Ameliorating the Annotation Bottleneck

Data Programming + Asynchronous Deep Learning

Chris Ré  Alex Ratner

Hazy Research @ Stanford
Outline

• Data Programming: Creating Large Training Sets, Quickly

Brief look:

• Omnivore: An Optimizer for Multi-Device Deep Learning on CPUs and GPUs
Data Programming

Creating Large Training Sets, Quickly
My Amazing Collaborators on Data Programming

Chris De Sa    Sen Wu    Henry Ehrenberg    Jason Fries    Jaeho Shin    Chris Ré
Data Programming: Key Idea

• PROBLEM: It’s difficult to get enough training data

• SOLUTION: We can let users programmatically generate large, noisy training sets
  • Users describe the process of labeling in standard scripting languages
  • Modern ML techniques can handle the noise!
Outline

1. Motivating Application: Knowledge Base Construction

2. The Data Programming Paradigm

3. As a Software Engineering Framework: DDLite
Knowledge Base Construction

Motivating Application
Knowledge Base Construction (KBC)

• Task: Populate fixed schema with *structured data* from *unstructured sources*

• KBC a critical component of analysis pipelines in domains such as:
  
  • Law Enforcement
  
  • Pharmacogenomics
  
  • Geology
Why is the KBC task important?

- Having a structured representation is critical for a range of downstream analysis tasks:
  - Macroscopic analyses
    - Ex: Paleo-biological climate analysis (Nelson et. al., 2016)
  - Link prediction
    - Ex: Protein-Protein interactions (ANAP, 2016)
  - Predictive Modeling
    - Ex: Lung cancer histology prediction (Yu et. al., 2016)
  - Entity or Relation-centric search
    - Ex: MEMEX anti-human trafficking work
Example: Chemical-Disease Relation Extraction from Text

 TITLE: 
 *Myasthenia gravis* presenting as weakness after *magnesium* administration.

 ABSTRACT: 
 We studied a patient with no prior history of neuromuscular disease who became virtually *quadriplegic* after parenteral *magnesium* administration for *preeclampsia*. The serum *magnesium* concentration was 3.0 mEq/L, which is usually well tolerated. The *magnesium* was stopped and she recovered over a few days. While she was weak, 2-Hz repetitive stimulation revealed a decrement without significant facilitation at rapid rates or after exercise, suggesting *postsynaptic neuromuscular blockade*. After her strength returned, repetitive stimulation was normal, but single fiber EMG revealed increased jitter and blocking. Her *acetylcholine* receptor antibody level was markedly elevated. Although *paralysis* after *magnesium* administration has been described in patients with known *myasthenia gravis*, it has not previously been reported to be the initial or only manifestation of the disease. Patients who are unusually sensitive to the neuromuscular effects of *magnesium* should be suspected of having an underlying disorder of neuromuscular transmission.

- **We define candidate entity mentions:**
  - **Chemicals**
  - **Diseases**

- **Goal:** Populate a relational schema with *relation mentions*

<table>
<thead>
<tr>
<th>ID</th>
<th>Chemical</th>
<th>Disease</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>magnesium</td>
<td>Myasthenia gravis</td>
<td>0.84</td>
</tr>
<tr>
<td>01</td>
<td>magnesium</td>
<td>quadriplegic</td>
<td>0.73</td>
</tr>
<tr>
<td>02</td>
<td>magnesium</td>
<td>paralysis</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Machine Learning Approach to KBC

TITeL:
Myasthenia gravis presenting as weakness after magnesium administration.

ABSTRACT:
We studied a patient with no prior history of neuromuscular disease who became virtually quadriplegic after parenteral magnesium administration for preeclampsia. The serum magnesium concentration was 3.0 mEq/L, which is usually well tolerated. The magnesium was stopped and she recovered over a few days. While she was weak, 2-Hz repetitive stimulation revealed a decrement without significant facilitation at rapid rates or after exercise, suggesting postsynaptic neuromuscular blockade. After her strength returned, repetitive stimulation was normal, but single fiber EMG revealed increased jitter and blocking. Her acetylcholine receptor antibody level was markedly elevated. Although paralysis after magnesium administration has been described in patients with known myasthenia gravis, it has not previously been reported to be the initial or only manifestation of the disease. Patients who are unusually sensitive to the neuromuscular effects of magnesium should be suspected of having an underlying disorder of neuromuscular transmission.
Machine Learning Approach to KBC

TITLE:
Myasthenia gravis presenting as weakness after magnesium administration.

