Non-Uniform SAG for Training CRFs

Mark Schmidt, Reza Babanezhad, Mohamed Ahmed
Ann Clifton, Anoop Sarkar, Aaron Defazio

University of British Columbia, Simon Fraser University

NIPS Structured Learning Workshop, 2016

Motivation: Structured Prediction

Classical supervised learning:

Input: 

Output: "P"

Motivation: Structured Prediction

Classical supervised learning:

Input: 

Output: "P"

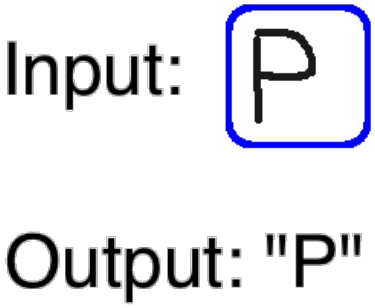
Structured prediction:

Input: 

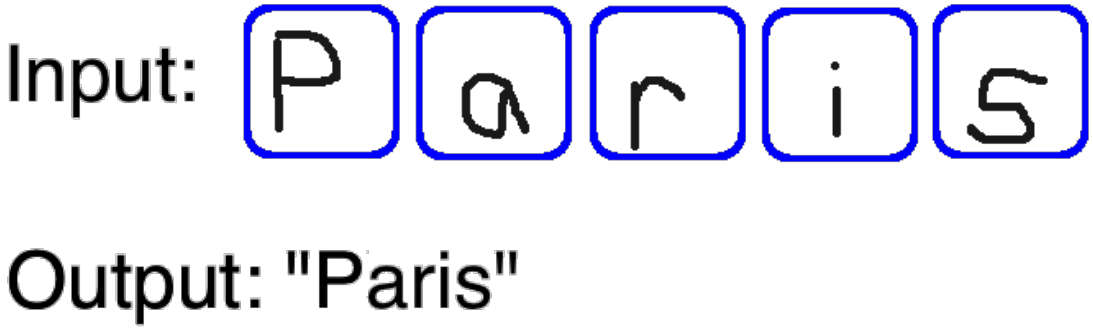
Output: "Paris"

Motivation: Structured Prediction

Classical supervised learning:



Structured prediction:



Other structure prediction tasks:

- Labelling all people/places in Wikiepdia, finding coding regions in DNA sequences, labelling all voxels in an MRI as normal or tumor, predicting protein structure from sequence, weather forecasting, translating from French to English, etc.

Motivation: Structured Prediction

Naive approaches to predicting letters y given images x:

- Multinomial logistic regression to predict word:

$$p(y|x, w) = \frac{\exp(w_y^T F(x))}{\sum_{y'} \exp(w_{y'}^T F(x))}.$$

Motivation: Structured Prediction

Naive approaches to predicting letters y given images x:

- Multinomial logistic regression to predict word:

$$p(y|x, w) = \frac{\exp(w_y^T F(x))}{\sum_{y'} \exp(w_{y'}^T F(x))}.$$

This requires parameter vector w_k for all possible words k.

Motivation: Structured Prediction

Naive approaches to predicting letters y given images x:

- Multinomial logistic regression to predict word:

$$p(y|x,w) = \frac{\exp(w_y^T F(x))}{\sum_{y'} \exp(w_{y'}^T F(x))}.$$

  This requires parameter vector w_k for all possible words k.

- Multinomial logistic regression to predict each letter:

$$p(y_j|x_j,w) = \frac{\exp(w_{y_j}^T F(x_j))}{\sum_{y_j'} \exp(w_{y_j'}^T F(x_j))}.$$

  This works if you are really good at predicting individual letters.

Motivation: Structured Prediction

Naive approaches to predicting letters y given images x:

- Multinomial logistic regression to predict word:

$$p(y|x,w) = \frac{\exp(w_y^T F(x))}{\sum_{y'} \exp(w_{y'}^T F(x))}.$$

  This requires parameter vector w_k for all possible words k.

- Multinomial logistic regression to predict each letter:

$$p(y_j|x_j,w) = \frac{\exp(w_{y_j}^T F(x_j))}{\sum_{y_j'} \exp(w_{y_j'}^T F(x_j))}.$$

  This works if you are really good at predicting individual letters.

  But this ignores dependencies between letters.

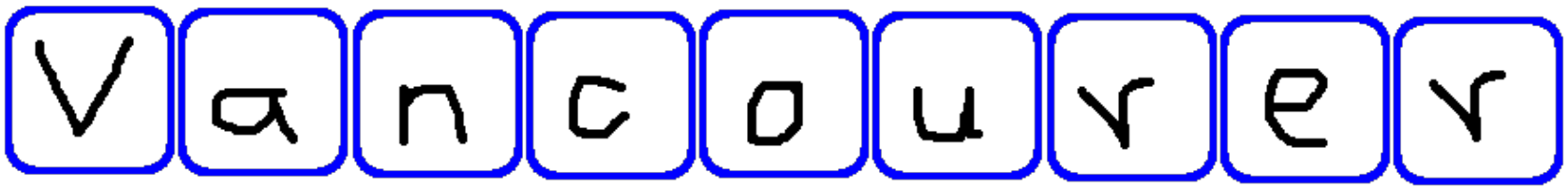
Motivation: Structured Prediction

- What letter is this?

Motivation: Structured Prediction

- What letter is this?

- What are these letters?



Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

where w are the parameters.

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.
- Examples of features $F(y,x)$:
  - $F(y_j,x)$: these features lead to a logistic model for each letter.

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.
- Examples of features $F(y,x)$:
  - $F(y_j,x)$: these features lead to a logistic model for each letter.
  - $F(y_{j-1},y_j,x)$: dependency between adjacent letters ('q-u').

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.
- Examples of features $F(y,x)$:
  - $F(y_j,x)$: these features lead to a logistic model for each letter.
  - $F(y_{j-1},y_j,x)$: dependency between adjacent letters ('q-u').
  - $F(y_{j-1},y_j,j,x)$: position-based dependency (French: 'e-r' ending).

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.

- Examples of features $F(y,x)$:
  - $F(y_j, x)$: these features lead to a logistic model for each letter.
  - $F(y_{j-1}, y_j, x)$: dependency between adjacent letters ('q-u').
  - $F(y_{j-1}, y_j, j, x)$: position-based dependency (French: 'e-r' ending).
  - $F(y_{j-2}, y_{j-1}, y_j, j, x)$: third-order and position (English: 'i-n-g' end).

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.

- Examples of features $F(y,x)$:
  - $F(y_j,x)$: these features lead to a logistic model for each letter.
  - $F(y_{j-1},y_j,x)$: dependency between adjacent letters ('q-u').
  - $F(y_{j-1},y_j,j,x)$: position-based dependency (French: 'e-r' ending).
  - $F(y_{j-2},y_{j-1},y_j,j,x)$: third-order and position (English: 'i-n-g' end).
  - $F(y \in \mathcal{D},x)$: is y in dictionary $\mathcal{D}$?

Conditional Random Fields

- Conditional random fields model targets y given inputs x using

$$p(y|x,w) = \frac{\exp(w^T F(y,x))}{\sum_{y'} \exp(w^T F(y,x))} = \frac{\exp(w^T F(y,x))}{Z}.$$

  where w are the parameters.
- Examples of features $F(y,x)$:
  - $F(y_j,x)$: these features lead to a logistic model for each letter.
  - $F(y_{j-1},y_j,x)$: dependency between adjacent letters ('q-u').
  - $F(y_{j-1},y_j,j,x)$: position-based dependency (French: 'e-r' ending).
  - $F(y_{j-2},y_{j-1},y_j,j,x)$: third-order and position (English: 'i-n-g' end).
  - $F(y \in \mathcal{D},x)$: is y in dictionary $\mathcal{D}$?
- CRFs are a ubiquitous tool in natural language processing:
  - Part-of-speech tagging, semantic role labelling, information extraction, shallow parsing, named-entity recognition, etc.

Optimization Formulation and Challenge

- Typically train using ℓ_2-regularized negative log-likelihood:

$$\min_w f(w) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|x_i, w).$$

Optimization Formulation and Challenge

- Typically train using ℓ_2-regularized negative log-likelihood:

$$\min_w f(w) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|x_i, w).$$

- Good news: $\nabla f(w)$ is Lipschitz-continuous, f is strongly-convex.

Optimization Formulation and Challenge

- Typically train using ℓ_2-regularized negative log-likelihood:

$$\min_w f(w) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|x_i, w).$$

- Good news: $\nabla f(w)$ is Lipschitz-continuous, f is strongly-convex.
- Bad news: evaluating $\log p(y_i|x_i, w)$ and its gradient is expensive.

Optimization Formulation and Challenge

- Typically train using ℓ_2-regularized negative log-likelihood:

$$\min_w f(w) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|x_i, w).$$

- Good news: $\nabla f(w)$ is Lipschitz-continuous, f is strongly-convex.
- Bad news: evaluating $\log p(y_i|x_i, w)$ and its gradient is expensive.
  - Chain-structures: run forward-backward on each example.

