Week 4: Extensions and Variations of Perceptron, and Practical Issues

Professor Liang Huang

some slides from A. Zisserman (Oxford)
Trivia: Grace Hopper and the first bug

- Edison coined the term “bug” around 1878 and since then it had been widely used in engineering.
- Hopper was associated with the discovery of the first computer bug in 1947 which was a moth stuck in a relay.
Week 4: Perceptron in Practice

- Problems with Perceptron
  - doesn’t converge with inseparable data
  - update might often be too “bold”
  - doesn’t optimize margin
  - result is sensitive to the order of examples

- Ways to alleviate these problems (without SVM/kernels)
  - Part II: voted perceptron and average perceptron
  - Part III: MIRA (margin-infused relaxation algorithm)
  - Part IV: Practical Issues and HW I
  - Part V: “Soft” Perceptron: Logistic Regression

“A ship in port is safe, but that is not what ships are for.”

– Grace Hopper (1906-1992)
Recap of Week 3

input: training data $D$
output: weights $w$
initialize $w \leftarrow 0$
while not converged
  for $(x, y) \in D$
    if $y(w \cdot x) \leq 0$
      $w \leftarrow w + yx$

“idealized” ML

<table>
<thead>
<tr>
<th>Input $x$</th>
<th>→</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output $y$</td>
<td>→</td>
<td>Model $w$</td>
</tr>
</tbody>
</table>

“actual” ML

<table>
<thead>
<tr>
<th>Input $x$</th>
<th>→</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output $y$</td>
<td>→</td>
<td>Model $w$</td>
</tr>
<tr>
<td>feature map $\phi$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>

depth learning $\approx$ representation learning

<table>
<thead>
<tr>
<th>Input $x$</th>
<th>→</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output $y$</td>
<td>→</td>
<td>Model $w$</td>
</tr>
<tr>
<td>feature map $\phi$</td>
<td>→</td>
<td></td>
</tr>
</tbody>
</table>
$ python perc_demo.py  
(requires numpy and matplotlib)
Part II: Voted and Averaged Perceptron

![Graph showing error rate over epochs for different models: vanilla perceptron, averaged perceptron, voted perceptron, random (unnorm), last (unnorm), and avg (unnorm). The graph indicates that voted perceptron has the lowest error rate compared to the others.]
**Brief History of Perceptron**

1959
- Rosenblatt invention
- Novikoff proof
- Minsky/Papert book killed it

1962
- Novikoff proof
- Minsky/Papert book killed it

1969
- Minsky/Papert book killed it

1970
- Cortes/Vapnik SVM

1997
- Cortes/Vapnik SVM

1999
- Freund/Schapire voted/avg: revived

1999*
- Freund/Schapire voted/avg: revived

2002
- Collins aggressive

2003
- Crammer/Singer MIRA

2005
- McDonald/Crammer/Pereira structured MIRA

2006
- Singer group aggressive

2007--2010*
- Singer group Pegasos

- subgradient descent
- online approx.
- max margin
- minibatch

1959
- Rosenblatt invention
- Novikoff proof
- Minsky/Papert book killed it

1997
- Cortes/Vapnik SVM

1999
- Freund/Schapire voted/avg: revived

2007--2010*
- Singer group Pegasos

- subgradient descent
- online approx.
- max margin
- minibatch

- conservative updates
- batch
- online
- minibatch

- +max margin
- +kernels
- +soft-margin

- AT&T Research
- ex-AT&T and students

*mentioned in lectures but optional (others papers all covered in detail)
problem: later examples dominate earlier examples

solution: voted perceptron (Freund and Schapire, 1999)

- record the weight vector after each example in $D$
  - not just after each update!

- and vote on a new example using $|D|$ models

- shown to have better generalization power

averaged perceptron (from the same paper)

- an approximation of voted perceptron
- just use the average of all weight vectors
- can be implemented efficiently
Voted Perceptron

Input: a labeled training set $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$
number of epochs $T$

Output: a list of weighted perceptrons $\langle (v_1, c_1), \ldots, (v_k, c_k) \rangle$

- Initialize: $k := 0$, $v_1 := 0$, $c_1 := 0$.
- Repeat $T$ times:
  - For $i = 1, \ldots, m$:
    * Compute prediction: $\hat{y} := \text{sign}(v_k \cdot x_i)$
    * If $\hat{y} = y$ then $c_k := c_k + 1$.
      else $v_{k+1} := v_k + y_i x_i$;
      $c_{k+1} := 1$;
      $k := k + 1$.$ \!

