Hands-on Learning to Search for Structured Joint Prediction

Kai-Wei Chang @UIUC->MSR-NE

Hal Daume III, He He, Sudha Rao @UMaryland

John Langford @MSR-NYC

git clone https://github.com/JohnLangford/vowpal_wabbit.git

NAACL, May 31
The Problem: Joint Prediction

How? Other answers
The Problem: Joint Prediction

How? Other answers

1. Each prediction is independent.
The Problem: Joint Prediction

1. Each prediction is independent.
The Problem: Joint Prediction

How? Other answers

1. Each prediction is independent.
3. Assume tractable graphical model, optimize.
The Problem: Joint Prediction

How? Other answers

1. Each prediction is independent.
3. Assume tractable graphical model, optimize.
What makes a good solution?

1. Programming complexity.
What makes a good solution?

1. **Programming complexity.** Most complex problems addressed independently—too complex to do otherwise.
What makes a good solution?

1. **Programming complexity.** Most complex problems addressed independently—too complex to do otherwise.

2. **Prediction accuracy.** It had better work well.
What makes a good solution?

1. **Programming complexity.** Most complex problems addressed independently—too complex to do otherwise.

2. **Prediction accuracy.** It had better work well.

3. **Train speed.** Debug/development productivity + maximum data input.
What makes a good solution?

1. **Programming complexity.** Most complex problems addressed independently—too complex to do otherwise.

2. **Prediction accuracy.** It had better work well.

3. **Train speed.** Debug/development productivity + maximum data input.

4. **Test speed.** Application efficiency
A program complexity comparison

![Graph showing comparison of lines of code for POS tagging across different models. CRFSGD, CRF++, S-SVM, and Search are compared.]
The Plan

Part 1

1. Machine Learning + Vowpal Wabbit Background
   1. Input format
   2. One-against-all Multiclass
   3. Regression

2. Joint prediction by Imitation Learning

3. Joint Prediction by Learning to Search

Part 2: Hands-on.

Let’s solve Named Entity Recognition.
Part of Speech Tagging

wget http://bit.ly/1FVkLEK
Part of Speech Tagging

wget http://bit.ly/1FVkLEK
unzip 1FVkLEK
Part of Speech Tagging

wget http://bit.ly/1FVkLEK
unzip 1FVkLEK
less wsj.train.vw

1 | w Despite
2 | w continuing
3 | w problems
1 | w in
4 | w its
5 | w newsprint
5 | w business

In general:
label | Namespace Feature
An approach: One-Against-All (OAA)

Create $k$ binary problems, one per class. For class $i$ predict "Is the label $i$ or not?"

$$(x, y) \mapsto \begin{cases} (x, 1(y = 1)) \\ (x, 1(y = 2)) \\ \vdots \\ (x, 1(y = k)) \end{cases}$$

Then reduce to regression.
1. Hashing: avoid dictionaries for simplicity/speed.
2. Online Learning: quick optimization.
3. Progressive Validation: test one ahead of train for unbiased perf.
**Key ideas:**

1. **Hashing:** avoid dictionaries for simplicity/speed.
2. **Online Learning:** quick optimization.
3. **Progressive Validation:** test one ahead of train for unbiased perf.
Cost-sensitive multiclass classification

Distribution $D$ over $X \times [0, 1]^k$, where a vector in $[0, 1]^k$ specifies the cost of each of the $k$ choices.

Find a classifier $h : X \rightarrow \{1, \ldots, k\}$ minimizing the expected cost

$$\text{cost}(c, D) = \mathbb{E}_{(x,c) \sim D}[c_{h(x)}].$$
Cost-sensitive multiclass classification

Distribution $D$ over $X \times [0, 1]^k$, where a vector in $[0, 1]^k$ specifies the cost of each of the $k$ choices.