Optimization Formulation and Challenge

- Typically train using ℓ_2-regularized negative log-likelihood:

$$\min_w f(w) = \frac{\lambda}{2}\|w\|^2 - \frac{1}{n}\sum_{i=1}^{n} \log p(y_i|x_i, w).$$

- Good news: $\nabla f(w)$ is Lipschitz-continuous, f is strongly-convex.
- Bad news: evaluating $\log p(y_i|x_i, w)$ and its gradient is expensive.
  - Chain-structures: run forward-backward on each example.
  - General features: exponential in tree-width of dependency graph.
  - A lot of work on approximate evaluation.
- This optimization problem remains a bottleneck.

Current Optimization Methods

- Lafferty et al. [2001] proposed an iterative scaling approach.
- Outperformed by L-BFGS quasi-Newton algorithm.

[Wallach, 2002, Sha Pereira, 2003]

  - Has a linear convergence rate: $O(\log(1/\epsilon))$ iterations required.

Current Optimization Methods

- Lafferty et al. [2001] proposed an iterative scaling approach.
- Outperformed by L-BFGS quasi-Newton algorithm.

[Wallach, 2002, Sha Pereira, 2003]

  - Has a linear convergence rate: $O(\log(1/\epsilon))$ iterations required.
  - But each iteration requires $\log p(y_i|x_i, w)$ for all n examples.

Current Optimization Methods

- Lafferty et al. [2001] proposed an iterative scaling approach.
- Outperformed by L-BFGS quasi-Newton algorithm.

[Wallach, 2002, Sha Pereira, 2003]

  - Has a linear convergence rate: $O(\log(1/\epsilon))$ iterations required.
  - But each iteration requires $\log p(y_i|x_i, w)$ for all n examples.

- To scale to large n, we looked at stochastic gradient methods.

[Vishwanathan et al., 2006]

  - Iteration cost is independent of n.

Current Optimization Methods

- Lafferty et al. [2001] proposed an iterative scaling approach.
- Outperformed by L-BFGS quasi-Newton algorithm.

[Wallach, 2002, Sha Pereira, 2003]

  - Has a linear convergence rate: $O(\log(1/\epsilon))$ iterations required.
  - But each iteration requires $\log p(y_i|x_i, w)$ for all n examples.

- To scale to large n, we looked at stochastic gradient methods.

[Vishwanathan et al., 2006]

  - Iteration cost is independent of n.
  - But has a sub linear convergence rate: $O(1/\epsilon)$ iterations required.
  - Or with constant step-size you get linear rate up to fixed tolerance.

Current Optimization Methods

- Lafferty et al. [2001] proposed an iterative scaling approach.
- Outperformed by L-BFGS quasi-Newton algorithm.

[Wallach, 2002, Sha Pereira, 2003]

  - Has a linear convergence rate: $O(\log(1/\epsilon))$ iterations required.
  - But each iteration requires $\log p(y_i|x_i, w)$ for all n examples.

- To scale to large n, we looked at stochastic gradient methods.

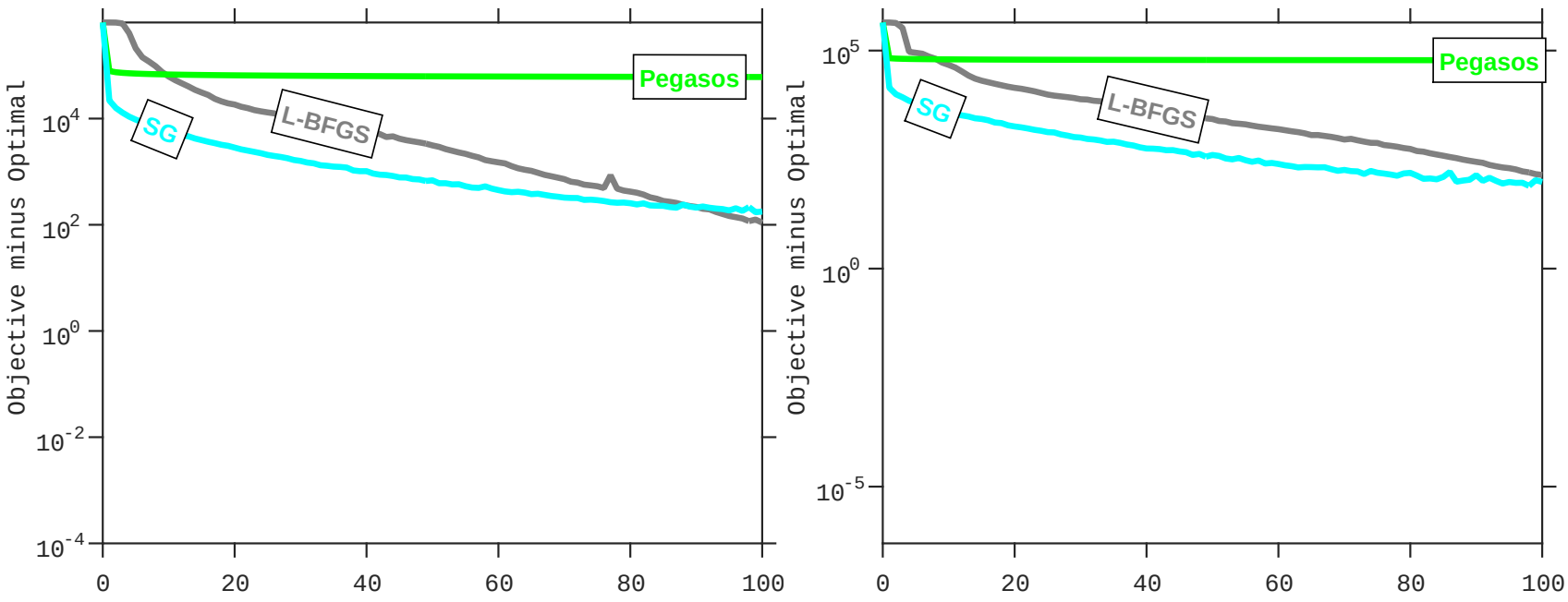
[Vishwanathan et al., 2006]

  - Iteration cost is independent of n.
  - But has a sub linear convergence rate: $O(1/\epsilon)$ iterations required.
  - Or with constant step-size you get linear rate up to fixed tolerance.

- These remain the strategies used by most implementations.
  - Many packages implement both strategies.
  - My codes still use L-BFGS because it's easier to tune.

L-BFGS vs. Stochastic Gradient

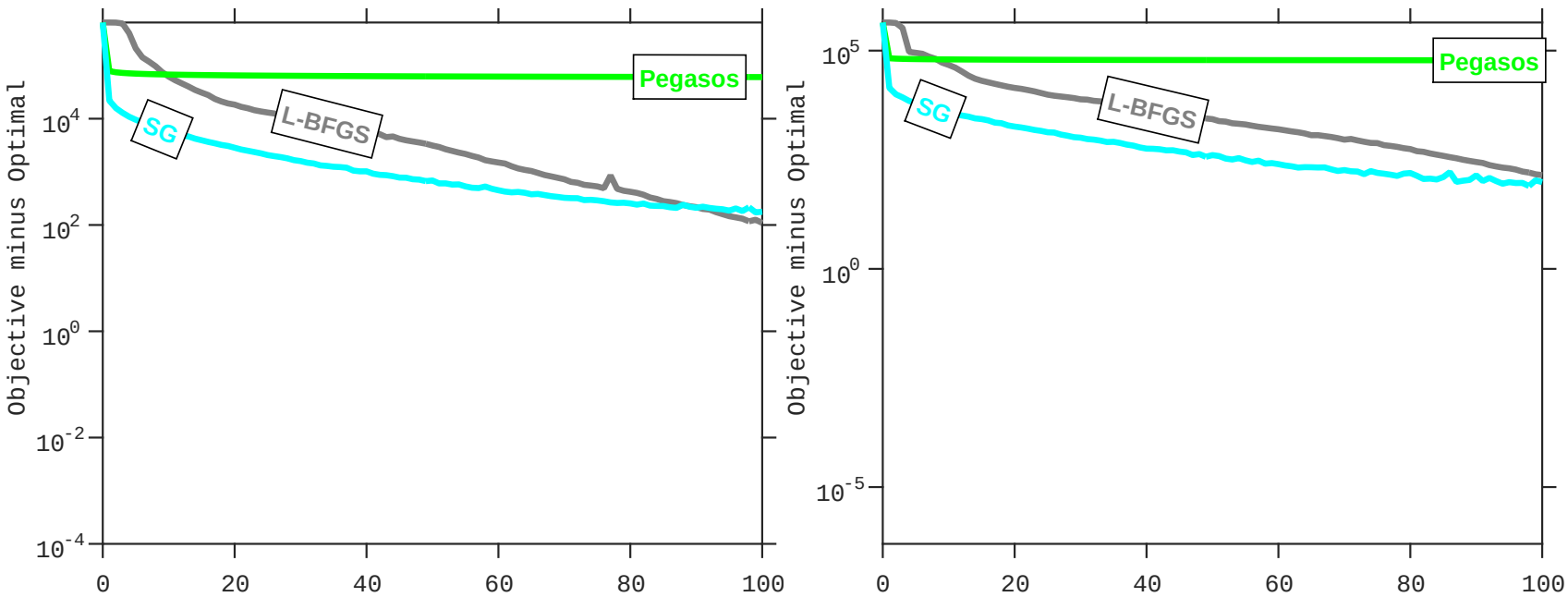
- L-BFGS has fast convergence but slow iterations.
- SG (decreasing α) has slow convergence but fast iterations.
- SG (constant α) has fast convergence but not to optimal.



