Deep Learning
10-715 Fall 2015

Alexander Smola
alex@smola.org

Office hours - after class in my office
Outline

• Basics
  • Perceptron (and convergence rule)
  • Stochastic gradient descent
  • Loss functions and objectives

• Deep Networks
  • Layers and Invariances
  • Advanced objectives

• State and Structure
  • Optimization
  • Autoregressive Models & Hidden State
  • Toolkits
1. Perceptron

Rosenblatt

Widom
Neurons and Learning
Biology and Learning

- **Basic Idea**
  - Good behavior should be rewarded, bad behavior punished (or not rewarded). This improves system fitness.
  - Killing a sabertooth tiger should be rewarded ...
  - Correlated events should be combined.
  - Pavlov’s salivating dog.

- **Training mechanisms**
  - Behavioral modification of individuals (learning) Successful behavior is rewarded (e.g. food).
  - Hard-coded behavior in the genes (instinct) The wrongly coded animal does not reproduce.
Neurons

- **Soma** (CPU)
  Cell body - combines signals

- **Dendrite** (input bus)
  Combines the inputs from several other nerve cells

- **Synapse** (interface)
  Interface and parameter store between neurons

- **Axon** (cable)
  May be up to 1m long and will transport the activation signal to neurons at different locations
Neurons

\[
f(x) = \sum_{i} w_i x_i = \langle w, x \rangle
\]
Perceptron

- Weighted linear combination
- Nonlinear decision function
- Linear offset (bias)
- Linear separating hyperplanes (spam/ham, novel/typical, click/no click)
- Learning - Estimating the parameters $w$ and $b$

$$f(x) = \sigma (\langle w, x \rangle + b)$$
Perceptron

Ham

Spam
Perceptron Algorithm
The Perceptron

**initialize** $w = 0$ and $b = 0$

repeat
  if $y_i [\langle w, x_i \rangle + b] \leq 0$ then
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly

- Nothing happens if classified correctly
- Weight vector is linear combination
- Classifier is linear combination of inner products

$$w = \sum_{i \in I} y_i x_i$$

$$f(x) = \sum_{i \in I} y_i \langle x_i, x \rangle + b$$
Convergence Theorem

• If there exists some \((w^*, b^*)\) with unit length and
  
  \[ y_i \left[ \langle x_i, w^* \rangle + b^* \right] \geq \rho \text{ for all } i \]

  then the perceptron converges to a linear separator after a number of steps bounded by

  \[
  \left( b^2 + 1 \right) \left( r^2 + 1 \right) \rho^{-2} \text{ where } \|x_i\| \leq r
  \]

• Dimensionality independent
• Order independent (i.e. also worst case)
• Scales with ‘difficulty’ of problem
Proof

Starting Point
We start from $w_1 = 0$ and $b_1 = 0$.

Step 1: Bound on the increase of alignment
Denote by $w_i$ the value of $w$ at step $i$ (analogously $b_i$).

Alignment: $\langle (w_i, b_i), (w^*, b^*) \rangle$

For error in observation $(x_i, y_i)$ we get

$$\langle (w_{j+1}, b_{j+1}) \cdot (w^*, b^*) \rangle$$
$$= \langle [(w_j, b_j) + y_i(x_i, 1)], (w^*, b^*) \rangle$$
$$= \langle (w_j, b_j), (w^*, b^*) \rangle + y_i \langle (x_i, 1) \cdot (w^*, b^*) \rangle$$
$$\geq \langle (w_j, b_j), (w^*, b^*) \rangle + \rho$$
$$\geq j \rho.$$

Alignment increases with number of errors.
Proof

Step 2: Cauchy-Schwartz for the Dot Product

\[ \langle (w_{j+1}, b_{j+1}) \cdot (w^*, b^*) \rangle \leq \| (w_{j+1}, b_{j+1}) \| \| (w^*, b^*) \| \]
\[ = \sqrt{1 + (b^*)^2} \| (w_{j+1}, b_{j+1}) \| \]

Step 3: Upper Bound on \( \| (w_j, b_j) \| \)

If we make a mistake we have

\[ \| (w_{j+1}, b_{j+1}) \|^2 = \| (w_j, b_j) + y_i(x_i, 1) \|^2 \]
\[ = \| (w_j, b_j) \|^2 + 2y_i \langle (x_i, 1), (w_j, b_j) \rangle + \| (x_i, 1) \|^2 \]
\[ \leq \| (w_j, b_j) \|^2 + \| (x_i, 1) \|^2 \]
\[ \leq j(R^2 + 1). \]

Step 4: Combination of first three steps

\[ j \rho \leq \sqrt{1 + (b^*)^2} \| (w_{j+1}, b_{j+1}) \| \leq \sqrt{j(R^2 + 1)((b^*)^2 + 1)} \]

Solving for \( j \) proves the theorem.
Consequences

• Only need to store errors. This gives a compression bound for perceptron.
• This is stochastic gradient descent on hinge loss
  \[ l(x_i, y_i, w, b) = \max(0, 1 - y_i [\langle w, x_i \rangle + b]) \]
• Fails miserably with noisy data