ABSTRACT:
We studied a patient with no prior history of neuromuscular disease who became virtually quadriplegic after parenteral magnesium administration for preeclampsia. The serum magnesium concentration was 3.0 mEq/L, which is usually well tolerated. The magnesium was stopped and she recovered over a few days. While she was weak, 2-Hz repetitive stimulation revealed a decrement without significant facilitation at rapid rates or after exercise, suggesting postsynaptic neuromuscular blockade. After her strength returned, repetitive stimulation was normal, but single fiber EMG revealed increased jitter and blocking. Her acetylcholine receptor antibody level was markedly elevated. Although paralysis after magnesium administration has been described in patients with known myasthenia gravis, it has not previously been reported to be the initial or only manifestation of the disease. Patients who are unusually sensitive to the neuromuscular effects of magnesium should be suspected of having an underlying disorder of neuromuscular transmission.

Candidate Extraction → Training Set
Machine Learning Approach to KBC

Title: Myasthenia gravis presenting as weakness after magnesium administration.

Abstract: We studied a patient with no prior history of neuromuscular disease who became virtually quadriplegic after parenteral magnesium administration for preeclampsia. The serum magnesium concentration was 3.0 mEq/L, which is usually well tolerated. The magnesium was stopped and she recovered over a few days. While she was weak, 2-Hz repetitive stimulation revealed a decrement without significant facilitation at rapid rates or after exercise, suggesting postsynaptic neuromuscular blockade. After her strength returned, repetitive stimulation was normal, but single fiber EMG revealed increased jitter and blocking. Her acetylcholine receptor antibody level was markedly elevated. Although paralysis after magnesium administration has been described in patients with known myasthenia gravis, it has not previously been reported to be the initial or only manifestation of the disease. Patients who are unusually sensitive to the neuromuscular effects of magnesium should be suspected of having an underlying disorder of neuromuscular transmission.

Example binary features:
- PHRASE_BTWN[“presenting as”]
- WORD_BTWN[“after”]
Machine Learning Approach to KBC

TITLE:
Myasthenia gravis presenting as weakness after magnesium administration.

ABSTRACT:
We studied a patient with no prior history of neuromuscular disease who became virtually quadriplegic after parenteral magnesium administration for preeclampsia. The serum magnesium concentration was 3.0 mEq/L, which is usually well tolerated. The magnesium was stopped and she recovered over a few days. While she was weak, 2-Hz repetitive stimulation revealed a decrement without significant facilitation at rapid rates or after exercise, suggesting postsynaptic neuromuscular blockade. After her strength returned, repetitive stimulation was normal, but single fiber EMG revealed increased jitter and blocking. Her acetylcholine receptor antibody level was markedly elevated. Although paralysis after magnesium administration has been described in patients with known myasthenia gravis, it has not previously been reported to be the initial or only manifestation of the disease. Patients who are unusually sensitive to the neuromuscular effects of magnesium should be suspected of having an underlying disorder of neuromuscular transmission.
Traditional ML Application Development Bottlenecks

1. Creating Training Sets:
   • Supervised learning approaches rely on large sets of hand-labeled *training* data

2. Feature Engineering:
   • Developers must specify a *reasonably-sized* set of *features* which represent the data points
The End of Feature Engineering?

Automated feature generation techniques—such as deep learning—seem increasingly viable
The End of Feature Engineering?

Automated feature generation techniques—such as deep learning—seem increasingly viable.