(Using $\alpha_t = \alpha/(\delta + \sqrt{t})$ gives intermediate performance.)

L-BFGS vs. Stochastic Gradient

- L-BFGS has fast convergence but slow iterations.
- SG (decreasing α) has slow convergence but fast iterations.
- SG (constant α) has fast convergence but not to optimal.



(Using $\alpha_t = \alpha/(\delta + \sqrt{t})$ gives intermediate performance.)

- Can we develop a method that outperforms these methods?

Better Stochastic Gradient Methods?

- 2007: summer project with Kevin Swersky on improving SG.
  - ASG: averaged stochastic gradient with large step-sizes.

    [Polyak & Juditsky, 1992, Bach & Moulines, 2011]
    - Typically outperform non-averaged SG, doesn't always beat L-BFGS.
  - Rejected NIPS paper.

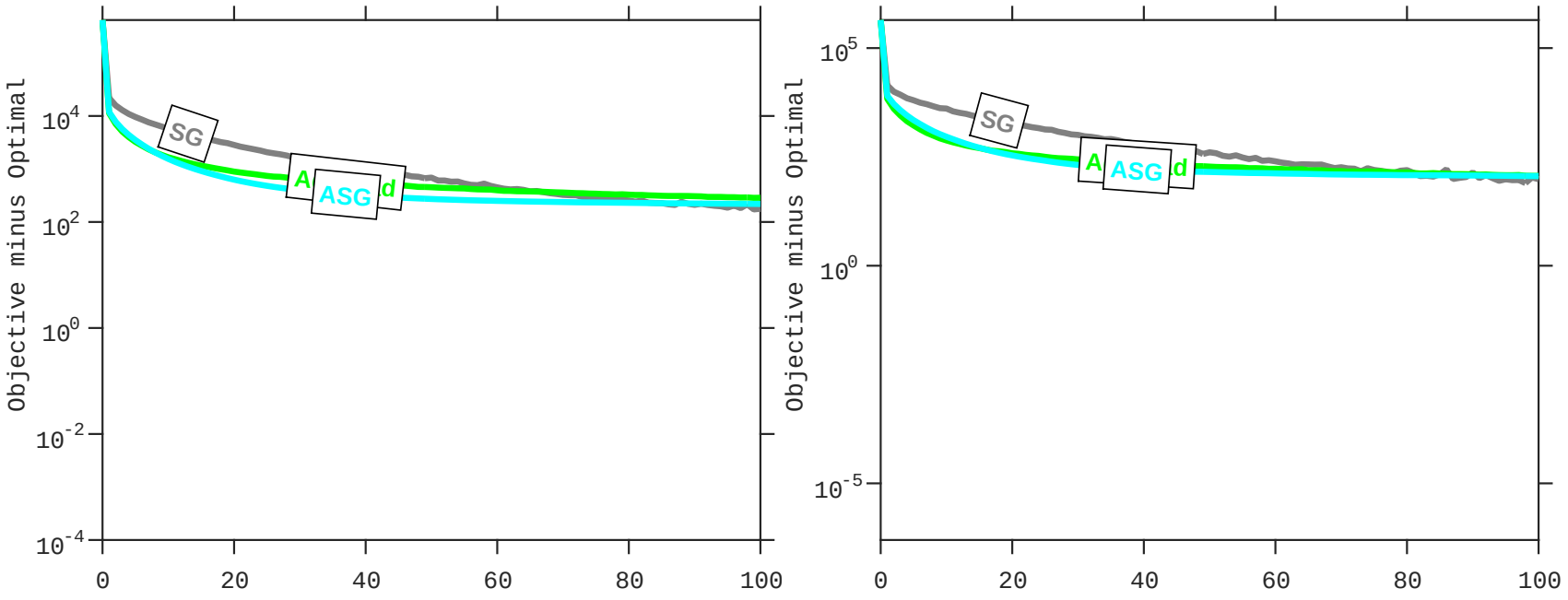
Better Stochastic Gradient Methods?

- 2007: summer project with Kevin Swersky on improving SG.
  - ASG: averaged stochastic gradient with large step-sizes.

    [Polyak & Juditsky, 1992, Bach & Moulines, 2011]
    - Typically outperform non-averaged SG, doesn't always beat L-BFGS.
  - Rejected NIPS paper.
- 2010: methods with improved regret.
  - AdaGrad: adaptive diagonal scaling. [Duchi et al., 2010]
    - Often improves performance over basic stochastic gradient.
    - Still has $O(1/\epsilon)$ rate and typically outperformed by ASG.

Comparison of Stochastic Gradient Methods

- Comparison of Pegasos, SG, ASG, and AdaGrad:

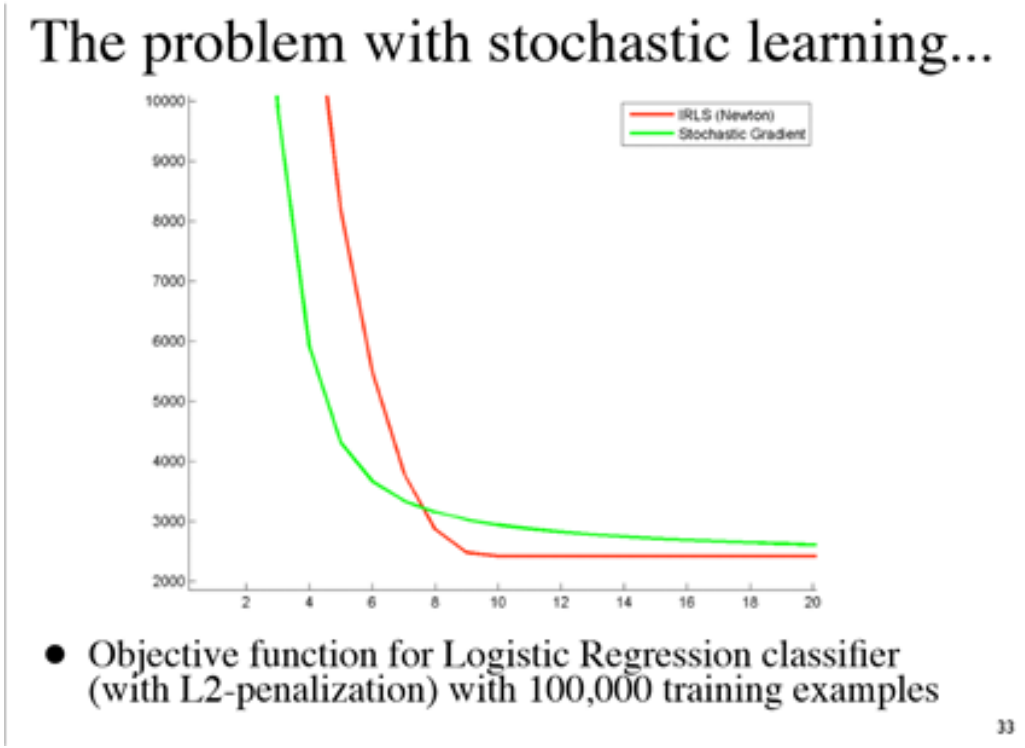


(Averaging did not improve performance of Pegasos.)

- ASG often outperforms SG and AdaGrad.

