic manipulations (completing the square), we obtain:

$$\begin{aligned}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle &= \frac{1}{\eta} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \eta \mathbf{v}_t \rangle \\ &= \frac{1}{2\eta} (-\|\mathbf{w}^{(t)} - \mathbf{w}^\star - \eta \mathbf{v}_t\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2 + \eta^2 \|\mathbf{v}_t\|^2) \\ &= \frac{1}{2\eta} (-\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2) + \frac{\eta}{2} \|\mathbf{v}_t\|^2,\end{aligned}$$

SGD Analysis

where the last equality follows from the definition of the update rule. Summing the equality over t, we have

$$\sum_{t=1}^{T} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle = \frac{1}{2\eta} \sum_{t=1}^{T} \left(-\|\mathbf{w}^{(t+1)} - \mathbf{w}^\star\|^2 + \|\mathbf{w}^{(t)} - \mathbf{w}^\star\|^2 \right) + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{v}_t\|^2. \tag{14.6}$$

The first sum on the right-hand side is a telescopic sum that collapses to

$$\|\mathbf{w}^{(1)} - \mathbf{w}^\star\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^\star\|^2.$$

SGD Analysis

Plugging this in Equation (14.6), we have

$$\begin{aligned}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t\rangle &= \frac{1}{2\eta}(\|\mathbf{w}^{(1)} - \mathbf{w}^\star\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^\star\|^2) + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2 \\ &\leq \frac{1}{2\eta}\|\mathbf{w}^{(1)} - \mathbf{w}^\star\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2 \\ &= \frac{1}{2\eta}\|\mathbf{w}^\star\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{v}_t\|^2,\end{aligned}$$

where the last equality is due to the definition $\mathbf{w}^{(1)} = 0$. This proves the first part of the lemma (Equation (14.5)). The second part follows by upper bounding $\|\mathbf{w}^\star\|$ by B, $\|\mathbf{v}_t\|$ by ρ, dividing by T, and plugging in the value of η. □

SGD Analysis

- Proof of theorem:

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}[f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star)] \leq \mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^\star))\right].$$

Since Lemma 14.1 holds for any sequence $\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_T$, it applies to SGD as well. By taking expectation of the bound in the lemma we have

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t\rangle\right] \leq \frac{B\,\rho}{\sqrt{T}}. \tag{14.9}$$

It is left to show that

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^\star))\right] \leq \mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t\rangle\right], \tag{14.10}$$

SGD Analysis

Using the linearity of the expectation we have

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}\left[\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle\right]=\frac{1}{T}\sum_{t=1}^{T}\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}[\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle].$$

Next, we recall the *law of total expectation*: For every two random variables α, β, and a function g, $\mathbb{E}_{\alpha}[g(\alpha)] = \mathbb{E}_{\beta}\,\mathbb{E}_{\alpha}[g(\alpha)|\beta]$. Setting $\alpha = \mathbf{v}_{1:t}$ and $\beta = \mathbf{v}_{1:t-1}$ we get that

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{v}_{1:T}}[\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle] &= \mathop{\mathbb{E}}_{\mathbf{v}_{1:t}}[\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle] \\ &= \mathop{\mathbb{E}}_{\mathbf{v}_{1:t-1}}\mathop{\mathbb{E}}_{\mathbf{v}_{1:t}}\left[\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle \,\middle|\, \mathbf{v}_{1:t-1}\right].\end{aligned}$$

Once we know $\mathbf{v}_{1:t-1}$, the value of $\mathbf{w}^{(t)}$ is not random any more and therefore

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:t-1}}\mathop{\mathbb{E}}_{\mathbf{v}_{1:t}}\left[\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathbf{v}_t\rangle \,\middle|\, \mathbf{v}_{1:t-1}\right]=\mathop{\mathbb{E}}_{\mathbf{v}_{1:t-1}}\left\langle \mathbf{w}^{(t)}-\mathbf{w}^{\star},\mathop{\mathbb{E}}_{\mathbf{v}_t}[\mathbf{v}_t \,|\, \mathbf{v}_{1:t-1}]\right\rangle .$$

SGD Analysis

Since $\mathbf{w}^{(t)}$ only depends on $\mathbf{v}_{1:t-1}$ and SGD requires that $\mathbb{E}_{\mathbf{v}_t}[\mathbf{v}_t \,|\, \mathbf{w}^{(t)}] \in \partial f(\mathbf{w}^{(t)})$ we obtain that $\mathbb{E}_{\mathbf{v}_t}[\mathbf{v}_t \,|\, \mathbf{v}_{1:t-1}] \in \partial f(\mathbf{w}^{(t)})$. Thus,

$$\mathop{\mathbb{E}}_{\mathbf{v}_{1:\,t-1}} \langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathop{\mathbb{E}}_{\mathbf{v}_t}[\mathbf{v}_t \,|\, \mathbf{v}_{1:\,t-1}] \rangle \;\geq\; \mathop{\mathbb{E}}_{\mathbf{v}_{1:\,t-1}} [f(\mathbf{w}^{(t)}) - f(\mathbf{w}^\star)].$$

