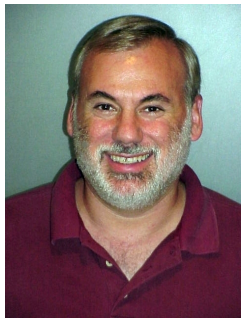
Large-Scale Matrix Factorization with Distributed Stochastic Gradient Descent

Rainer Gemulla

December 17, 2011

Peter J. Haas Yannis Sismanis Christina Teflioudi Faraz Makari




Outline

Outline

Matrix completion visualized

Original image

Matrix completion visualized

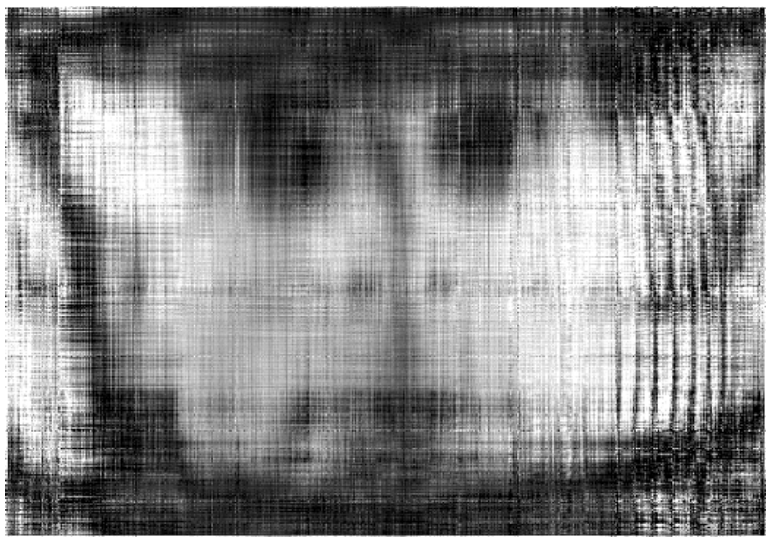
Original image

Partially observed image

Matrix completion visualized

Original image

Partially observed image



Reconstructed image

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar	**The Matrix**	**Up**
Alice		**4**	**2**
Bob	**3**	**2**	
Charlie	**5**		**3**

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)		**4**	**2**
Bob (*1.21*)	**3**	**2**	
Charlie (*2.30*)	**5**		**3**

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)		**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	
Charlie (*2.30*)	**5** (*5.2*)		**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})^2$$

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i - \mathbf{m}_j - [\mathbf{WH}]_{ij})^2$$

- Bias

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j)\in Z} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i - \mathbf{m}_j - [\mathbf{WH}]_{ij})^2 + \lambda\left(\|\mathbf{W}\| + \|\mathbf{H}\| + \|\mathbf{u}\| + \|\mathbf{m}\|\right)$$

- Bias, regularization

Matrix completion for recommender systems

- Discover latent factors ($r = 1$)

	Avatar (*2.24*)	**The Matrix** (*1.92*)	**Up** (*1.18*)
Alice (*1.98*)	? (*4.4*)	**4** (*3.8*)	**2** (*2.3*)
Bob (*1.21*)	**3** (*2.7*)	**2** (*2.3*)	? (*1.4*)
Charlie (*2.30*)	**5** (*5.2*)	? (*4.4*)	**3** (*2.7*)

- Minimum loss

$$\min_{\mathbf{W},\mathbf{H},\mathbf{u},\mathbf{m}} \sum_{(i,j,t)\in Z_t} (\mathbf{V}_{ij} - \mu - \mathbf{u}_i(t) - \mathbf{m}_j(t) - [\mathbf{W}(t)\mathbf{H}]_{ij})^2 + \lambda\left(\|\mathbf{W}(t)\| + \|\mathbf{H}\| + \|\mathbf{u}(t)\| + \|\mathbf{m}(t)\|\right)$$

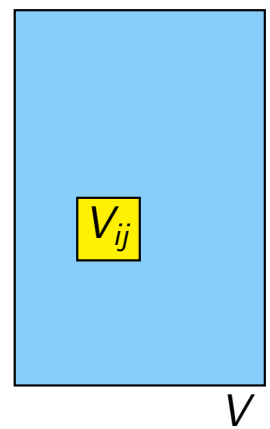
- Bias, regularization, time, ...

Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...

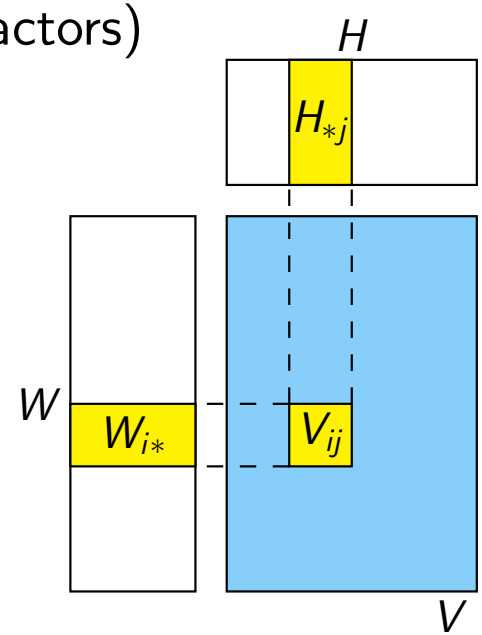
Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)



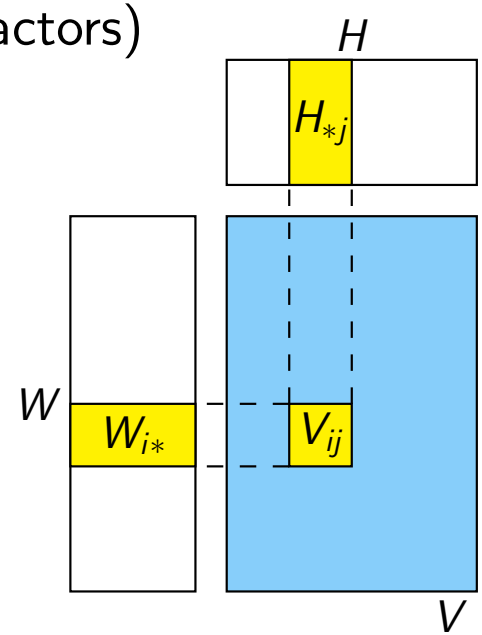
Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)
- Parameter space
  - $\mathbf{W}$: row factors (e.g., $m \times r$ latent customer factors)
  - $\mathbf{H}$: column factors (e.g., $r \times n$ latent movie factors)



Generalized Matrix Factorization

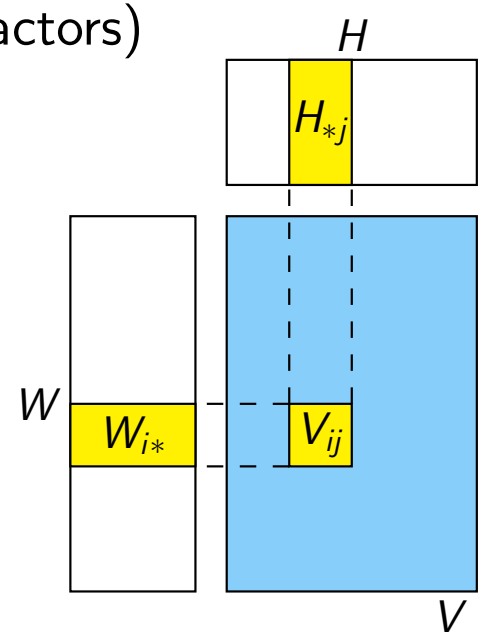
- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)
- Parameter space
  - $\mathbf{W}$: row factors (e.g., $m \times r$ latent customer factors)
  - $\mathbf{H}$: column factors (e.g., $r \times n$ latent movie factors)
- Model
  - $L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$: loss at element (i, j)
  - Includes prediction error, regularization, auxiliary information, ...
  - Constraints (e.g., non-negativity)



Generalized Matrix Factorization

- A general machine learning problem
  - Recommender systems, text indexing, face recognition, ...
- Training data
  - $\mathbf{V}$: $m \times n$ input matrix (e.g., rating matrix)
  - Z: *training set* of indexes in $\mathbf{V}$ (e.g., subset of known ratings)
- Parameter space
  - $\mathbf{W}$: row factors (e.g., $m \times r$ latent customer factors)
  - $\mathbf{H}$: column factors (e.g., $r \times n$ latent movie factors)
- Model
  - $L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$: loss at element (i, j)
  - Includes prediction error, regularization, auxiliary information, ...
  - Constraints (e.g., non-negativity)
- Find best model

$$\underset{\mathbf{W},\mathbf{H}}{\operatorname{argmin}} \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$



Successful Applications

- Movie recommendation (Netflix, competition papers)
  - $>$12M users, $>$20k movies, 2.4B ratings (projected)
  - 36GB data, 9.2GB model (projected)
  - Latent factor model
- Website recommendation (Microsoft, WWW10)
  - 51M users, 15M URLs, 1.2B clicks
  - 17.8GB data, 161GB metadata, 49GB model
  - Gaussian non-negative matrix factorization
- News personalization (Google, WWW07)
  - Millions of users, millions of stories, ? clicks
  - Probabilistic latent semantic indexing

Successful Applications

- Movie recommendation (Netflix, competition papers)
  - >12M users, >20k movies, 2.4B ratings (projected)
  - 36GB data, 9.2GB model (projected)
  - Latent factor model
- Website recommendation (Microsoft, WWW10)
  - 51M users, 15M URLs, 1.2B clicks
  - 17.8GB data, 161GB metadata, 49GB model
  - Gaussian non-negative matrix factorization
- News personalization (Google, WWW07)
  - Millions of users, millions of stories, ? clicks
  - Probabilistic latent semantic indexing

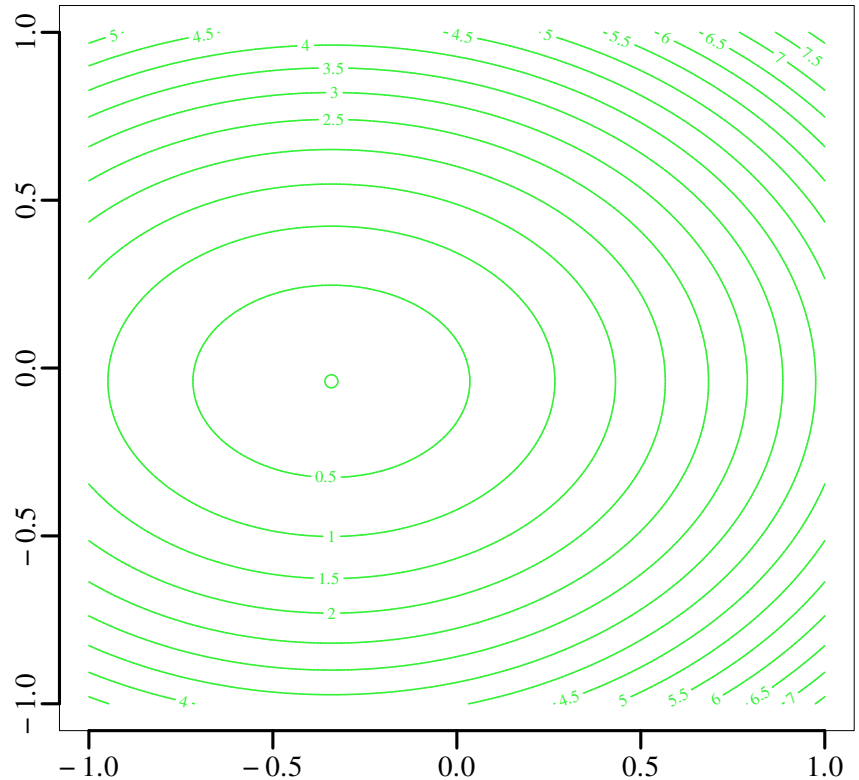
Distributed processing is necessary!

- Big data
- Large models
- Expensive computations

Outline

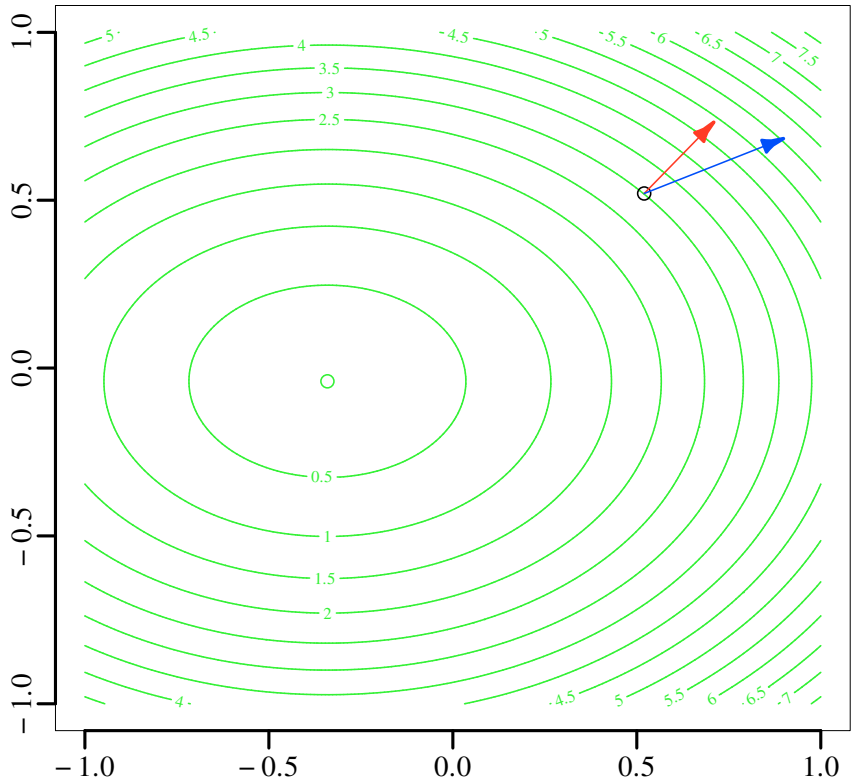
Stochastic Gradient Descent

- Find minimum θ^* of function L



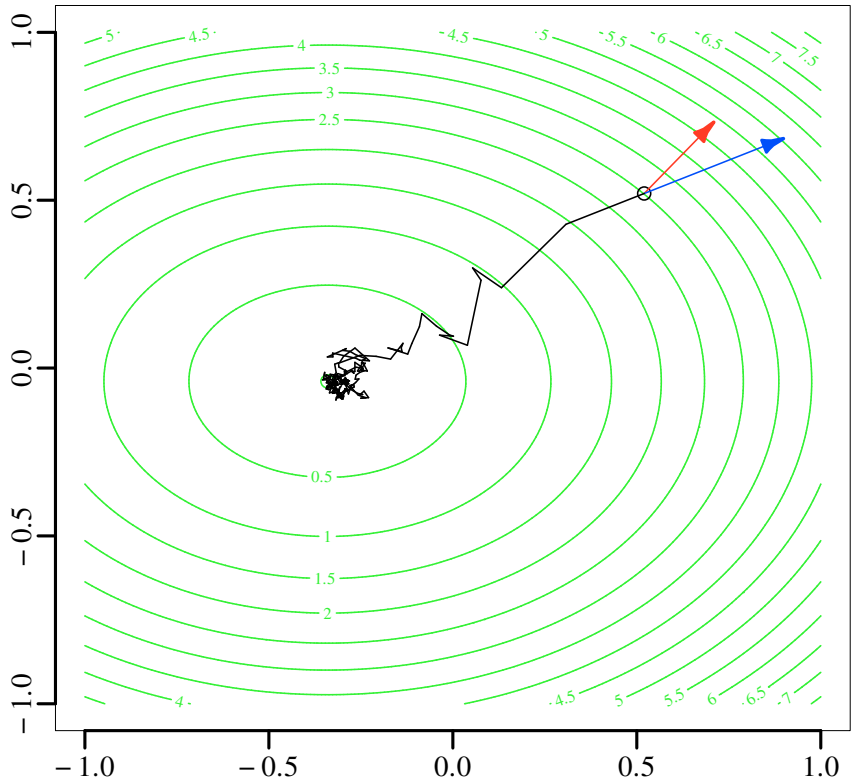
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$