However, these approaches rely on massive training sets.
Challenges of Training Set Creation & Management

• Domain expertise is necessary to annotate properly

• Application schemas evolve during use

Training set creation needs to be **expert-driven** and **dynamic**
The Data Programming Approach
Related work


- **Crowdsourcing** *(Dawid 1979, Karger 2011, Berend 2014, ...)*

- **Co-training** *(Blum 1998, ...)*

- **Boosting** *(Freund, Balsubramani 2015, ...)*

- **Learning with noisy labels** *(Bootkrajang 2012, Lugosi 1992, ...)*

We build on a rich body of prior work in this general area
Programmatic Training Set Creation & Management

• In the data programming approach, domain-expert users write **labeling functions** which each label some subset of the data

\[ \lambda_i(x) \rightarrow \{-1,0,1\} \]

• **Labeling functions** are black-box functions which describe a process of labeling a training set

Users can quickly create a **large** but **noisy** training set using **standard scripting languages**
Labeling Functions

• Traditional “distant supervision” rule relying on external KB

```
def lf1(x):
cid = (x.chemical_id, x.disease_id)
return 1 if cid in KB else 0
```

• Pattern-based heuristic

```
def lf2(x):
m = re.search(r'\.*causes\.*', x.between)
return 1 if m else 0
```

• Hybrid labeling fn:

```
def lf3(x):
m = re.search(r'\.*after\.*', x.between)
b = x.disease_idx < x.chemical_idx
cid = (x.chemical_id, x.disease_id)
return 1 if m and b and in KB else 0
```

• Etc...

```
from bio_functions import black_box

def lf4(x):
return black_box(x)
```
Labeling Function Dependencies

• Users can also define dependencies between the labeling functions.

```python
def lf1(x):
    m = re.search(r'.*cause.*', x.between)
    return 1 if m else 0

def lf2(x):
    m = re.search(r'.*not cause.*', x.between)
    return 1 if m else 0
```

These can be manually specified or derived from static analysis—we have the functional forms.
The Data Programming Pipeline

• Given:
  
  • Unlabeled data $x = \{x^{(1)}, \ldots, x^{(N)}\}$ having unknown true labels $y^{(i)}$

  • A set of labeling functions $\lambda = \{\lambda_1, \ldots, \lambda_M\}$

  • A set of labeling function dependencies $G$
The Data Programming Pipeline

• Given:

  • Unlabeled data $x = \{x^{(1)}, \ldots, x^{(N)}\}$ having unknown true labels $y^{(i)}$

  • A set of labeling functions $\lambda = \{\lambda_1, \ldots, \lambda_M\}$

  • A set of labeling function dependencies $G$

Will leave out here for simplicity of presentation!
The Data Programming Pipeline

• Do:
  1. Learn a generative model of the training set labeling process
The Data Programming Pipeline

• Do:
  1. Learn a generative model of the training set labeling process
     • → Producing a set of *noisy* training labels
The Data Programming Pipeline

• Do:
  1. Learn a generative model of the training set labeling process
     • → Producing a set of noisy training labels
  2. Use these to train a **noise-aware** discriminative model

![Diagram showing the pipeline with labeling functions, features, and the relationship between them.](image)
Step 1: Modeling a Noisy Training Set Generation Process

• Learn the model $\pi = P(y, \Lambda)$ using MLE
  
  • Model the LFs as biased coin-flips
  
  • Intuition: Majority vote--estimate labeling function accuracy based on overlaps / conflicts
  
  • Similar to crowdsourcing methods, except the scaling is different (small number of LFs, large number of labels each)
  
• Produce a set of noisy training labels

$$\mu_{\pi}(y, \lambda) = P_{(y,\Lambda)\sim\pi}(y | \Lambda = \lambda(x))$$
Step 2: Training a Noise-Aware Model

• In a traditional supervised learning setting, we would learn from ground-truth labels:

\[
\hat{w} = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} l(x^{(i)}, y^{(i)})
\]

• Here, we learn from the noisy labels:

\[
\hat{w} = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_{(y,A) \sim \pi}[l(x^{(i)}, y^{(i)} = y)]
\]

Only requires simple tweak to loss function!
Scaling with Unlabeled Data

**Thm** Independent Case

*If:

1. $\pi$ can be represented by our model family
2. Our noise-aware risk minimizer has bounded risk
3. $y \perp f(x) \mid \lambda(x)$
4. We have a sufficient number of LFs with enough coverage & accuracy

