Distributed Training Strategies for the Structured Perceptron

Ryan McDonald Keith Hall Gideon Mann

Google, Inc., New York / Zurich

Presenter: Mahsa Yarmohammadi

Perceptron Algorithm

- An algorithm for supervised classification
- Classify one input into one of several outputs
- Example in POS tagging:

  input:

  "*fruit flies fast*"

  possible outputs:

  "*NN NNS VB*"

  "*NN VB RB*"

  "*NN NNS JJ*"

Perceptron Algorithm

- Supervised: input-output training instances

$$(\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}$$

- For all training instances:
  - Predict the best output ($\mathbf{y}'$)
  - Penalize features of $\mathbf{y}'$
  - Reward features of the truth ($\mathbf{y}_t$)

Perceptron Algorithm

- Predict the best output ($\mathbf{y}'$)

$$\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$$

Perceptron Algorithm

- Predict the best output ($\mathbf{y}'$)

$$\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$$

weight vector

feature vector

Perceptron Algorithm

- Predict the best output ($\mathbf{y}'$)

$$\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$$

- Reward/Penalize features

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$$

Perceptron Algorithm

Perceptron($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)

$\mathbf{w}^{(0)} = \mathbf{0};\ k = 0$

for $n : 1..N$

for $t : 1..T$

Let $\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$

if $\mathbf{y}' \neq \mathbf{y}_t$

$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$

$k = k + 1$

return $\mathbf{w}^{(k)}$

Perceptron Algorithm

- Can be costly to train
  - increasing size of training data
  - complexity is proportional to inference
  - non-linear increasing complexity with sequence length
- Distributed training strategies to reduce training time

Distributed Perceptron

- Two strategies:
  - Parameter Mixing
  - Iterative Parameter Mixing

- Evaluate on two problems:
  - Named Entity Recognition (NER)
  - Dependency Parsing

Parameter Mixing

1) Divide training data $\mathcal{T}$ into S shards

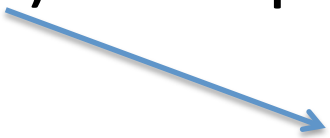
2) Train perceptron on each shard in parallel

3) Final $\mathbf{w}$ is a weighted mixture of perceptrons

Parameter Mixing

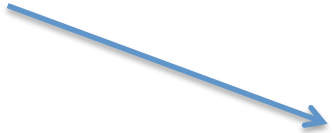
1) Divide training data $\mathcal{T}$ into S shards

2) Train perceptron on each shard in parallel



mapper

3) Final $\mathbf{w}$ is a weighted mixture of perceptrons



reducer

Parameter Mixing

PerceptronParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)

Shard $\mathcal{T}$ into S pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$

$\mathbf{w}^{(i)} = \text{Perceptron}(\mathcal{T}_i)$

$\mathbf{w} = \sum_i \mu_i \mathbf{w}^{(i)}$

return $\mathbf{w}$

Parameter Mixing

- Advantages:
  - straight-forward
  - scalable to extremely large data sets
  - resource (network usage) efficient
- Disadvantage:
  - sub-optimal: does not necessarily return a *separating* weight vector

Separable Training Set

- $\mathcal{T}$ is separable with margin γ if there is $\mathbf{u}$ ($\|\mathbf{u}\| = 1$)

$$\mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}') \geq \gamma$$

$$\text{for all} (\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}, \text{ and for all } \mathbf{y}' \in \mathcal{Y}_t \text{ such that } \mathbf{y}' \neq \mathbf{y}_t$$

Separable Training Set

- $\mathcal{T}$ is separable with margin γ if there is $\mathbf{u}$ ($\|\mathbf{u}\| = 1$)

$$\mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{u} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}') \geq \gamma$$

$$\text{for all} (\mathbf{x}_t, \mathbf{y}_t) \in \mathcal{T}, \text{ and for all } \mathbf{y}' \in \mathcal{Y}_t \text{ such that } \mathbf{y}' \neq \mathbf{y}_t$$

- If $\mathcal{T}$ is separable, then the perceptron

1) will converge in a finite time
2) will produce a separating $\mathbf{w}$

Iterative Parameter Mixing

1) Divide training data $\mathcal{T}$ into S shards

2) Train single-epoch perceptron on each shard

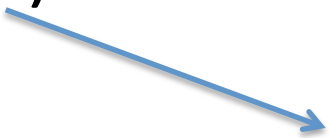
3) Mix the weights

4) Re-send the new weight $\mathbf{w}$ to each shard,
   go to 2

Iterative Parameter Mixing

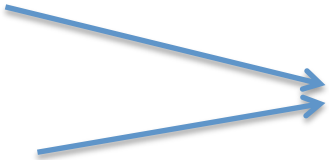
1) Divide training data $\mathcal{T}$ into S shards

2) Train single-epoch perceptron on each shard



mapper

3) Mix the weights



reducer

4) Re-send the new weight $\mathbf{w}$ to each shard,

go to 2

Iterative Parameter Mixing

PerceptronIterParamMix($\mathcal{T} = \{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{|\mathcal{T}|}$)

Shard $\mathcal{T}$ into S pieces $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_S\}$

$\mathbf{w} = \mathbf{0}$

for $n : 1..N$

$\mathbf{w}^{(i,n)} =$ OneEpochPerceptron($\mathcal{T}_i, \mathbf{w}$)