Stochastic Gradient Descent

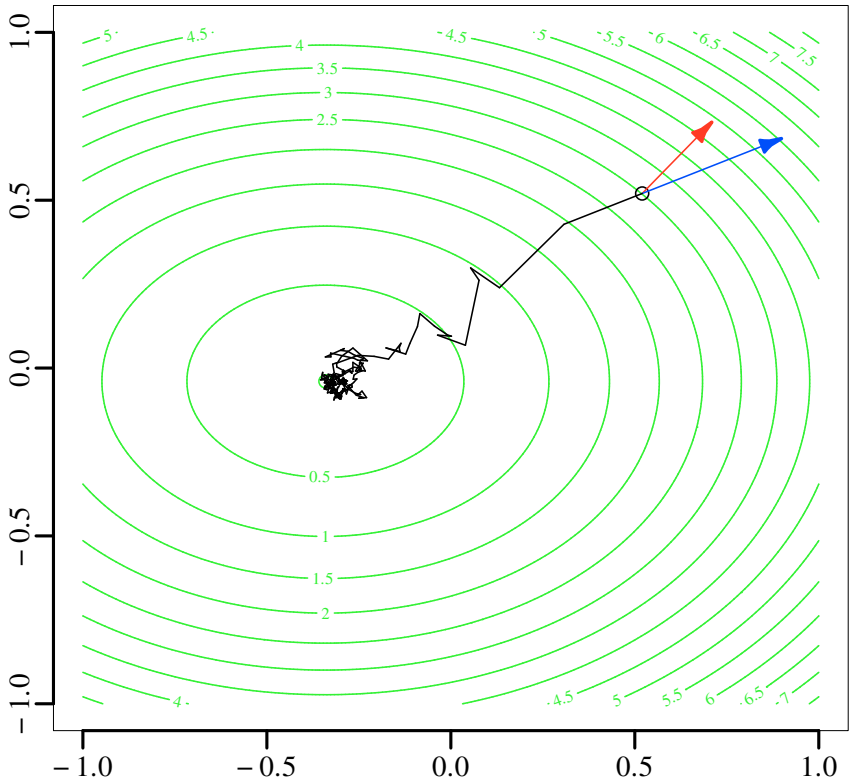
- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Move "approximately" downhill



Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Move "approximately" downhill
- Stochastic difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

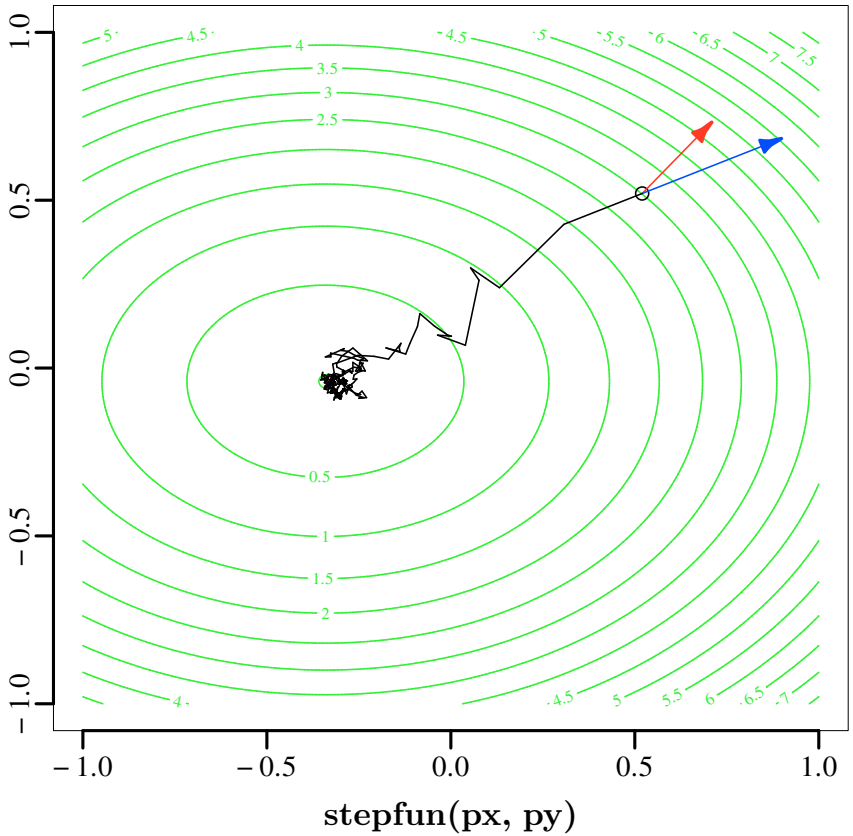


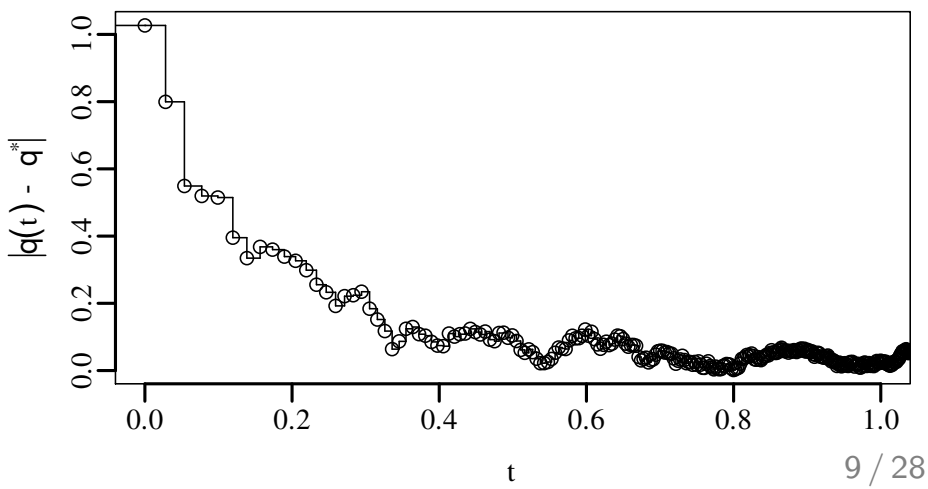
Stochastic Gradient Descent

- Find minimum θ^* of function L
- Pick a starting point θ_0
- Approximate gradient $\hat{L}'(\theta_0)$
- Move "approximately" downhill
- Stochastic difference equation

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- Under certain conditions, asymptotically approximates (continuous) gradient descent

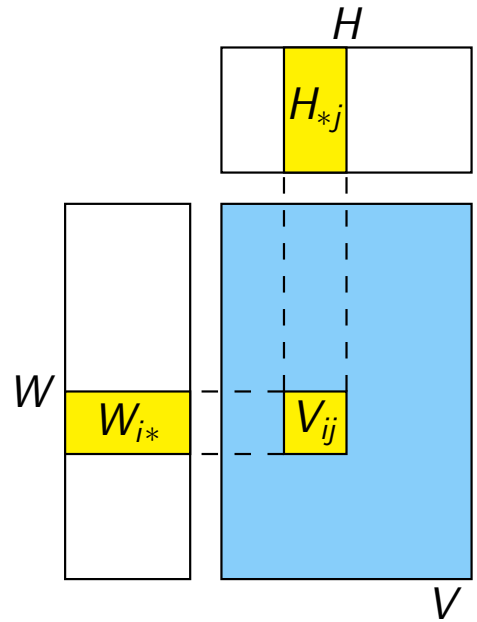




Stochastic Gradient Descent for Matrix Factorization

- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j) \in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

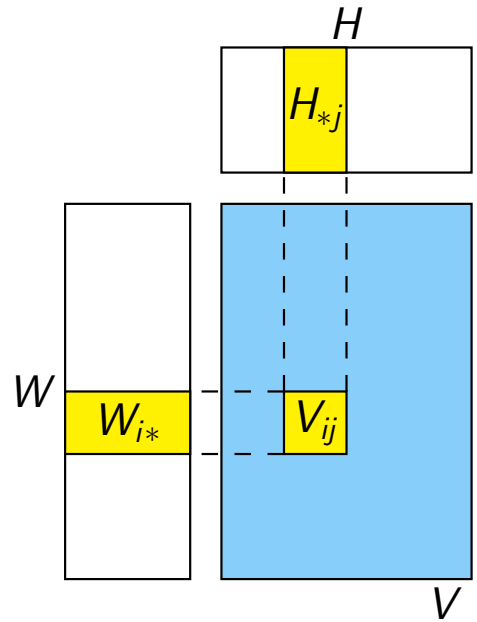


Stochastic Gradient Descent for Matrix Factorization

- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j) \in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j) \in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$



Stochastic Gradient Descent for Matrix Factorization

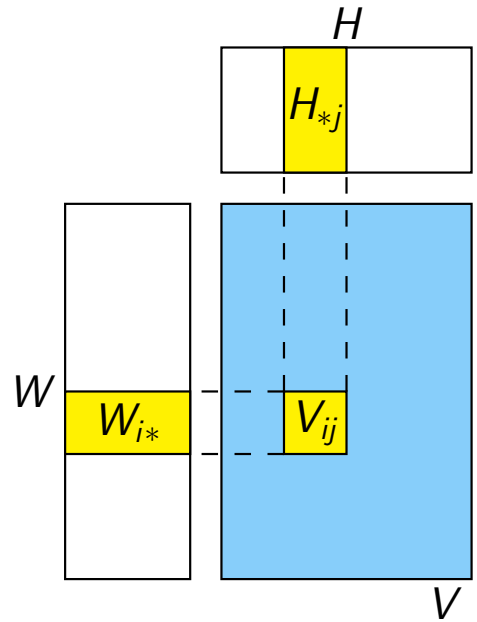
- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$\hat{L}'(\theta, z) = N L'_{i_z j_z}(\mathbf{W}_{i_z *}, \mathbf{H}_{*j_z}),$$

where $N = |Z|$



Stochastic Gradient Descent for Matrix Factorization

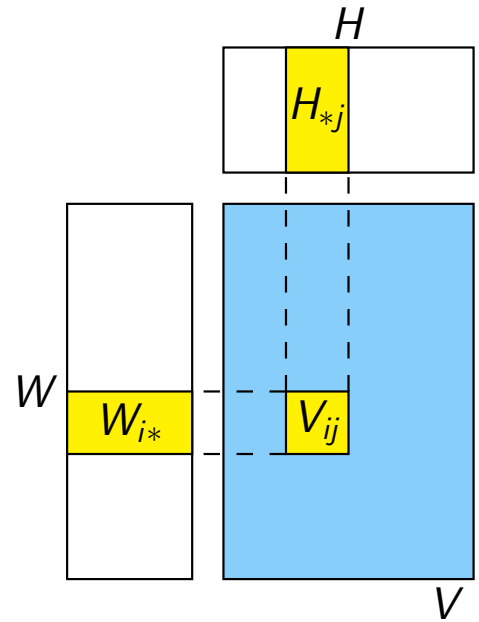
- Set $\theta = (\mathbf{W}, \mathbf{H})$ and use

$$L(\theta) = \sum_{(i,j)\in Z} L_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

$$L'(\theta) = \sum_{(i,j)\in Z} L'_{ij}(\mathbf{W}_{i*}, \mathbf{H}_{*j})$$

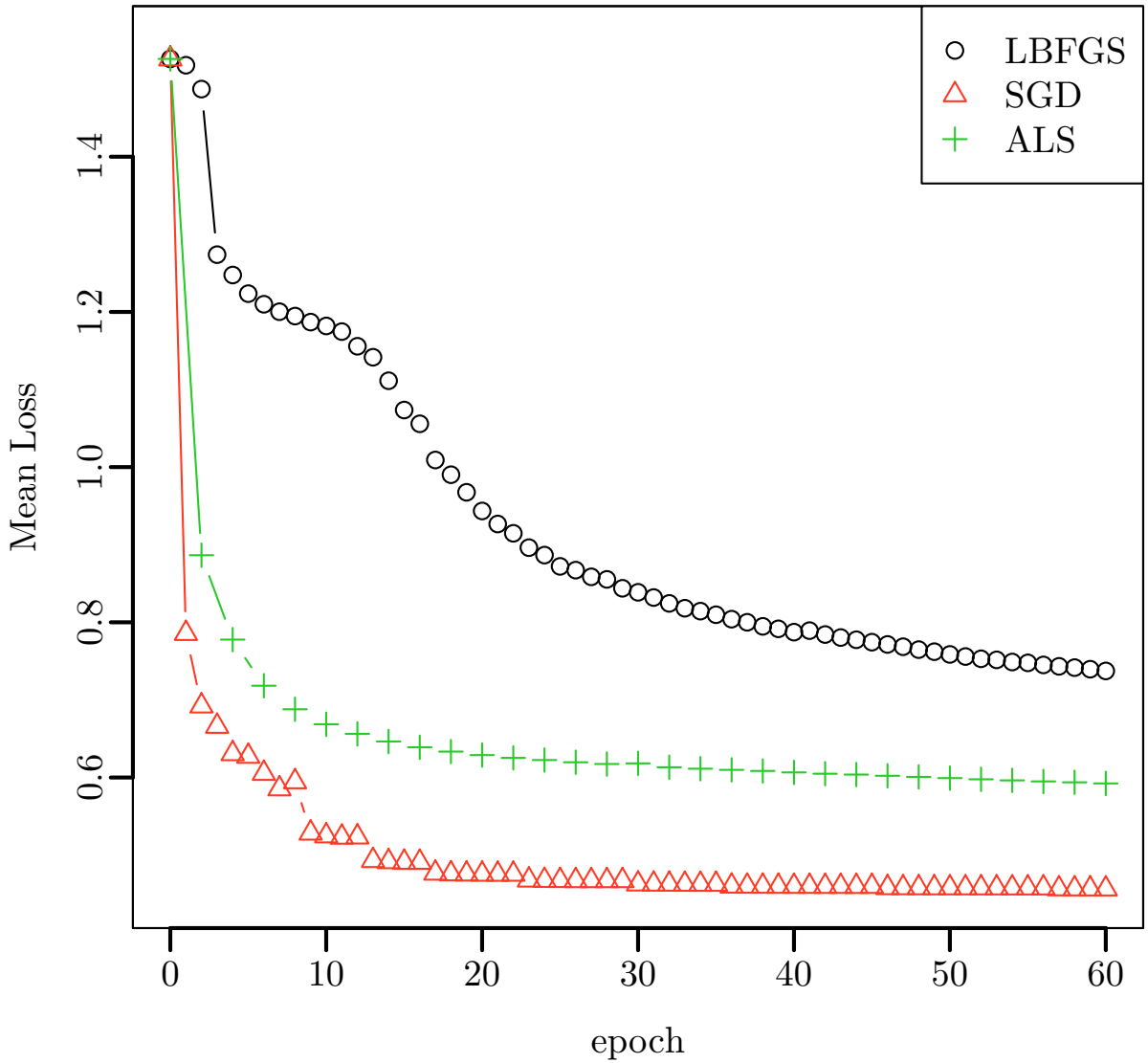
$$\hat{L}'(\theta, z) = N L'_{i_z j_z}(\mathbf{W}_{i_z *}, \mathbf{H}_{*j_z}),$$

where $N = |Z|$



- SGD epoch
  1. Pick a random entry $z \in Z$
  2. Compute approximate gradient $\hat{L}'(\theta, z)$
  3. Update parameters
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n, z)$$
  4. Repeat N times

Stochastic Gradient Descent on Netflix Data



Outline

Averaging Techniques

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

How to distribute?

Averaging Techniques

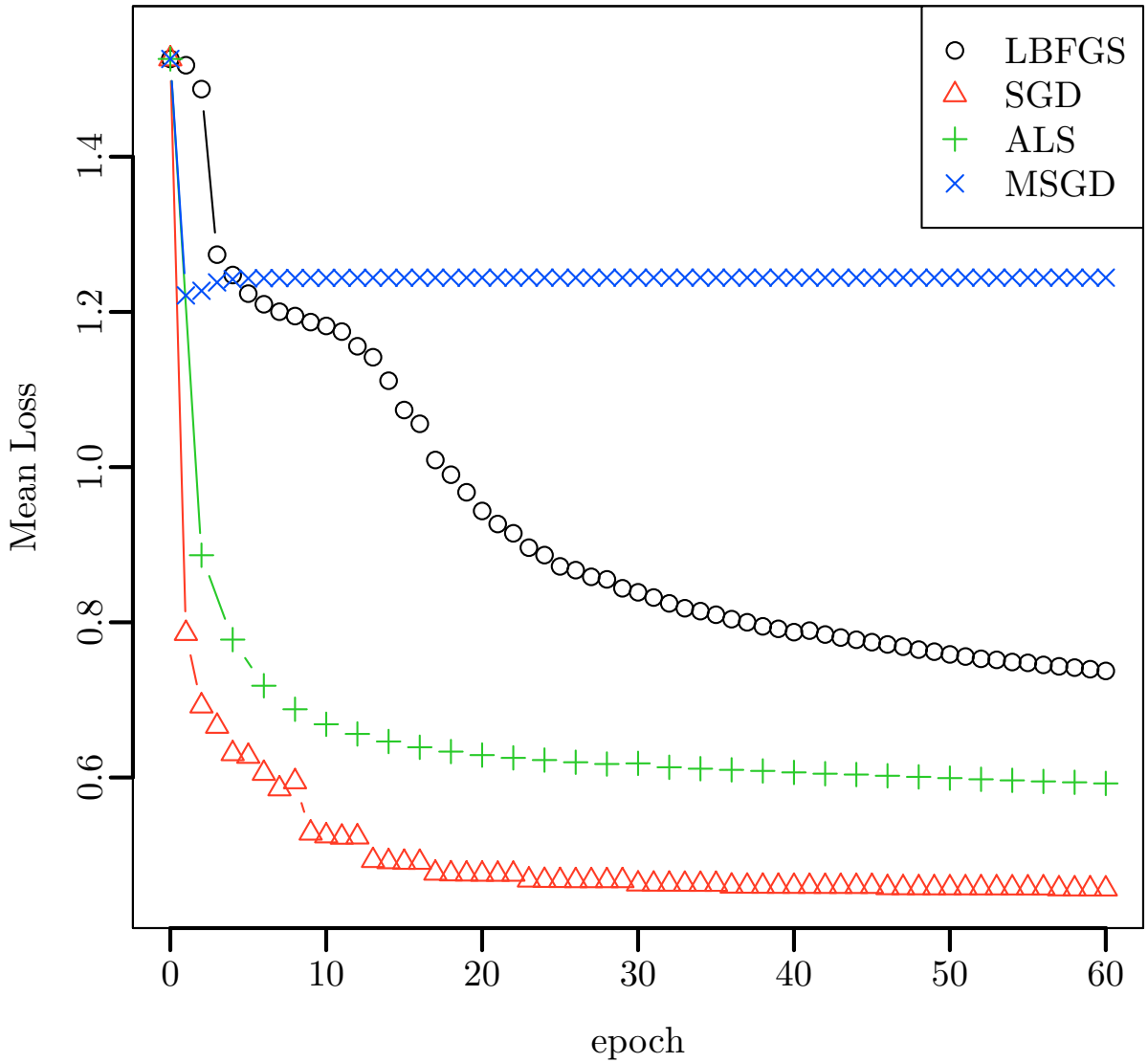
- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results

Averaging Techniques



Averaging Techniques

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results
  - Does not converge to correct solution!