Overall, we have shown that

$$\begin{aligned} \mathop{\mathbb{E}}_{\mathbf{v}_{1:T}} [\langle \mathbf{w}^{(t)} - \mathbf{w}^\star, \mathbf{v}_t \rangle] &\geq \mathop{\mathbb{E}}_{\mathbf{v}_{1:t-1}} [f(\mathbf{w}^{(t)}) - f(\mathbf{w}^\star)] \\ &= \mathop{\mathbb{E}}_{\mathbf{v}_{1:T}} [f(\mathbf{w}^{(t)}) - f(\mathbf{w}^\star)] \,. \end{aligned}$$

Summing over t, dividing by T, and using the linearity of expectation, we get that Equation (14.10) holds, which concludes our proof. □

ACCELERATED GRADIENT DESCENT

Stochastic gradient descent

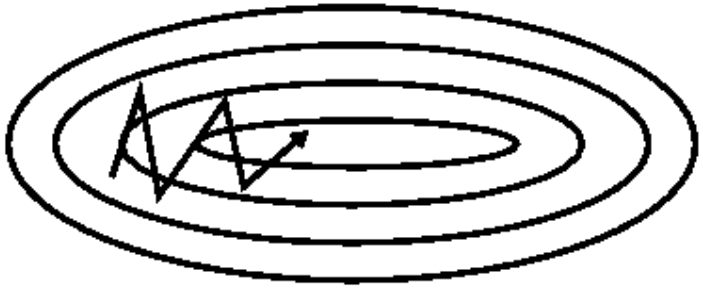
- Idea: Perform a parameter update for each training example *x(i)* and label *y(i)*
- Update: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x(i), y(i))$
- Performs redundant computations for large datasets

Momentum gradient descent

- Idea: Overcome ravine oscillations by momentum
- Update:
  - $v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta)$
  - $\theta = \theta - v_t$

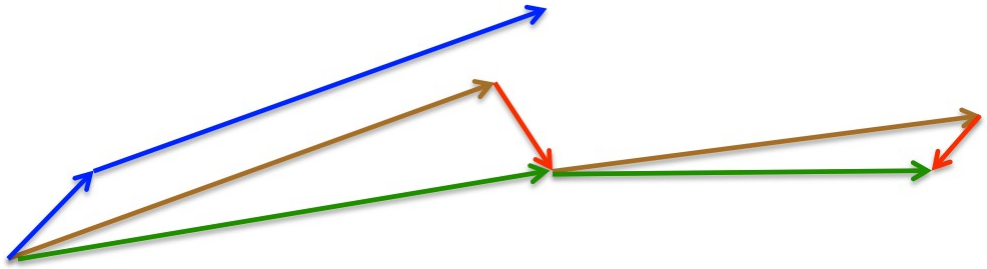
SGD

SGD with momentum



Nesterov accelerated gradient

- Ideas:
  1. Big jump in the direction of the previous accumulated gradient & measure the gradient
  2. Then make a correction.



- Update:
  - $v_t = \gamma v_{t-1} + \eta \cdot \nabla_\theta J(\theta - \gamma v_{t-1})$
  - $\theta = \theta - v_t$

AdaGrad

- Adapts the learning rate to the parameters
- Smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features
- larger updates (i.e. high learning rates) for parameters associated with infrequent features

Update:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

RMSprop

- Idea: Use the second moment of gradient vector to estimate the magnitude of update in a given direction.
- Update:
  - $E[g^2]_t = 0.9\, E[g^2]_{t-1} + 0.1\, g_t^2$
  - $\Delta\theta_t = -\eta / \sqrt{(E[g^2]_t + \epsilon)} \odot g_t$

ADAM (Adaptive moment)

- Idea: In addition to storing an exponentially decaying average of past squared gradients like RMSprop, Adam also keeps an exponentially decaying average of past gradients.
- Updates:
  - $m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$
  - $v_t = \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2$
  - $\hat{m}_t = m_t / (1 - \beta_1^t)$
  - $\hat{v}_t = v_t / (1 - \beta_2^t)$
  - $\vartheta_{t+1} = \vartheta_t - (\eta / (\sqrt{\hat{v}_t} + \epsilon))\, \hat{m}_t$

ADAM (Adaptive moment)