$\mathbf{w} = \sum_i \mu_{i,n} \mathbf{w}^{(i,n)}$

return $\mathbf{w}$

OneEpochPerceptron($\mathcal{T}$, $\mathbf{w}^*$)

$\mathbf{w}^{(0)} = \mathbf{w}^*;\ k = 0$

for $t : 1..T$

Let $\mathbf{y}' = \arg\max_{\mathbf{y}'} \mathbf{w}^{(k)} \cdot \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$

if $\mathbf{y}' \neq \mathbf{y}_t$

$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \mathbf{f}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(\mathbf{x}_t, \mathbf{y}')$

$k = k + 1$

return $\mathbf{w}^{(k)}$

Iterative Parameter Mixing

- Advantage:
  - optimal: returns a *separating* weight vector
  - significantly reduces training time
- Disadvantage:
  - increasing network usage

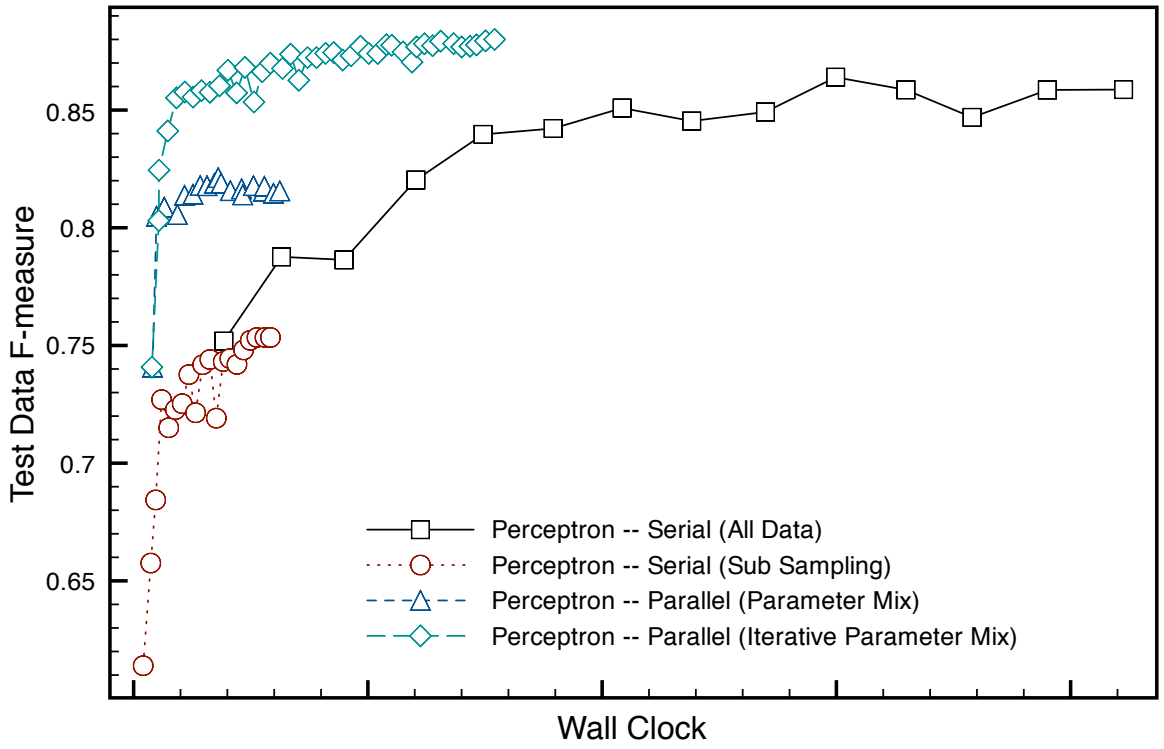
Experiments

- Standard/Averaged perceptron in 4 systems:
  - Serial – All Data
  - Serial – Sub Sampling
  - Parallel (Parameter Mixing)
  - Parallel (Iterative Parameter Mixing)

