Using Fast Weights to Improve Persistent Contrastive Divergence

Presented by Tijmen Tieleman
Joint work with Geoff Hinton
University of Toronto

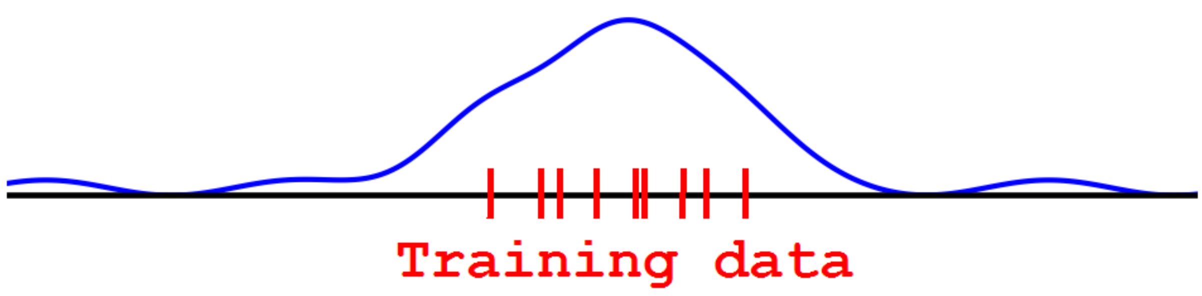
What this is about

- An algorithm for <u>unsupervised</u> learning: modeling data density. Not classification or regression.
- Using Markov Random Fields
  - Also known as:
    - Energy-based models
    - Log-linear models
    - Products of experts, or product models
  - Such as: Restricted Boltzmann Machines