Find a classifier $h : X \rightarrow \{1, \ldots, k\}$ minimizing the expected cost

$$\text{cost}(c, D) = E_{(x,c) \sim D}[c_{h(x)}].$$

1. Is this packet \{normal, error, attack\}?
2. A key primitive for learning to search.
Use in VW

Label information via sparse vector.
A test example:
|Namespace Feature
A test example with only classes 1,2,4 valid:
1: 2: 4: |Namespace Feature
A training example with only classes 1,2,4 valid:
1:0.4 2:3.1 4:2.2 |Namespace Feature
Use in VW

Label information via sparse vector.
A test example:

|Namespace Feature
A test example with only classes 1,2,4 valid:

1: 2: 4: |Namespace Feature

A training example with only classes 1,2,4 valid:

1:0.4 2:3.1 4:2.2 |Namespace Feature

Methods:

- csoaa k cost-sensitive OAA prediction. $O(k)$ time.
- csoaa_ldf Label-dependent features OAA.
- wap_ldf LDF Weighted-all-pairs.
The reduction to regression

tar -xvzf 1EvYlm9
zless VW_raw/rcv1.train.raw.txt.gz
The reduction to regression

tar -xvzf 1EvYlm9
zless VW_raw/rcv1.train.raw.txt.gz

```
1 | tuesday year million short compan ...
-1 | econom stock rate month year invest ...
...```

...
tar -xvzf 1EvYlm9
zless VW_raw/rcv1.train.raw.txt.gz
1  tuesday year million short compan ...
-1  econom stock rate month year invest ...
...
vw -c rcv1.train.raw.txt -b 22 --ngram 2 --skips 4 -l 0.25 --binary
generates good solution.
Features: a vector $x \in \mathbb{R}^n$
Label: $y \in \mathbb{R}$
Goal: Learn $w \in \mathbb{R}^n$ such that $\hat{y}_w(x) = \sum_i w_i x_i$ is close to $y$. 
Online Linear Learning

Start with $\forall i : \ w_i = 0$

Repeatedly:

1. Get features $x \in \mathbb{R}^n$.
2. Make linear prediction $\hat{y}_w(x) = \sum_i w_i x_i$.
3. Observe label $y \in \mathbb{R}$.
4. Update weights so $\hat{y}_w(x)$ is closer to $y$.

Example: $w_i \leftarrow w_i + \eta(y - \hat{y})x_i$. 
Repeated squared loss updates

Squared Loss
Repeated update

loss
prediction when y=1
The Plan

1. Part 1
   1. Machine Learning + Vowpal Wabbit Background
   2. Joint prediction by Imitation Learning
   3. Joint Prediction by Learning to Search

   Let’s solve Named Entity Recognition.
Joint prediction via imitation learning

Part of Speech Tagging

NLP algorithms use a kitchen sink of features

Dependency Parsing

NLP algorithms use a kitchen sink of features
Joint prediction via imitation learning

Joint Prediction Haiku

A joint prediction
Across a single input
Loss measured jointly
An analogy from playing Mario

From Mario AI competition 2009

Input:

Output:
Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

High level goal:
Watch an expert play and learn to mimic her behavior
Vanilla supervised learning

1. Collect trajectories from expert $\pi^*$
   - Trajectory = sequence of state/action pairs over time