**do NOT train your avatar with perceptrons**
Stochastic Gradient with Hinge Loss

\[ l(x_i, y_i, w, b) = \max(0, 1 - y_i [\langle w, x_i \rangle + b]) \]
Hardness

hard

easy
2. Stochastic Gradient Descent
Objectives

- **(Linear) Function**
  \[ f(x) = \langle w, x \rangle + b \]
  We need to define what to do with it

- **Regression**
  Real valued \( y \), goodness measured by \( y - f(x) \)

\[
\begin{align*}
l(y, f(x)) &= |y - f(x)| & \text{absolute value} \\
l(y, f(x)) &= \frac{1}{2} (y - f(x))^2 & \text{least mean squares} \\
l(y, f(x)) &= \max(0, |y - f(x)|) & \text{\( \epsilon \)-insensitive} \\
l(y, f(x)) &= \begin{cases} |y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\ 0.5(y - f(x))^2 & \text{otherwise} \end{cases} & \text{Huber}
\end{align*}
\]
\[ l(y, f(x)) = |y - f(x)| \]
\[ l(y, f(x)) = \frac{1}{2} (y - f(x))^2 \]
\[ l(y, f(x)) = \max(0, |y - f(x)|) \]
\[ l(y, f(x)) = \begin{cases} 
|y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\
0.5(y - f(x))^2 & \text{otherwise}
\end{cases} \]
Regression applications

- **Stock market prediction**
  (often better to convert to log-price)

- **Image Superresolution**
  (regress from lower dimensional to higher dimensional image - Laplacian pyramid)

- **Recommendation and rating**
  (Netflix, Spotify, Yelp, Amazon)

- **Embeddings**
  Feature vectors for words, profiles, images
Objectives

• Classification
  • Binary y, e.g. \{apples, oranges\}
  • Multiple categories, e.g. \{red, green, blue\}
  • Ordinal relationship, e.g. \{A, B, C, D, Fail\}

\[
l(y, f(x)) = \log \left( 1 + e^{-yf(x)} \right)
\]

\[
l(y, f(x)) = \max(0, 1 - yf(x))
\]

\[
l(y, f(x, \cdot)) = \log \sum_{y'} e^{f(x, y')} - f(x, y)
\]

\[
l(y, f(x, \cdot)) = \max_{y'} \left[ f(x, y') + \Delta(y, y') \right] - f(x, y)
\]
Logistic Regression

\[
p(y|f(x)) = \frac{1}{1 + e^{-yf(x)}}
\]
\[
\max(0, 1 - f(x)) \quad \text{SVM}
\]

\[
\log \left( 1 + e^{-f(x)} \right)
\]

Binary Classification Losses
Risk

• Ideal Case
Minimize expected risk

\[ R[f] := \mathbb{E}_x \left[ \mathbb{E}_{y|x} [l(y, f(x))] \right] \]

• Reality
We only have samples (x,y) drawn from distribution. Hence minimize empirical risk

\[ R_{\text{emp}}[f] := \frac{1}{m} \sum_{i=1}^{m} l(y_i, f(x_i)) \]

• Challenges
  Overfitting (regularization), Redundancy (SGD)
Stochastic Gradient Descent
Recall ... Perceptron

initialize $w = 0$ and $b = 0$
repeat
  if $y_i [\langle w, x_i \rangle + b] \leq 0$ then
    $w \leftarrow w + y_i x_i$ and $b \leftarrow b + y_i$
  end if
until all classified correctly
Stochastic gradient descent

- Empirical risk

\[ R_{\text{emp}}[f] := \frac{1}{m} \sum_{i=1}^{m} l(y_i, f(x_i)) \]

- Stochastic gradient descent (pick random x,y)

\[ w_{t+1} \leftarrow w_t - \eta_t \partial_f(x_i) l(y_i, f(x_i)) \partial_w f(x_i) \]

- Often we require that parameters are restricted to some convex set X, hence we project on it

Rate is \( O \left( t^{-\frac{1}{2}} \right) \)

faster for strong convexity
Convergence in Expectation

\[ E_{\bar{\theta}} [l(\bar{\theta})] - l^* \leq \frac{R^2 + L^2 \sum_{t=0}^{T-1} \eta_t^2}{2 \sum_{t=0}^{T-1} \eta_t} \]

where

\[ l(\theta) = E_{(x,y)} [l(y, \langle \phi(x), \theta \rangle)] \quad \text{and} \quad l^* = \inf_{\theta \in X} l(\theta) \quad \text{and} \quad \bar{\theta} = \frac{\sum_{t=0}^{T-1} \theta_t \eta_t}{\sum_{t=0}^{T-1} \eta_t} \]

- **Proof**
  
  Show that parameters converge to minimum

\[ \theta^* \in \arg\min_{\theta \in X} l(\theta) \quad \text{and set} \quad r_t := ||\theta^* - \theta_t|| \]