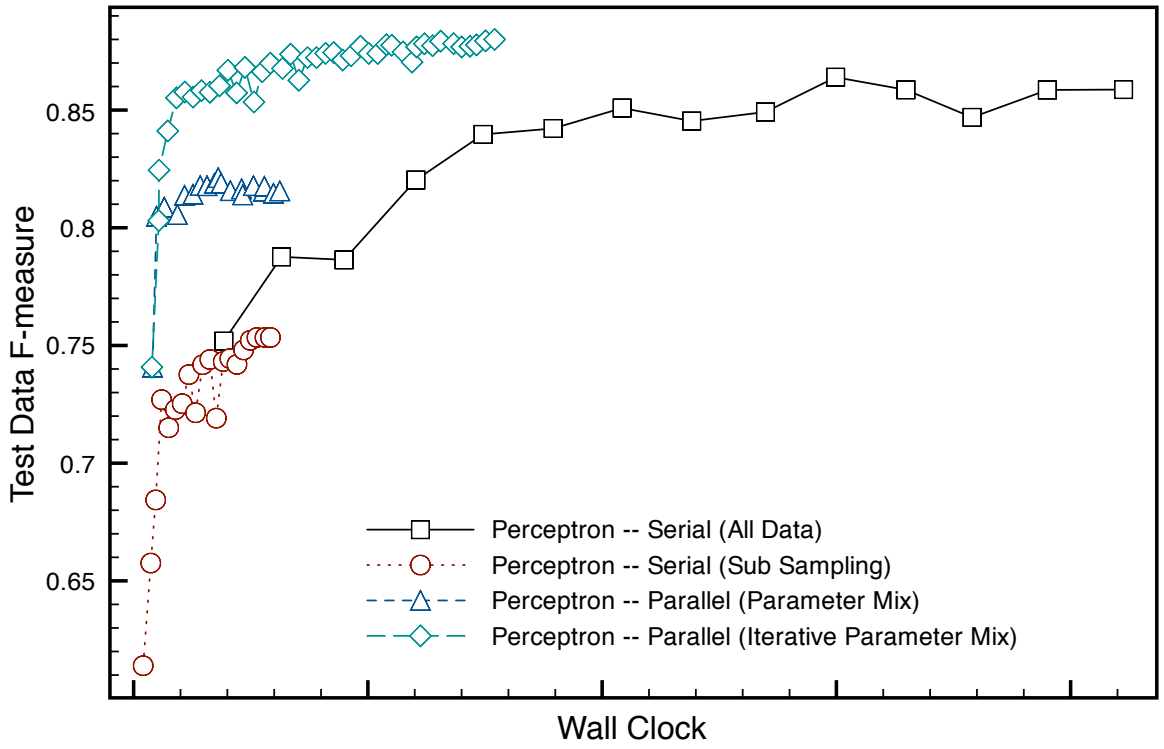
Experiments

- Standard/Averaged perceptron in 4 systems:
  - Serial – All Data
  - Serial – Sub Sampling
  - Parallel (PM)
  - Parallel (IPM)
- Two Problems:
  - NER – CoNLL 2033 shared task
  - Dependency parsing – Prague Dependency Treebank