*Then:* $\tilde{O}(\epsilon^{-2})$ *unlabeled* training points allow the algorithm to achieve $O(\epsilon)$ generalization risk

*• Using SGD + Gibbs sampling (for ex, see: De Sa, ICML 2016)*

This is the same asymptotic scaling as in supervised methods!
Scaling with Unlabeled Data

**Thm** Independent Case

**If:**
1. \( \pi \) can be represented by our model family
2. Our noise-aware risk minimizer has bounded risk
3. \( y \perp f(x) | \lambda(x) \)
4. We have a sufficient number of LFs with enough coverage & accuracy

**Then:** \( \tilde{O}(\epsilon^{-2}) \) **unlabeled** training points allow the algorithm to achieve \( O(\epsilon) \) generalization risk
   - Using SGD + Gibbs sampling (for ex, see: De Sa, ICML 2016)

This is the same asymptotic scaling as in supervised methods!

I.e. our labeling functions provide sufficient information to distinguish the class (where \( f \) are the end model features)

And, similar result when dependencies provided...
Scaling with Unlabeled Data

**Thm** Independent Case

*If:*

1. $\pi$ can be represented by our model family
2. Our noise-aware risk minimizer has bounded risk
3. $y \perp f(x) \mid \lambda(x)$
4. We have a sufficient number of LFs with enough coverage & accuracy

*Then:* $\tilde{O}(\epsilon^{-2})$ **unlabeled** training points allow the algorithm to achieve $O(\epsilon)$ generalization risk

- Using SGD + Gibbs sampling (for ex, see: De Sa, ICML 2016)

This is the same asymptotic scaling as in supervised methods!

---


And, similar result when dependencies provided...
Experimental Validation against Programmatic Supervision Baselines

• We compared with prior programmatic approaches in three KBC applications:

<table>
<thead>
<tr>
<th>Application</th>
<th># of LFs</th>
<th>Coverage</th>
<th>Overlap</th>
<th>Conflict</th>
<th>F1 Score Improvement HT</th>
<th>LSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBP (News)</td>
<td>40</td>
<td>29.39</td>
<td>1.38</td>
<td>0.15</td>
<td>1.92</td>
<td>3.12</td>
</tr>
<tr>
<td>Genomics</td>
<td>146</td>
<td>53.61</td>
<td>26.71</td>
<td>2.05</td>
<td>1.59</td>
<td>0.47</td>
</tr>
<tr>
<td>Pharmacogenomics</td>
<td>7</td>
<td>7.70</td>
<td>0.35</td>
<td>0.32</td>
<td>3.60</td>
<td>4.94</td>
</tr>
</tbody>
</table>

• We show that data programming boosts performance on existing distantly-supervised applications
  • New state-of-the-art on 2014 TAC-KBP challenge

Data programming *resolves* overlapping supervision rules for the user- easier and more powerful!
Evidence that Data Programming Plays Well with Deep Learning

• Using with deep learning for automated representation learning, we see strong improvements
  • +3% F1 on TAC-KBP
  • +6% F1 on TAC-KBP over a supervised LSTM baseline

<table>
<thead>
<tr>
<th>Application</th>
<th># of LFs</th>
<th>Coverage</th>
<th>Overlap</th>
<th>Conflict</th>
<th>F1 Score Improvement HT</th>
<th>LSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBP (News)</td>
<td>40</td>
<td>29.39</td>
<td>1.38</td>
<td>0.15</td>
<td>1.92</td>
<td>3.12</td>
</tr>
<tr>
<td>Genomics</td>
<td>146</td>
<td>53.61</td>
<td>26.71</td>
<td>2.05</td>
<td>1.59</td>
<td>0.47</td>
</tr>
<tr>
<td>Pharmacogenomics</td>
<td>7</td>
<td>7.70</td>
<td>0.35</td>
<td>0.32</td>
<td>3.60</td>
<td>4.94</td>
</tr>
</tbody>
</table>

Noise aware deep learning + large noisy training sets seem to be a promising combination!
DDLite: Putting humans in a different part of the loop

Data Programming as a Software Engineering Framework:

See our code @ [https://github.com/HazyResearch/ddlite](https://github.com/HazyResearch/ddlite)!
Notebook-based Environment

- Focus on rapid, simple-to-use prototyping for non-ML-experts

- Python based: easy integration of domain-specific resources:
  - Modules, libraries
  - Ontologies, dictionaries, lexicons
  - Existing knowledgebases
  - Etc...