Averaging Techniques

- SGD steps depend on each other

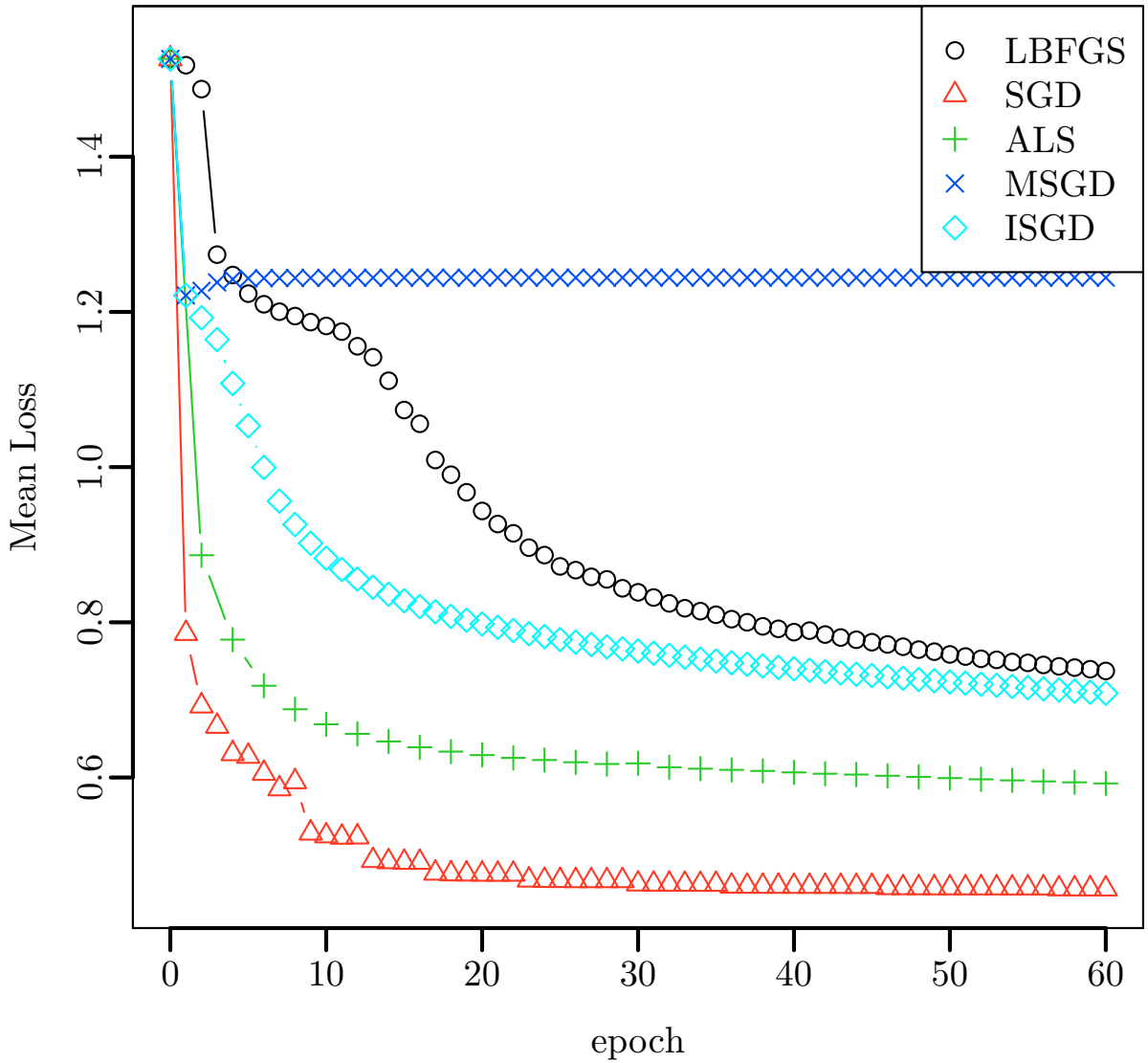
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results
  - Does not converge to correct solution!

- Iterative Parameter mixing (ISGD)
  - *Map*: Run independent instances of SGD on subsets of the data (for some time)
  - *Reduce*: Average results
  - Repeat

Averaging Techniques



Averaging Techniques

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

How to distribute?

- Parameter mixing (MSGD)
  - *Map*: Run independent instances of SGD on subsets of the data (until convergence)
  - *Reduce*: Average results
  - Does not converge to correct solution!

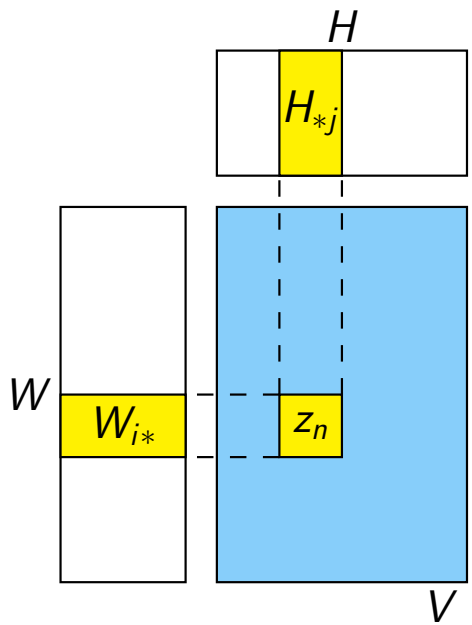
- Iterative Parameter mixing (ISGD)
  - *Map*: Run independent instances of SGD on subsets of the data (for some time)
  - *Reduce*: Average results
  - Repeat
  - Converges slowly!

Problem Structure

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

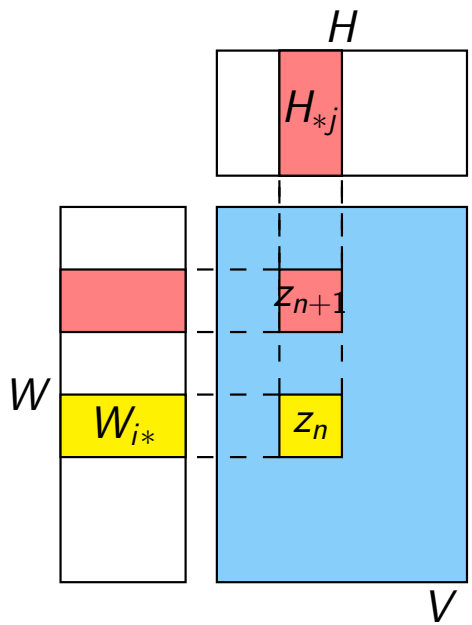


Problem Structure

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

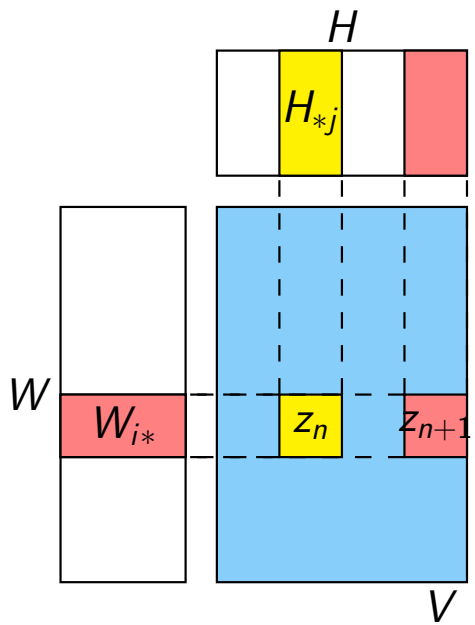


Problem Structure

- SGD steps depend on each other

$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$

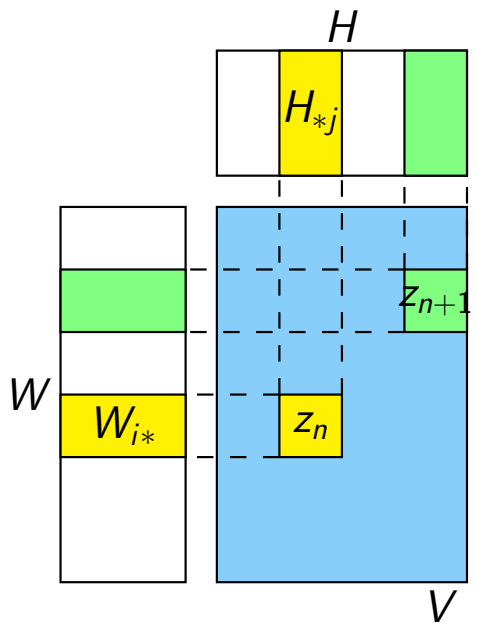


Problem Structure

- SGD steps depend on each other

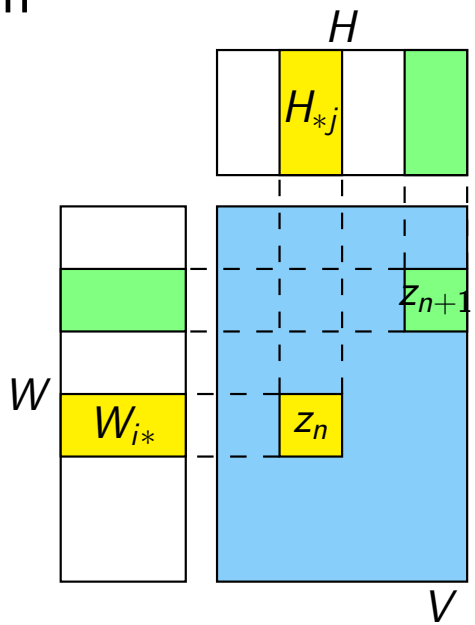
$$\theta_{n+1} = \theta_n - \epsilon_n \hat{L}'(\theta_n)$$

- An SGD step on example $z \in Z$ ...
  1. Reads $W_{i_z *}$ and $H_{* j_z}$
  2. Performs gradient computation $L'_{ij}(W_{i_z *}, H_{* j_z})$
  3. Updates $W_{i_z *}$ and $H_{* j_z}$
- Not all steps are dependent



Interchangeability

- Two elements $z_1, z_2 \in Z$ are *interchangeable* if they share neither row nor column

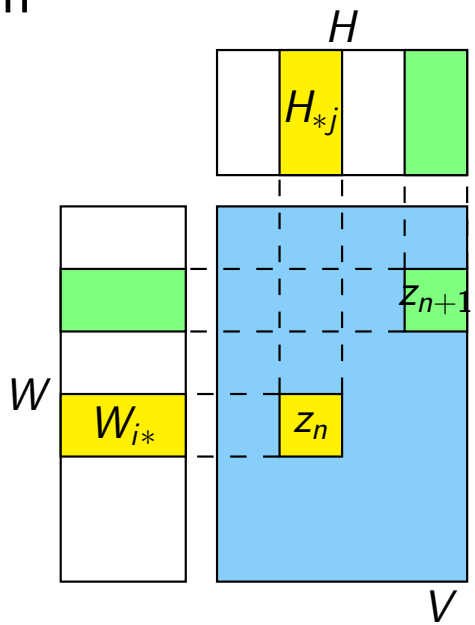


- When z_n and z_{n+1} are interchangeable, the SGD steps

$$\theta_{n+1} = \theta_n - \epsilon \hat{L}'(\theta_n, z_n)$$

Interchangeability

- Two elements $z_1, z_2 \in Z$ are *interchangeable* if they share neither row nor column

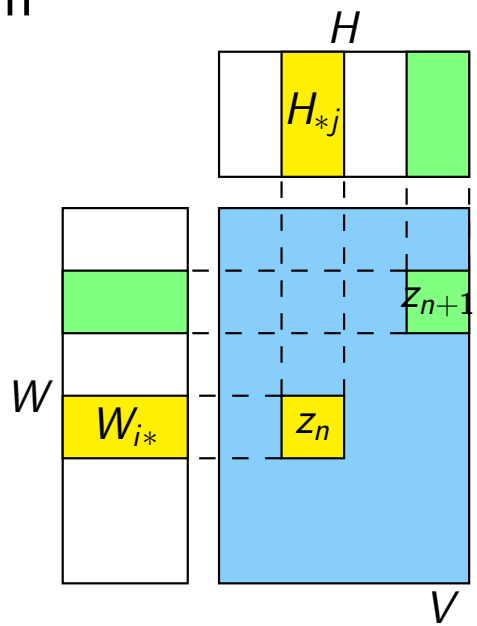


- When z_n and z_{n+1} are interchangeable, the SGD steps

$$\theta_{n+2} = \theta_n - \epsilon \hat{L}'(\theta_n, z_n) - \epsilon \hat{L}'(\theta_{n+1}, z_{n+1})$$

Interchangeability

- Two elements $z_1, z_2 \in Z$ are *interchangeable* if they share neither row nor column



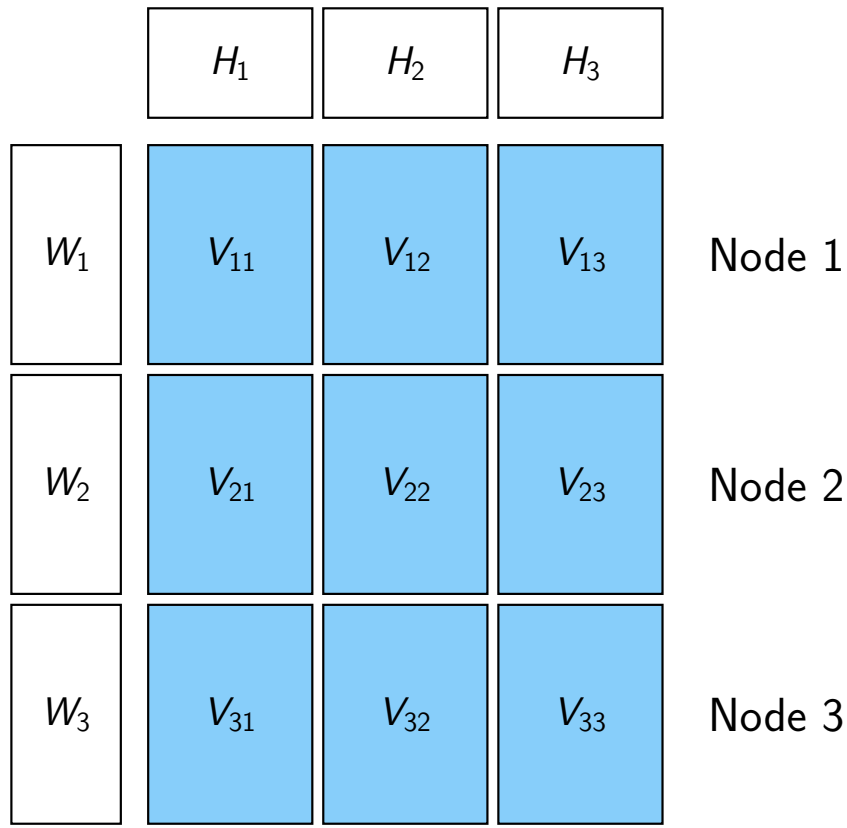
- When z_n and z_{n+1} are interchangeable, the SGD steps

$$\begin{aligned}\theta_{n+2} &= \theta_n - \epsilon\hat{L}'(\theta_n, z_n) - \epsilon\hat{L}'(\theta_{n+1}, z_{n+1}) \\ &= \theta_n - \epsilon\hat{L}'(\theta_n, z_n) - \epsilon\hat{L}'(\theta_n, z_{n+1}),\end{aligned}$$

become parallelizable!