   - States are represented as feature vectors
     - Incorporates current “observations” …
     - … and any past decisions

2. Store as dataset $D = \{ (s, \pi^*(s)) \mid s \sim \pi^* \}$

3. Train classifier $\pi$ on $D$
   - Let $\pi$ play the game!
Training (expert)

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
Test-time execution (classifier)

Supervised Approach after 100K Training Samples

Video credit: Stéphane Ross, Geoff Gordon and Drew Bagnell
What's the (biggest) failure mode?

- The expert never gets stuck next to pipes
- => Classifier doesn't learn to recover!
Imitation learning: DAgger

1. Collect trajectories from expert $\pi^*$
2. Dataset $D_0 = \{ (s, \pi^*(s)) | s \sim \pi^* \}$
3. Train $\pi_1$ on $D_0$
4. Collect new trajectories from $\pi_1$
   - But let the expert steer!
5. Dataset $D_1 = \{ (s, \pi^*(s)) | s \sim \pi_1 \}$
6. Train $\pi_2$ on $D_0 \cup D_1$

- In general:
  - $D_n = \{ (s, \pi^*(s)) | s \sim \pi_n \}$
  - Train $\pi_n$ on $\cup_{i<n} D_i$

If $N = T \log T$, $L(\pi_n) < T \epsilon_N + O(1)$ for some $n$
Test-time execution (DAgger)
What's the biggest failure mode?

- Classifier only sees “right” versus “not-right”
- No notion of “better” or “worse”
- No “partial credit”
- Must have a single “target” answer
Joint prediction via learning to search

Part of Speech Tagging

NLP algorithms use a kitchen sink of features

Dependency Parsing

*ROOT*

NLP algorithms use a kitchen sink of features
Learning to search

1. Generate an initial trajectory using a *rollin policy*

2. Foreach state $R$ on that trajectory:
   a) Foreach possible action $a$ (one-step deviations)
      i. Take that action
      ii. Complete this trajectory using a rollout policy
      iii. Obtain a final loss
   b) Generate a cost-sensitive classification example:
      \[ ( \Phi(R), \langle c_a \rangle_{a \in A} ) \]
Choosing the rollin/rollout policies

- Three basic options:
  - The currently learned policy ("learn")
  - The reference/expert policy ("ref")
  - A stochastic mixture of these ("mix")

<table>
<thead>
<tr>
<th></th>
<th>Out</th>
<th>Ref</th>
<th>Mix</th>
<th>Learn</th>
</tr>
</thead>
<tbody>
<tr>
<td>In</td>
<td>Inconsistent</td>
<td>One-step fail</td>
<td>Inconsistent</td>
<td>Inconsistent</td>
</tr>
<tr>
<td>Ref</td>
<td>Inconsistent</td>
<td>One-step fail</td>
<td>Inconsistent</td>
<td>Inconsistent</td>
</tr>
<tr>
<td>Learn</td>
<td>One-step fail</td>
<td></td>
<td>Good</td>
<td>Really hard</td>
</tr>
</tbody>
</table>

Note: if the reference policy is *optimal* then: In=Learn & Out=Ref is also a good choice.

Sanity check: which of these is closest to DAgger?
The oracle (reference) policy gives the true label for the corresponding word.

Sanity check: why/when is this optimal?