NER – Standard perceptron

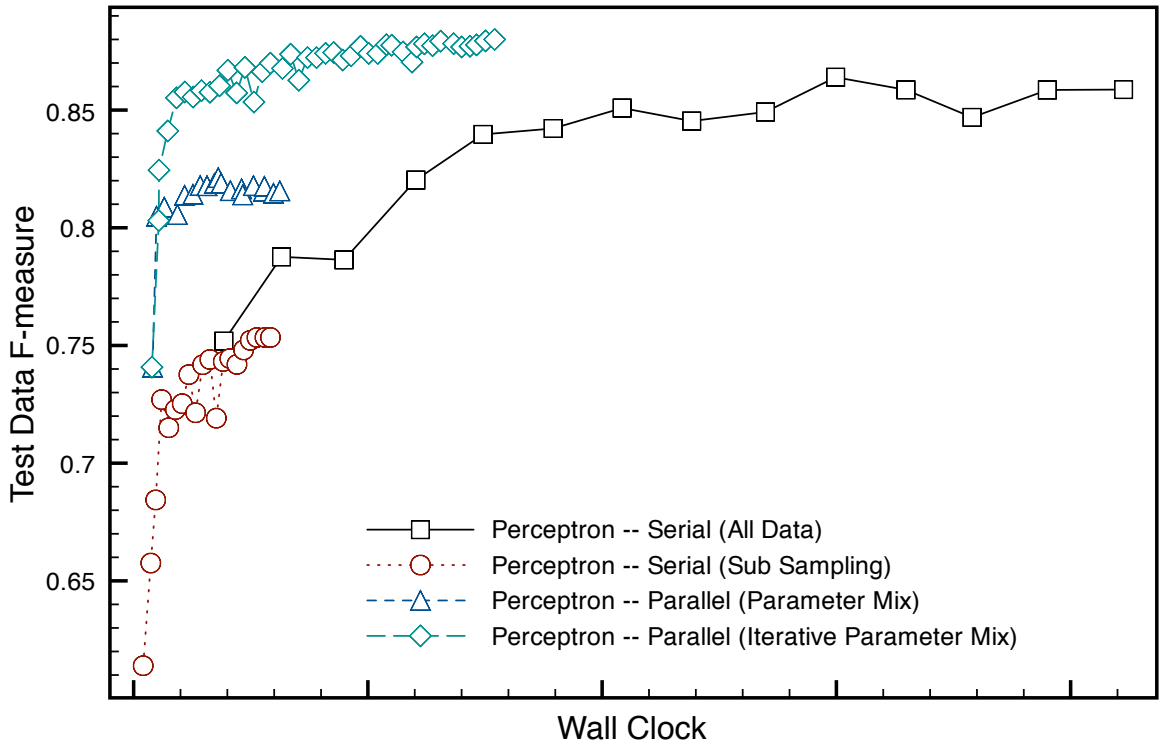


NER – Standard perceptron



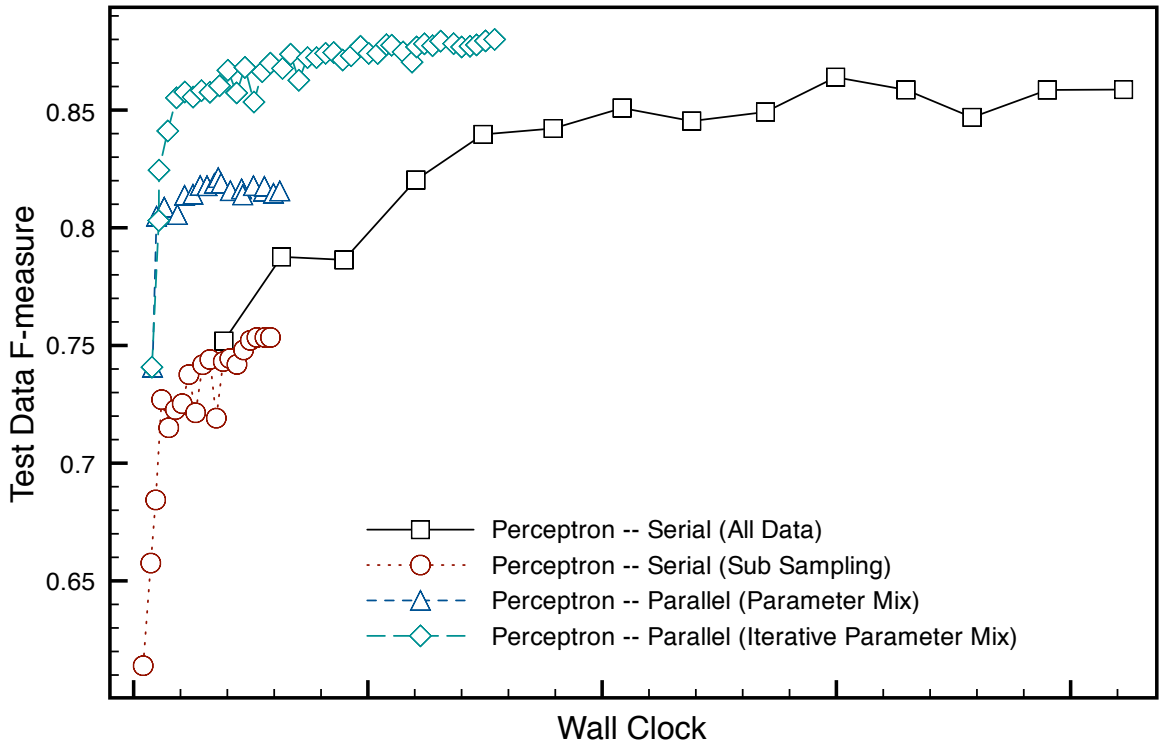
- Training on a single shard performs worse than all data

NER – Standard perceptron



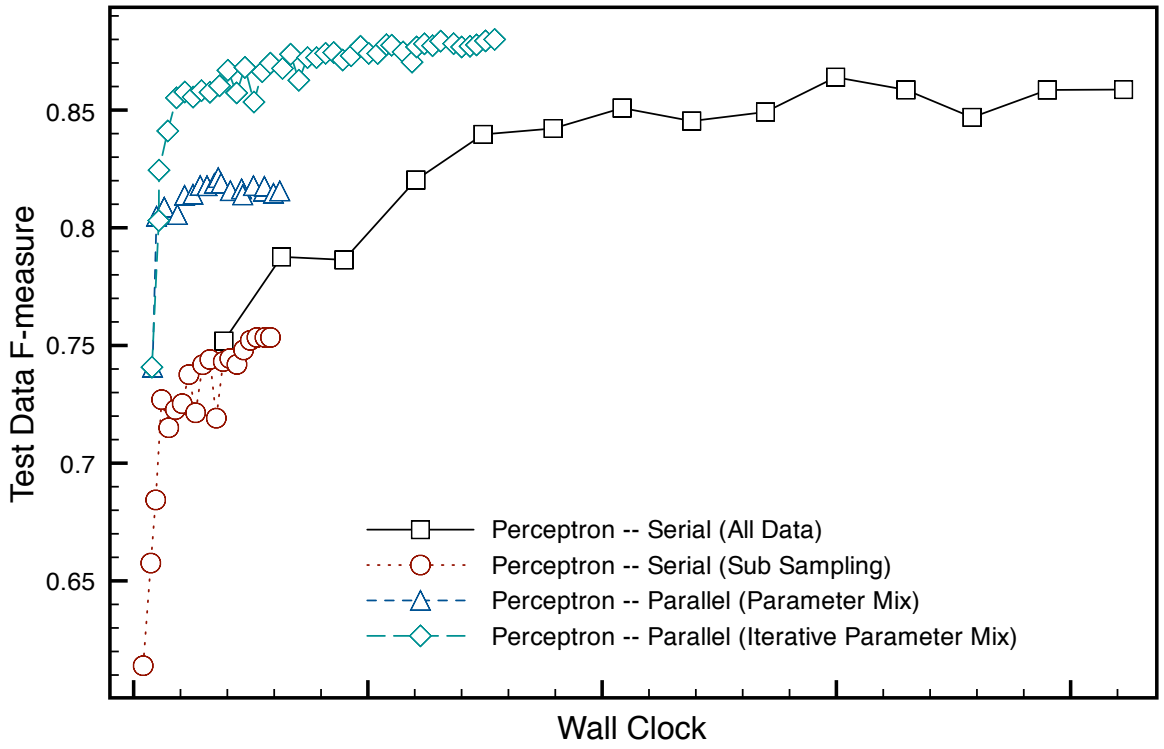
- PM performs better than single-shard but worse than all data