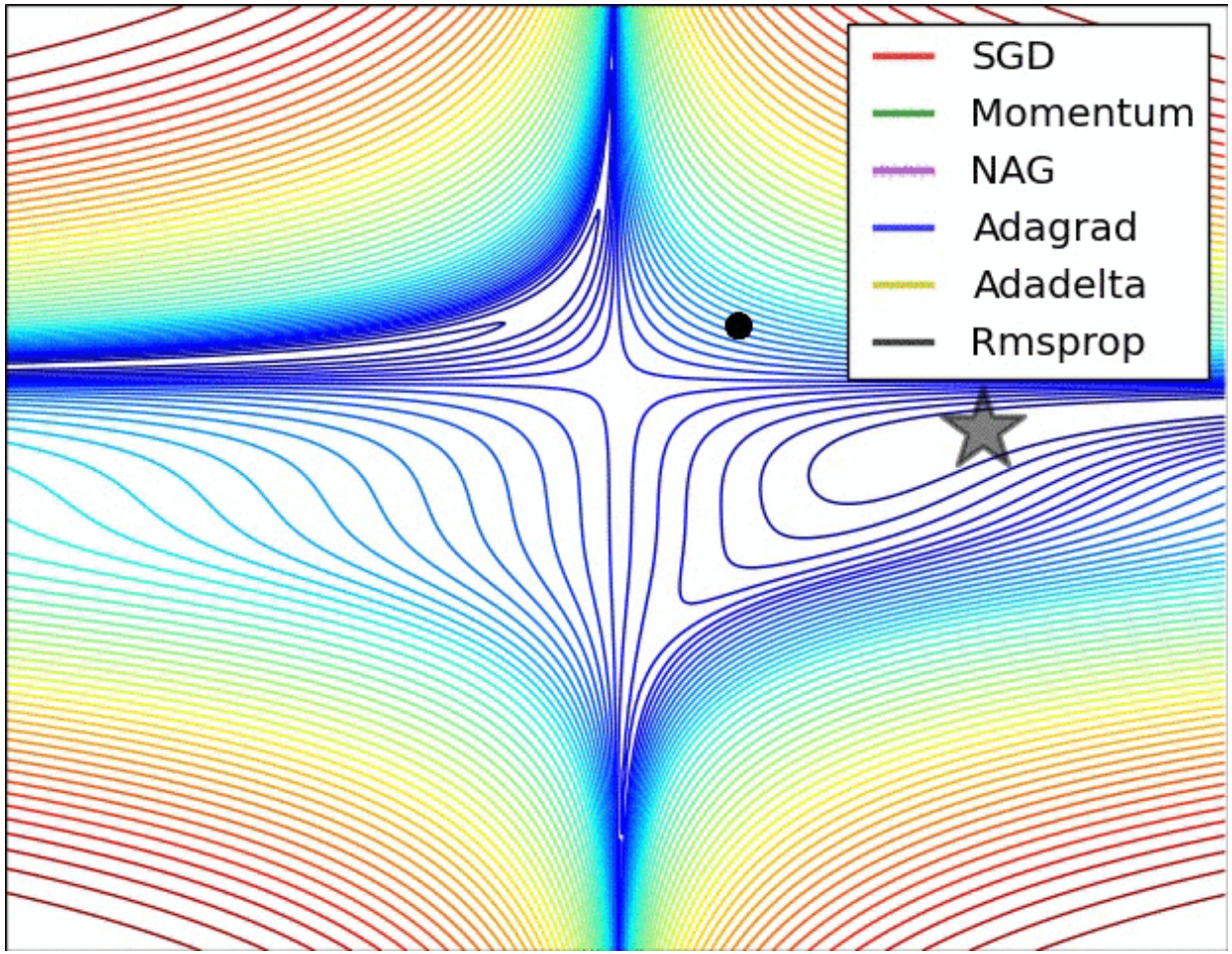
- Updates imply:

$$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2$$

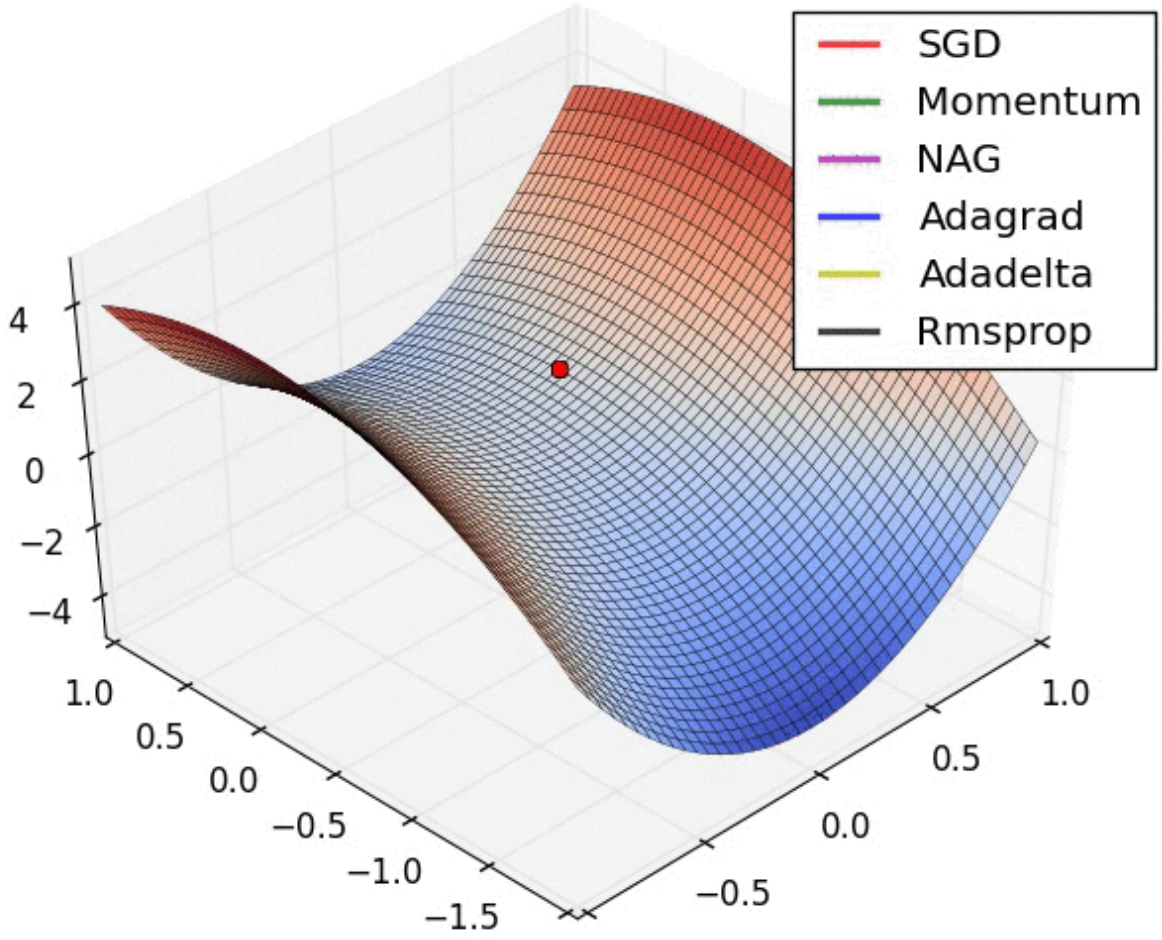
- Bias in expection:

$$\begin{aligned}
\mathbb{E}[v_t] &= \mathbb{E}\left[(1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2\right] \\
&= \mathbb{E}[g_t^2] \cdot (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \\
&= \mathbb{E}[g_t^2] \cdot (1 - \beta_2^t) + \zeta
\end{aligned}$$

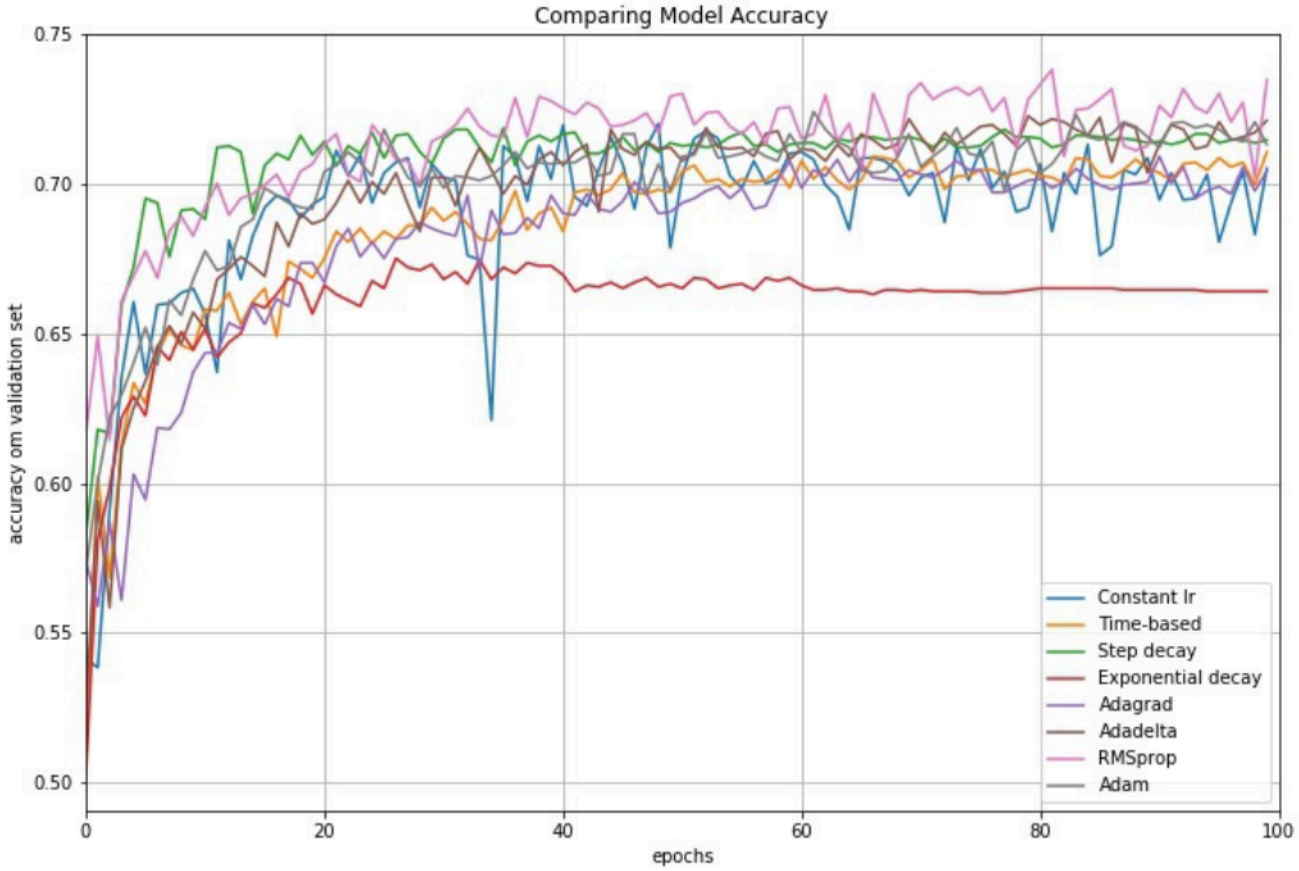
Visualization



Visualization



Enhancements comparison



LINEAR RATE METHODS

Stochastic Averaged Gradient

- **Can we have a rate of $O(\rho^t)$ with only 1 gradient evaluation per iteration?**
  - YES! The stochastic average gradient (SAG) algorithm:
    - Randomly select i_t from $\{1, 2, \ldots, N\}$ and compute $f'_{i_t}(x^t)$.

$$x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^{N} y_i^t$$

    - **Memory**: $y_i^t = \nabla f_i(x^t)$ from the last t where i was selected.
      [Le Roux et al., 2012]
  - Stochastic variant of increment average gradient (IAG).
    [Blatt et al., 2007]
  - Assumes gradients of non-selected examples don't change.
  - Assumption becomes accurate as $||x^{t+1} - x^t|| \to 0$.