Motivation for New Methods

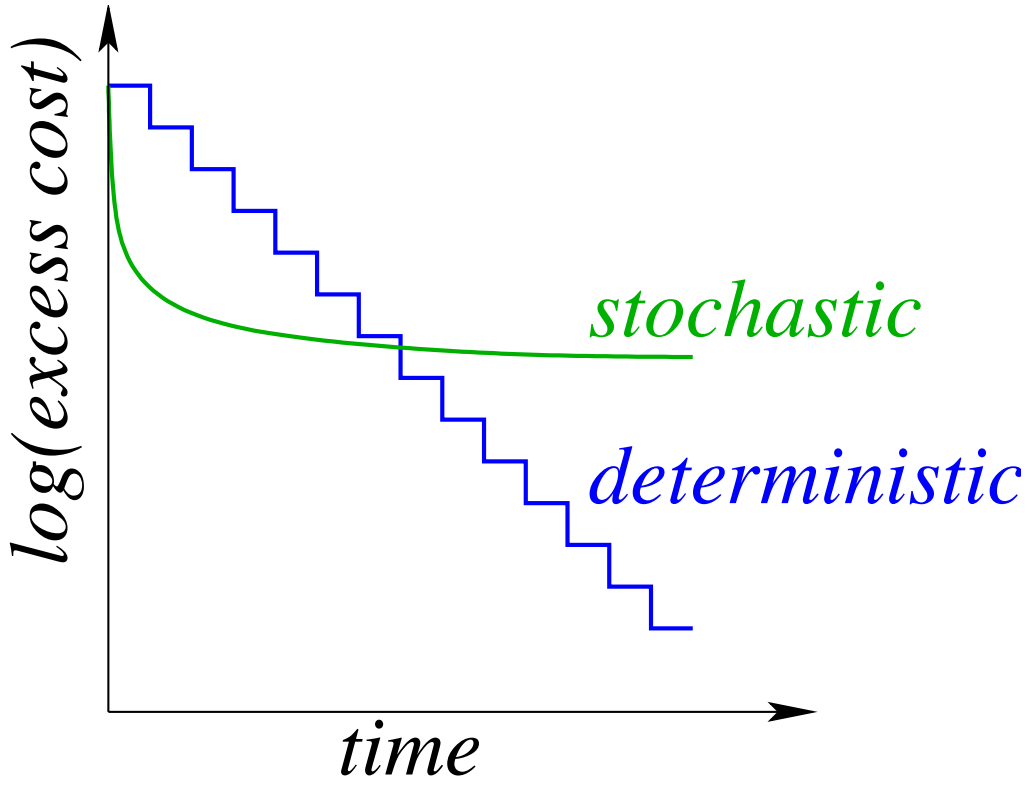
- 2008: proposed to explore hybrid methods in my PhD proposal:



- Also rejected! “Too hard, focus on existing projects”.

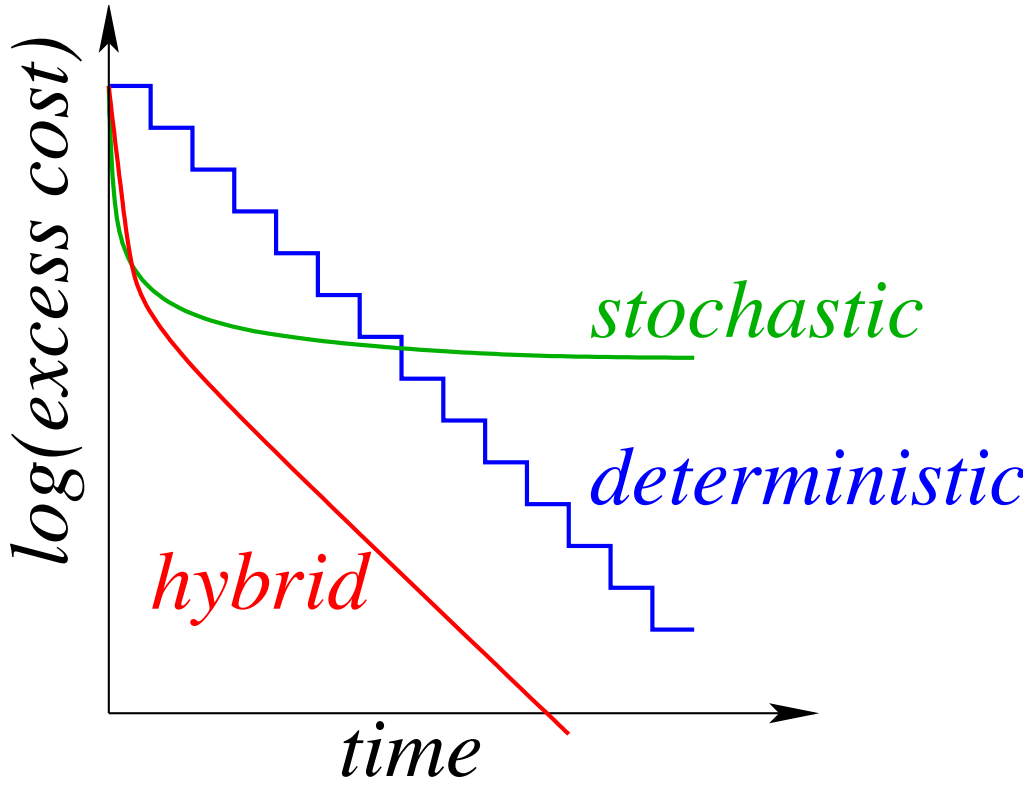
Motivation for New Methods

- Deterministic methods requires $O(\log(1/\epsilon))$ with $O(N)$.
- Stochastic methods requires $O(1/\epsilon)$ iterations with $O(1)$.



Motivation for New Methods

- Deterministic methods requires $O(\log(1/\epsilon))$ with $O(N)$.
- Stochastic methods requires $O(1/\epsilon)$ iterations with $O(1)$.



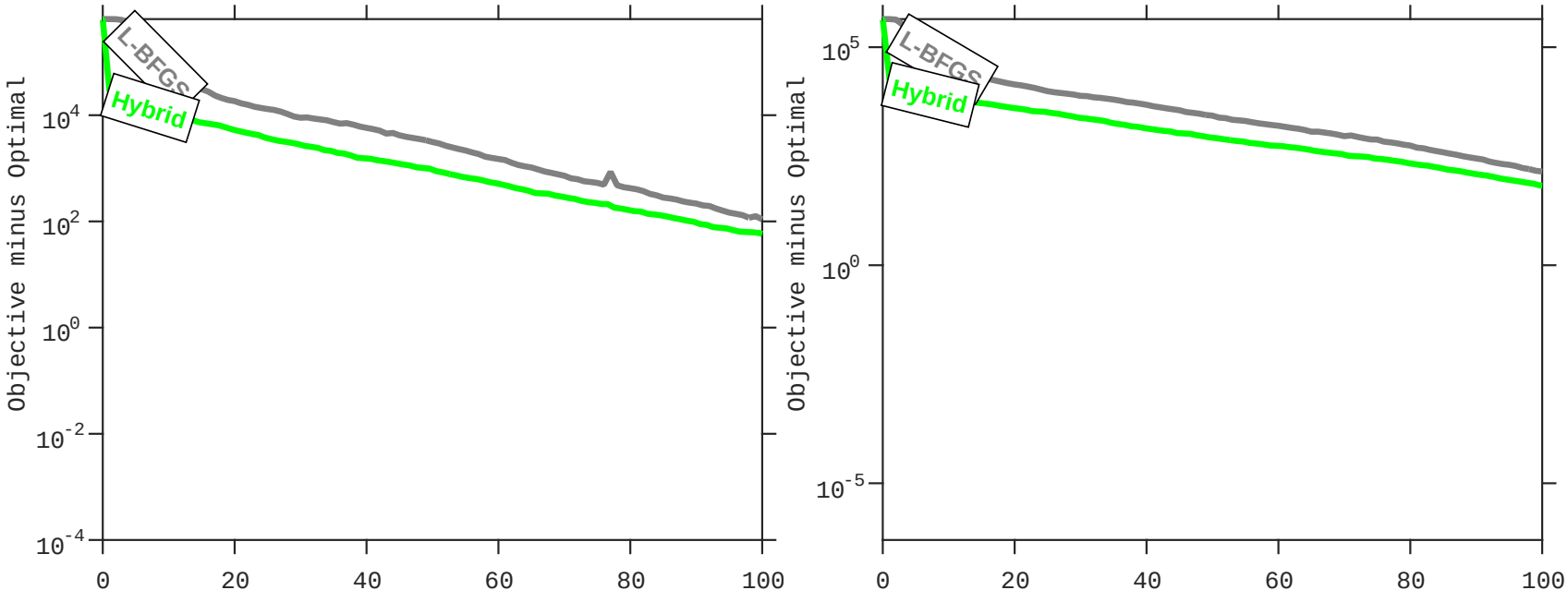
- Can we develop a method that outperforms both?

Comparison of L-BFGS Methods

- 2010: Hybrid of L-BFGS and stochastic gradient.