Nesterov and Vial 10.1016/j.automatica.2008.01.017
Proof

\[ r_{t+1}^2 = \| \pi_X [\theta_t - \eta_t g_t] - \theta^* \|^2 \]

\[ \leq \| \theta_t - \eta_t g_t - \theta^* \|^2 \]

\[ = r_t^2 + \eta_t^2 \| g_t \|^2 - 2 \eta_t \langle \theta_t - \theta^*, g_t \rangle \]

hence \( \mathbb{E} [r_{t+1}^2 - r_t^2] \leq \eta_t^2 L^2 + 2 \eta_t [l^* - \mathbb{E}[l(\theta_t)]] \]

\[ \leq \eta_t^2 L^2 + 2 \eta_t [l^* - \mathbb{E}[l(\bar{\theta})]] \]

- Summing over inequality for \( t \) proves claim
- This yields randomized algorithm for minimizing objective functions (try log times and pick the best / or average median trick)
Rates

- **Guarantee**
  
  \[ E_{\theta} [l(\theta)] - l^* \leq \frac{R^2 + L^2 \sum_{t=0}^{T-1} \eta_t^2}{2 \sum_{t=0}^{T-1} \eta_t} \]

- **If we know R, L, T pick constant learning rate**

  \[ \eta = \frac{R}{L\sqrt{T}} \]  
  and hence \[ E_{\theta}[l(\theta)] - l^* \leq \frac{R[1 + 1/T]L}{2\sqrt{T}} < \frac{LR}{\sqrt{T}} \]

- **If we don’t know T pick**  
  \[ \eta_t = O(t^{-\frac{1}{2}}) \]  
  This costs us an additional log term

  \[ E_{\theta}[l(\theta)] - l^* = O \left( \frac{\log T}{\sqrt{T}} \right) \]
Strong Convexity

\[ l_i(\theta') \geq l_i(\theta) + \langle \partial_\theta l_i(\theta), \theta' - \theta \rangle + \frac{1}{2} \lambda \| \theta - \theta' \|^2 \]

- Use this to bound the expected deviation

\[
r_{t+1}^2 \leq r_t^2 + \eta_t^2 \| g_t \|^2 - 2\eta_t \langle \theta_t - \theta^*, g_t \rangle \\
\leq r_t^2 + \eta_t^2 L^2 - 2\eta_t [ l_t(\theta_t) - l_t(\theta^*) ] - 2\lambda \eta_t r_k^2
\]

hence \( \mathbb{E}[r_{t+1}^2] \leq (1 - \lambda h_t) \mathbb{E}[r_t^2] - 2\eta_t [ \mathbb{E}[l(\theta_t)] - l^* ] \)

- Exponentially decaying averaging

\[
\bar{\theta} = \frac{1 - \sigma}{1 - \sigma T} \sum_{t=0}^{T-1} \sigma^{T-1-t} \theta_t
\]

and plugging this into the discrepancy yields

\[
l(\bar{\theta}) - l^* \leq \frac{2L^2}{\lambda T} \log \left[ 1 + \frac{\lambda RT^{1/2}}{2L} \right] \quad \text{for} \quad \eta = \frac{2}{\lambda T} \log \left[ 1 + \frac{\lambda RT^{1/2}}{2L} \right]
\]
More variants

- Adversarial guarantees
  \[
  \theta_{t+1} \leftarrow \pi_x [\theta_t - \eta_t \partial_\theta (y_t, \langle \phi(x_t), \theta_t \rangle)]
  \]
  has low regret (average instantaneous cost) for arbitrary orders (useful for game theory)
- Ratliff, Bagnell, Zinkevich
  \( O(t^{-1/2}) \) learning rate
- Shalev-Shwartz, Srebro, Singer (Pegasos)
  \( O(t^{-1}) \) learning rate (but need constants)
- Bartlett, Rakhlin, Hazan
  (add strong convexity penalty)
In a Nutshell
Key Components

• Data & distribution
  (image, ‘cat’), (email, ‘spam’), ((user,movie), ★★★)

• Loss Function
  least mean squares, logistic, multiclass, …

• Function class
  linear, nonlinear, kernels, deep

• Optimization procedure
  Stochastic gradient descent, batch, sampling

• Capacity control
  Norm of coefficients, operator, dropout
Key Components

- **Data & distribution**
  \((\text{cat}, \text{cat'})\)

- **Loss Function**
  \[
  \log \left( 1 + e^{-yf(x)} \right)
  \]

- **Function class**
  \[
  f(x) = \langle w, x \rangle
  \]

- **Optimization procedure**
  \[
  w \leftarrow w + \eta \frac{yx}{1 + eyf(x)}
  \]

- **Capacity control**
  \[
  w \leftarrow (1 - \eta \lambda)w + \eta \frac{yx}{1 + eyf(x)}
  \]
**Model Selection**

- **Simple model**  
  Linear function in $\mathbb{R}$, $n$ observations, no problem  
  (e.g. estimating voltage in sockets)

- **High dimensional model**  
  Linear function in $\mathbb{R}^d$, $n < d$ observations  
  The problem is underdetermined