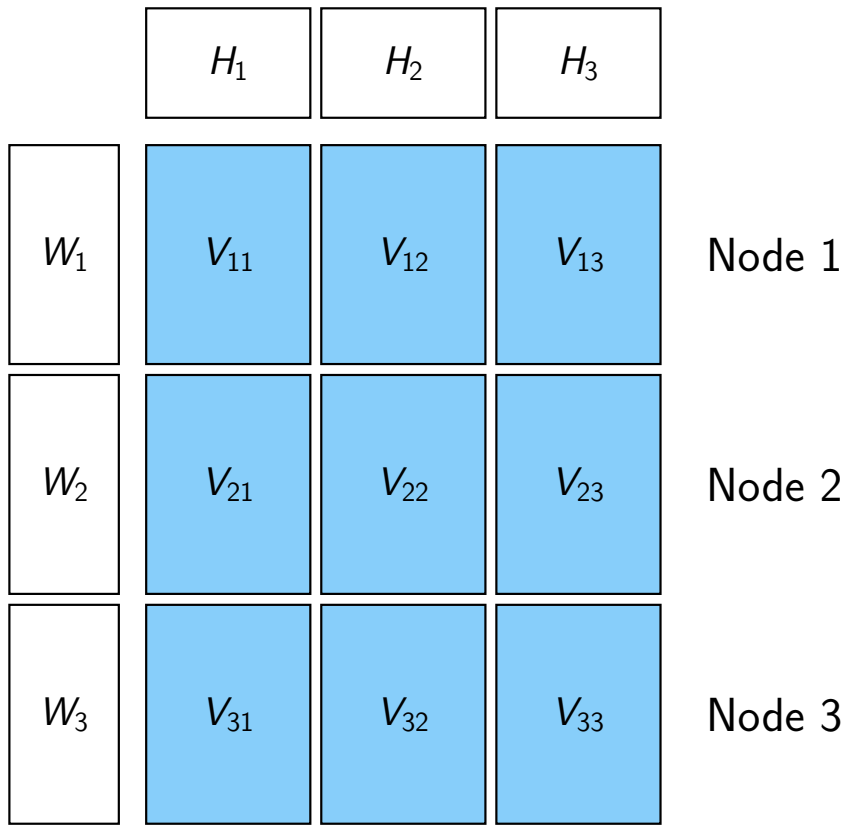
Exploitation

- Block and distribute the input matrix $\mathbf{V}$



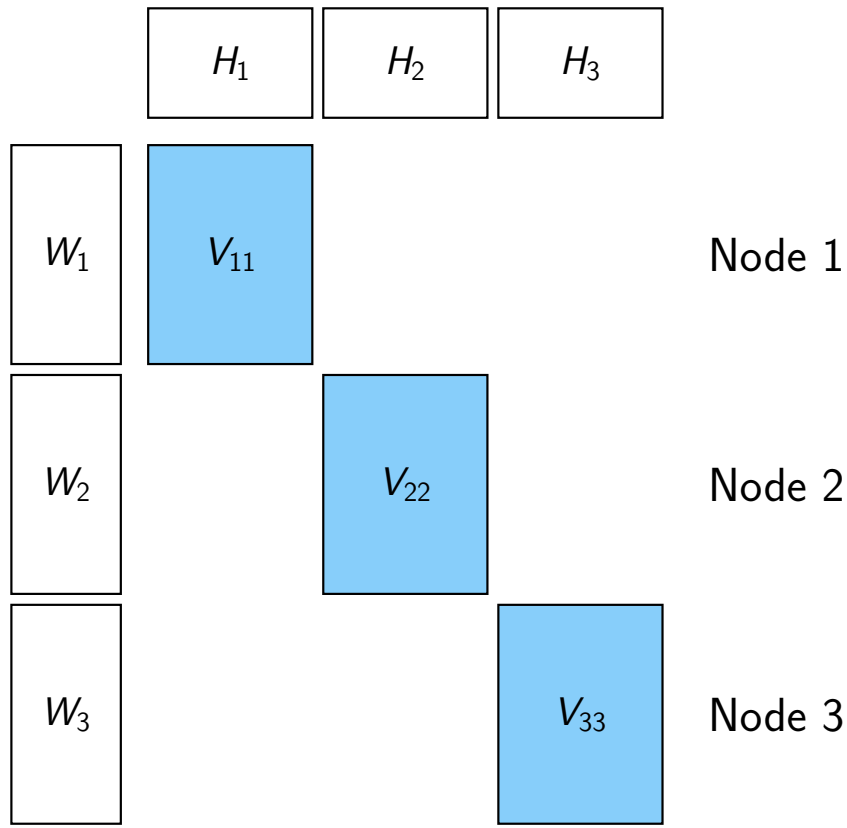
Exploitation

- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
     - Steps 1–3 form a *cycle*



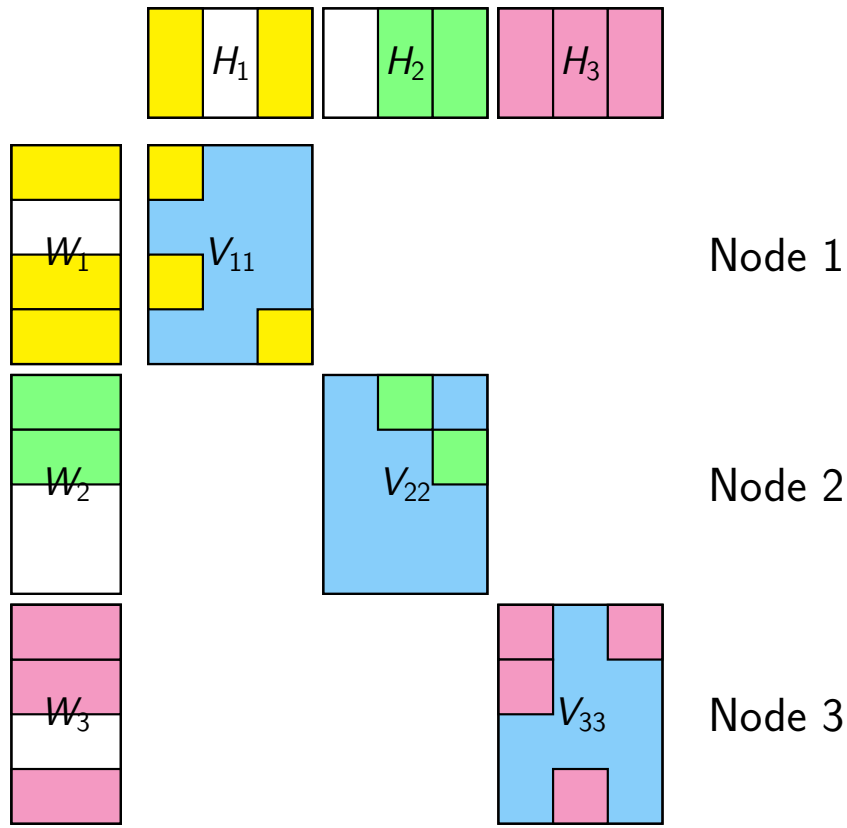
Exploitation

- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
  - Steps 1–3 form a *cycle*



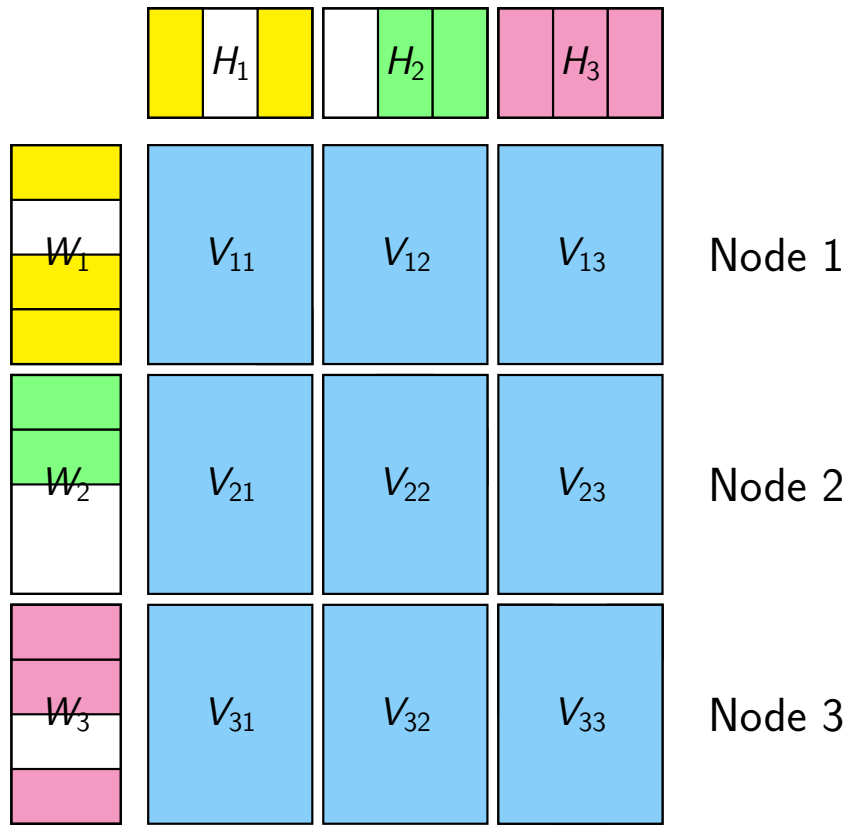
Exploitation

- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
     - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



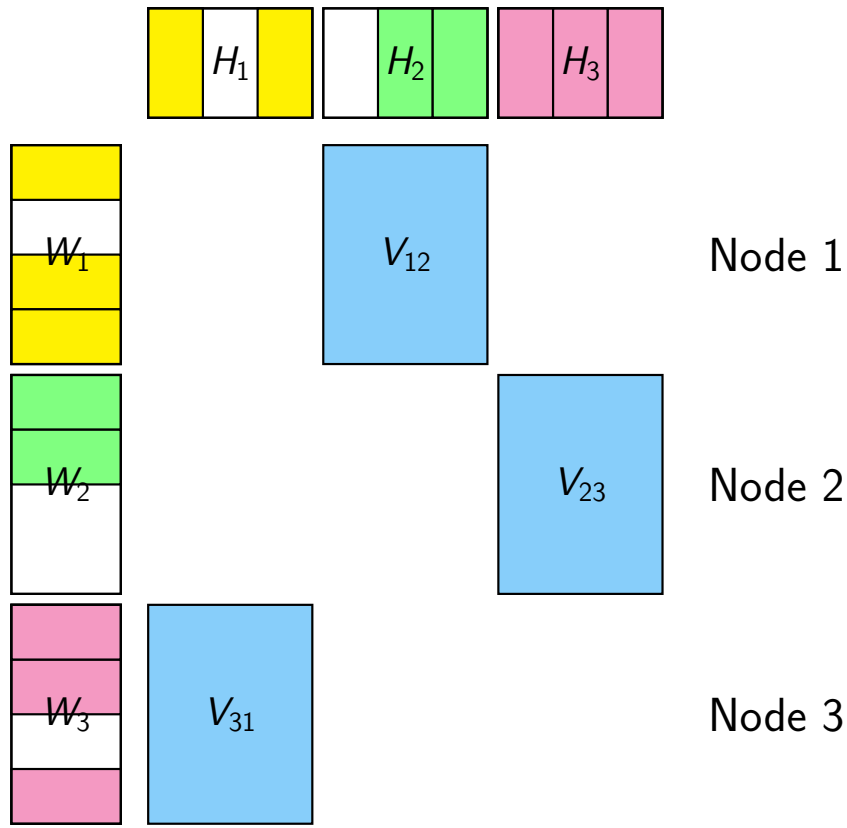
Exploitation

- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
     - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block



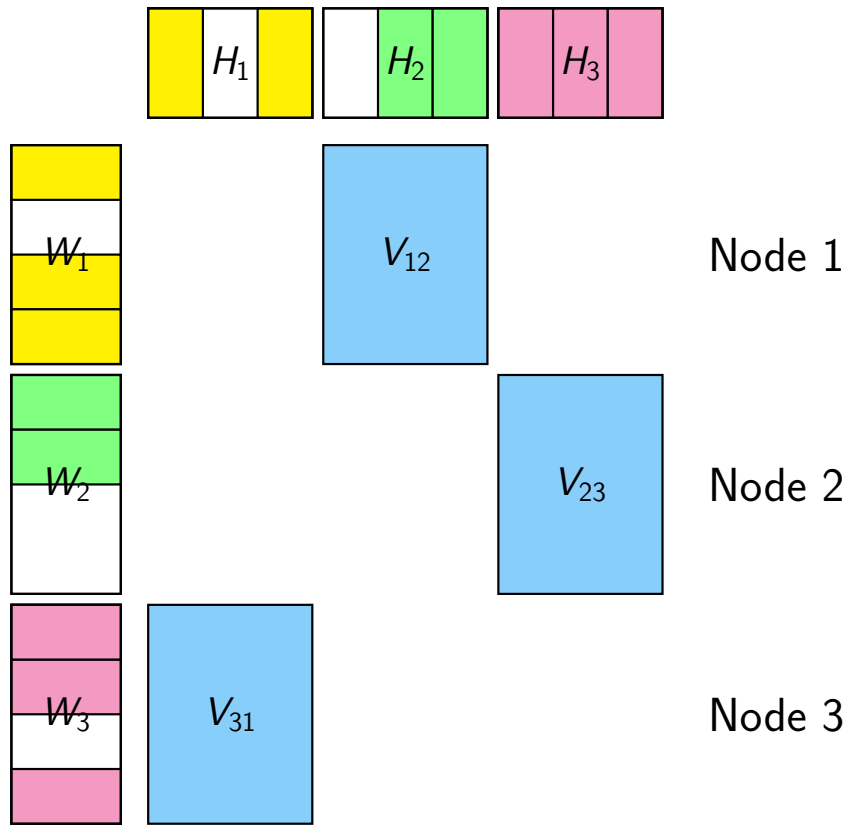
Exploitation

- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
     - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block