[Frielander & Schmidt, 2012]

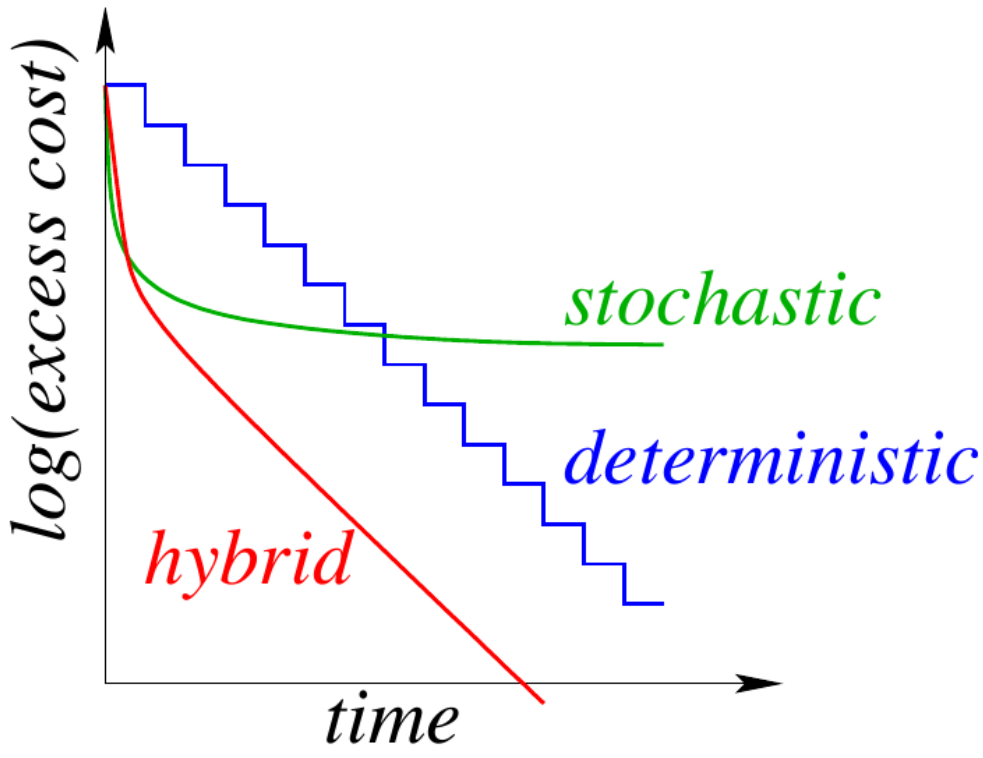
  - Key idea: control variance of gradient by growing batch size.
  - $O(\log(1/\epsilon))$ rate but cheaper in early iterations.



- Hybrid often outperforms L-BFGS, but not by very much.

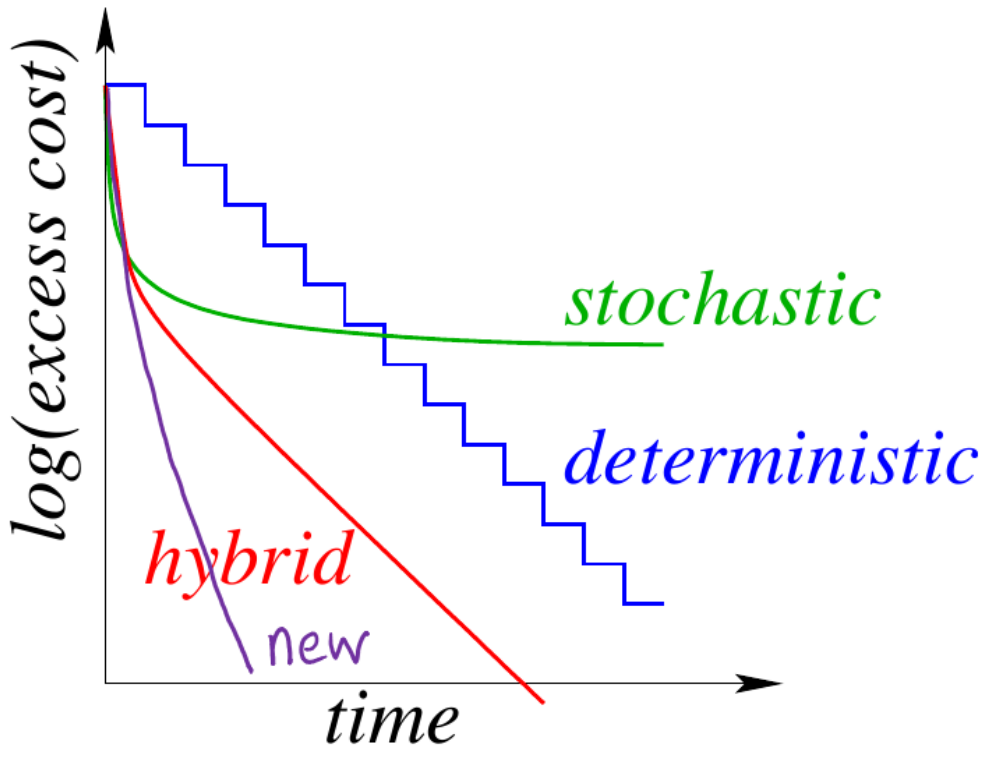
Motivation for New Methods

- Deterministic methods requires $O(\log(1/\epsilon))$ with $O(N)$.
- Stochastic methods requires $O(1/\epsilon)$ iterations with $O(1)$.



Motivation for New Methods

- Deterministic methods requires $O(\log(1/\epsilon))$ with $O(N)$.
- Stochastic methods requires $O(1/\epsilon)$ iterations with $O(1)$.



- Can we have $O(1)$ cost and only $O(\log(1/\epsilon))$ iterations?

Online Exponentiated Gradient

- OEG: online exponentiated gradient.

[Collin et al., 2008]

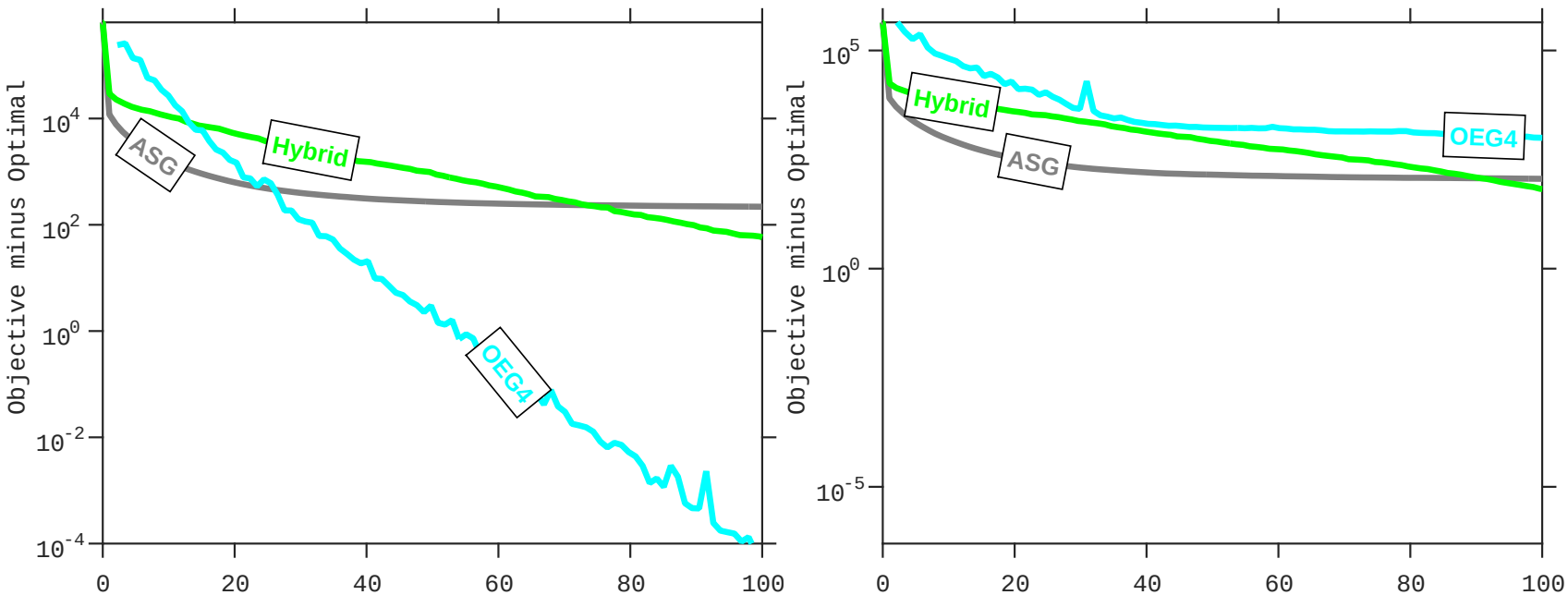
  - $O(\log(1/\epsilon))$ iterations for dual problem with $O(1)$ cost.
  - In theory, the rate of deterministic with the cost of stochastic.

Online Exponentiated Gradient

- OEG: online exponentiated gradient.

[Collin et al., 2008]

  - $O(\log(1/\epsilon))$ iterations for dual problem with $O(1)$ cost.
  - In theory, the rate of deterministic with the cost of stochastic.
  - Sometimes great and sometimes poor performance.



- Best of hybrid vs. ASG vs. OEG is problem dependent.
- Fancier methods do not give consistent/significant improvement.

A New Hope: Linearly-Convergent Stochastic Gradient

- Recent new stochastic algorithms for minimizing finite sums,

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  requiring $O(\log(1/\epsilon))$ iterations with $O(1)$ cost.

A New Hope: Linearly-Convergent Stochastic Gradient

- Recent new stochastic algorithms for minimizing finite sums,

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  requiring $O(\log(1/\epsilon))$ iterations with $O(1)$ cost.

- Stochastic average gradient (SAG): [Le Roux et al., 2012]

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

  where iteration sets $s_i^t = \nabla f_i(x^t)$ for random i (o.w., $s_i^t = s_i^{t-1}$).

A New Hope: Linearly-Convergent Stochastic Gradient

- Recent new stochastic algorithms for minimizing finite sums,

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  requiring $O(\log(1/\epsilon))$ iterations with $O(1)$ cost.