NER – Standard perceptron



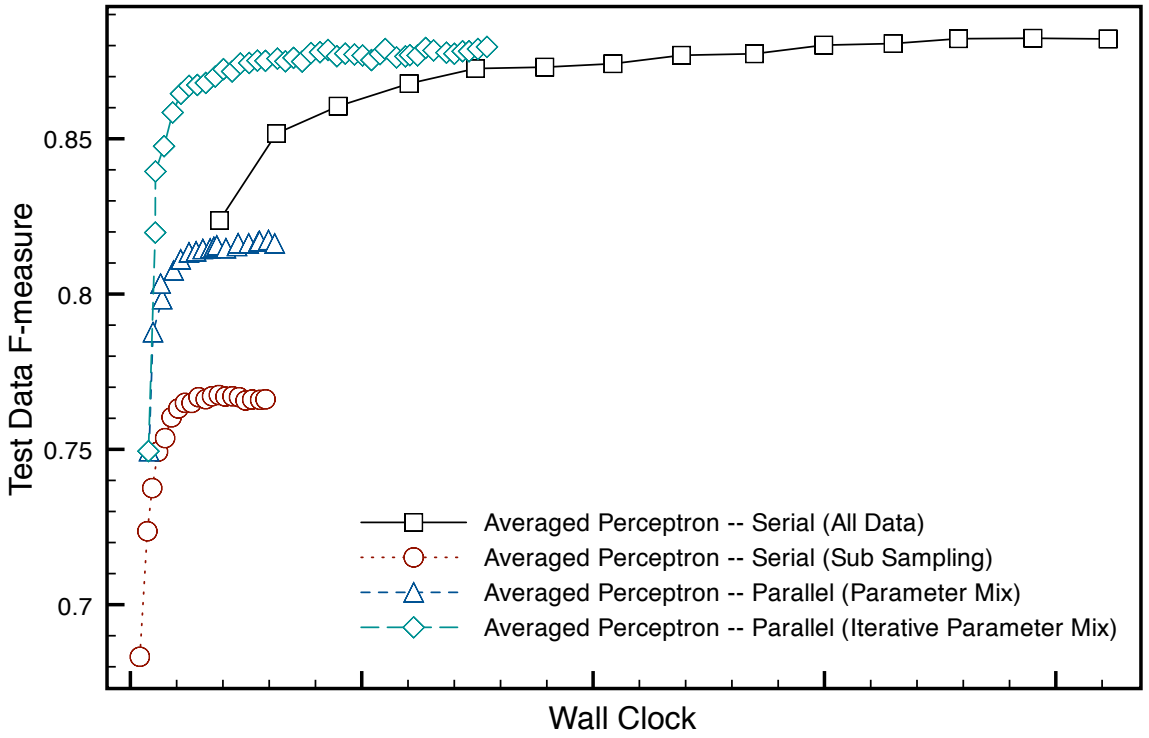
- IPM performs as good as/better than all data