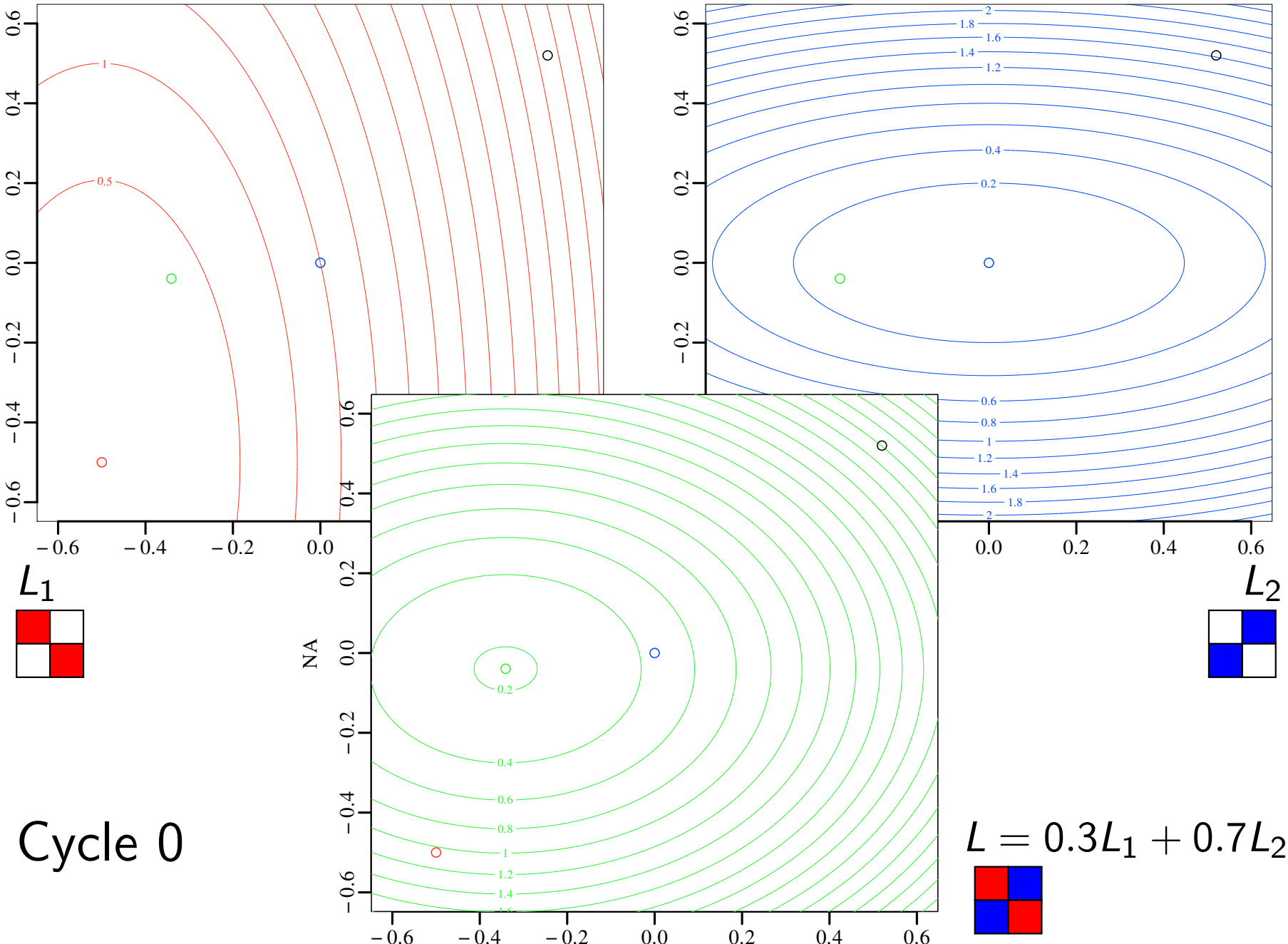


Exploitation

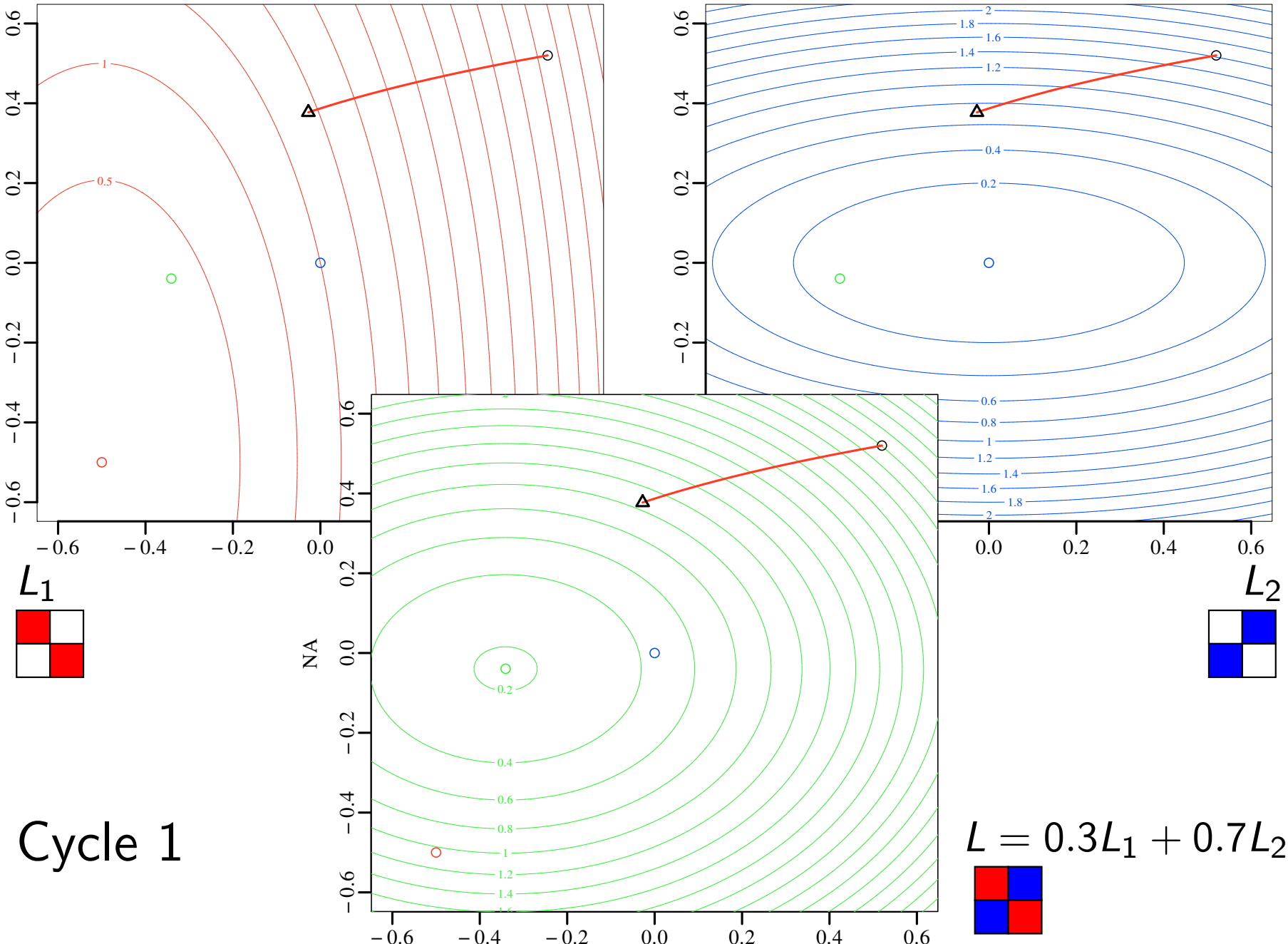
- Block and distribute the input matrix **V**
- High-level approach (Map only)
  1. Pick a "diagonal"
  2. Run SGD on the diagonal (in parallel)
  3. Merge the results
  4. Move on to next "diagonal"
     - Steps 1–3 form a *cycle*
- Step 2:
  Simulate sequential SGD
  - Interchangeable blocks
  - Throw dice of how many iterations per block
  - Throw dice of which step sizes per block
- Instance of "stratified SGD"
- Provably correct



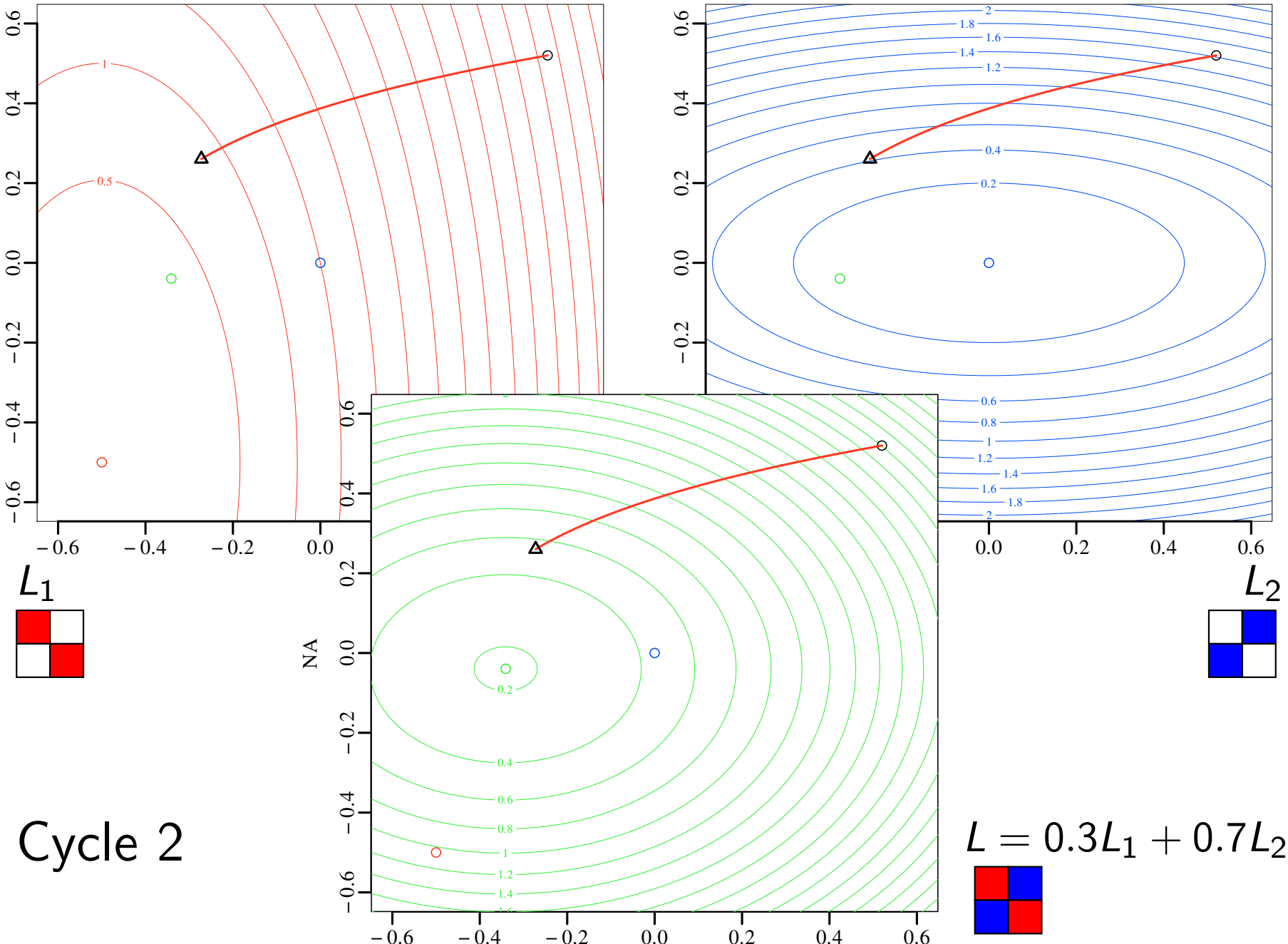
How does it work?



How does it work?



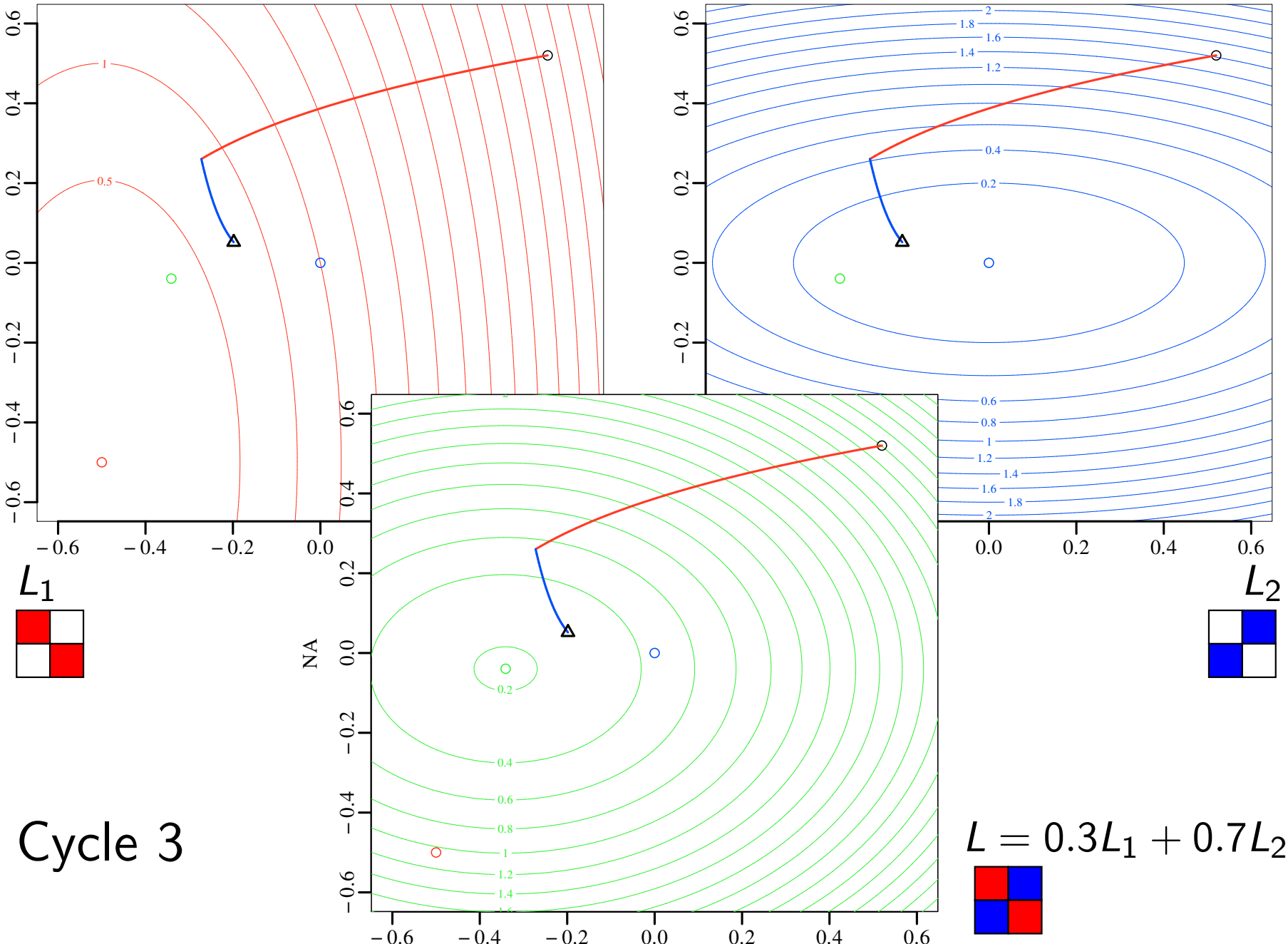
How does it work?



Cycle 2

$L = 0.3L_1 + 0.7L_2$

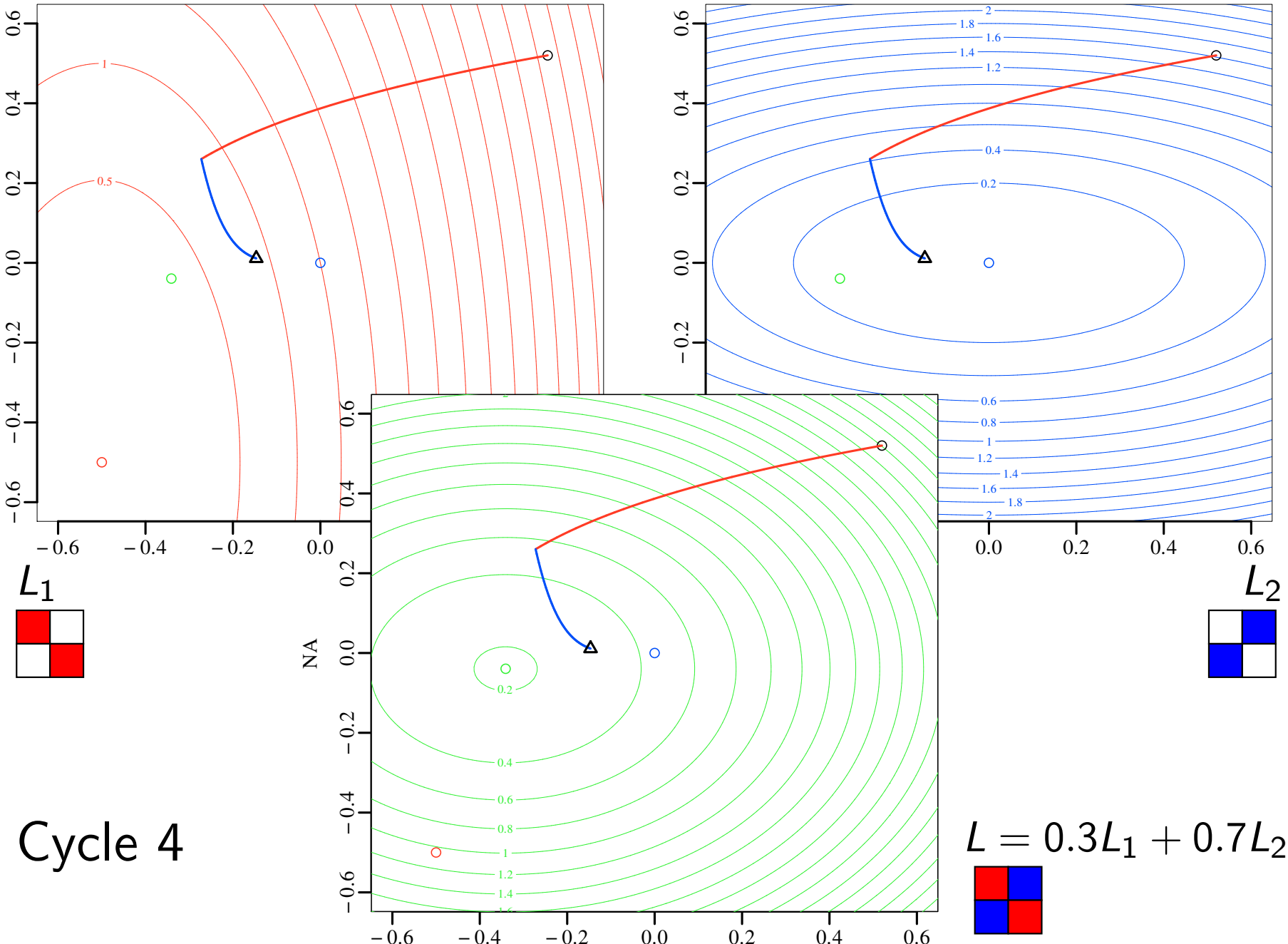
How does it work?