- Stochastic average gradient (SAG): [Le Roux et al., 2012]

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

  where iteration sets $s_i^t = \nabla f_i(x^t)$ for random i (o.w., $s_i^t = s_i^{t-1}$).

- Similar rate to full gradient but iterations are n times cheaper.

A New Hope: Linearly-Convergent Stochastic Gradient

- Recent new stochastic algorithms for minimizing finite sums,

$$\min_w f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

  requiring $O(\log(1/\epsilon))$ iterations with $O(1)$ cost.

- Stochastic average gradient (SAG): [Le Roux et al., 2012]

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

  where iteration sets $s_i^t = \nabla f_i(x^t)$ for random i (o.w., $s_i^t = s_i^{t-1}$).

- Similar rate to full gradient but iterations are n times cheaper.

- Unlike EG, adaptive to strong-convexity.

Comparison of Convergence Rates

Number of iterations to reach an accuracy of ϵ:

Deterministic:	$O(n\sqrt{\frac{L}{\mu}}\log(1/\epsilon))$	(primal)
Stochastic	$O(\frac{\sigma^2}{\mu\epsilon} + \sqrt{\frac{L}{\mu}}\log(1/\epsilon))$	(primal)
Dual stochastic EG	$O((n + \frac{L}{\lambda})\log(1/\epsilon))$	(dual)
SAG	$O((n + \frac{L}{\mu})\log(1/\epsilon))$	(primal)

Comparison of Convergence Rates

Number of iterations to reach an accuracy of ϵ:

Deterministic:	$O(n\sqrt{\frac{L}{\mu}}\log(1/\epsilon))$	(primal)
Stochastic	$O(\frac{\sigma^2}{\mu\epsilon} + \sqrt{\frac{L}{\mu}}\log(1/\epsilon))$	(primal)
Dual stochastic EG	$O((n+\frac{L}{\lambda})\log(1/\epsilon))$	(dual)
SAG	$O((n+\frac{L}{\mu})\log(1/\epsilon))$	(primal)

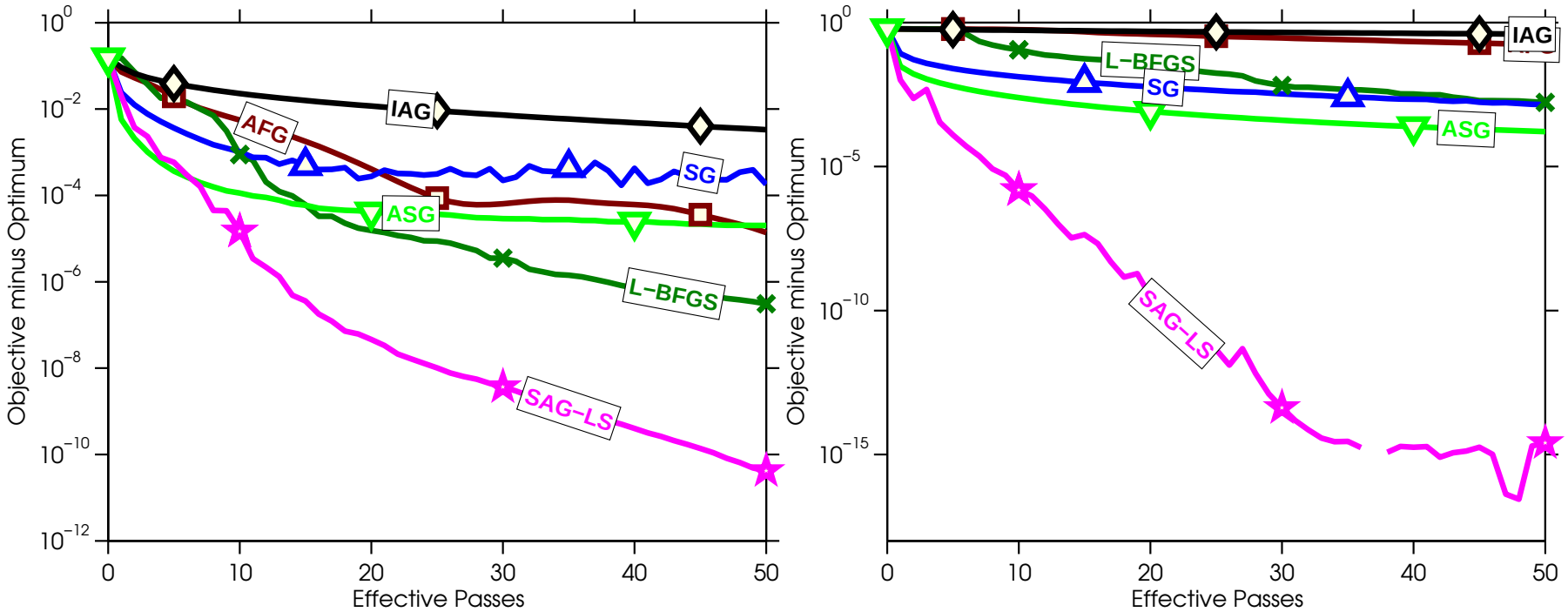
Similar to deterministic methods, SAG can adapt to problem:

- SAG automatically adapts to local μ at solution.
- Practical implementations try to automatically adapt to L, too.

Strong empirical performance for independent classification.

SAG for Logistic Regression

- Performance on logistic regression problems:



- SAG starts fast and stays fast.

Addressing the Memory Requirements

- Could this algorithm consistently outperform old CRF methods?

Addressing the Memory Requirements

- Could this algorithm consistently outperform old CRF methods?
- First, we need to address that SAG requires storing n gradients,

$$s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k),$$

for some previous k, which do not have a nice structure.

Addressing the Memory Requirements

- Could this algorithm consistently outperform old CRF methods?
- First, we need to address that SAG requires storing n gradients,

$$s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k),$$

  for some previous k, which do not have a nice structure.
- We could use SVRG/mixedGrad:

[Johnson & Zhang, 2013, Mahdavi et al, 2013,]

  - Similar convergence rate but without memory requirement.

Addressing the Memory Requirements

- Could this algorithm consistently outperform old CRF methods?
- First, we need to address that SAG requires storing n gradients,

$$s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k),$$

  for some previous k, which do not have a nice structure.
- We could use SVRG/mixedGrad:

[Johnson & Zhang, 2013, Mahdavi et al, 2013,]

  - Similar convergence rate but without memory requirement.
  - But requires extra evaluations of $\nabla \log p(y_i|x_i, w^t)$ per iteration.

Addressing the Memory Requirements

- The deterministic gradient update can be written:

$$w^{t+1} = w^t - \alpha\lambda w^t + \frac{\alpha}{n}\sum_{i=1}^{n} \nabla \log p(y_i|x_i, w^t).$$

Addressing the Memory Requirements

- The deterministic gradient update can be written:

$$w^{t+1} = w^t - \alpha\lambda w^t + \frac{\alpha}{n}\sum_{i=1}^{n} \nabla \log p(y_i|x_i, w^t).$$

- The SAG update:

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

where $s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k)$ for some previous k.

Addressing the Memory Requirements

- The deterministic gradient update can be written:

$$w^{t+1} = w^t - \alpha\lambda w^t + \frac{\alpha}{n}\sum_{i=1}^{n} \nabla \log p(y_i|x_i, w^t).$$

- The SAG update:

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

where $s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k)$ for some previous k.

- A modified update where we don't approximate the regularizer:

$$w^{t+1} = w^t - \alpha\lambda w^t - \frac{\alpha}{n}\sum_{i=1}^{n} g_i^t,$$

where $g_i^t = -\nabla \log p(y_i|x_i, w^k)$ for some previous k.

Addressing the Memory Requirements

- The deterministic gradient update can be written:

$$w^{t+1} = w^t - \alpha\lambda w^t + \frac{\alpha}{n}\sum_{i=1}^{n} \nabla \log p(y_i|x_i, w^t).$$

- The SAG update:

$$w^{t+1} = w^t - \frac{\alpha}{n}\sum_{i=1}^{n} s_i^t,$$

where $s_i^t = \lambda w^k - \nabla \log p(y_i|x_i, w^k)$ for some previous k.

- A modified update where we don't approximate the regularizer:

$$w^{t+1} = w^t - \alpha\lambda w^t - \frac{\alpha}{n}\sum_{i=1}^{n} g_i^t,$$

where $g_i^t = -\nabla \log p(y_i|x_i, w^k)$ for some previous k.

- The g_i^t have a nice structure, and regularizer update is efficient.