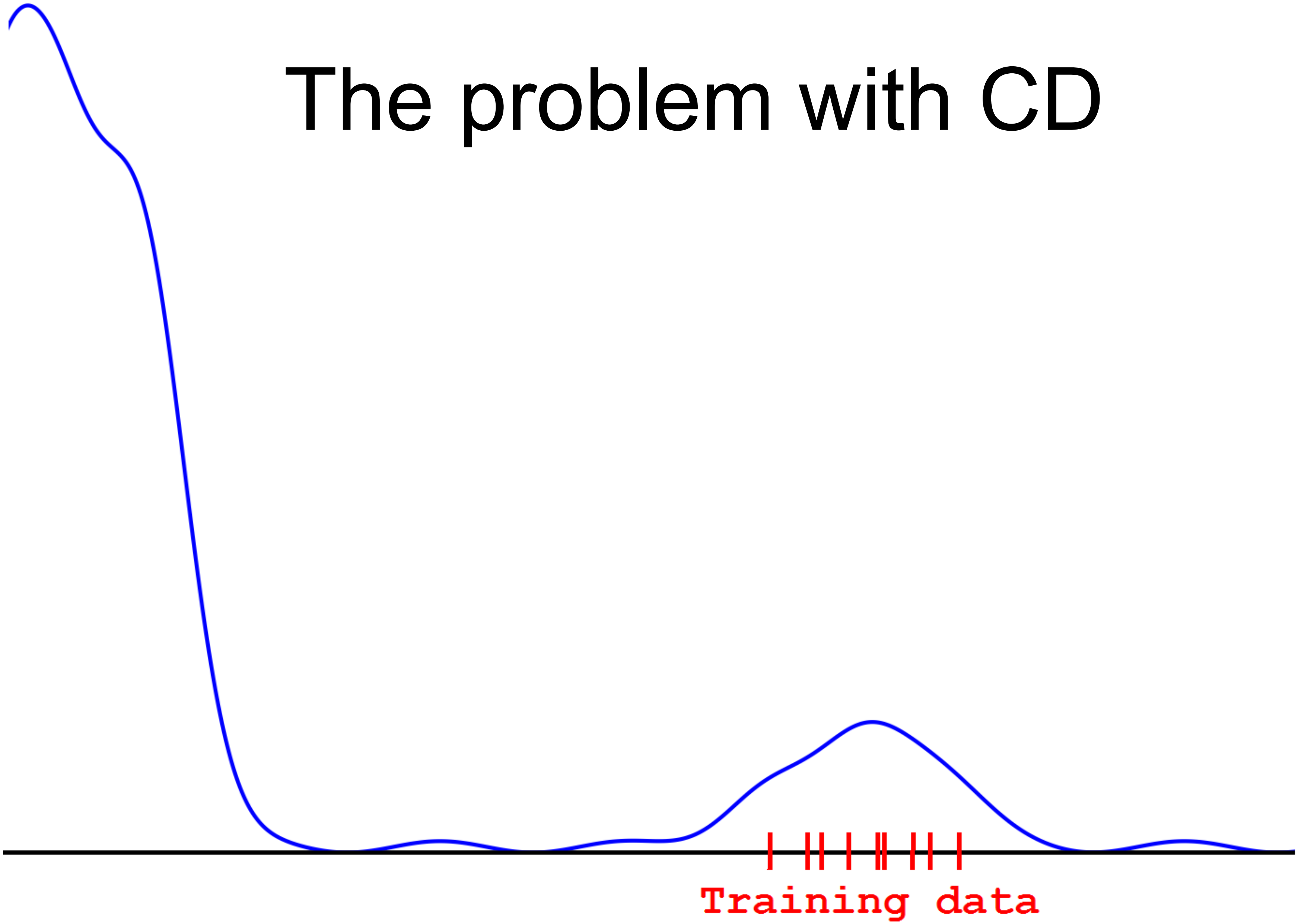
MRF Learning

- Increase probability where training data is
- Decrease probability when the model assigns much probability
  - Problem: finding those places (sampling) is intractable
- CD or PL: use surrogate samples, a few steps away from the training data.
  - It works ☺ Best application paper etc…
  - But it’s not perfect

The problem with CD



The problem with CD



PCD (part 1)

- Gradient descent is iterative.
  - We can reuse data from the previous gradient estimate.
- Use a Markov Chain for getting samples.
- Plan: keep the Markov Chain close to equilibrium.
- Do a few transitions after each weight update.
  - Thus the Chain catches up after the model changes.
- Do not reset the Markov Chain after a weight update (hence ‘Persistent’ CD).
- Thus we always have samples from very close to the model.

PCD (part 2)

- If we would not change the model at all, we would have <u>exact</u> samples (after burn-in). It would be a regular Markov Chain.