Slide taken from Mark Schmidt

SAG Convergence Rate

- If each f_i' is $L-$continuous and f is strongly-convex, with $\alpha_t = 1/16L$ SAG has

$$\mathbb{E}[f(x^t) - f(x^*)] \leqslant \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8N}\right\}\right)^t C,$$

where

$$C = [f(x^0) - f(x^*)] + \frac{4L}{N}\|x^0 - x^*\|^2 + \frac{\sigma^2}{16L}.$$

- Linear convergence rate but only 1 gradient per iteration.
  - For well-conditioned problems, constant reduction per pass:

$$\left(1 - \frac{1}{8N}\right)^N \leq \exp\left(-\frac{1}{8}\right) = 0.8825.$$

  - For ill-conditioned problems, almost same as deterministic method (but N times faster).

SAG Convergence Rate

- Assume that $N = 700000$, $L = 0.25$, $\mu = 1/N$:
  - Gradient method has rate $\left(\frac{L-\mu}{L+\mu}\right)^2 = 0.99998$.
  - Accelerated gradient method has rate $\left(1 - \sqrt{\frac{\mu}{L}}\right) = 0.99761$.
  - SAG (N iterations) has rate $\left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8N}\right\}\right)^N = 0.88250$.
  - *Fastest possible* first-order method: $\left(\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)^2 = 0.99048$.
- SAG beats two lower bounds:
  - Stochastic gradient bound (of $O(1/t)$).
  - Deterministic gradient bound (for typical L, μ, and N).
- Number of f_i' evaluations to reach ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$.
  - Gradient: $O(N\frac{L}{\mu}\log(1/\epsilon))$.
  - Accelerated: $O(N\sqrt{\frac{L}{\mu}}\log(1/\epsilon))$.
  - SAG: $O(\max\{N, \frac{L}{\mu}\}\log(1/\epsilon))$.

SAG Convergence Rate

- Use SGD for well conditioned problems.
- Use Accelerated SGD for ill-conditioned problems where N is lower than $O(\sqrt{C})$.
- Otherwise use SAG.

SAG Implementation

- Basic SAG algorithm:
  - while(1)
  - Sample i from $\{1, 2, \ldots, N\}$.
  - Compute $f_i'(x)$.
  - $d = d - y_i + f_i'(x)$.
  - $y_i = f_i'(x)$.
  - $x = x - \frac{\alpha}{N} d$.
- Practical variants of the basic algorithm allow:
  - Regularization.
  - Sparse gradients.
  - Automatic step-size selection.
    - Common to use adaptive step-size procedure to estimate L.
  - Termination criterion.
    - Can use $\|x^{t+1} - x^t\|/\alpha = \frac{1}{n} d \approx \|\nabla f(x^t)\|$ to decide when to stop.
  - Acceleration [Lin et al., 2015].
  - Adaptive non-uniform sampling [Schmidt et al., 2013].

SAG Implementation

- Does re-shuffling and doing full passes work better?
  - For classic SG: Maybe?
    - Noncommutative arithmetic-geometric mean inequality conjecture.

      [Recht & Ré, 2012]
  - For SAG: NO.
  - Performance is intermediate between IAG and SAG.
- Can non-uniform sampling help?
  - For classic SG methods, can only improve constants.
  - For SAG, bias sampling towards Lipschitz constants L_i,