Large Margin Classification Using the Perceptron Algorithm

YOAV FREUND
AT&T Labs, Shannon Laboratory, 180 Park Avenue, Room A205, Florham Park, NJ 07932-0971

ROBERT E. SCHAPIRE
AT&T Labs, Shannon Laboratory, 180 Park Avenue, Room A279, Florham Park, NJ 07932-0971

Prediction
Given: the list of weighted perceptrons: $\langle (v_1, c_1), \ldots, (v_k, c_k) \rangle$
an unlabeled instance: $x$
compute a predicted label $\hat{y}$ as follows:

$$s = \sum_{i=1}^{k} c_i \text{sign}(v_i \cdot x); \quad \hat{y} = \text{sign}(s).$$

if correct, increase the current model’s # of votes; otherwise create a new model with 1 vote
Experiments
Averaged Perceptron

- voted perceptron is not scalable
- and does not output a single model
- avg perceptron is an approximation of voted perceptron
- actually, summing all weight vectors is enough; no need to divide

\[
\begin{align*}
\mathbf{w}(1) &= \Delta \mathbf{w}^{(1)} \\
\mathbf{w}(2) &= \Delta \mathbf{w}^{(2)} \\
\mathbf{w}(3) &= \Delta \mathbf{w}^{(3)} \\
\mathbf{w}(4) &= \Delta \mathbf{w}^{(4)}
\end{align*}
\]

initialize \( \mathbf{w} \leftarrow 0; \; \mathbf{w}_s \leftarrow 0 \)
while not converged
for \((x, y) \in D\)
  if \(y(\mathbf{w} \cdot x) \leq 0\)
    \(\mathbf{w} \leftarrow \mathbf{w} + yx\)
  \(\mathbf{w}_s \leftarrow \mathbf{w}_s + \mathbf{w}\)
output: summed weights \(\mathbf{w}_s\)

*after each example, not after each update!*
Efficient Implementation of Averaging

- naive implementation (running sum $w_s$) doesn’t scale either
- OK for low dim. (HW1); too slow for high-dim. (HW3)
- very clever trick from Hal Daumé (2006, PhD thesis)

Initialize $w \leftarrow 0; w_a \leftarrow 0; c \leftarrow 0$

While not converged
  For $(x, y) \in D$
    If $y(w \cdot x) \leq 0$
      $w \leftarrow w + yx$
      $w_a \leftarrow w_a + cyx$
    $c \leftarrow c + 1$

Output: $cw - w_a$

After each update, not after each example!
Part III: MIRA

- perceptron often makes bold updates (over-correction)
- and sometimes too small updates (under-correction)
- but hard to tune learning rate
- “just enough” update to correct the mistake?

\[ w' \leftarrow w + \frac{y - w \cdot x}{\|x\|^2} x \]

easy to show:

\[ w' \cdot x = (w + \frac{y - w \cdot x}{\|x\|^2} x) \cdot x = y \]

margin-infused relaxation algorithm (MIRA)
Example: Perceptron under-correction
MIRA: just enough

\[
\begin{align*}
\min_{w'} & \|w' - w\|^2 \\
\text{s.t.} & \quad w' \cdot x \geq 1
\end{align*}
\]

minimal change to ensure functional margin of 1 (dot-product \(w' \cdot x = 1\))

\[\text{MIRA} \approx \text{1-step SVM}\]

functional margin: \(y(w \cdot x)\)

geometric margin: \(\frac{y(w \cdot x)}{\|w\|}\)
\[ \min_{\mathbf{w}'} \| \mathbf{w}' - \mathbf{w} \|^2 \]
\[ \text{s.t. } \mathbf{w}' \cdot \mathbf{x} \geq 1 \]

minimal change to ensure functional margin of 1
(dot-product \( \mathbf{w}' \cdot \mathbf{x} = 1 \))

\[ \text{MIRA } \approx \text{ 1-step SVM} \]