- The model changes <u>slightly</u>,
  – So the Markov Chain is always <u>a little</u> behind.

PCD Pseudocode

- Initialize 100 Markov Chains, *negData,* arbitrarily
- Initialize the model, *theta*, with small random weights
- Repeat:
  - Get *positive gradient* on a batch of training data
  - Get *negative gradient* on *negData*.
  - Do learning on *theta* using the difference of gradients.
  - Update *negData* using a Gibbs update on the current model.

Let’s take a step back

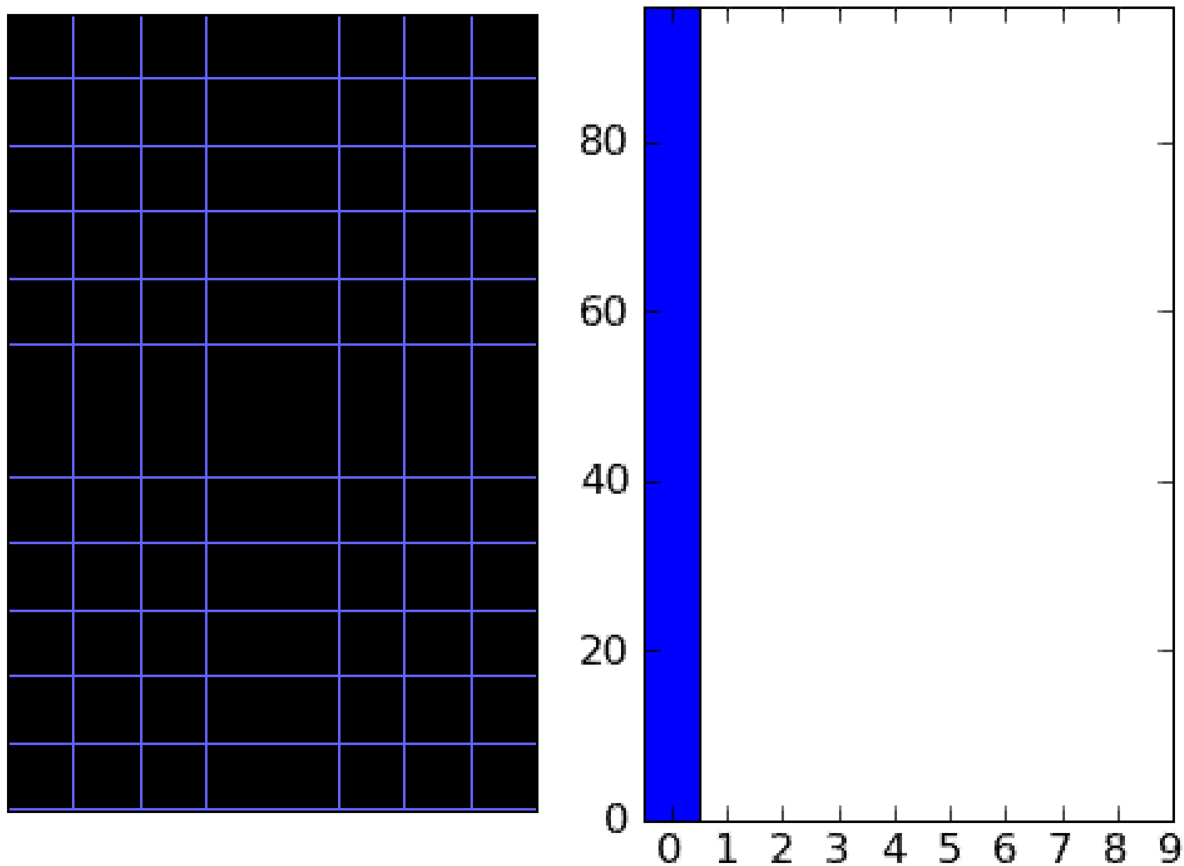
- Gradient descent is iterative.
  - We can reuse data from the previous estimate.
- Use a Markov Chain for getting samples.
- Plan: keep the Markov Chain close to equilibrium.
- Do a few transitions after each weight update.
  - Thus the Chain catches up after the model changes.
- Do not reset the Markov Chain after a weight update (hence ‘Persistent’ CD).
- Thus we always have samples from very close to the model.