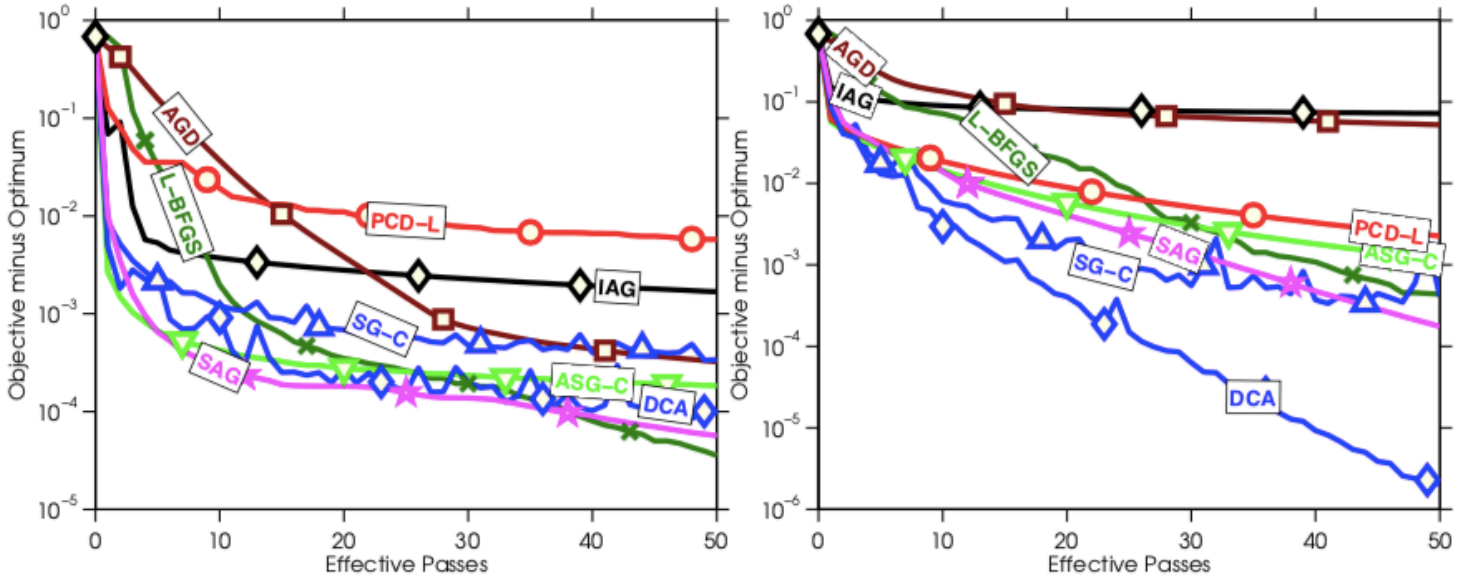
    $$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L_i \|x - y\|.$$

    improves rate to depend on L_{mean} instead of L_{max}.

    (with bigger step size)
  - Adaptively estimate L_i as you go. (see paper/code).
  - Slowly learns to ignore well-classified examples.

SAG with Adaptive Non-Uniform Sampling

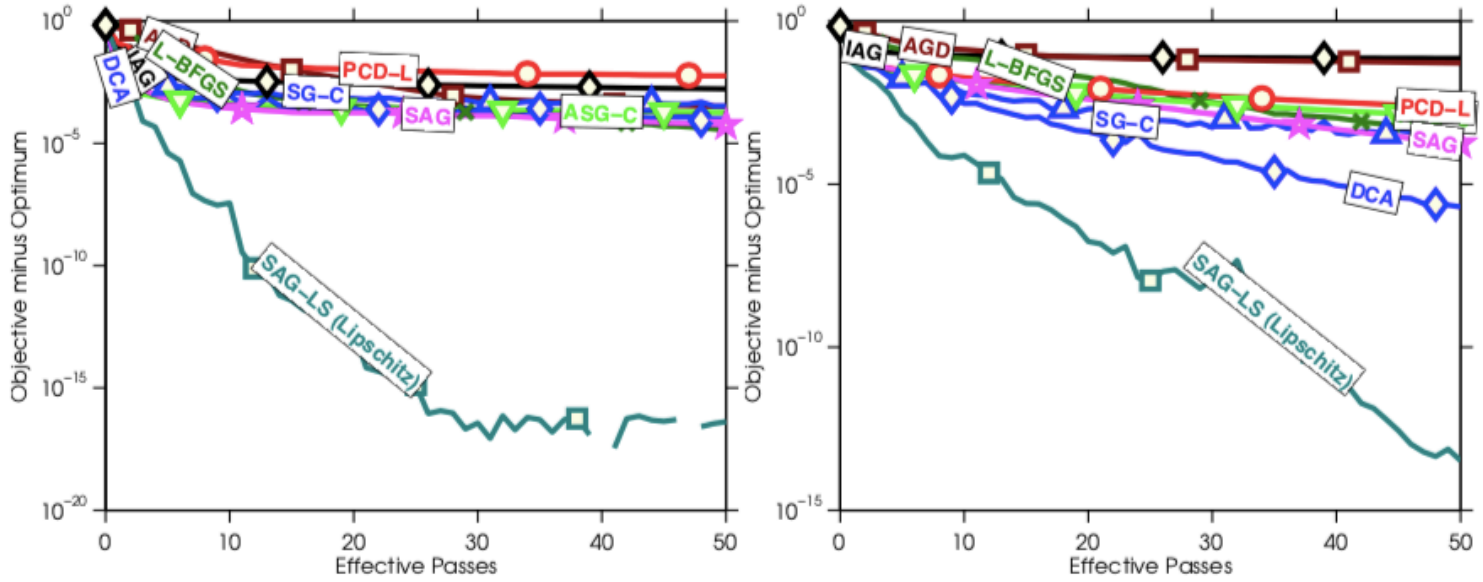
- protein ($n = 145751$, $p = 74$) and sido ($n = 12678$, $p = 4932$)



- Datasets where SAG had the worst relative performance.

SAG with Non-Uniform Sampling

- protein $(n = 145751,\ p = 74)$ and sido $(n = 12678,\ p = 4932)$



- Adaptive non-uniform sampling helps a lot.

Stochastic Variance Reduced GD

SVRG algorithm:

- Start with x_0
- for $s = 0, 1, 2 \ldots$
  - $d_s = \frac{1}{N} \sum_{i=1}^{N} f_i'(x_s)$
  - $x^0 = x_s$
  - for $t = 1, 2, \ldots m$
    - Randomly pick $i_t \in \{1, 2, \ldots, N\}$
    - $x^t = x^{t-1} - \alpha_t (f_{i_t}'(x^{t-1}) - f_{i_t}'(x_s) + d_s)$.
  - $x_{s+1} = x^t$ for random $t \in \{1, 2, \ldots, m\}$.