- **Even for $d = O(n)$ not enough data to handle noise**  
  - Only small number of nonzeros (sparsity)  
  - Only small coefficients (norm)  
  - Insensitive to local changes (dropout)
Outline

• Basics
  • Perceptron (and convergence rule)
  • Stochastic gradient descent
  • Loss functions and objectives

• Deep Networks
  • Layers and Invariances
  • Advanced objectives

• State and Structure
  • Optimization
  • Autoregressive Models & Hidden State
  • Toolkits

Marianas Labs
3. Backprop
## A brief history of computers

<table>
<thead>
<tr>
<th></th>
<th>1970s</th>
<th>1980s</th>
<th>1990s</th>
<th>2000s</th>
<th>2010s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>$10^2$</td>
<td>$10^3$</td>
<td>$10^5$</td>
<td>$10^8$</td>
<td>$10^{11}$</td>
</tr>
<tr>
<td>RAM</td>
<td>?</td>
<td>1MB</td>
<td>100MB</td>
<td>10GB</td>
<td>1TB</td>
</tr>
<tr>
<td>CPU</td>
<td>?</td>
<td>10MF</td>
<td>1GF</td>
<td>100GF</td>
<td>1PF GPU</td>
</tr>
</tbody>
</table>

- Data grows at higher exponent
- Moore’s law (silicon) vs. Kryder’s law (disks)
- Early algorithms data bound, now CPU/RAM bound

Marianas Labs

Carnegie Mellon University
Perceptron

\[ y(x) = \sigma(\langle w, x \rangle) \]
Nonlinearities via Layers

\[ y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle) \]
\[ y_2(x) = \sigma(\langle w_2, y_1 \rangle) \]

\[ y_{1i} = k(x_i, x) \]

Kernels

Deep Nets

optimize all weights
Nonlinearities via Layers

\[ y_{1i}(x) = \sigma(\langle w_{1i}, x \rangle) \]
\[ y_{2i}(x) = \sigma(\langle w_{2i}, y_{1i} \rangle) \]
\[ y_{3i}(x) = \sigma(\langle w_{3i}, y_{2i} \rangle) \]
Multilayer Perceptron

- **Layer Representation**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]

- (typically) iterate between linear mapping \( Wx \) and nonlinear function

- **Loss function** \( l(y, y_i) \) to measure quality of estimate so far
Multilayer Perceptron

- **Layer Representation**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]

- **Change in Objective**
  \[ g_j = \partial_{W_j} l(y, y_i) \]

- **Chain Rule**
  \[ \partial_x [f_2 \circ f_1](x) = [\partial_{f_1} f_2 \circ f_1](x) [\partial_x f_1](x) \]
Multilayer Perceptron

- **Layer Representation**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]

- **Gradients**
  \[ \partial_{x_i} y_i = W_i \]
  \[ \partial_{W_i} y_i = x_i \]
  \[ \partial_{y_i} x_{i+1} = \sigma'(y_i) \]
  \[ \implies \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top \]
Backpropagation

- **Layers & Gradients**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]
  \[
  \frac{\partial x_i y_i}{\partial x_i} = W_i \quad \frac{\partial W_i y_i}{\partial W_i} = x_i \\
  \frac{\partial y_i x_{i+1}}{\partial y_i} = \sigma'(y_i) \quad \frac{\partial x_i x_{i+1}}{\partial x_i} = \sigma'(y_i) W_i^T
  \]

- **Backpropagation**

  \[
  \frac{\partial x_i l(y, y_n)}{\partial x_i} = \frac{\partial x_{i+1} l(y, y_n)}{\partial x_i} x_{i+1} \\
  = \frac{\partial x_{i+1} l(y, y_n)}{\partial x_{i+1}} \sigma'(y_i) W_i^T \\
  = \frac{\partial y_i l(y, y_n)}{\partial y_i} \frac{\partial W_i y_i}{\partial W_i} \\
  = \frac{\partial y_i l(y, y_n)}{\partial y_i} \sigma'(y_i) x_i^T
  \]
Backpropagation

- **Layers & Gradients**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]
  \[ \partial x_i y_i = W_i \]
  \[ \partial W_i y_i = x_i \]
  \[ \partial y_i x_{i+1} = \sigma'(y_i) \]
  \[ \Longrightarrow \partial x_i x_{i+1} = \sigma'(y_i) W_i^\top \]

- **Backpropagation**
  \[ g_n = l'(y, y_n) W_n \]
  \[ g_i = g_{i+1} \sigma'(y_i) W_i^\top \]
  \[ \partial W_i l(y, n_n) = g_{i+1} \sigma'(y_i) x_i^\top \]
Optimization

- **Layer Representation**
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]

- **Gradient descent**
  \[ W_i \leftarrow W_i - \eta \partial_{W_i} l(y, y_n) \]

- **Second order method**
  (use higher derivatives)

- **Stochastic gradient descent**
  (use only one sample)

- **Minibatch** (small subset)
4. Layers & Symmetries
Fully Connected

- Forward mapping
  \[ y_i = W_i x_i \]
  \[ x_{i+1} = \sigma(y_i) \]
  with subsequent nonlinearity