User interaction with the system is *observational*—uses standard tools, from which system extracts implicit model
Focused around Labeling Function Development Process

• Feature engineering is difficult for users
  • What makes a “good” feature? Is a tricky statistical function of the training set & model

• Labeling functions have clear optimality criterion: that they label data correctly

Core hypothesis: Creating labeling functions easier than feature engineering
Labeling Function Development Workflow

(a) Labeled ground truth for development

```
In [44]: def LF_cancer(c):
   ...:     "Mention contains 'cancer' in lemmas"
   ...:     return 1 if re.search("\w+cancer", c.mention(attrib='lemmas'))
   ...:     else 0
   ...
```

(b) Labeling functions written in Python

```
def LF.noun_phrases(c):
    "Noun phrase of at least 2 tokens"
    pos_tags = c.mention(attrib='pos')
    ptn = r"\[NN\]+\[JJ\]+\[FW\]\]
    w = np.all([re.search(ptn, t) is not None
                for t in pos_tags])
    return 1 if w and len(pos_tags) > 1
   ...:     else 0
   ...
```

(c) Metrics for labeling functions

![Metrics chart]

---

Extract Candidates → Candidate Mentions → Extract Features

Label Development Set → Generate Training Set → Train & Predict

Evaluate Results → Candidates Mentions with Probabilities

Repeat LF Development if not Satisfactory, else Success! Move to Next Task
Labeling functions written in notebook environment as simple scripts

```python
def LF_neg_suffix(m):
    terms = ['deficiency', 'the', 'the', 'of', 'to', 'a']
    rw = right_window(m, window=1)
    if len(rw) > 0 and rw[0].lower() in terms:
        return FALSE
    return ABSTAIN

def LF_non_common_disease(m):
    'Non common disease'
    return FALSE if m.mention().lower() in non_common_disease else ABSTAIN

def LF_non_disease_acronyms(m):
    'Non common disease acronyms'
    return FALSE if m.mention() in non_disease_acronyms else ABSTAIN

def LF_intensity_modifier(m):
    'Do not include intensity modifiers (e.g., "severe <DISEASE NAME>")'
    return FALSE if intensity_terms.intersection([m.lemmas[i] for i in m.idxs]) else ABSTAIN

def LF_pos_in(m):
    'Candidates beginning with a preposition or subordinating conjunction'
    poses = [m.lemmas[i] for i in m.idxs]
    return FALSE if 'IN' in poses[0] else ABSTAIN

def LF_genetics_terminology(m):
    terms = set(['polymerase chain reaction', 'PCR', 'pcr'])
    phrase = m.lemmas[i] for i in m.idxs)
    return FALSE if phrase in terms else ABSTAIN

def LF_gene_chromosome_link(m):
    'Mentions of the form "Huntington Disease gene"
    w = 10
    genetics_terms = set(['gene', 'chromosome'])
    diseases_terms = set(['disease', 'syndrome', 'disorder'])
    context = left_window(m, window=w) + right_window(m, window=w)
```
Labeling Function Metrics and Visualizations

• For corpus-level views, e.g.
  • How much coverage?
  • How much conflict?

• For label function ranking and prioritization:
Using DDLite & Data Programming in Practice

• We have been using in “hackathons” with collaborators extracting various relations:
  • Anatomical location - sensation
  • Drug - drug
  • Protein – protein
  • Tumor marker – disease

• On a disease extraction benchmark, we see promise using no gold data:

<table>
<thead>
<tr>
<th>System</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANNER (Dogan, 2012)</td>
<td>83.80</td>
<td>80.00</td>
<td>81.80</td>
</tr>
<tr>
<td>DDLite: Single Hackathon</td>
<td>76.30</td>
<td>69.20</td>
<td>72.00</td>
</tr>
<tr>
<td>DDLite: Expert tuned (~3 days)</td>
<td>81.30</td>
<td>78.10</td>
<td>80.60</td>
</tr>
</tbody>
</table>
Next steps

• Can we get near- or better-than-benchmark scores within hours, by non-expert users?