Requires 2 gradients per iteration and occasional full passes,
but only requires storing d_s and x_s.

Practical issues similar to SAG (acceleration versions, automatic step-size/termination, handles sparsity/regularization, non-uniform sampling, mini-batches).

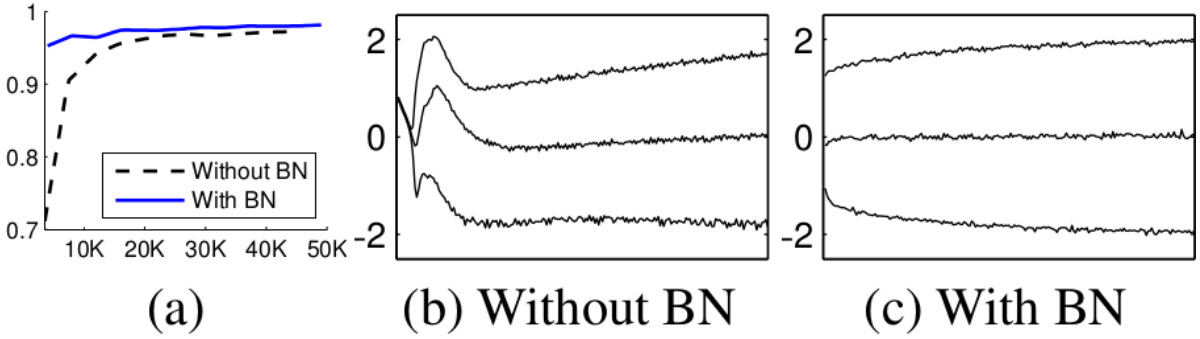
BATCH NORMALIZATION

Slides taken from Jude Shavlik: http://pages.cs.wisc.edu/~shavlik/cs638_cs838.html

Batch normalization: Other benefits in practice

- BN reduces training times. (Because of less Covariate Shift, less exploding/vanishing gradients.)
- BN reduces demand for regularization, e.g. dropout or L2 norm.
  - Because the means and variances are calculated over batches and therefore every normalized value depends on the current batch. I.e. the network can no longer just memorize values and their correct answers.)
- BN allows higher learning rates. (Because of less danger of exploding/vanishing gradients.)
- BN enables training with saturating nonlinearities in deep networks, e.g. sigmoid. (Because the normalization prevents them from getting stuck in saturating ranges, e.g. very high/low values for sigmoid.)

Batch normalization: Better accuracy , faster.



BN applied to MNIST (a), and activations of a randomly selected neuron over time (b, c), where the middle line is the median activation, the top line is the 15th percentile and the bottom line is the 85th percentile.

Why the naïve approach Does not work?

- Normalizes layer inputs to zero mean and unit variance. *whitening*.
- Naive method: Train on a batch. Update model parameters. Then normalize. Doesn't work: Leads to exploding biases while distribution parameters (mean, variance) don't change.

  - If we do it this way gradient always ignores the effect that the normalization for the next batch would have
  - i.e. : "The issue with the above approach is that the gradient descent optimization does not take into account the fact that the normalization takes place"

Doing it the “correct way” Is too expensive!

- A proper method has to include the current example batch *and* somehow all previous batches (all examples) in the normalization step.
- This leads to calculating in covariance matrix and its inverse square root. That's expensive. The authors found a faster way!

The issue with the above approach is that the gradient descent optimization does not take into account the fact that the normalization takes place. To address this issue, we would like to ensure that, for any parameter values, the network *always* produces activations with the desired distribution. Doing so would allow the gradient of the loss with respect to the model parameters to account for the normalization, and for its dependence on the model parameters Θ. Let again x be a layer input, treated as a vector, and $\mathcal{X}$ be the set of these inputs over the training data set. The normalization can then be written as a transformation

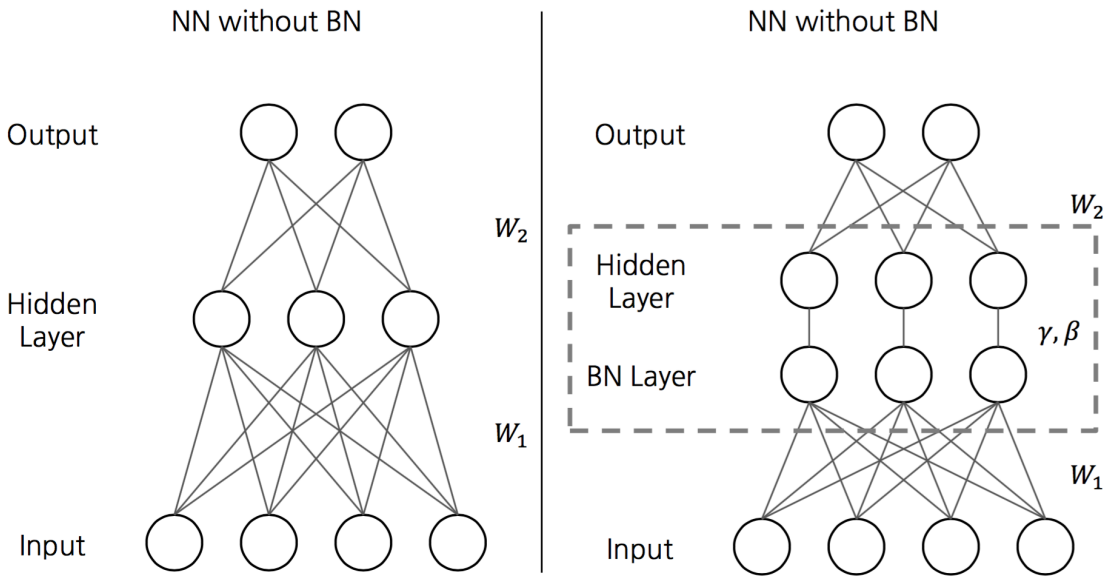
$$\hat{\mathrm{x}} = \mathrm{Norm}(\mathrm{x}, \mathcal{X})$$

