- Assignment 2 out!
- Project groups extension → until Oct 7th
- OH / Setup meeting by email Oct 17th — Introduction
Propose topic, group (2 sentences) – Oct 7
Project Proposal (2 pages) – Oct 17

Introduction
Related Work
Timeline (with eval plan)
WRITING AN INTRODUCTION

1-2 paras: what is the problem you are solving
why is it important (need citations)

1-2 paras: How other people solve and why they fall short

1-2 paras: How do you plan on solving it and why your approach is better

1 para: Anticipated results or what experiments you will use
WRITING RELATED WORK

Group related work into two/three buckets (1-2 para per bucket)

Explain what the papers / projects do
Why are they different / insufficient
Scalable Storage Systems

Datacenter Architecture

Computational Engines

Machine Learning

SQL

Resource Management

Streaming

Graph

Applications

Spark

MapReduce

Mesos

DRF

Datacenter Architecture
MACHINE LEARNING

Classification

Recommendation

Gmail → Spam filter

IMAGENET

NETFLIX

amazon
Optimization

Function

Model

Data (Examples)

Regularization

\[
\min_{w \in \mathbb{R}^d} \sum_{i=1}^{N} f(w, z_i) + P(w)
\]
What is convex?
Linear Regression, Linear SVM
Kernel SVMs, Logistic Regression,

What is not convex?
Graph mining, Deep Learning
GRADIENT DESCENT

\[ w^{(k+1)} = w^{(k)} - \alpha_k \nabla f(w^{(k)}) \]

Initialize \( w \)

For many iterations:

- Compute Gradient
- Update model

End

Random initialization

Step size

How many data points?

GD: All of your training data (n)

IGD: One training data point (1)

SGD: Stochastic Gradient (\( B = 128 \) or 512)
INCREMENTAL GRADIENT DESCENT

\[ w^{(k+1)} = w^{(k)} - \alpha_k \nabla f_{\eta(k)}(w^{(k)}) \]

Initialize \( w \)

For many iterations:

- Pick one point
- Compute Gradient
- Update model

End
<table>
<thead>
<tr>
<th>Analytics Task</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistic Regression (LR)</td>
<td>$\sum_i \log(1 + \exp(-y_i w^T x_i)) + \mu |\vec{w}|_1$</td>
</tr>
<tr>
<td>Classification (SVM)</td>
<td>$\sum_i (1 - y_i w^T x_i)_+ + \mu |\vec{w}|_1$</td>
</tr>
<tr>
<td>Recommendation (LMF)</td>
<td>$\sum_{(i,j) \in \Omega} (L_i^T R_j - M_{ij})^2 + \mu |L, R|_F^2$</td>
</tr>
<tr>
<td>Labeling (CRF) [48]</td>
<td>$\sum_k \left[ \sum_j w_j F_j(y_k, x_k) - \log Z(x_k) \right]$</td>
</tr>
<tr>
<td>Kalman Filters</td>
<td>$\sum_{t=1}^T |C w_t - f(y_t)|<em>2^2 + |w_t - A w</em>{t-1}|_2^2$</td>
</tr>
<tr>
<td>Portfolio Optimization</td>
<td>$p^T w + w^T \Sigma w$ s.t. $w \in \Delta$</td>
</tr>
</tbody>
</table>
Bismarck Architecture

Specification

Bismarck

IGD Aggregate

Loss Aggregate

Model

Tables

Training data

Storage layer

Selecting

Gradient

Updating

Compute layer
BISMARCK: USER DEFINED AGGREGATE

Three steps:

1. initialize(state)
   - Initialize the model
     - Zero weights / random weights

2. transition(state, data)
   - Input: \( <\text{data}, \text{label}> \)
   - Update model based on data

3. terminate(state)
   - Loss value \(< \|\| \)
   - Fixed num iterations / epochs
LR_Transition(ModelCoef *w, Example e) {
    wx = Dot_Product(w, e.x);
    sig = Sigmoid(-wx * e.y);
    c = stepsize * e.y * sig;
    Scale_And_Add(w, e.x, c); ... }

DATA ORDERING

Random sampling
- Sample \textit{without replacement}
- Shuffle the data after each \textit{epoch}

Shuffle once
- Avoids pathological ordering
- Much cheaper
RESERVOIR SAMPLING

Select first $m$ items

On the $k^{th}$ additional item

$s = \text{random in } [0, m + k)$

if $s < m$

   Put in slot $s$

else

   Drop the item
PARALLEL GRADIENTS

Shared Memory:
- Compute gradients in parallel
- Average their updates
- Or update in parallel
  - Locks?
    - A I C : Atomic Increment Instructions
      - Fast, implemented in hardware
    - Lock-free: Iteration times very fast, conflicts?
DISCUSSION

https://forms.gle/nFNEi2NZMNhZio1f7
How would an implementation of GD look in Spark? Try to sketch an implementation. What would be similar / different to Bismarck?

```
data: sc.textFile("")
data: shuffle()
grad: data.map(x => gradComp(x))
  .reduce(+-+)
model: model - step * grad
loop: broadcast
→ check for termination
```
What are some ML scenarios where Bismarck architecture might prove to be limited?

Non convex: Converge? Shuffle once back free

Skewed dataset: Conflicts → lock free?

Overfitting?

Model fits in memory?

Is limited?
The graphs show the log-likelihood over epochs and time, comparing different shuffling strategies:

- **Clustered**: 185 epochs, log-likelihood starts high and decreases gradually.
- **ShuffleOnce**: 47 epochs, shows a significant drop compared to Clustered.
- **ShuffleAlways**: 35 epochs, maintains a relatively high log-likelihood throughout.

The time graphs indicate:
- **Clustered**: 9.3 seconds, showing a consistent decrease.
- **ShuffleOnce**: 2.4 seconds, with a sharper decrease.

Handwritten notes indicate:
- "least epoch"
Next class: Parameter Server
Assignment 2 out!
Project Proposal
  Groups by Oct 7
  2 pager by Oct 17