- Backprop gradients
  \[ \partial_{x_i} x_{i+1} = \sigma'(y_i) W_i^\top \]
  \[ \partial_{W_i} x_{i+1} = \sigma'(y_i) x_i^\top \]

- General purpose layer

Marianas Labs
Example - Word Vectors

- Full vocabulary (say 30k words)
- One-hot encoding for words
  \((0,0,0,0,1,0,0,0,0)\) \((0,0,0,0,0,0,0,1,0)\)
  
  - `cat`  
  - `dog`  

- Map words into vector space via \(y = Wx\)
  
  - `laptop`
  - `dinosaur`
  - `dog`
  - `cat`

- Postprocess as normal.
Rectified Linear Unit (ReLu)

- Forward mapping
  
  \[ y_i = W_i x_i \]
  
  \[ x_{i+1} = \sigma(y_i) \]

  with subsequent nonlinearity

- Gradients vanish at tails
- Solution - replace by \( \max(0, x) \)
  
  - Derivative is in \{0; 1\}
  
  - Sparsity of signal

  (Nair & Hinton, machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf)
Where is Wally?
No need to learn a Wally model for each location. All are the same!
LeNet for OCR (1990s)
Convolutional Layers

• Images have translation invariance (to some extent)
• Low level is mostly edge and feature detectors
• Usually via convolution (plus nonlinearity)
Convolutional Layers

• **Feature Locality**
  Relevant information only in neighborhood of pixel

\[ y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ij,ab} x_{i+a,j+b} \]

• **Translation Invariance**
  Weights invariant relative to shift in point of view

\[ y_{ij} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{ab} x_{i+a,j+b} \]
Convolutional Layers

- Images have translation invariance
- Forward (brute force BLAS beats convolution)
  \[ y_i = x_i \circ W_i \]
  \[ x_{i+1} = \sigma(y_i) \]
- Backward gradients: convolve appropriately (see homework)
Subsampling & MaxPooling

- Multiple convolutions blow up dimensionality

- Subsampling - average over patches (this works decently)
- MaxPooling - pick the maximum over patches (often non overlapping ones)
• Multiple convolutions blow up dimensionality

- Subsampling

\[ y_{ij} = \frac{1}{4} \left[ x_{2i,2j} + x_{2i+1,2j} + x_{2i,2j+1} + x_{2i+1,2j+1} \right] \]

- MaxPooling

\[ y_{ij} = \max \left[ x_{2i,2j}, x_{2i+1,2j}, x_{2i,2j+1}, x_{2i+1,2j+1} \right] \]
### Depth vs. Width

- Longer range effects
- Many narrow convolutions
- Few wide convolutions
- More nonlinearities work better (same number of parameters)

Simonyan and Zisserman

Fancy structures

- Compute different filters
- Compose one big vector from all of them
- Layer this iteratively

Szegedy et al. arxiv.org/pdf/1409.4842v1.pdf
Fancy structures

- Compute different filters
- Compose one big vector from all of them
- Layer this iteratively

Szegedy et al. arxiv.org/pdf/1409.4842v1.pdf

Marianas Labs
def SimpleFactory(data, ch_1x1, ch_3x3):
    global concat_cnt
    param = {}
    # 1x1
    param["kernel"] = (1, 1)
    param["num_filter"] = ch_1x1
    param["pad"] = (0, 0)
    param["stride"] = (1, 1)
    param["act_type"] = "relu"
    param["data"] = data
    conv1x1 = ConvFactory(**param)

    # 3x3
    param["kernel"] = (3, 3)
    param["num_filter"] = ch_3x3
    param["pad"] = (1, 1)
    conv3x3 = ConvFactory(**param)

    #concat
    concat = mx.symbol.Concat(*[conv1x1, conv3x3], name="concat%d" % concat_cnt)
    concat_cnt += 1
    return concat
Convolutions alone cannot handle this!
Text Convolutions

3.2.2
Convolutions on Text

- Embed words into vector space
- Nonlinear function based on inputs
- What to do about sequential structure of text?
- What about typos, omissions?

... this is a long sentence and it just keeps on going ...

Marianas Labs
Outer Convolutions

- Different length sequences lead to different size of convolutions.

A very long and sad story

Marianas Labs
• Pad with zeros

Marianas Labs
• Need to shorten sequence, e.g. max pooling. But insufficient if only pick the largest from sequence.
k-max Pooling

Intuition - pick out the most important tokens

Choose the k largest elements (length adjusted)
Convolutions on Text

Kalchbrenner, Grefenstette, Blunsom ‘14
Approximate symmetries
Approximate Symmetries

- Image rotation

- Text
  I like the flowers. Thanks. I like the flowers. I like the red flowers.