Really?

- Gradient descent is iterative.
  - We can reuse data from the previous estimate.
- Use a Markov Chain for getting samples.
- Plan: keep the Markov Chain close to equilibrium.
- Do a few transitions after each weight update.
  - Thus the Chain catches up after the model changes.
- Do not reset the Markov Chain after a weight update (hence ‘Persistent’ CD).
- Thus we always have samples from very close to the model.

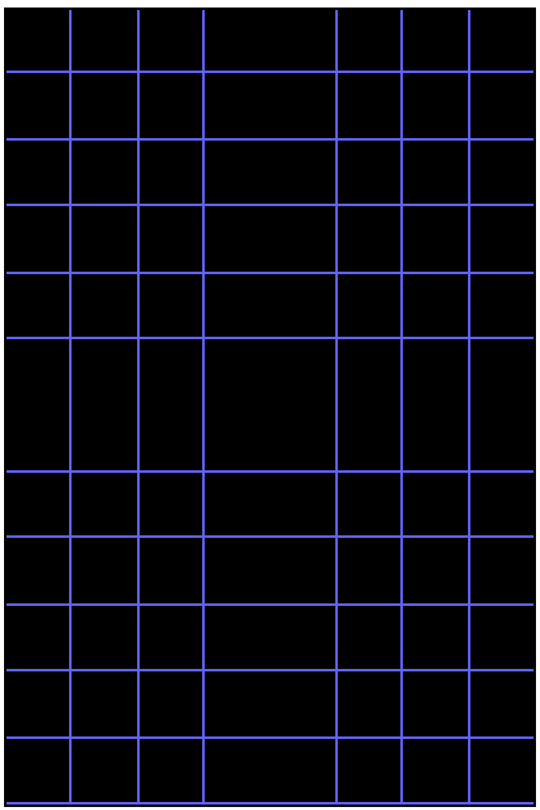
The mixing rate

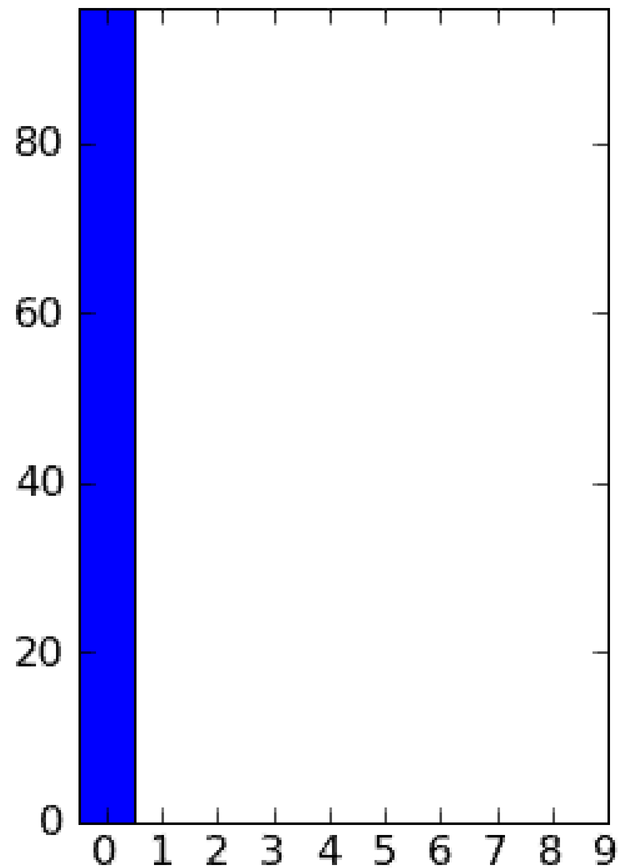
- Markov Chain (i.e. PCD with learning rate 0)