NER – Standard perceptron



- PM and IPM return better classifiers much quicker than all data

NER – Averaged perceptron

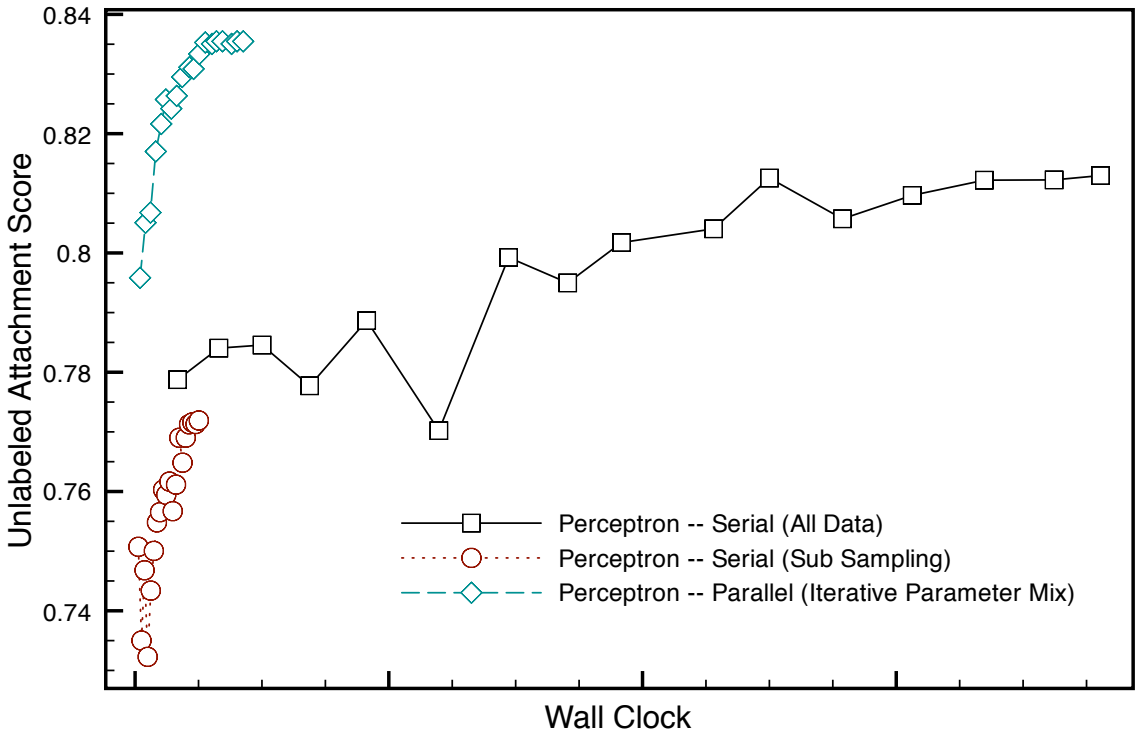


- The same patterns hold for averaged perceptron

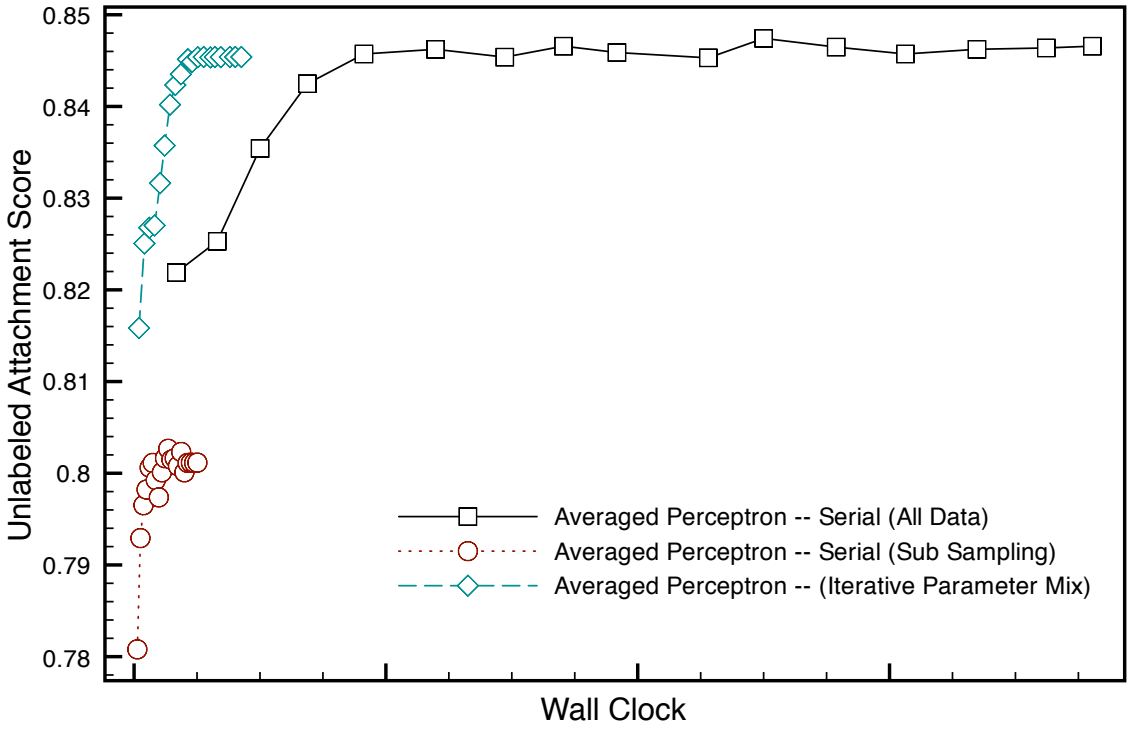
NER – Standard/Averaged perceptron

	Reg. Perceptron F-measure	Avg. Perceptron F-measure
Serial (All Data)	85.8	88.2
Serial (Sub Sampling)	75.3	76.6
Parallel (Parameter Mix)	81.5	81.6
Parallel (Iterative Parameter Mix)	87.9	88.1

Dep. Parsing – Standard perceptron



Dep. Parsing – Averaged perceptron



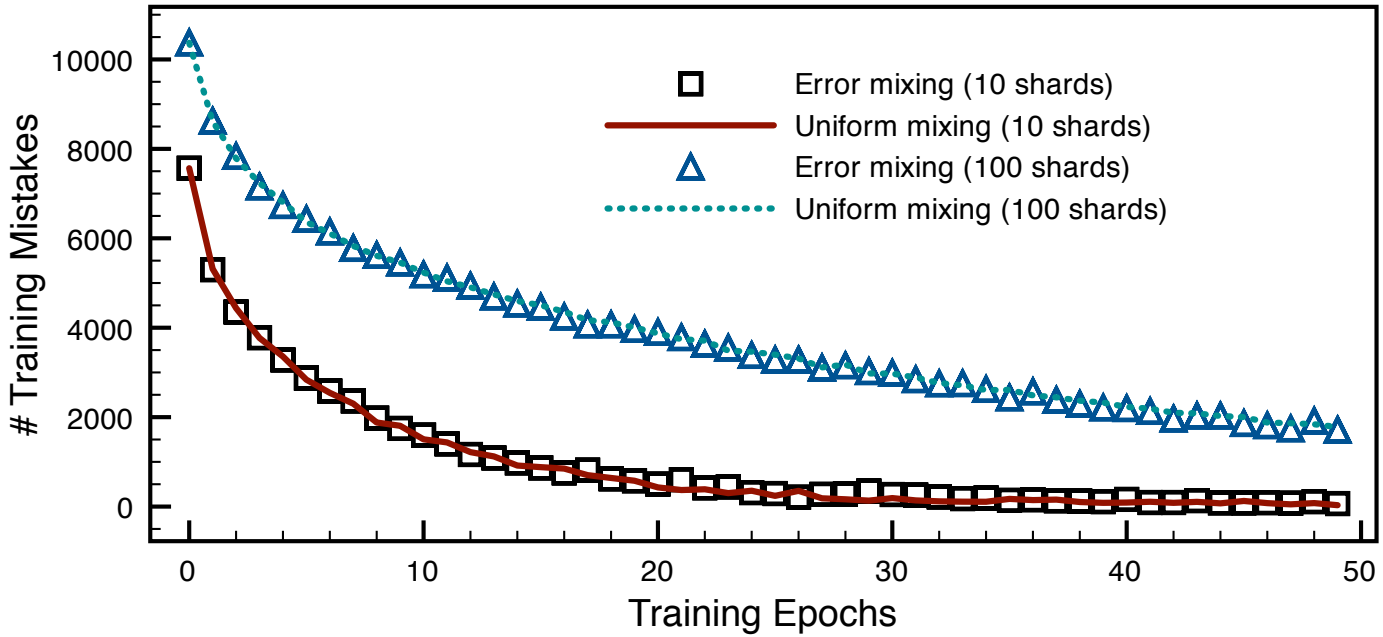
Dep. Parsing – Standard/Averaged perceptron