- Audio
  - Pitch and duration
  - Background noise
Features from group theory

• Averaging

\[ f(x) \rightarrow \bar{f}(x) := \sum_{g \in G} f(g \circ x) \]

Sums over all transformations due to group

\[ \bar{f}(g' \circ x) = \sum_{g \in G} f(g \circ g' \circ x) = \sum_{g'' \in G} f(g'' \circ x) = \bar{f}(x) \]

• Maximum (puzzle - approximate by averaging)

\[ f(x) \rightarrow \bar{f}(x) := \max_{g \in G} f(g \circ x) \]

Again, invariance since \( g \circ g' \in G \)
Features from group theory

- **Averaging**

  \[ f(x) \longrightarrow \bar{f}(x) := \sum_{g \in G} f(g \circ x) \]

  For loss functions

  \[ \bar{l}(x, y, f) := \sum_{g \in G} l(g \circ x, y, f) - \Delta(g) \]

- **Maximum**

  \[ f(x) \longrightarrow \bar{f}(x) := \max_{g \in G} f(g \circ x) \]

  \[ \bar{l}(x, y, f) := \max_{g \in G} l(g \circ x, y, f) - \Delta(g) \]
Virtual Observations

• Summing over entire group is dangerous (6 vs 9)

\[ \bar{l}(x, y, f) := \sum_{g \in G} l(g \circ x, y, f) - \Delta(g) \]

• Generate transformed instances (virtual SVs) only within ‘reasonable’ range

\[ \bar{l}(x, y, f) := \sum_{g \in G} p(g)l(g \circ x, y, f) - \Delta(g) \]

• E.g. Baidu speech recognizer (background noise), early Le Cun object classifier (clutter)
Backdrop for whole system training

Le Cun, Bottou, Bengio, Haffner, 2001
yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Marianas Labs
Whole system training

- Layers need not be ‘neural networks’
  - Rankers
  - Segmenters
  - Finite state automata
- Jointly train a full OCR system

Le Cun, Bottou, Bengio, Haffner, 2001
yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Marianas Labs
5. Advanced Objectives
Recall - Regression

- Real valued $y$, quality measured by $y - f(x)$

\[
l(y, f(x)) = |y - f(x)|
\]

- Vectorial data

\[
l(y, f(x)) = \frac{1}{2} (y - f(x))^2
\]

- $\epsilon$-insensitive

\[
l(y, f(x)) = \max(0, |y - f(x)|)
\]

- Huber

\[
l(y, f(x)) = \begin{cases} 
|y - f(x)| - 0.5 & \text{if } |y - f(x)| > 1 \\
0.5(y - f(x))^2 & \text{otherwise}
\end{cases}
\]
Recall - Classification

- Binary $y$, e.g. \{apples, oranges\}
- Multiple categories, e.g. \{red, green, blue\}
- Ordinal relationship, e.g. \{A, B, C, D, Fail\}

\[
l(y, f(x)) = \log \left(1 + e^{-yf(x)}\right)
\]

\[
l(y, f(x)) = \max(0, 1 - yf(x))
\]

\[
l(y, f(x, \cdot)) = \log \sum_{y'} e^{f(x, y')} - f(x, y)
\]

\[
l(y, f(x, \cdot)) = \max_{y'} [f(x, y') + \Delta(y, y')] - f(x, y)
\]
Multiclass Classification

- Multiclass classification (softmax)
  Multinomial exponential model

\[
p(y|x) = \frac{e^{f(x,y)}}{\sum_{y'} e^{f(x,y')}}
\]

\[
- \log p(y|x) = \log \sum_{y'} e^{f(x,y')} - f(x, y)
\]

- Example - words, e.g. to predict the next word.

Oh wow, I found my ...

{cat, dog, car, laptop, dinosaur, theorem ...}
Principal Component Analysis

- Regress from observation to itself
- Lower-dimensional layer is bottleneck
- Linear transfer function

\[
\sum_i\|x_i - WVx_i\|^2_2 = \text{tr} XX^\top - 2 \text{tr} XMX^\top + \text{tr} XMM^\top X^\top
\]
Principal Component Analysis

\[
\sum \| x_i - WV x_i \|^2 = \text{tr} \ XX^\top - 2 \text{tr} \ XWV X^\top + \text{tr} \ XV^\top W^\top WV X^\top \\
= \text{tr} \ Q - 2 \text{tr} \ M^\top Q + \text{tr} \ M^\top QM \\
\partial_M [\cdot] = -2Q + M^\top Q + MQ
\]

M symmetric low rank constraint

Optimal M is projection. PCA!
Autoencoder

• Regress from observation to itself ($y_n = x_1$)
• Lower-dimensional layer is bottleneck
• **Nonlinearity per layer**
• Often trained iteratively (one layer at a time)
• Extracts approximate sufficient statistic of data (reconstruct $x$ from itself)
Autoencoder

- Encode \( x \) to code \( c \)
- Decode \( c \) to data \( x \)

- Code \( c \) has all information needed about \( x \)

\[
p(y|x) = p(y|x, c(x)) \approx p(y|x(c), c)
\]

- We can synthesize more data by finding new \( c' \)
  (density model for code)
• Encode x to code c
• Decode c to data y
• Code c has all information needed about x

\[ p(y|x) = p(y|x, c(x)) \approx p(y|x(c), c) \]

• We can synthesize new data y’ by encoding new data x’ via c(x’)