The mixing rate

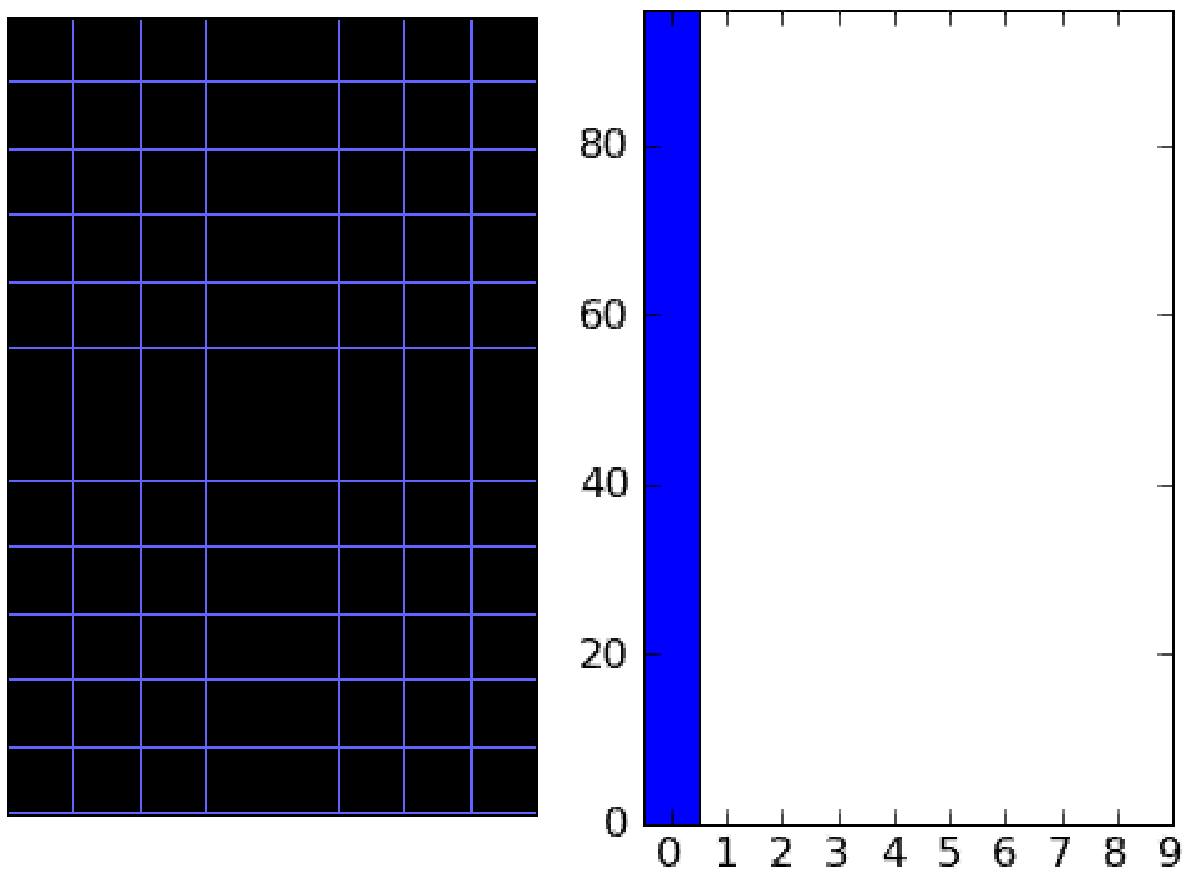
- Learning rate: 0.00003





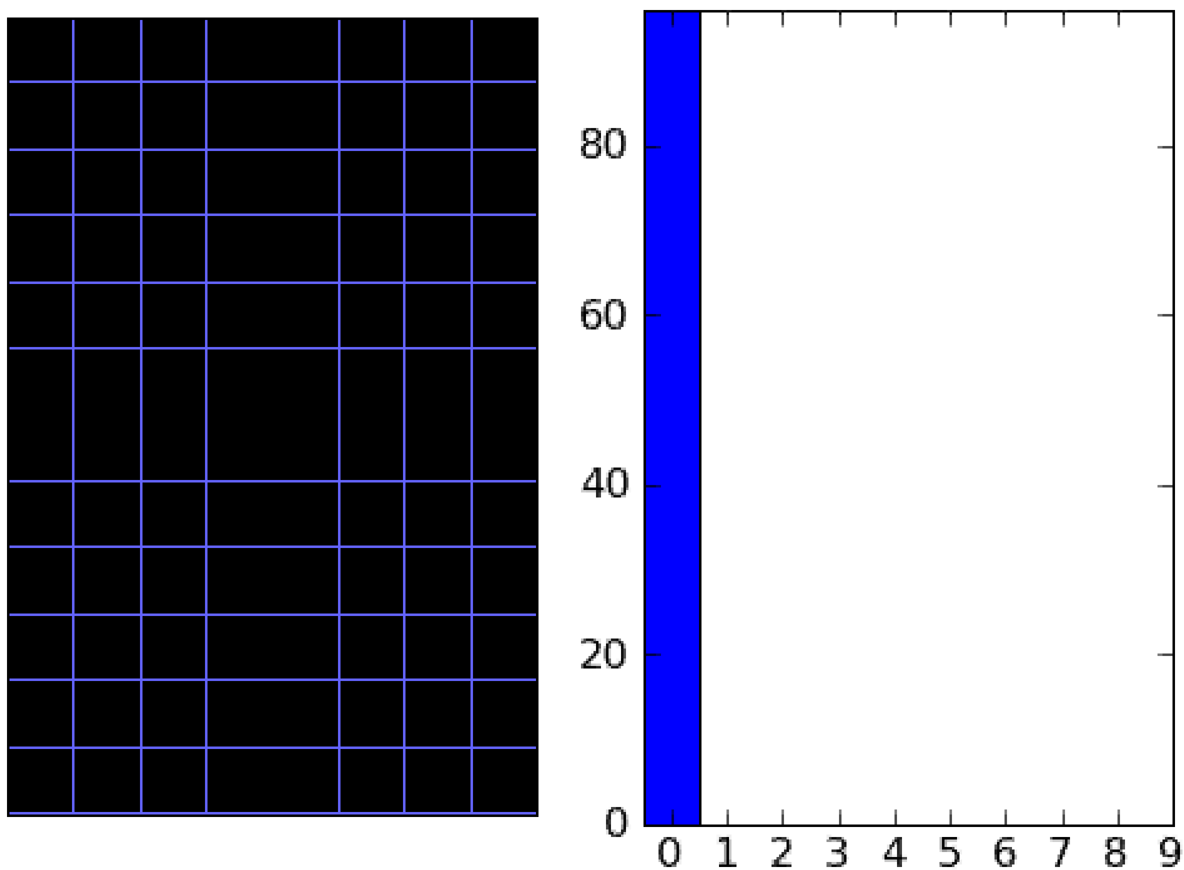
The mixing rate

- Learning rate: 0.0001



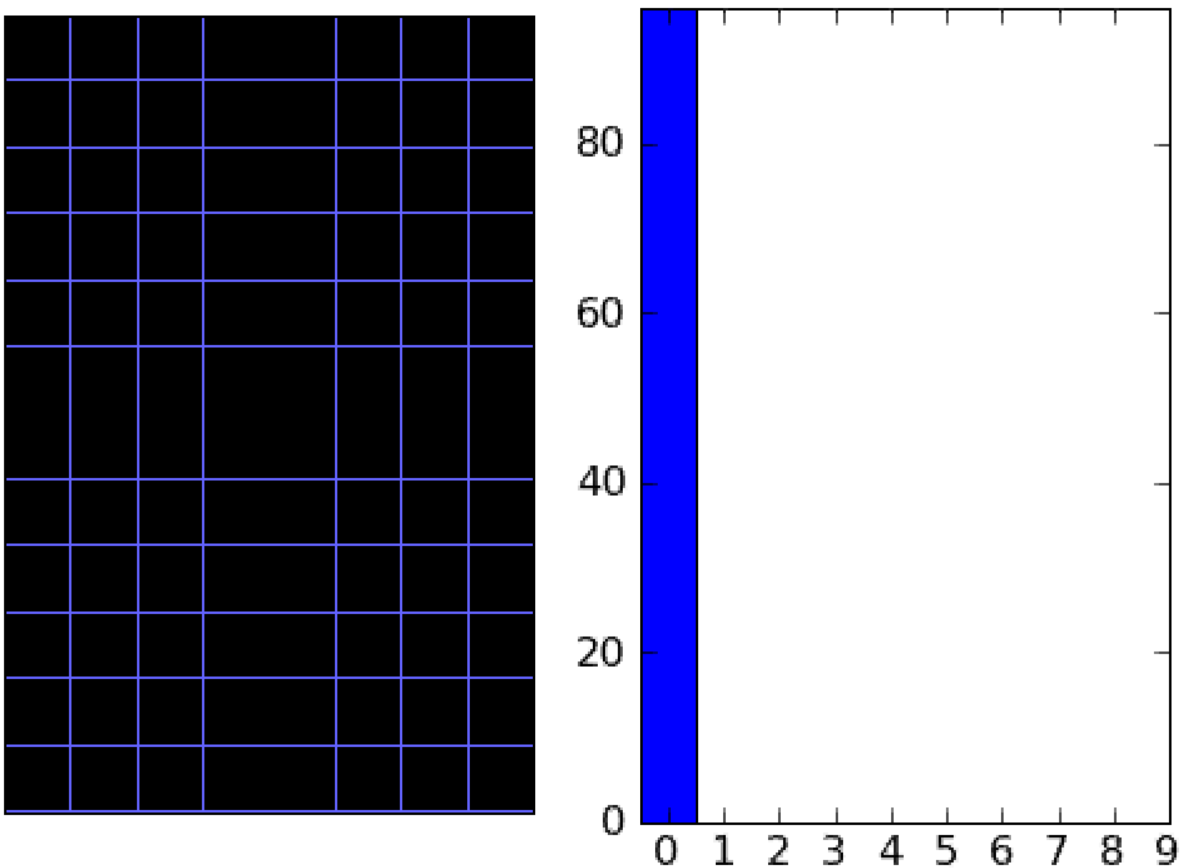
The mixing rate

- Learning rate: 0.0003



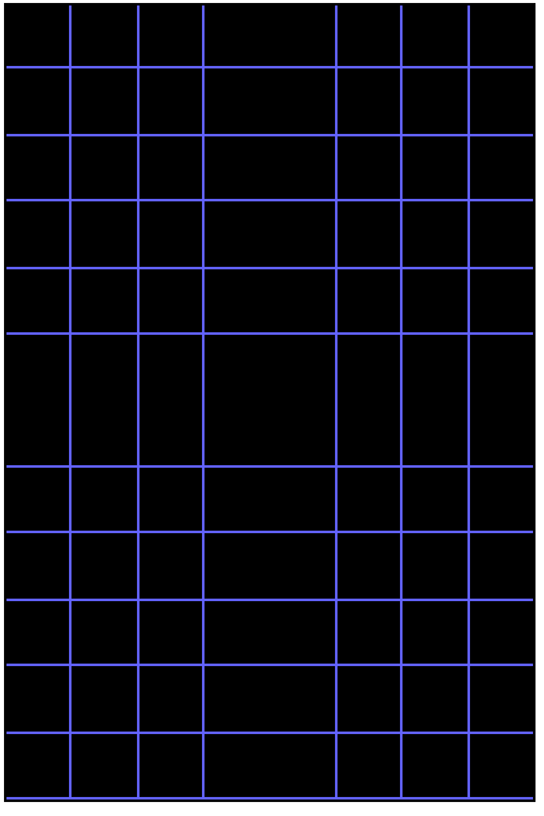