which depends not only on the given training example x but on all examples $\mathcal{X}$ – each of which depends on Θ if x is generated by another layer. For backpropagation, we would need to compute the Jacobians $\frac{\partial \mathrm{Norm}(\mathrm{x}, \mathcal{X})}{\partial \mathrm{x}}$ and $\frac{\partial \mathrm{Norm}(\mathrm{x}, \mathcal{X})}{\partial \mathcal{X}}$; ignoring the latter term would lead to the explosion described above. Within this framework, whitening the layer inputs is expensive, as it requires computing the covariance matrix $\mathrm{Cov}[\mathrm{x}] = \mathrm{E}_{\mathrm{x} \in \mathcal{X}}[\mathrm{x}\mathrm{x}^T] - \mathrm{E}[\mathrm{x}]\mathrm{E}[\mathrm{x}]^T$ and its inverse square root, to produce the whitened activations $\mathrm{Cov}[\mathrm{x}]^{-1/2}(\mathrm{x} - \mathrm{E}[\mathrm{x}])$, as well as the derivatives of these transforms for backpropagation. This motivates us to seek an alternative that performs input normalization in a way that is differentiable and does not require the analysis of the entire training set after every parameter update.

The proposed solution: To add an extra regu

we introduce, for each activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}, \beta^{(k)}$, which scale and shift the normalized value:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}.$$



A new layer is added so the gradient can “see” the normalization and make adjustments if needed.

Algorithm Summary: Normalization via Mini-Batch Statistics

- Each feature (component) is normalized individually
- Normalization according to:
  - componentNormalizedValue = (componentOldValue - E[component]) / sqrt(Var(component))
- A new layer is added so the gradient can "see" the normalization and made adjustments if needed.
  - The new layer has the power to learn the identity function to de-normalize the features if necessary!
  - Full formula: newValue = gamma * componentNormalizedValue + beta (gamma and beta learned per component)
- E and Var are estimated for each mini batch.
- BN is fully differentiable.

The Batch Transformation: formally from the paper.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x_i} \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x_i} + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

The full algorithm as proposed in the paper

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

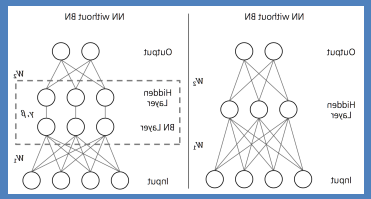
$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

Alg 1 (previous slide)

Architecture modificati



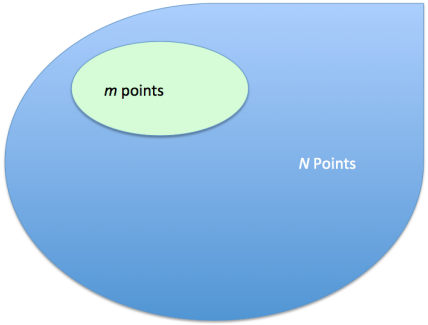
Note that BN(x) is differe during test…

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

Vs.

$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

Populations stats vs. sample stats



- In algorithm 1, we are estimating the true mean and variance over the entire population for a given batch.

- When doing inference you're minibatching your way through the entire dataset, you're calculating statistics on a per sample/batch basis. We want our sample statistics to be *unbiased* to population statistics.

	Population Statistics over *N*	Sample/Batch Statistics over *m*
Mean Estimate	$\mu = \frac{1}{N}\sum_i x_i$	$\bar{x} = \frac{1}{m}\sum_i x_i$
Variance Estimate	$\sigma = \frac{1}{N}\sum_i (x_i - \mu)^2$	$\sigma_B = \frac{1}{m-1}\sum_i (x_i - \bar{x})^2$

ACCELERATING BN NETWORKS

Batch normalization only not enough!

- Increase learning rate.
- Remove Dropout.
- Shuffle training examples more thoroughly
- Reduce the L2 weight regularization.
- Accelerate the learning rate decay.
- Reduce the photometric distortions.

References:

- SGD proof by Yuri Nesterov.
- MMDS http://www.mmds.org/
- Mini-course by Mark Schmidt: https://www.cs.ubc.ca/~schmidtm/SVAN16/
- *Blog of Sebastian Ruder http://ruder.io/optimizing-gradient-descent/*
- *Learning rate comparison https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1*