	Reg. Perceptron Unlabeled Attachment Score	Avg. Perceptron Unlabeled Attachment Score
Serial (All Data)	81.3	84.7
Serial (Sub Sampling)	77.2	80.1
Parallel (Iterative Parameter Mix)	83.5	84.5

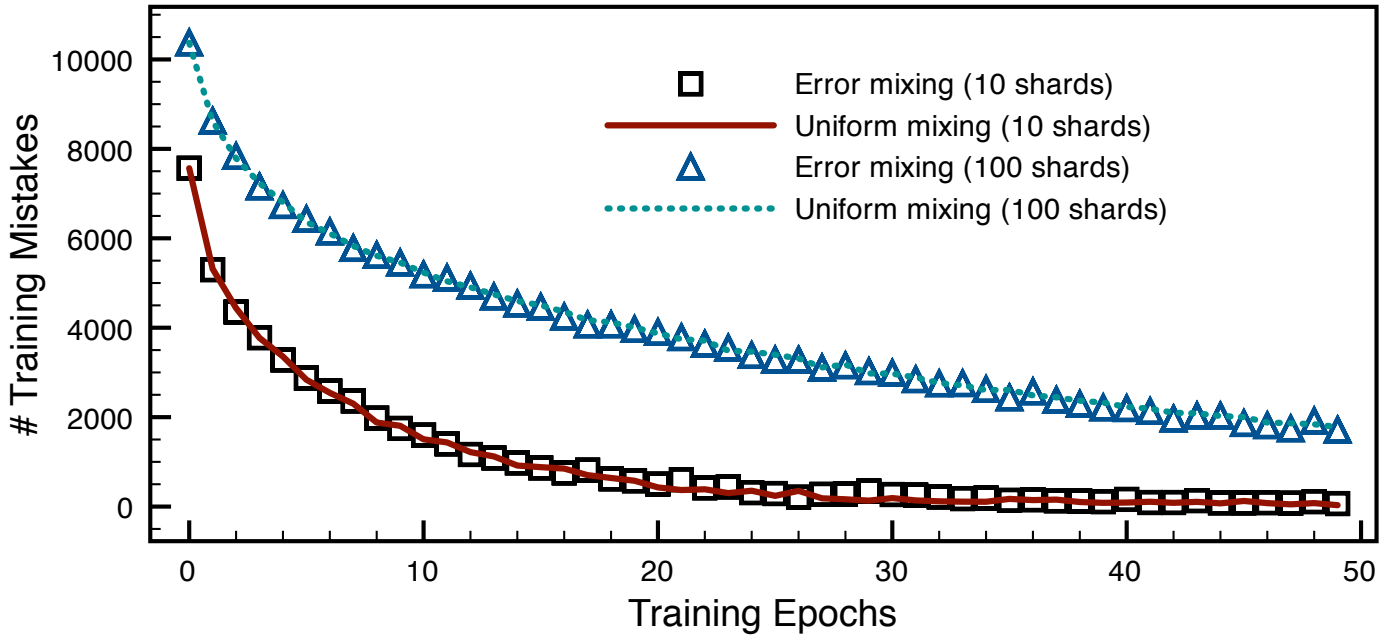
IPM Convergence

- # shards
  - 10
  - 100
- Mixing strategy
  - uniform ($\mu_{i,n}=1/S$)
  - error based ($\mu_{i,n}=k_{i,n}/k_n$)
- # epochs (*N*)
  - 0 .. 50