• Can we provide better metrics and/or more intuitive visualizations?

• Can we automatically suggest new labeling functions?

• Can we increase our understanding of the connection between the core data programming formulation and the actual user experience?
Brief look:

Omnivore

An Optimizer for Multi-Device Deep Learning on CPUs and GPUs
My Amazing Lab-mates

Stefan Hadjis
Ce Zhang
Ioannis Mitliagkas
Dan Iter
Chris Ré
Outline

• Single-machine optimizations

• Multi-device tradeoffs: hardware vs. statistical efficiency

• Optimizer based on *asynchrony as an implicit momentum term*
We focus on Convolutional Neural Networks

Most popular type of network; work applicable to others as well
Single Machine: Batch Lowering + Dense Matrix Multiply

FLOP-proportionate throughput on GPUs and CPUs!
Single Machine: Batch Lowering + Dense Matrix Multiply

FLOP-proportionate throughput on GPUs and CPUs!
Single Machine: Batch Lowering + Dense Matrix Multiply

FLOP-proportionate throughput on GPUs and CPUs!
Single Machine: Batch Lowering + Dense Matrix Multiply

FLOP-proportionate throughput on GPUs and CPUs!
Single Machine: Batch Lowering + Dense Matrix Multiply

Key parameter: Batch size $b$

FLOP-proportionate throughput on GPUs and CPUs!
Multi-machine: Hardware v. Statistical Efficiency

Hardware Efficiency \( \times \) Statistical Efficiency = Total Time to Final Loss

(a) Hardware Efficiency
(b) Statistical Efficiency
(c) Total Time to Final Loss

FC Saturation

Normalized time/step

Normalized # of iter to conv.

Normalized time to conv.
Stochastic Gradient Descent (SGD)

Recall:

\[ f(w) = \frac{1}{n} \sum_{i=1}^{n} f(w; z_i) \]

**Objective we want to minimize**

\( z_i \) can be data point or batch

\[ w_{t+1} = w_t - \alpha_t \nabla_w f(w_t; z_{i_t}) \]

**SGD update**

\( \alpha_t \) is step size, \( i_t \) is batch used for \( t \)

Standard practice: Introducing an *explicit momentum term* to the update:

\[ w_{t+1} - w_t = \mu L (w_t - w_{t-1}) - \alpha_t \nabla_w f(w_t; z_{i_t}) \]

Usually just fixed at 0.9!
Asynchrony as an *implicit* momentum term

- If times to compute gradients are *exponentially distributed and independent*:

\[
\mathbb{E}[w_{t+1} - w_t] = \left(1 - \frac{1}{M}\right) \mathbb{E}[w_t - w_{t-1}] - \frac{1}{M} \alpha \mathbb{E}\nabla w f(w_t)
\]

An (implicit) momentum term!

Asynchrony can be viewed as an implicit momentum term- how does this help?
Simple Sketch of Optimizer Procedure

1. Start with the maximum # of compute groups (*full asynchrony*)

2. Tune momentum with grid search.

3. *If momentum = 0 is optimal, → too much momentum from asynchrony!*
   • *Reduce the number of compute groups & repeat from step 2*

**Intuition:** Zero explicit momentum would imply higher-than-optimal implicit momentum, impacting statistical efficiency
Theoretical and Experimental Results

Latest experiments show we can use *negative* explicit momentum to cancel some of the implicit momentum. We can push the limit of asynchrony further down!
Takeaways

• We can accelerate non-expert developers of ML applications by having them focus on programmatically generating training data

• We can use automated feature-generation methods—such as deep learning methods—to obviate feature engineering
  • (in many common cases)

• We can train these networks at scale using optimized *asynchronous* learning techniques

Thanks!