```python
def _run(self, sentence):
    out = []
    for n in range(len(sentence)):
        pos, word = sentence[n]
        ex = example({'w': [word]})
        pred = predict(ex, pos)
        out.append(pred)
    loss( # of pred != pos )
    return out
```

- The oracle (reference) policy gives the true label for the corresponding word.
- Sanity check: why/when is this optimal?
Optimal policies

**Given:**
- Training input $x$
- State $R$
- Loss function

**Return the action $a$ that:**
- (If all future actions are taken optimally)
- Minimizes the corresponding loss
Optimal policies for harder problems

- Consider word-based machine translation

- You want to write

- But what does the optimal policy do?

F: Marie programme l'ordinateur
E: Mary programs the computer

State R: Mary ______
State R': The computer ______
State R'': Aardvarks ______

F, ref = input
E = [ <s> ]
i = 1
cov = {}

while |cov| != |F|:
a = predict(cov, ???)
e = predict(Fa, ???)
cov[a] = true
E.push(e)
i += 1

loss(1 - BLEU(E, ref))
return E
How can you do this for Mario?

Input:

Output:

Jump in \{0,1\}
Right in \{0,1\}
Left in \{0,1\}
Speed in \{0,1\}

Reference policy is constructed on-the-fly:
At each state, execute a depth-4 BFS
At each of the 64k leaves, evaluate
Choose initial action that leads to local optimum
Key concepts and commentary

- Rollin / rollout / one-step deviations
- Reference policy / optimal policy
- Joint loss

Tips:
- Defining a good reference can be tricky:
  - If optimal, do: in=learn, out=ref\|none
  - If suboptimal, do: in=learn, out=mix
- Can only learn to avoid compounding errors given the right features
Coming up next....

- Instantiating these ideas in vw
- During the break, please:
  
git clone git@github.com:JohnLangford/vowpal_wabbit.git
make
make python
cd python
python test.py
python test_search.py

- And ask us any questions you might have!
- When we return, we'll build some predictors!
A short reading list

• **DAgger (imitation learning from oracle):**
  A reduction of imitation learning and structured prediction to no-regret online learning
  Ross, Gordon & Bagnell, *AIStats 2011*

• **AggreVaTe (roughly “DAgger with rollouts”):**
  Reinforcement and imitation learning via interactive no-regret learning
  Ross & Bagnell, *arXiv:1406.5979*

• **LOLS (analysis of rollin/rollout, lower bounds, suboptimal reference):**
  Learning to search better than your teacher
  Chang, Krishnamurthy, Agarwal, Daumé III & Langford, *ICML 2015*

• **Imperative learning to search (programming framework, sequence labeling results):**
  Efficient programmable learning to search
  Chang, Daumé III, Langford & Ross, *arXiv:1406.1837*

• **State of the art dependency parsing in ~300 lines of code:**
  Learning to search for dependencies
  Chang, He, Daumé III & Langford, *arXiv:1503.05615*

• **Efficiently computing an optimal policy for shift-reduce dependency parsing:**
  A tabular method for dynamic oracles in transition-based parsing
  Goldberg, Sartorio & Satta, *TACL 2014*
Agenda

• Command-line usage of vw-l2s for canonical tasks:
  • Sequence labeling
  • Sequence span labeling
  • Graph labeling

• Intro to pyvw (vw in python interface)

• Learning to search in pyvw
  • Part of speech tagging walk-through
  • Named entity recognition exercise
Part of speech tagging on one slide

wget http://bit.ly/1FVkLEK
unzip 1FVkLEK

vw --search 45
   --search_task sequence
   --search_rollin learn
   --search_rollout none
   --affix -2w,+2w
   --spelling w
   --search_history_length 2
   --search_neighbor_features -1:p,1:p,-1:w,1:w
   -b 26
   -f wsj.train.model
   -d wsj.train.vw

... patience ...

vw -i wsj.train.model
   -p wsj.test.pred
   -d wsj.test.vw
   -t
Sequence span labeling

optional: --search_task sequencespan

- --search_span_bilou

- Plus special BIO encoding of labels:
  - “Out” = 1
  - “Begin-X” = any even # at least 2
  - “In-X” = “Begin-X” + 1
Graph labeling

--search_task graph

- Data encoding; for each graph:
  - List of nodes with labels and features
  - List of (hyper)edges with features

- See search_graph.cc for more docs
Intro to pyvw

- From vowpal_wabbit directory, run:
  ```
  cd python
  make
  python test.py
  ```

  If that doesn't work, look on with your neighbor

- If you have iPython installed, run:
  ```
  ipython notebook VW_in_Python.ipynb
  ```

- Or view at: http://tinyurl.com/pyvwintro
Pythonic part of speech tagging

• Open notebook
  Learning_to_Search.ipynb

or view at
  http://tinyurl.com/pyvwsearch
Your homework assignment

- Download: http://hal3.name/ner.zip

- Let's build a named entity recognizer!

- Files:
  - `ner.py` basic scaffolding
  - `ner_assignment.txt` your homework
  - `ner_solution.py` my solution to your homework
  - `moredata.py` a larger dataset to play with
We're here to help!