Cycle 3

$L = 0.3L_1 + 0.7L_2$

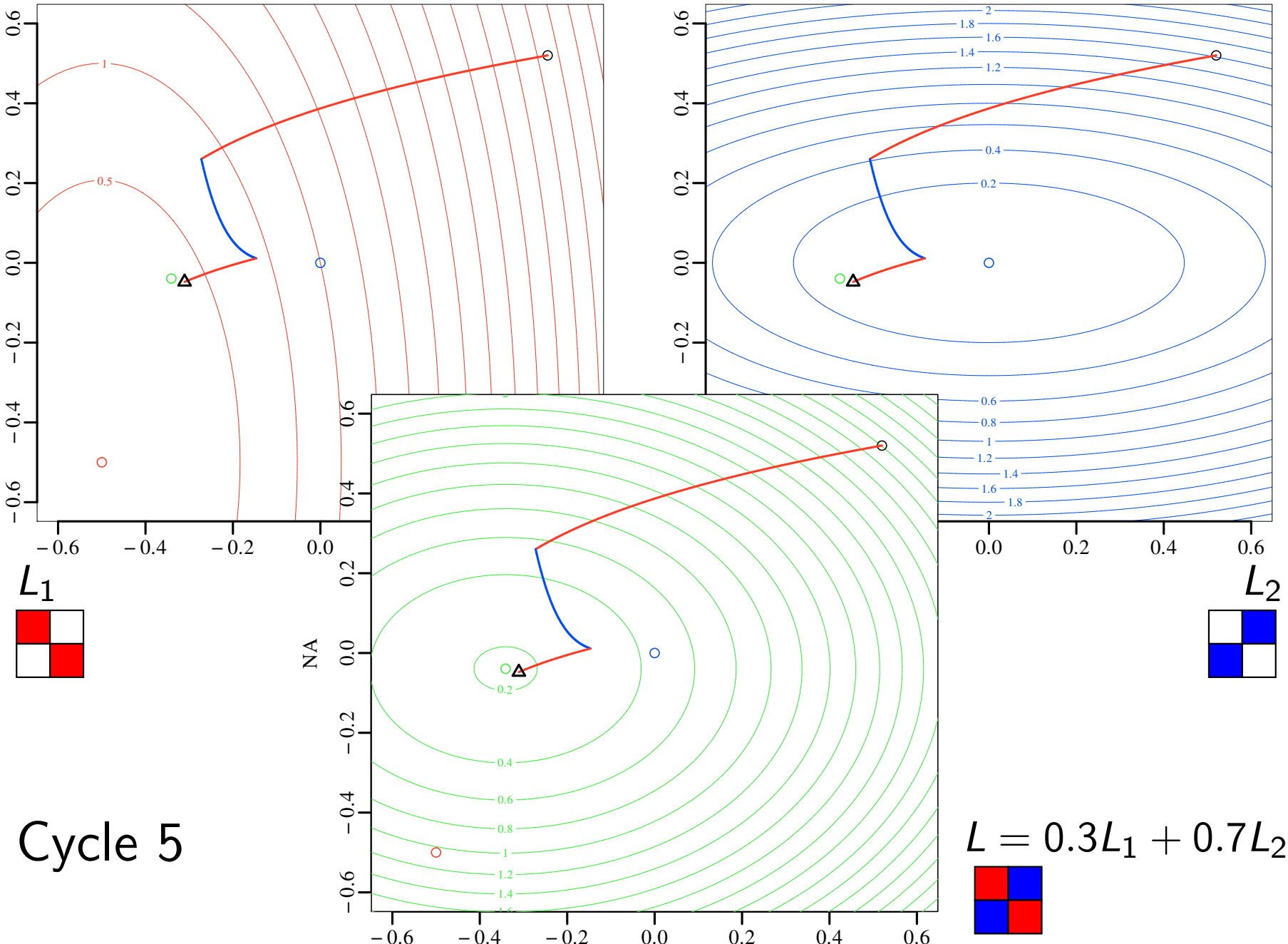
How does it work?



Cycle 4

$L = 0.3L_1 + 0.7L_2$

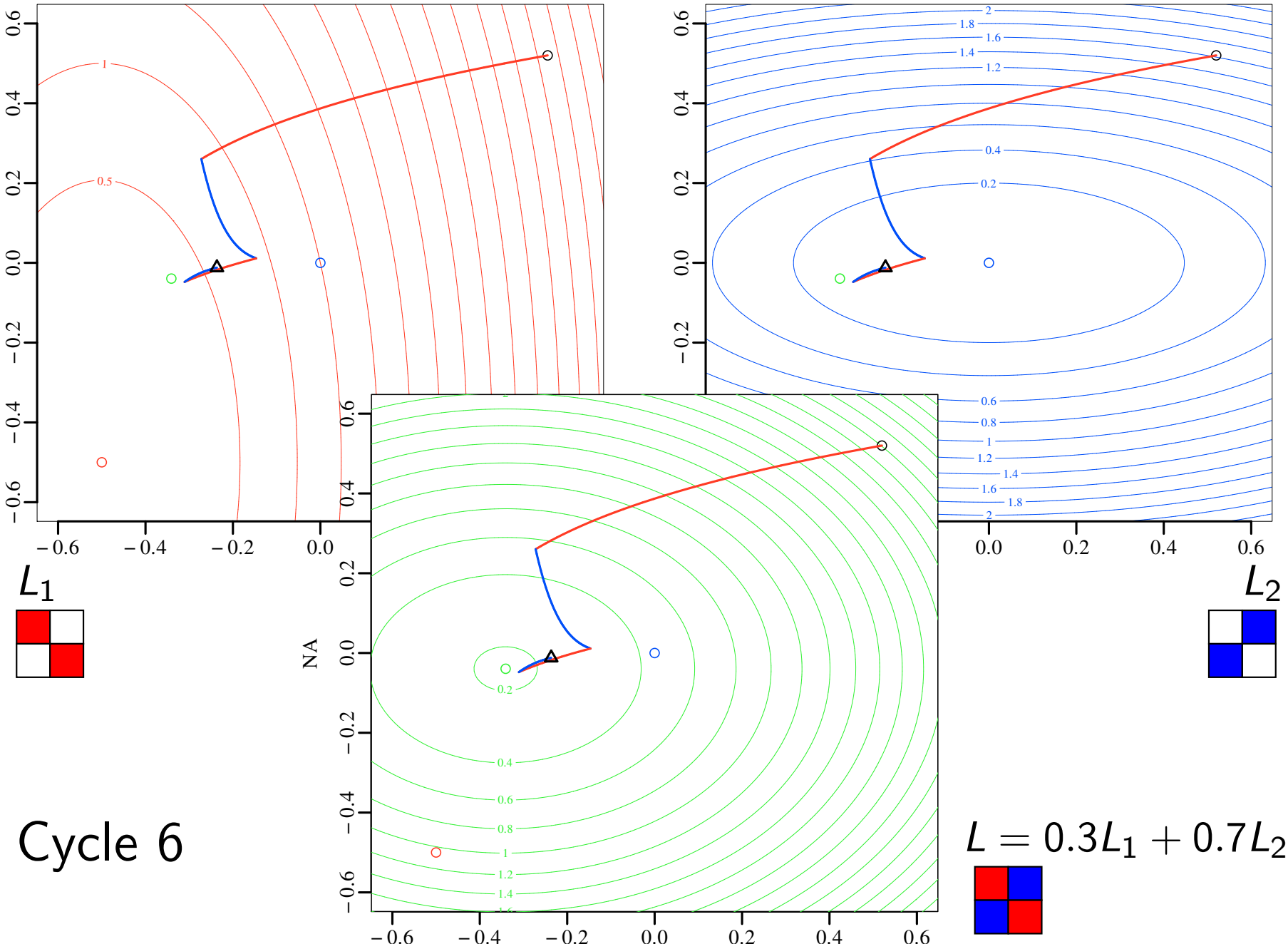
How does it work?



Cycle 5

$L = 0.3L_1 + 0.7L_2$

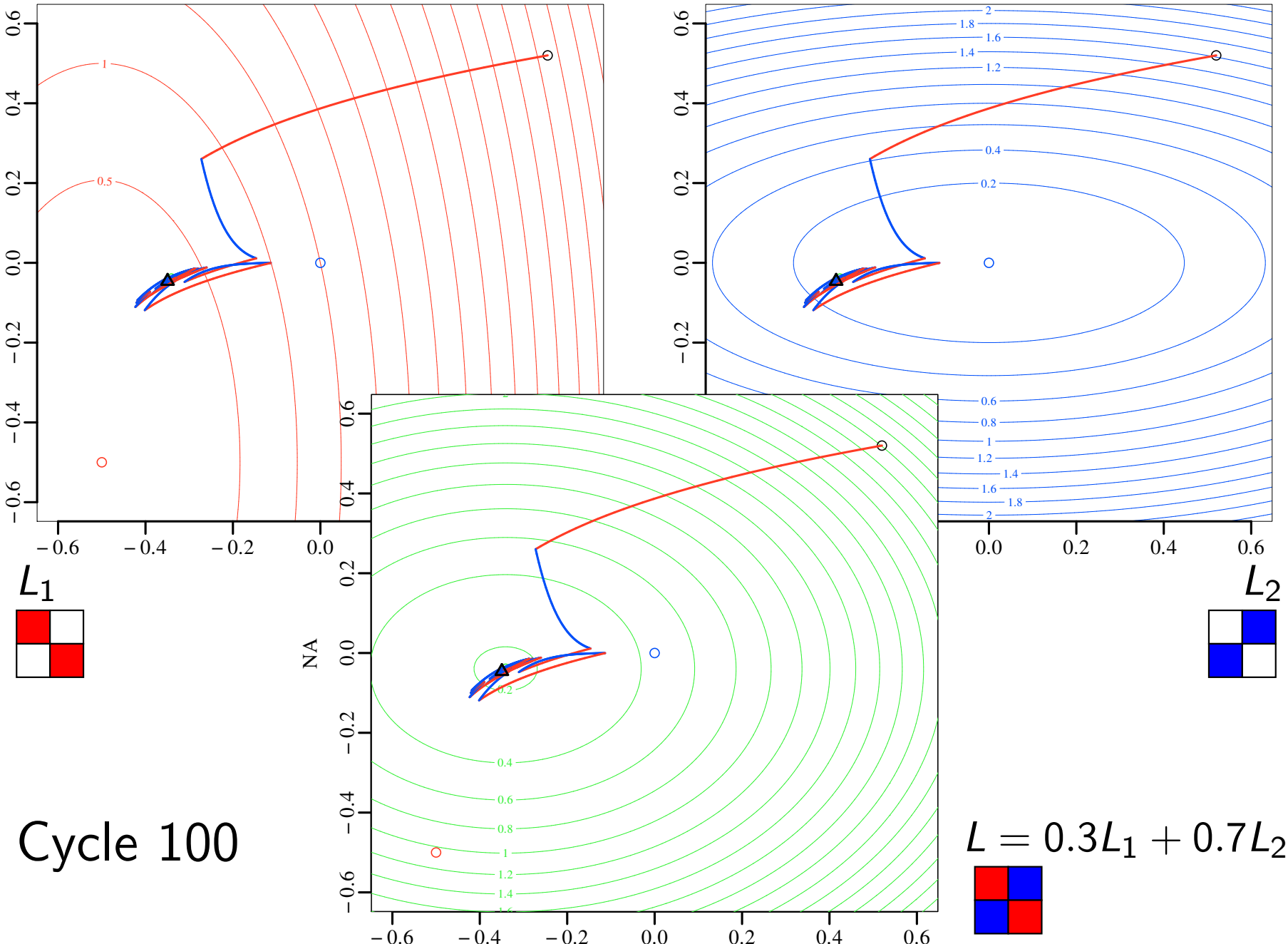
How does it work?



Cycle 6

$L = 0.3L_1 + 0.7L_2$

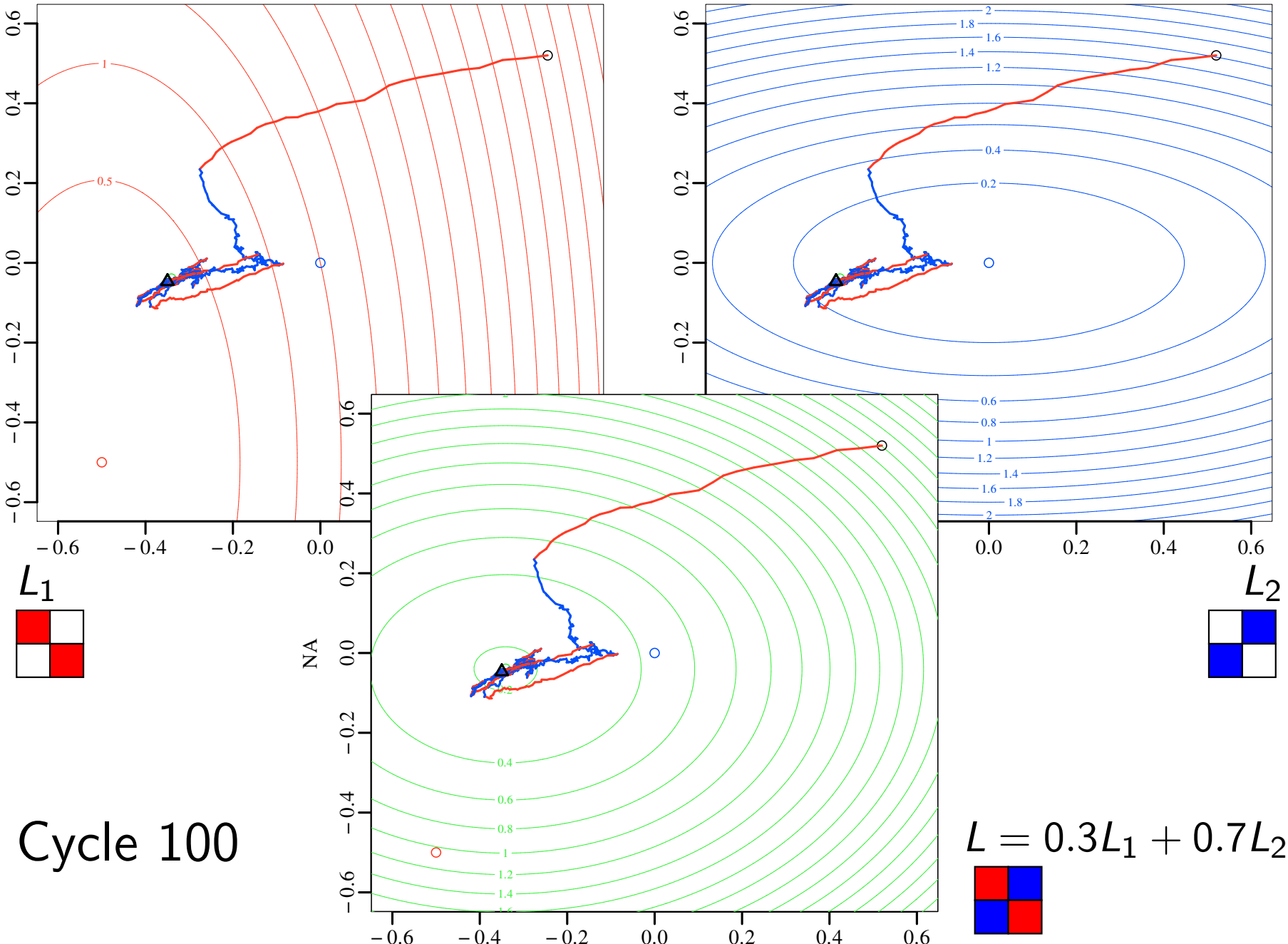
How does it work?