The mixing rate

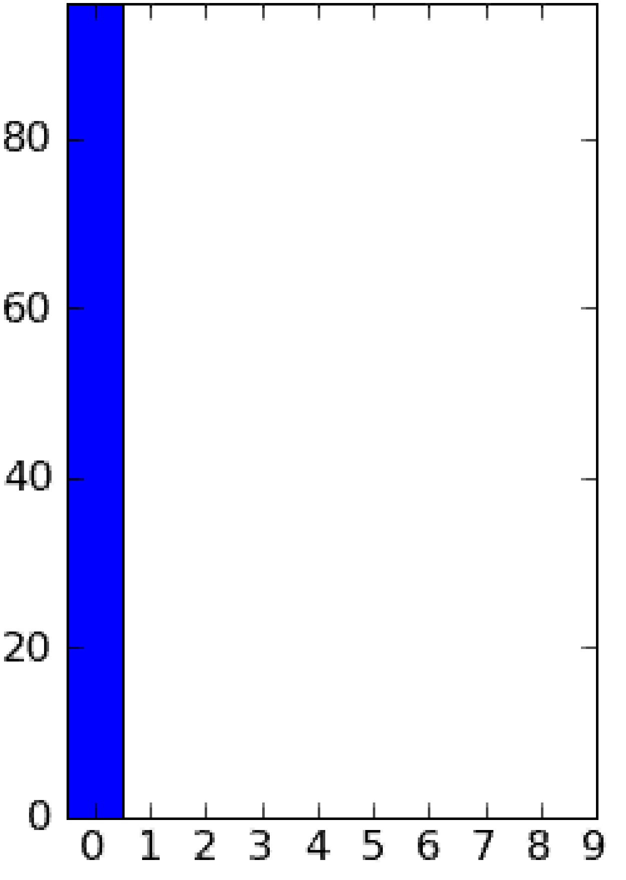
- Learning rate: 1



The mixing rate

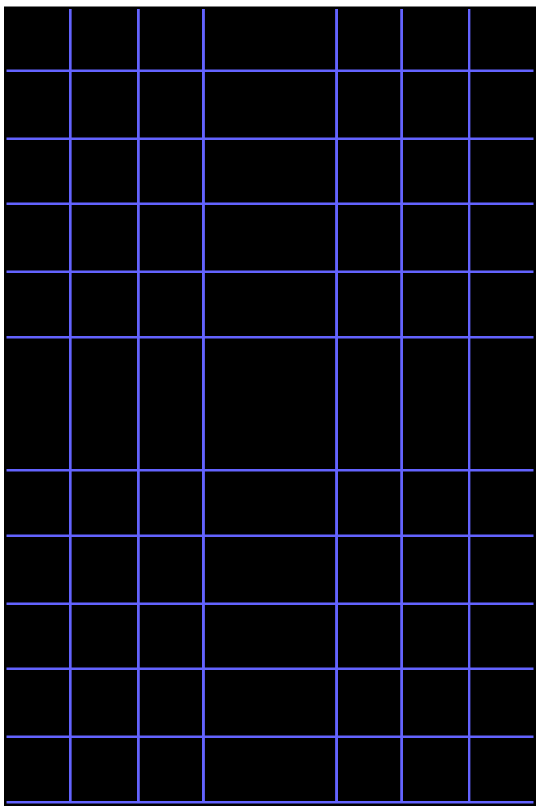
- Learning rate: 3

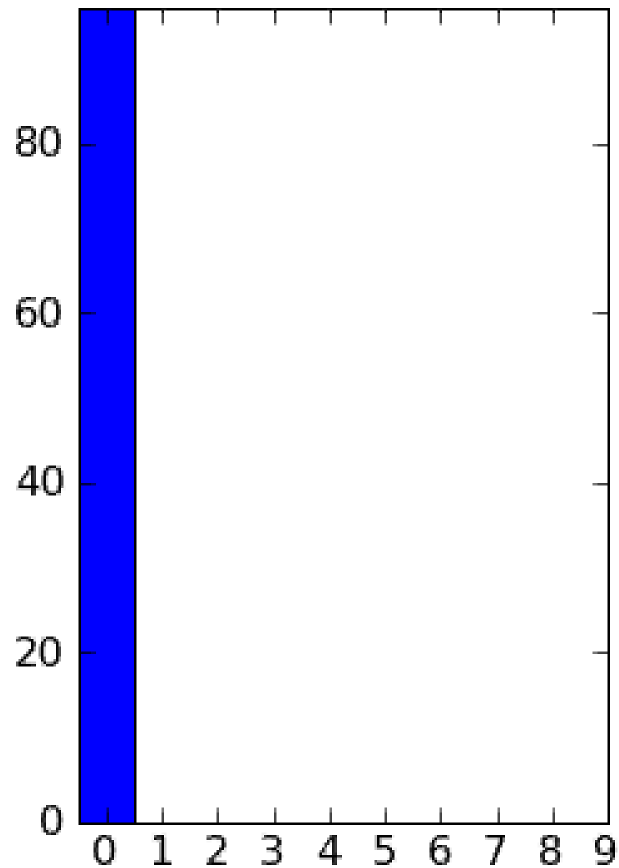




The mixing rate

- Learning rate: 10





Learning accelerates mixing

- Negative phase: “wherever *negData* is, reduce probability *(do unlearning)”*
- Suddenly, those *negData* Markov Chains are in a low probability area
- Therefore, they quickly move away, to a higher probability area
- Repeat… repeat…
- Similar but deterministic: “Herding Dynamical Weights to Learn” by Max Welling (poster today)