Addressing the Memory Requirements

- Consider a chain-structured CRF model of the form

$$p(y|x,w) \propto \exp\left(\sum_{j=1}^{V} x_j^T w_{y_j} + \sum_{j=1}^{V-1} w_{y_j,y_{j+1}}\right).$$

Addressing the Memory Requirements

- Consider a chain-structured CRF model of the form

$$p(y|x,w) \propto \exp\left(\sum_{j=1}^{V} x_j^T w_{y_j} + \sum_{j=1}^{V-1} w_{y_j,y_{j+1}}\right).$$

- The gradient with respect to a particular vector w_k is

$$\nabla_{w_k} \log\ p(y|x,w) = \sum_{j=1}^{V} x_j \left[\mathbb{I}(y_j = k) - p(y_j = k|x,w)\right].$$

Addressing the Memory Requirements

- Consider a chain-structured CRF model of the form

$$p(y|x,w) \propto \exp\left(\sum_{j=1}^{V} x_j^T w_{y_j} + \sum_{j=1}^{V-1} w_{y_j,y_{j+1}}\right).$$

- The gradient with respect to a particular vector w_k is

$$\nabla_{w_k} \log\ p(y|x,w) = \sum_{j=1}^{V} x_j \left[\mathbb{I}(y_j = k) - p(y_j = k|x,w)\right].$$

- The modified SAG algorithm needs to update the sum,

$$\sum_{i=1}^{n} g_i^{t+1} = \sum_{i=1}^{n} [g_i^t] + g_i^{t+1} - g_i^t.$$

Addressing the Memory Requirements

- Consider a chain-structured CRF model of the form

$$p(y|x,w) \propto \exp\left(\sum_{j=1}^{V} x_j^T w_{y_j} + \sum_{j=1}^{V-1} w_{y_j,y_{j+1}}\right).$$

- The gradient with respect to a particular vector w_k is

$$\nabla_{w_k} \log\ p(y|x,w) = \sum_{j=1}^{V} x_j \left[\mathbb{I}(y_j = k) - p(y_j = k|x,w)\right].$$

- The modified SAG algorithm needs to update the sum,

$$\sum_{i=1}^{n} g_i^{t+1} = \sum_{i=1}^{n} [g_i^t] + g_i^{t+1} - g_i^t.$$

- To do this, we only need to store the unary marginals.

Addressing the Memory Requirements

- Consider a chain-structured CRF model of the form

$$p(y|x,w) \propto \exp\left(\sum_{j=1}^{V} x_j^T w_{y_j} + \sum_{j=1}^{V-1} w_{y_j,y_{j+1}}\right).$$

- The gradient with respect to a particular vector w_k is

$$\nabla_{w_k} \log\ p(y|x,w) = \sum_{j=1}^{V} x_j \left[\mathbb{I}(y_j = k) - p(y_j = k|x,w)\right].$$

- The modified SAG algorithm needs to update the sum,

$$\sum_{i=1}^{n} g_i^{t+1} = \sum_{i=1}^{n} [g_i^t] + g_i^{t+1} - g_i^t.$$

- To do this, we only need to store the unary marginals.
- General pairwise graphical models require $O(VK + EK^2)$.
- Unlike basic SAG, no dependence on number of features.

Practical issues: setting the step size and stopping

Traditional sources of frustration for stochastic gradient users:

1. Need to choose between slow convergence or oscillations.
2. Setting the sequence of step-sizes.
3. Deciding when to stop.

Practical issues: setting the step size and stopping

Traditional sources of frustration for stochastic gradient users:

1. Need to choose between slow convergence or oscillations.
2. Setting the sequence of step-sizes.
3. Deciding when to stop.

These are easier to address in methods like SAG:

1. Faster convergence rates.
2. Allow a constant step-size ($\alpha = 1/L$).
3. Approximate the full gradient for deciding when to stop.

Practical issues: setting the step size and stopping

No manual step-size tuning, we approximate L as we go:

- Start with $L = 1$.

Practical issues: setting the step size and stopping

No manual step-size tuning, we approximate L as we go:

- Start with $L = 1$.
- If $\|f_i'(x)\|^2 \geq \delta$, increase L until we satisfy:

$$f_i(x - \frac{1}{L} f_i'(x)) \leq f_i'(x) - \frac{1}{2L} \|f_i'(x)\|^2.$$

(Lipschitz approximation procedure from FISTA)

Practical issues: setting the step size and stopping

No manual step-size tuning, we approximate L as we go:

- Start with $L = 1$.
- If $\|f_i'(x)\|^2 \geq \delta$, increase L until we satisfy:

$$f_i(x - \frac{1}{L} f_i'(x)) \leq f_i'(x) - \frac{1}{2L} \|f_i'(x)\|^2.$$

  (Lipschitz approximation procedure from FISTA)

- Decrease L between iterations.
  (makes algorithm adaptive to local L)

Practical issues: setting the step size and stopping

No manual step-size tuning, we approximate L as we go:

- Start with $L = 1$.
- If $\|f_i'(x)\|^2 \geq \delta$, increase L until we satisfy:

$$f_i(x - \frac{1}{L} f_i'(x)) \leq f_i'(x) - \frac{1}{2L} \|f_i'(x)\|^2.$$

  (Lipschitz approximation procedure from FISTA)

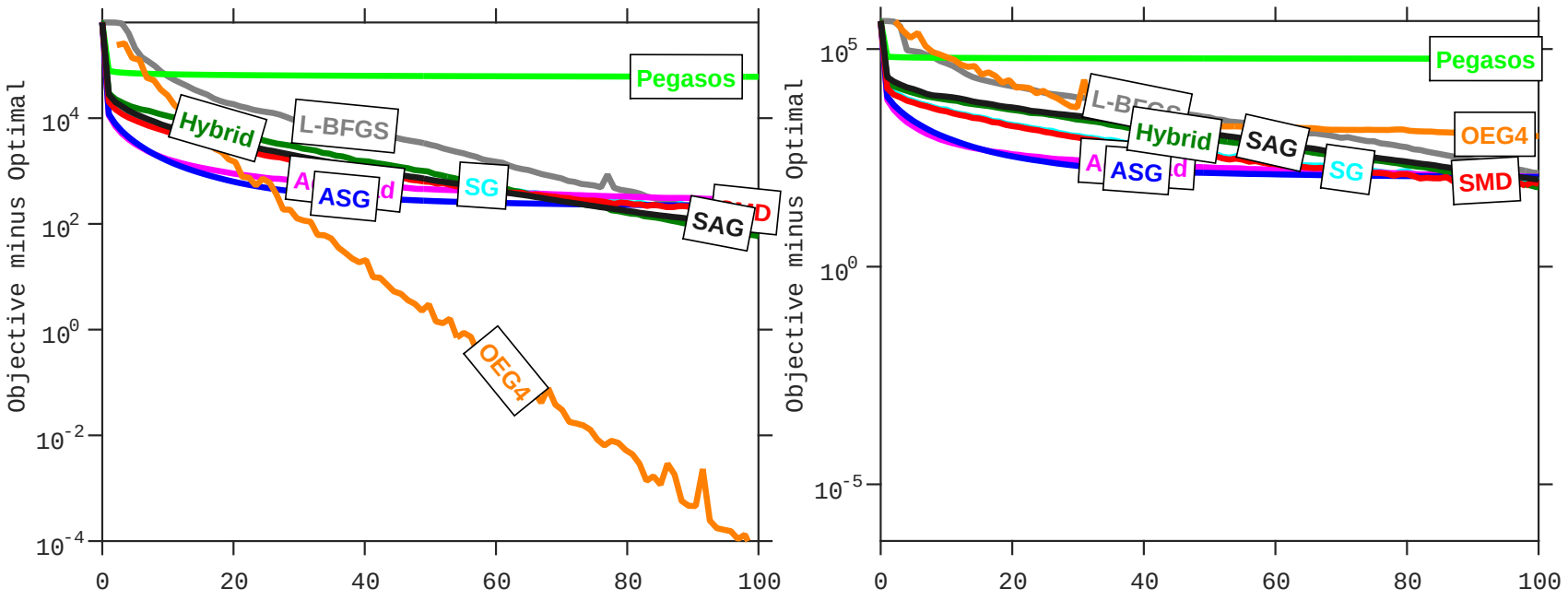
- Decrease L between iterations.
  (makes algorithm adaptive to local L)

Performance is better than using $1/L$ for global L.

- Similar to choosing the optimal step-size.

Comparison of SAG to existing methods

- Comparison of SAG and state of the art methods.



- Sometimes better and sometimes worse than existing methods.
- Have we really made so little progress???

Non-Uniform Sampling (NUS)

- Maybe random sampling is too naive?
- Can we instead do non-uniform sampling?
  - Sample some training examples more often than others.

Non-Uniform Sampling (NUS)

- Maybe random sampling is too naive?
- Can we instead do non-uniform sampling?
  - Sample some training examples more often than others.
- Key idea:
  - Bias sampling towards examples whose gradients change quicky.
  - “If the gradient changes slowly, don’t sample it as often”.

Non-Uniform Sampling (NUS)

- Maybe random sampling is too naive?
- Can we instead do non-uniform sampling?
  - Sample some training examples more often than others.
- Key idea:
  - Bias sampling towards examples whose gradients change quicky.
  - "If the gradient changes slowly, don't sample it as often".
- Implemented by biasing sampling towards Lipschitz constants:
  - High Lipschitz constant $\rightarrow$ gradient can change quickly.

Non-Uniform Sampling (NUS)

- Recent works show this improves various methods:
  - [Strohmer & Vershynin, 2009, Nesterov, 2010, Schmidt et al, 2013, Xiao & Zhang, 2014, Needell et al., 2014].
- Does SAG converge with NUS?