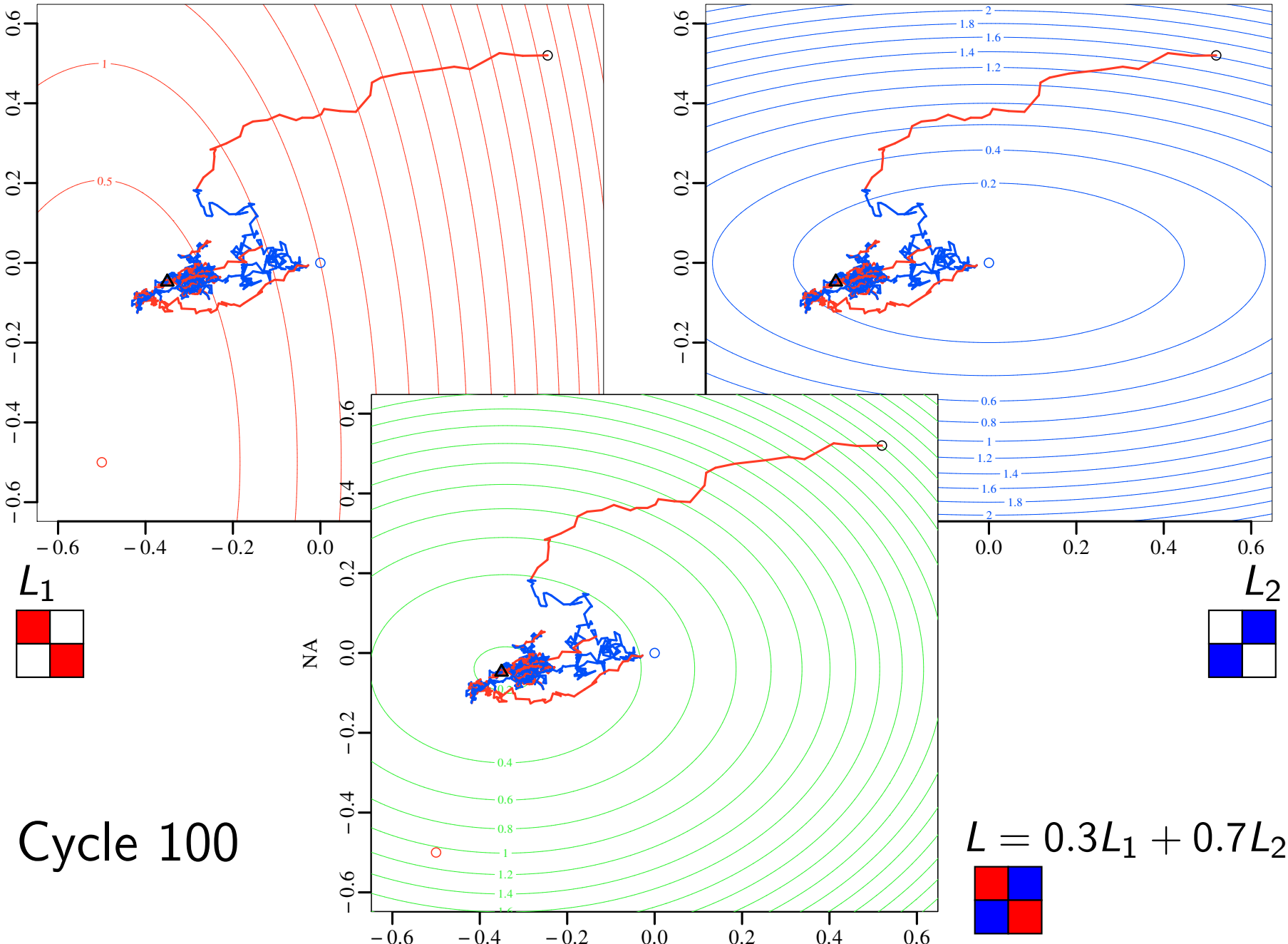
Cycle 100

$L = 0.3L_1 + 0.7L_2$

How does it work?

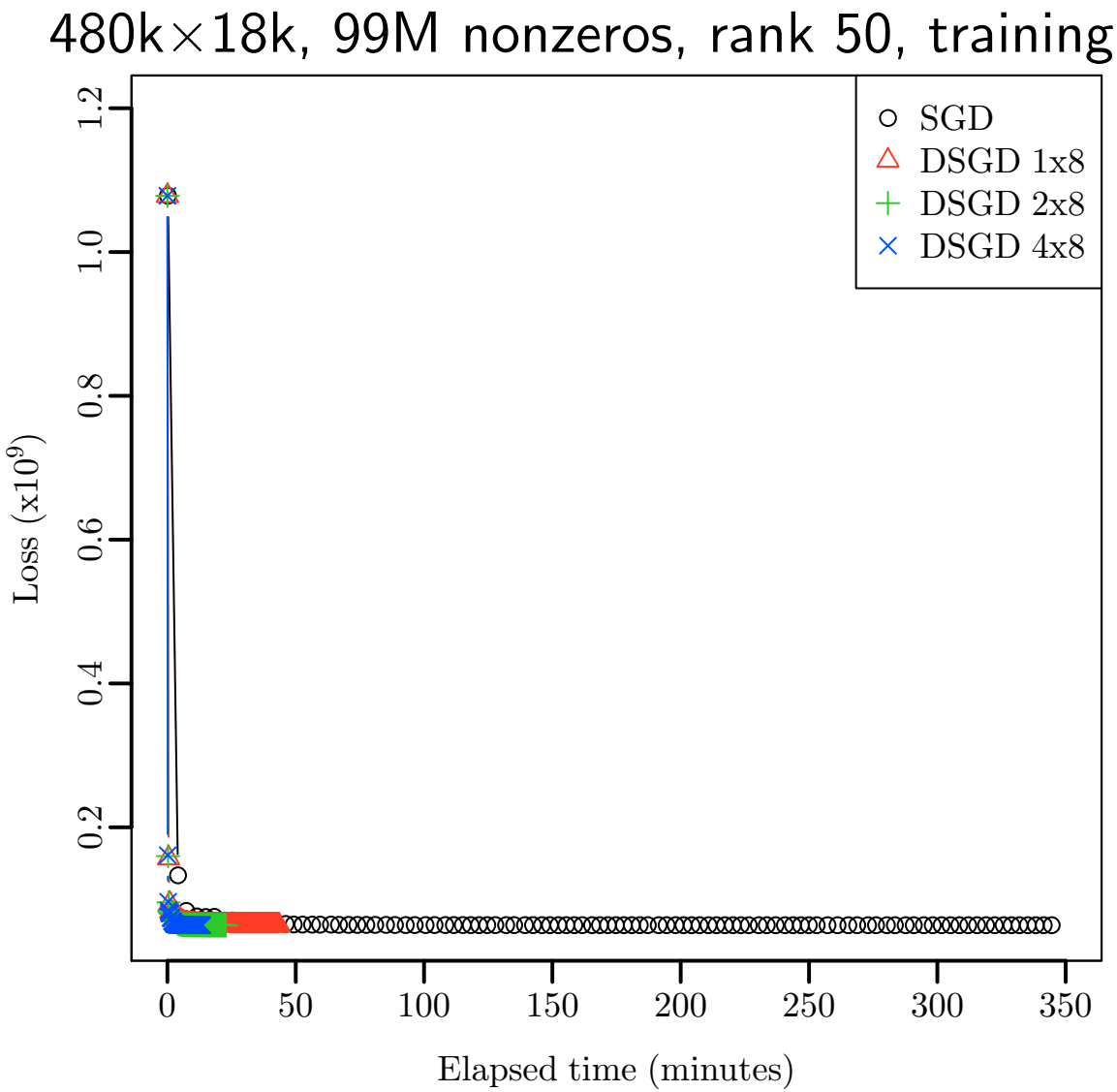


How does it work?