New idea

- Learning makes the chain mix
- Faster learning makes the chain mix faster…
- …but going too fast will mess up the learning
- Keep <u>separate</u> 'fast' weights that learn rapidly
  - Keep them close to the regular weights
  - The chains mix using the fast weights
  - The chains provide good samples for learning
- On the regular weights, the learning rate is small.

FPCD Pseudocode

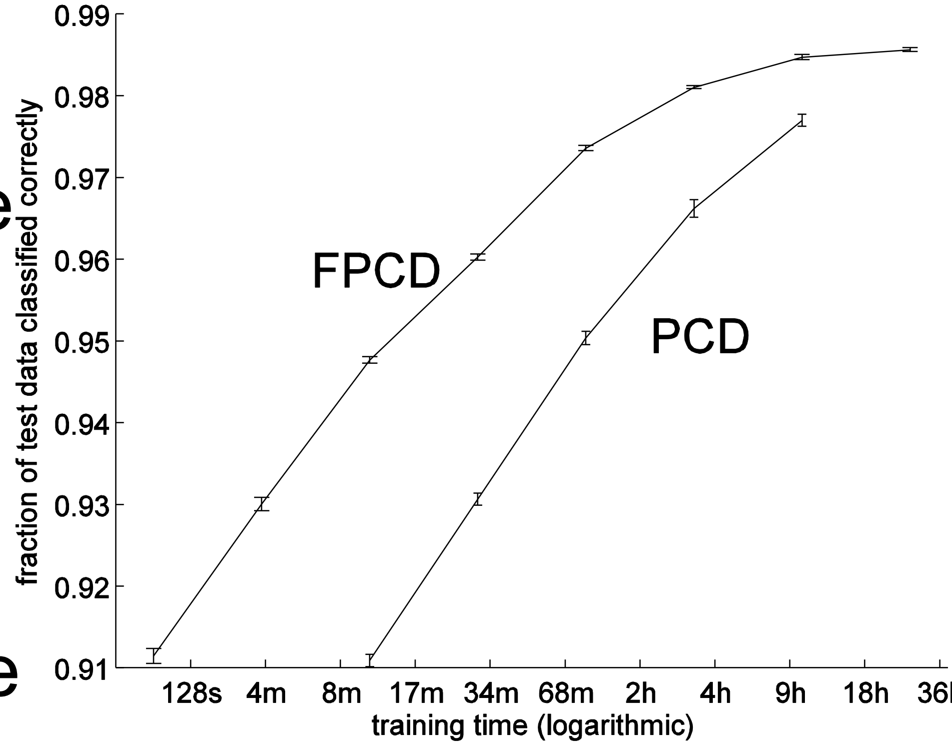
- Initialization:
  - *regular theta* = small random weights.
  - *fast theta* = all zeros.
  - Initialize *negData* arbitrarily.
- Repeat:
  - Get the *positive gradient* on a batch of training data.
  - Get the *negative gradient* on *negData.*
  - Do learning on both *regular theta* and *fast theta* using the difference of gradients (but with different LR's).
  - Update *negData* using a Gibbs update, with parameters *regular theta* + *fast theta*.
  - Update: f*ast theta* ← *fast theta* * 0.95

Additional notes

- When *fast theta* is all zeros, FPCD == PCD.
- There should be different learning rates for *regular theta* and *fast theta*.
  - *Regular theta* must learn slowly, to prevent getting bad models.
  - *Fast theta* must learn rapidly, to enable fast mixing.

Results

- FPCD helps a lot…
- …when not many updates can be done (i.e. large data dimensionality).
- For small models (e.g. toy) with lots of training time → same as PCD.
- More results: poster.



Conclusion

- FPCD = Fast PCD = Persistent Contrastive Divergence using Fast weights
- Use fast learning-causing-mixing, but not on the regular model parameters.

The End

The mixing rate

- Markov Chain (i.e. PCD with learning rate 0)