Marianas Labs
‘Synesthesia’

- Different data sources
  - Images and captions
  - Natural language queries and SQL queries
  - Movies and actions
- Generative embedding for both entities
- Minimize distance between pairs
- Need to prevent clumping all together

Marianas Labs
‘Synesthesia’

- Different data sources
  - Images and captions
  - Natural language queries and SQL queries
  - Movies and actions

\[
\text{max}(0, \text{margin} + d(a, b) - d(a, n))
\]

large margin of similarity


Marianas Labs
In a Nutshell
Key Components

• **Backprop**
  Optimize whole system by propagating gradients

• **Layers**
  Adapted to data type and invariance (+ macros)

• **Nonlinearities (max pooling, k-max)**
  Select subset of attributes (algebraic IF)

• **Loss Function**
  • Multiclass, large margin, regression
  • Autoencoder, translator
Outline

• Basics
  • Perceptron (and convergence rule)
  • Stochastic gradient descent
  • Loss functions and objectives
• Deep Networks
  • Layers and Invariances
  • Advanced objectives
• State and Structure
  • Optimization
  • Autoregressive Models & Hidden State
  • Toolkits

Marianas Labs
6. Optimization
Recall - Stochastic Gradient Descent

- Update parameters according to
  \[ w_{t+1} \leftarrow w_t - \eta_t \nabla f(x_i) l(y_i, f(x_i)) \partial_w f(x_i) \]
- Rate of decay for learning rate
- Adjust rate for each layer
- Adjust each parameter individually
- Minibatch size, momentum terms
- Lots of things that can (should) be adjusted (Bayesian optimization, e.g. Spearmint, MOE)

Senior, Heigold, Ranzato and Yang, 2013
Minibatch

- Aggregate gradients before applying them
  Small number of instances \((x_i, y_i) \ldots (x_{i+b}, y_{i+b})\)
- Reduces variance in gradients
- Better for vectorization (GPUs)
  vector, vector < vector, matrix < matrix, matrix
  \(\langle x, x' \rangle < Mx < MX\)
- Large minibatch may need large memory
  (and slow updates).
- Magic numbers are 64 to 256 on GPUs

Senior, Heigold, Ranzato and Yang, 2013
Learning rate decay

- **Constant**
  (requires schedule for piecewise constant, tricky)

- **Useful hack**
  Constant until no more improvement on validation set

- **Polynomial decay**

  \[ \eta(t) = \frac{\alpha}{(\beta + t)^\gamma} \]

  Recall exponent of 0.5 for conventional SGD, 1 for strong convexity. Bottou picks 0.75

- **Exponential decay**

  \[ \eta(t) = \alpha e^{-\beta t} \]

  not recommended since quite aggressive
AdaGrad

- **Adaptive learning rate** (preconditioner)
  \[ \eta_{ij}(t) = \frac{\eta_0}{\sqrt{K + \sum_t g_{ij}^2(t)}} \]

- For directions with large gradient, decrease learning rate aggressively to avoid instability
- If gradients start vanishing, learning rate decrease reduces, too
- **Local variant**
  \[ \eta_{ij}(t) = \frac{\eta_t}{\sqrt{K + \sum_{t'=t-\tau}^t g_{ij}^2(t')}} \]

Duchi, Hazan, Singer, 2010
http://www.magicbroom.info/Papers/DuchiHaSi10.pdf
Momentum

- Average over recent gradients
- Helps with local minima
- Flat (noisy) gradients

\[ m_t = (1 - \lambda)m_{t-1} + \lambda g_t \]
\[ w_t \leftarrow w_t - \eta_t g_t - \tilde{\eta}_t m_t \]

- Can lead to oscillations for large momentum
- Nesterov’s accelerated gradient

\[ m_{t+1} = \mu m_t + \epsilon g(w_t - \mu m_t) \]
\[ w_{t+1} = w_t - m_{t+1} \]
Capacity control

• Minimizing loss can lead to overfitting
• Weight decay
  \[ w_t \leftarrow w_t - \eta_t g_t \]
  \[ w_t \leftarrow (1 - \lambda)w_t - \eta_t g_t \]
• Parameter clipping
• Overheated GPU
• Numerical instabilities
• Hacky but necessary (e.g. deep architectures)
Dropout

- **Avoid parameter sensitivity**  
  (small changes in value shouldn’t change result)
- **Distributed representation**  
  (information carried by more than 1 dimension)
- **Randomized sparsification**

\[ y_{ti} = \xi_{ti} y_{ti} \quad \text{where} \quad \begin{cases} 
  \Pr(\xi_{ti} = \pi^{-1}) &= \pi \\
  \Pr(\xi_{ti} = 0) &= 1 - \pi 
\end{cases} \]

- **Same trick works for matrix W, too.**  
  DropConnect gives slightly better performance.

http://cs.nyu.edu/~wanli/dropc/
Srivastava, Hinton, Krizhevski, Sutskever, Salakhutdinov
http://jmlr.org/papers/v15/srivastava14a.html
Dropout & DropConnect