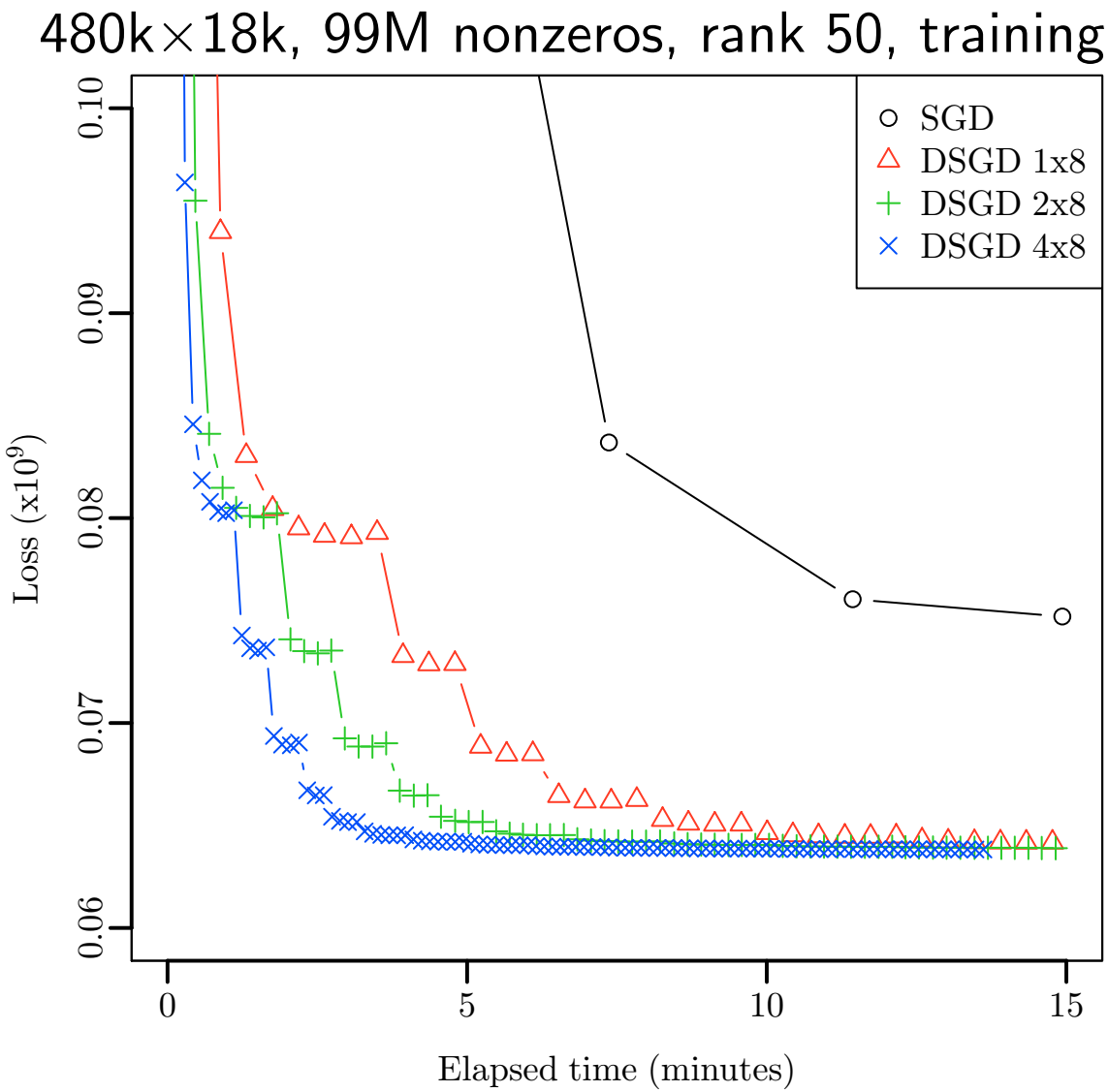


Outline

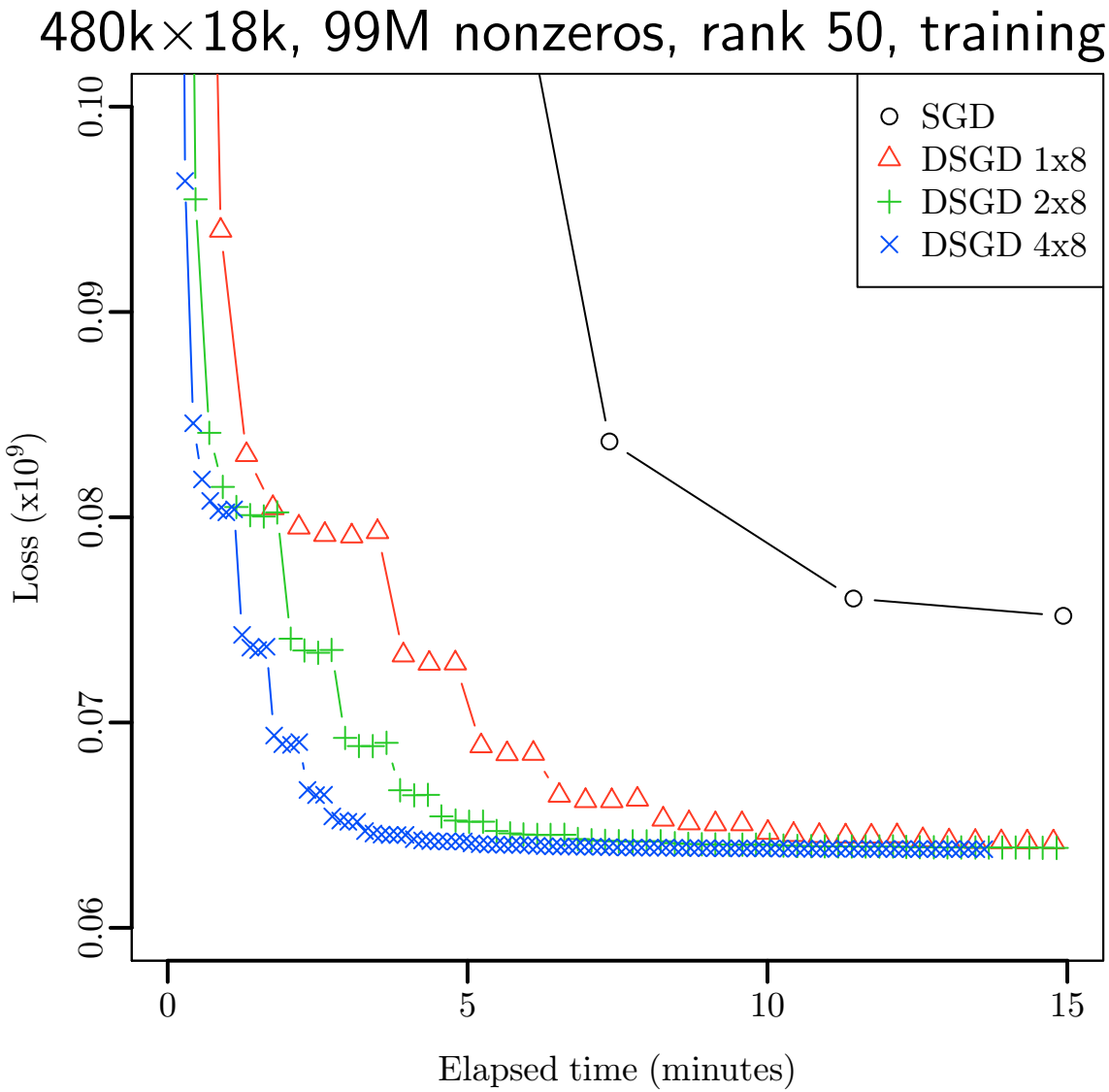
Netflix, Nzsl+L2



Netflix, Nzsl+L2

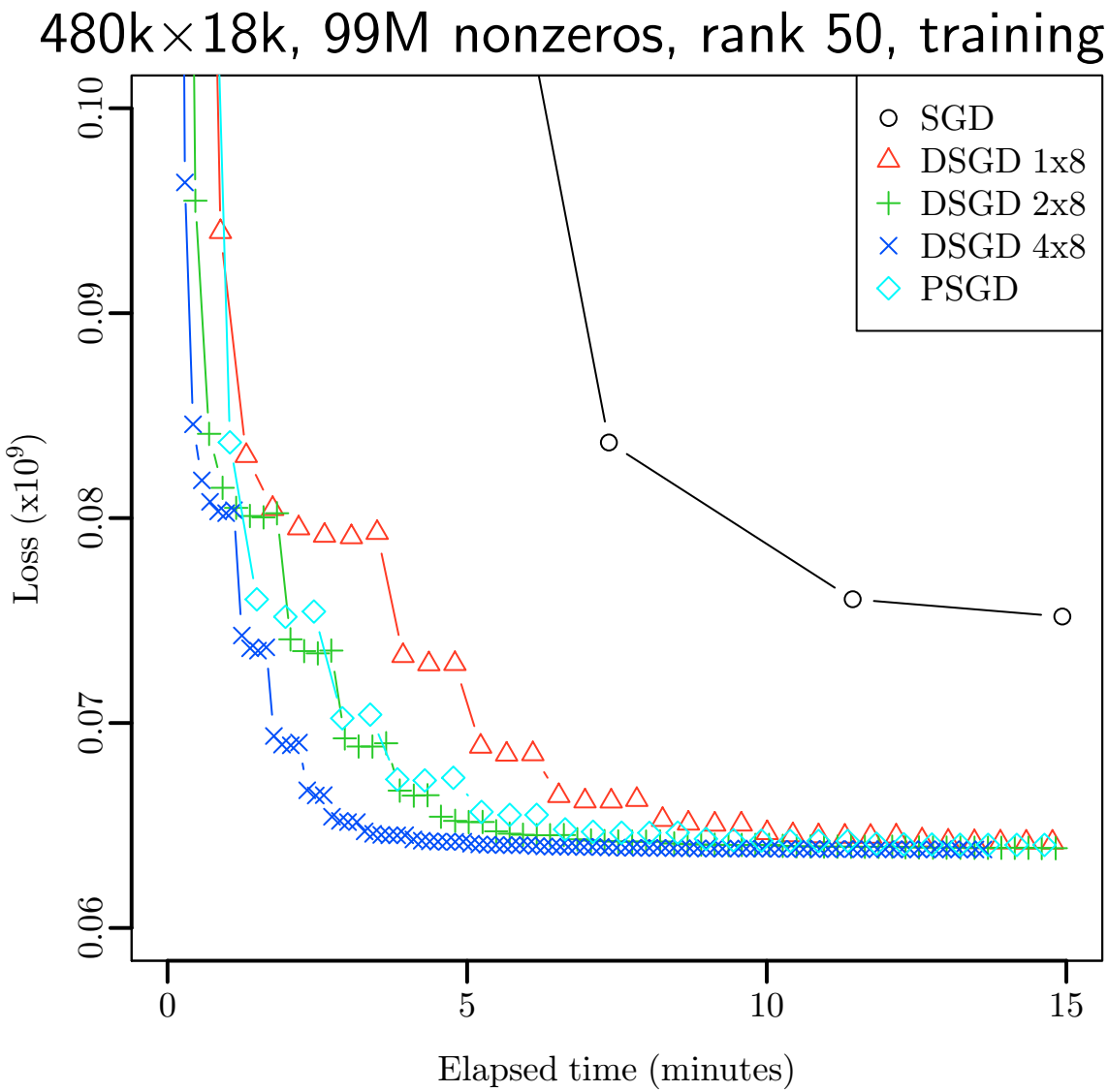


Netflix, Nzsl+L2

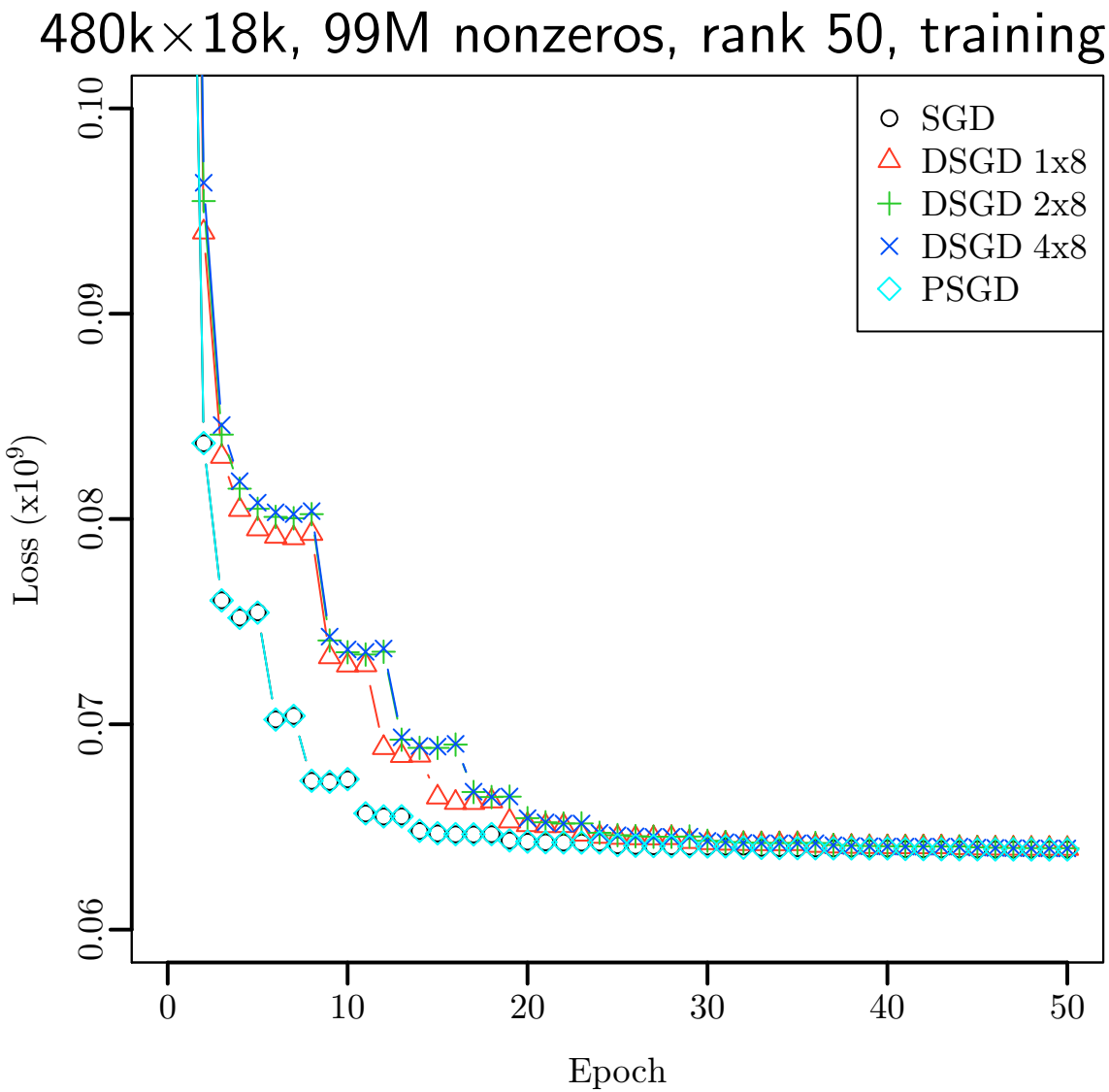


DSGD converges significantly faster than SGD.

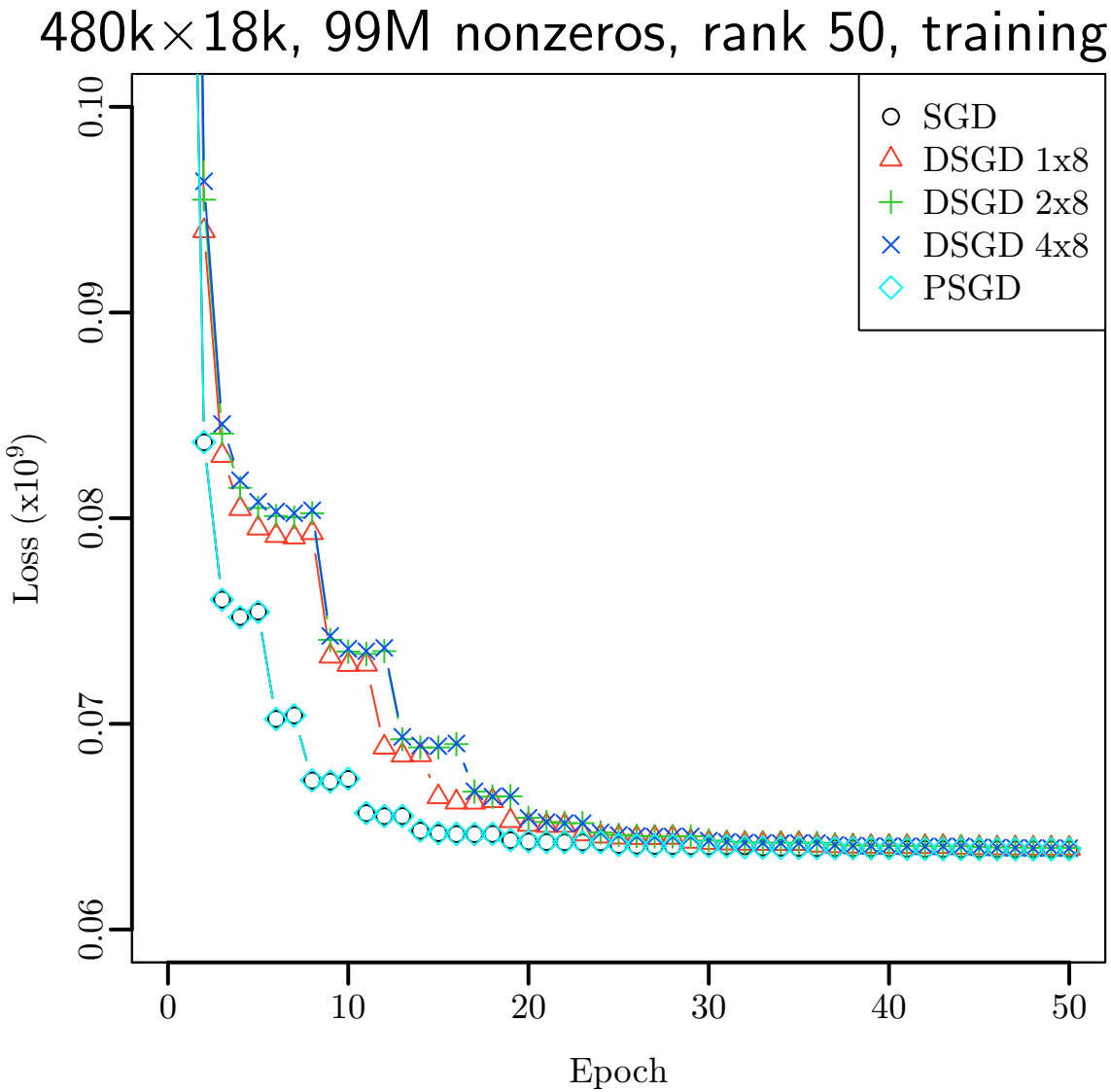
Netflix, Nzsl+L2



Netflix, Nzsl+L2

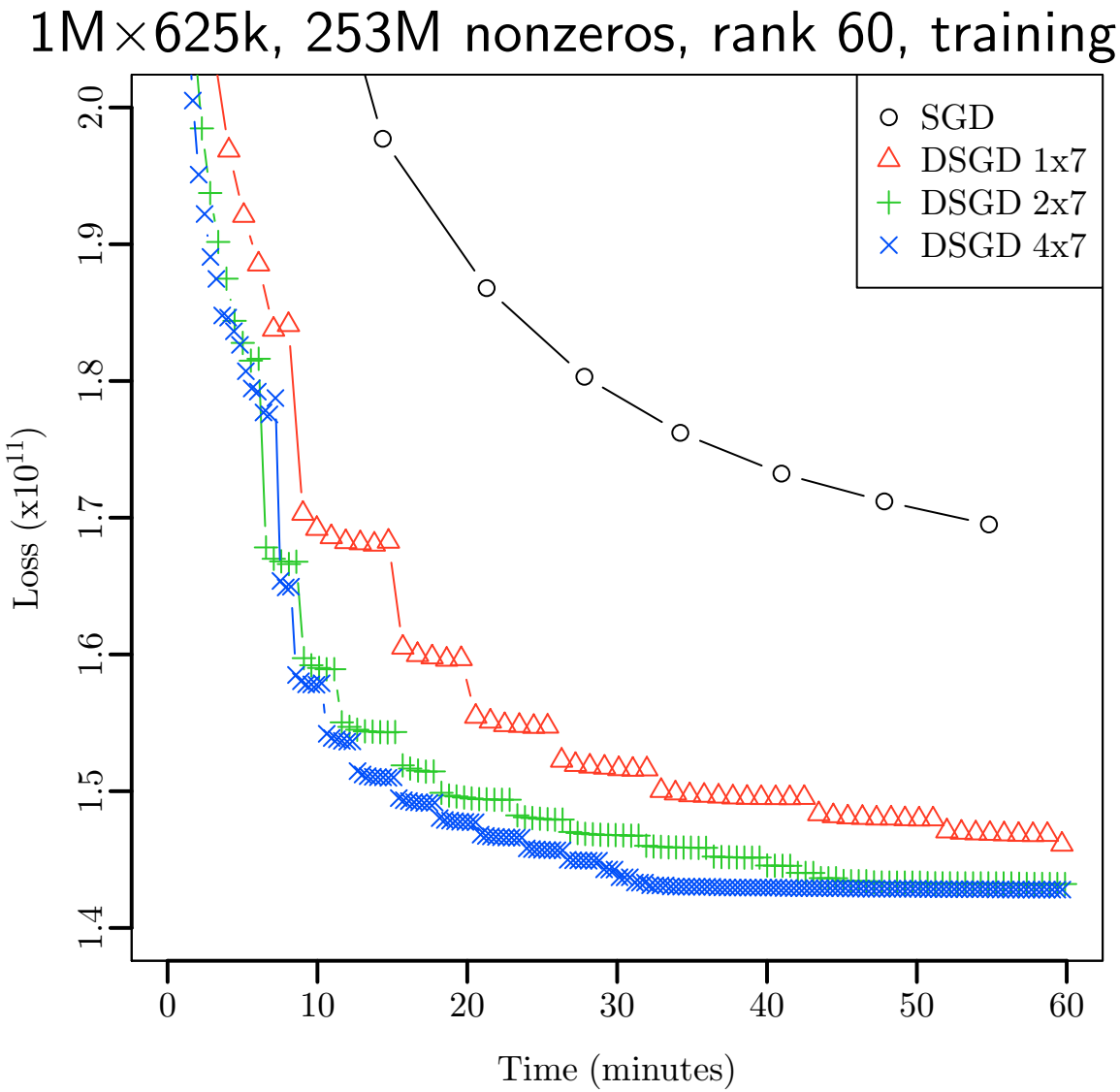


Netflix, Nzsl+L2

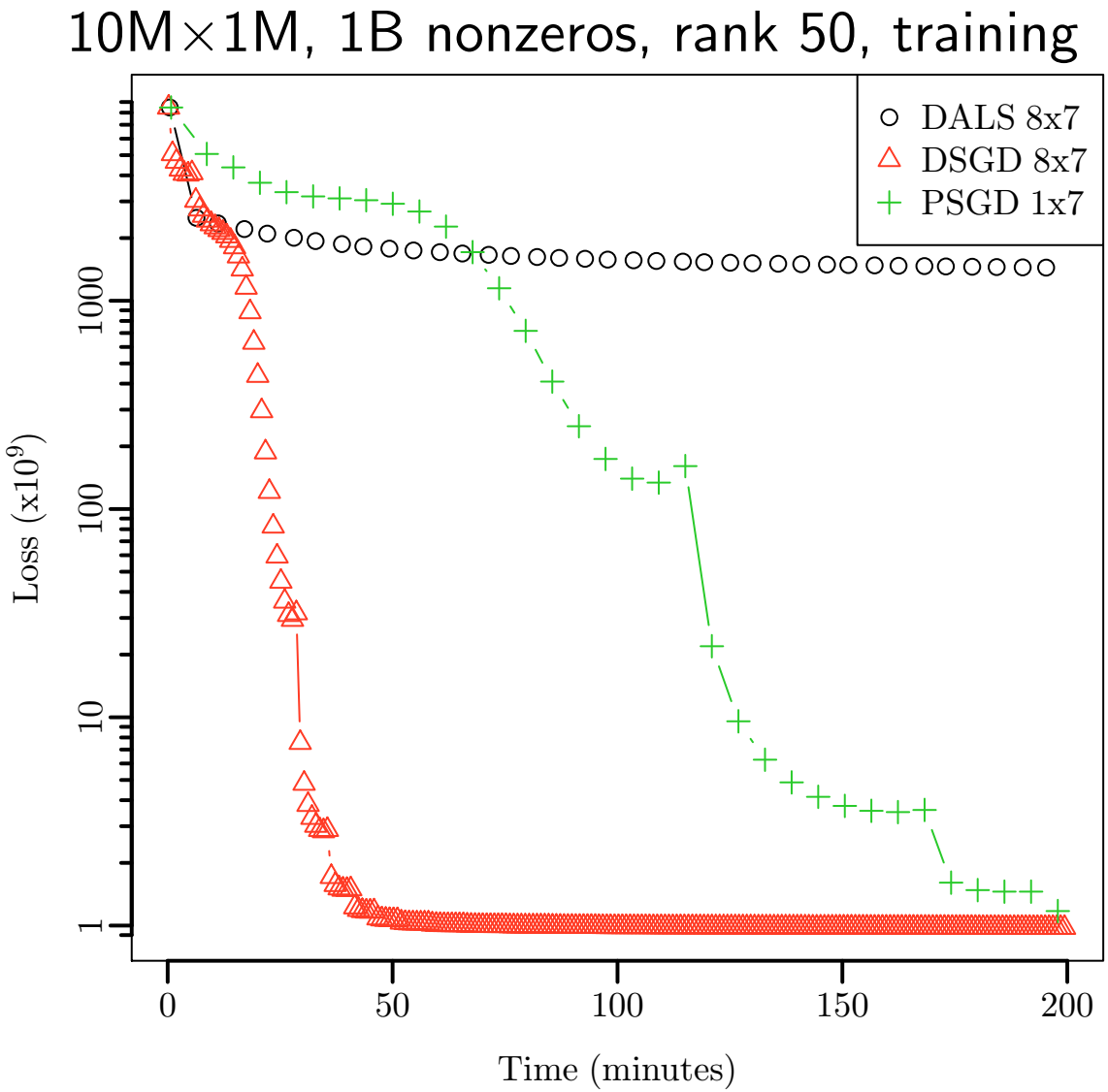


Stratification is (currently) not for free.

KDD Cup, Nzsl+Nzl2+Bias



Synthetic data, Nzsl+L2



Outline

Summary

- Matrix factorization
  - Widely applicable via customized loss functions
  - Large instances (millions $\times$ millions with billions of entries)
- Distributed Stochastic Gradient Descent
  - Simple and versatile
  - Avoids averaging via novel "stratified SGD" variant
  - Achieves
    - Fully distributed data/model
    - Fully distributed processing
    - Competitive to alternative algorithms
    - Fast, scalable
- Future Directions
  - Improved stratification
  - Simultaneous computation & communication
  - Stratification for other models
  - ...

Thank you!