IPM Convergence

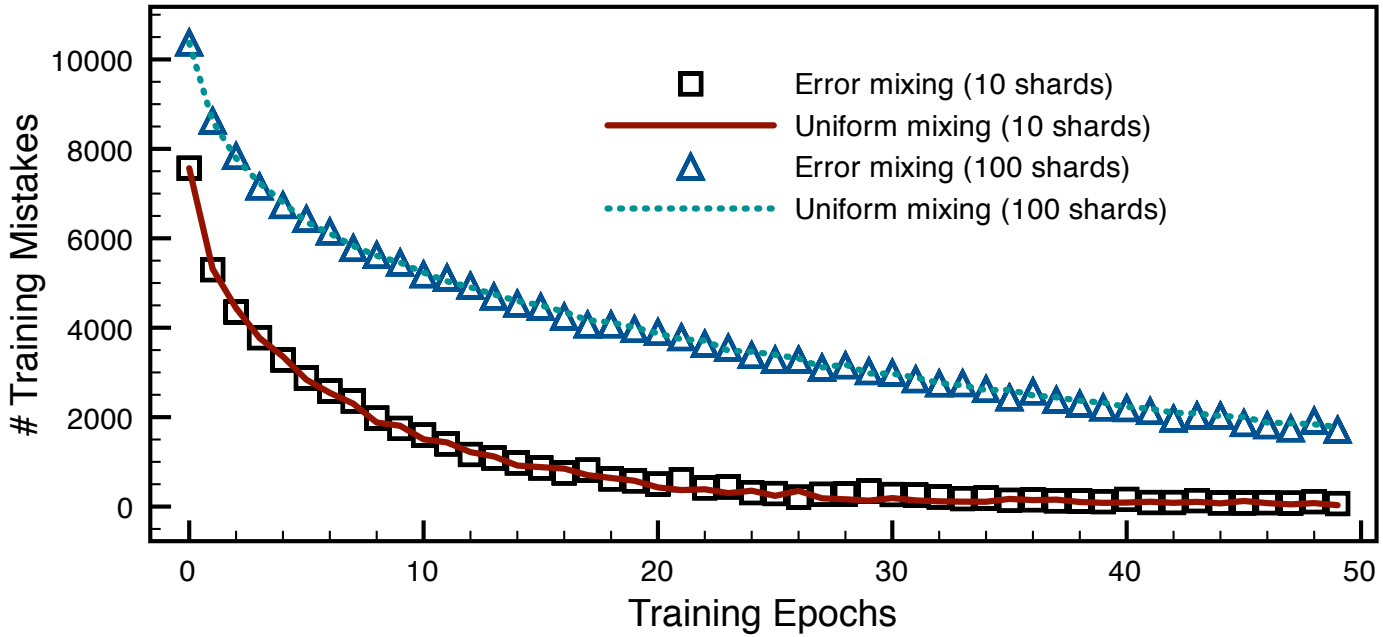


IPM Convergence



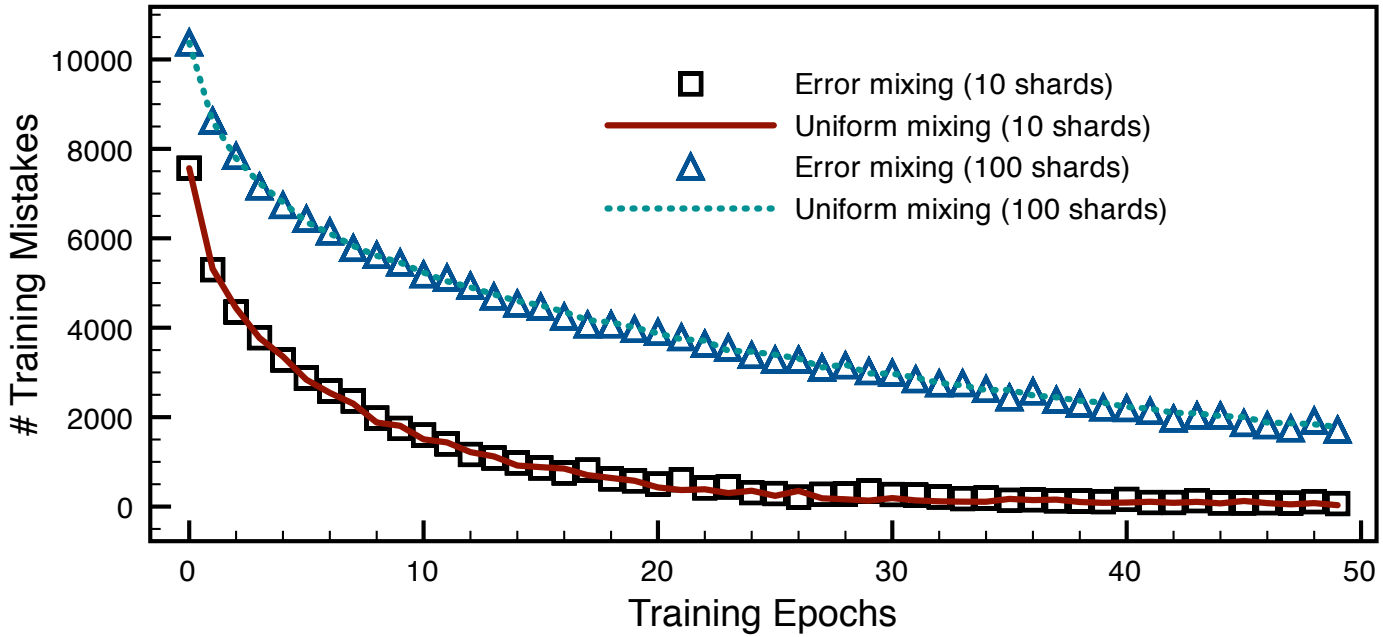
- Mixing strategy makes little difference

IPM Convergence



- Increasing # shards slows down convergence when # epochs increases

IPM Convergence



- Trade-off between slower convergence and quicker ephocs

Conclusions

- Iterative parameter mixing is
  - guaranteed to separate the data
  - significantly reduces training time
- Trade-off between distributing computation and slower convergence relative to # shards
- Theoretical proofs of convergence in the paper