Functional margin: \( y(\mathbf{w} \cdot \mathbf{x}) \)
Geometric margin: \( \frac{y(\mathbf{w} \cdot \mathbf{x})}{\|\mathbf{w}\|} \)
Optional: Aggressive MIRA

- aggressive version of MIRA
  - also update if correct but not confident enough
    - i.e., functional margin \((y \mathbf{w} \cdot \mathbf{x})\) not big enough
  - \(p\)-aggressive MIRA: update if \(y (\mathbf{w} \cdot \mathbf{x}) < p\) \(\ (0 \leq p < 1)\)
  - MIRA is a special case with \(p=0\): only update if misclassified!

- update equation is same as MIRA
  - i.e., after update, functional margin becomes 1

- larger \(p\) leads to a larger geometric margin but slower convergence
Demo
Part IV: Practical Issues

“A ship in port is safe, but that is not what ships are for.”

– Grace Hopper (1906-1992)

- you will build your own linear classifiers for HW2 (same data as HW1)
- slightly different binarizations
  - for k-NN, we binarize all categorical fields but keep the two numerical ones
  - for perceptron (and most other classifiers), we binarize numerical fields as well
- why? hint: larger “age” always better? more “hours” always better?
Useful Engineering Tips:
 averging, shuffling, variable learning rate, fixing feature scale

- averaging helps significantly; MIRA helps a tiny little bit
  - perceptron < MIRA < avg. perceptron ≈ avg. MIRA ≈ SVM
- shuffling the data helps hugely if classes were ordered (HW1)
  - shuffling before each epoch helps a little bit
- variable (decaying) learning rate often helps a little
  - 1/(total#updates) or 1/(total#examples) helps
- any requirement in order to converge?
  - how to prove convergence now?
- centering of each dimension helps (Ex1/HW1)
  - why? => smaller radius, bigger margin!
- unit variance also helps (why?) (Ex1/HW1)
  - 0-mean, 1-var => each feature ≈ a unit Gaussian
Feature Maps in Other Domains

• how to convert an image or text to a vector?

28x28 grayscale image

\[ x \in \mathbb{R}^{784} \]

23x23 RGB image

\[ x \in \mathbb{R}^{23 \times 23 \times 3} \]

• image

<table>
<thead>
<tr>
<th>“a”</th>
<th>“abbreviations”</th>
<th>“zoology”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

• text

“one-hot” representation of words
(all binary features)

in deep learning there are other feature maps
Part V: Perceptron vs. Logistic Regression

- logistic regression is another popular linear classifier
- can be viewed as “soft” or “probabilistic” perceptron
- same decision rule (sign of dot-product), but prob. output

\[ f(x) = \text{sign}(w \cdot x) \]

perceptron

\[ f(x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}} \]

logistic regression
Logistic vs. Linear Regression

- Linear regression is regression applied to real-valued output using linear function.
- Logistic regression is regression applied to 0-1 output using the sigmoid function.

[Diagram showing linear and logistic regression with 1 and 2 features]
Why Logistic instead of Linear

- linear regression easily dominated by distant points
- causing misclassification

\[ \sigma(wx + b) \text{ fit to } y \]
\[ wx + b \text{ fit to } y \]

- fit of \( wx + b \) dominated by more distant points
- causes misclassification
- instead LR regresses the sigmoid to the class data

Why Logistic instead of Linear

- linear regression easily dominated by distant points
- causing misclassification
Why 0/1 instead of +/-1

- perc: \( y = +1 \) or \( -1 \); logistic regression: \( y = 1 \) or 0
- reason: want the output to be a probability
- decision boundary is still linear: \( p(y=1 \mid x) = 0.5 \)
Logistic Regression: Large Margin

- perceptron can be viewed roughly as “step” regression
- logistic regression favors large margin; SVM: max margin
- in practice: perc. << avg. perc. $\approx$ logistic regression $\approx$ SVM
perceptron
1958

logistic regression
1958

cond. random fields
2001

structured perceptron
2002

structured SVM
2003

voted/avg. perceptron
1999

kernels
1964

SVM
1964;1995

multilayer perceptron

深層學習
~1986; 2006-now