Non-Uniform Sampling (NUS)

- Recent works show this improves various methods:
  - [Strohmer & Vershynin, 2009, Nesterov, 2010, Schmidt et al, 2013, Xiao & Zhang, 2014, Needell et al., 2014].
- Does SAG converge with NUS?
  - Not known, and seems hard to prove.

Non-Uniform Sampling (NUS)

- Recent works show this improves various methods:
  - [Strohmer & Vershynin, 2009, Nesterov, 2010, Schmidt et al, 2013, Xiao & Zhang, 2014, Needell et al., 2014].
- Does SAG converge with NUS?
  - Not known, and seems hard to prove.
- We instead analyzed the SAGA variant with NUS.

[Defazio et al., 2014]

  - Proved $O(\log(1/\epsilon))$ rate for any reasonable NUS method.
  - Proved that rate is faster with Lipschitz sampling.

Adaptive Non-Uniform Sampling

- Still not practical.
  - Global Lipschitz constants are hard to get.
  - Even if you have them, it doesn't help much.

Adaptive Non-Uniform Sampling

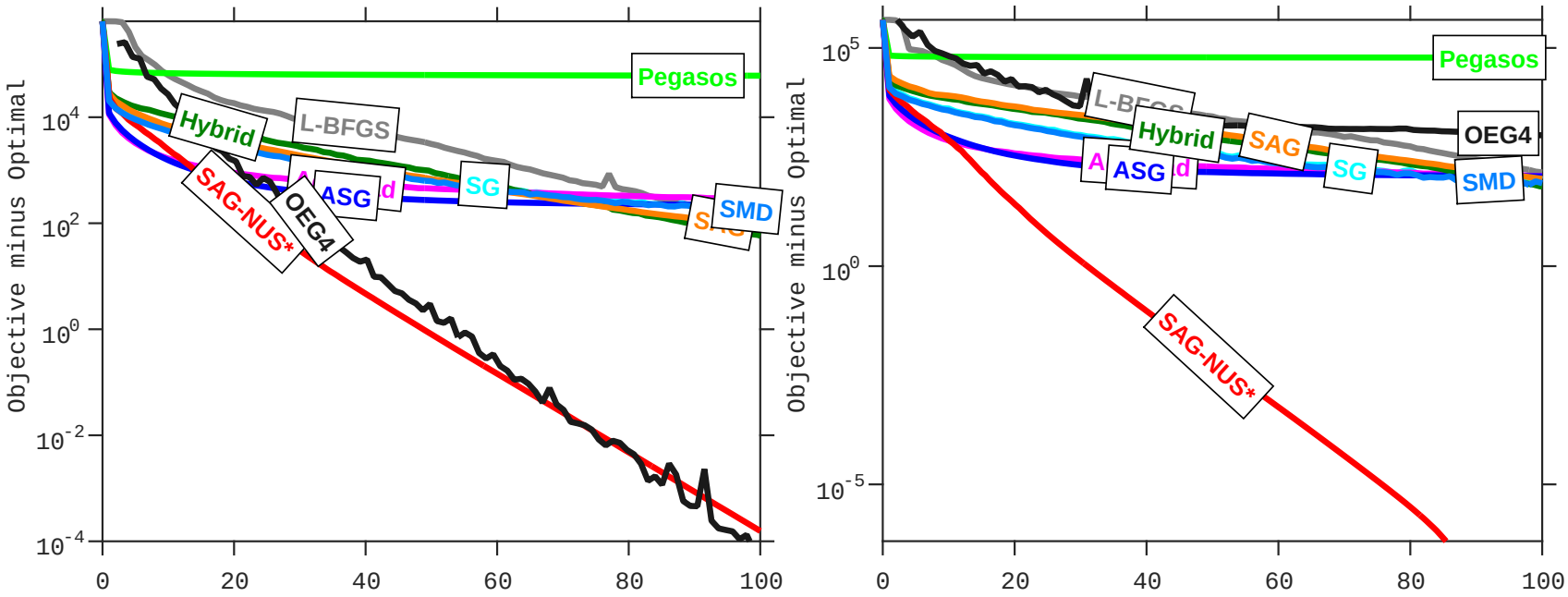
- Still not practical.
  - Global Lipschitz constants are hard to get.
  - Even if you have them, it doesn't help much.
- But it works better if you try to estimate local Lipschitz constant:
  - We estimate each L_i using similar Lipschitz approximation method.
  - Adapts to the local distribution of L_i at the solution.

Adaptive Non-Uniform Sampling

- Still not practical.
  - Global Lipschitz constants are hard to get.
  - Even if you have them, it doesn't help much.
- But it works better if you try to estimate local Lipschitz constant:
  - We estimate each L_i using similar Lipschitz approximation method.
  - Adapts to the local distribution of L_i at the solution.
- Why should the local L_i values work?
  - For correctly-classified examples, L_i is near zero.
  - Algorithm focuses on incorrectly-classified examples.

Comparison of SAG-NUS to existing methods

- Comparison of SAG with NUS to existing methods:

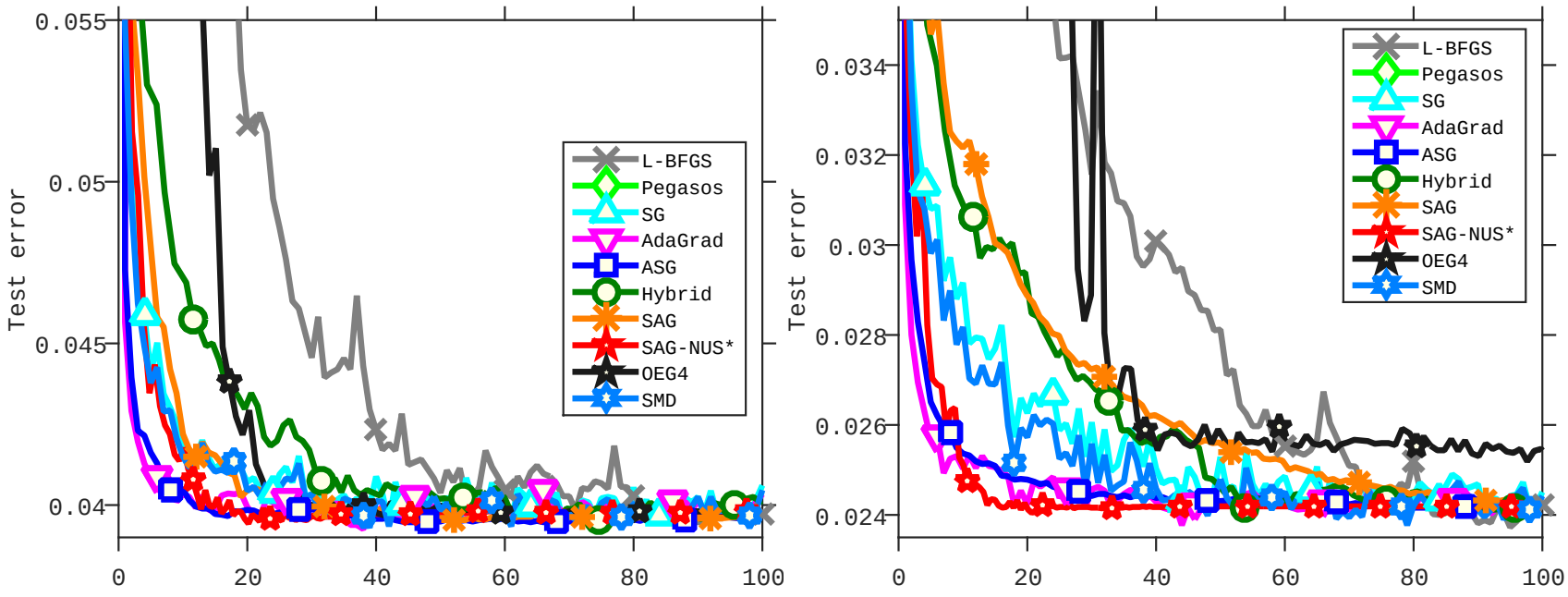


(NUS did not improve performance of SG.)

- Similar or significantly better than best of previous methods.

Comparison of SAG-NUS to existing methods

- Test error:



(NUS did not improve performance of SG.)

Discussion

- We explored applying SAG to train CRFs.
  - With a few modifications, the memory issue is not an issue.
  - Allows adaptive step-size and has a stopping criterion.
  - With NUS, substantially improves on state of the art.
  - SAG4CRF code available on my webpage.

Discussion

- We explored applying SAG to train CRFs.
  - With a few modifications, the memory issue is not an issue.
  - Allows adaptive step-size and has a stopping criterion.
  - With NUS, substantially improves on state of the art.
  - SAG4CRF code available on my webpage.

- Various extensions are possible:
  - Could use non-smooth regularizers via proximal/ADMM versions.
  - Faster methods may be possible via acceleration/Newton.
  - Method should work with approximate inference.
  - For conditional neural fields and variants like FCNs+CRFs:
    - Need SVRG to deal with the memory.