Regular

Dropout

DropConnect

Marianas Labs

Carnegie Mellon University
Training with Dropout

Without dropout

With dropout
7. Memory & State
Marianas Labs

Latent State Models

Autoregressive

Latent State Models

LRR Receptor-like Kinase
TIR-NBS-LRR Disease resistance
Retrotransposon associated
Other
STS
Prediction

- stock price
- gene finding
- speech recognition
- activity segmentation
- named entities
State and models

• IID data
  • Classification
  • Regression
  • Feature representation …

• Most of the data isn’t IID
  • Sequence annotation (tagging, parsing)
  • Sequence generation (translation)
  • Summarization
  • Image annotation (content extraction)

• Alternatives: dynamic programs / stepwise prediction
Autoregressive Models / RNN

• Time series of observations
  \[ \ldots X_{t-2}, X_{t-1}, X_t, X_{t+1}, X_{t+2} \ldots \]

• Estimate \( x_{t+1} = f(x_t, \ldots, x_{t-\tau}) \) e.g. via deep net

bull or bear market?
Autoregressive Models

- Time series of observations
  \[ \ldots \ X_{t-2}, X_{t-1}, X_t, X_{t+1}, X_{t+2} \ldots \]
- Estimate \( x_{t+1} = f(x_t, \ldots, x_{t-\tau}) \) e.g. via deep net
- Problem
  - Hard to encode latent state (e.g. parity)
  - Hard to encode long range context/knowledge
- Solution - latent state models
  \[
  x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau}) \\
  z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})
  \]
  also called Recurrent Neural Networks
Autoregressive Models / RNN

\[ x_{t+1} = f(x_t, \ldots, x_{t-\tau}) \]

\[ z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau}) \]

\[ z_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau}) \]
Autoregressive Models / RNN

- Sequence of observations
- Gradients need to propagate back through $t$
- Gradient may vanish. Makes model difficult to train. Due to stability condition on gradients

$$x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$
$$z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})$$
• Gradients need to propagate back through time

\[
x_{t+1} = f(x_t, \ldots, x_{t-\tau})
\]

\[
= f(f(x_{t-1}, \ldots, x_{t-\tau-1}), x_{t-1}, \ldots, x_{t-\tau}) = \ldots
\]

\[
\partial_{x_{t-a}} x_t = f'_t \cdot f'_{t-2} \cdot \text{gradients may vanish / explode}
\]

\[
x_{t+1} = f(x_t, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})
\]

\[
z_{t+1} = g(x_{t+1}, \ldots, x_{t-\tau}, z_t, \ldots, z_{t-\tau})
\]
Long Short Term Memory
LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update (like FlipFlop)
  Hochreiter & Schmidhuber, 1998
LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

\[
i_t = \sigma(W_i(x_t, h_{t-1}) + b_i) \\
f_t = \sigma(W_f(x_t, h_{t-1}) + b_f) \\
z_t = f_t * z_{t-1} + i_t * \tanh(W_z(x_t, h_{t-1}) + b_z) \\
o_t = \sigma(W_o(x_t, h_{t-1}, z_t) + b_f) \\
h_t = o_t * \tanh z_t
\]
LSTM (Long Short Term Memory)

- Sequence of observations
  - Latent state has custom update semantics

\[
\begin{align*}
  i_t &= \sigma(W_i(x_t, h_{t-1}) + b_i) \\
  f_t &= \sigma(W_f(x_t, h_{t-1}) + b_f) \\
  z_t &= f_t \cdot z_{t-1} + i_t \cdot \tanh(W_z(x_t, h_{t-1}) + b_z) \\
  o_t &= \sigma(W_o(x_t, h_{t-1}, z_t) + b_f) \\
  h_t &= o_t \cdot \tanh(z_t)
\end{align*}
\]
LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

\[
\begin{align*}
    i_t &= \sigma(W_i(x_t, h_{t-1}) + b_i) \\
    f_t &= \sigma(W_f(x_t, h_{t-1}) + b_f) \\
    z_t &= f_t \cdot z_{t-1} + i_t \cdot \tanh(W_z(x_t, h_{t-1}) + b_z) \\
    o_t &= \sigma(W_o(x_t, h_{t-1}, z_t) + b_f) \\
    h_t &= o_t \cdot \tanh(z_t)
\end{align*}
\]
LSTM (Long Short Term Memory)

- Sequence of observations
  Latent state has custom update semantics

**sequence generation**

**sequence classification**
LSTM (Long Short Term Memory)

- Group LSTM cells into layer
- Multiple layers
- Can model different scales of dynamics
“Synesthesia”

- Sequence embedding via LSTM
- Enforce closeness between LSTM state to obtain similarity between sequences
Much more

- Memory is area of active research
  - Neural Turing Machine
    http://arxiv.org/abs/1410.5401
  - Memory Networks
    http://arxiv.org/abs/1410.3916
  - Attention models (Kyunghyun Cho)
8. Tools
Quick overview

• Caffe http://caffe.berkeleyvision.org/
  Efficient for convolutional models / images

• Torch http://torch.ch/
  Very efficient. But you must LIKE Lua …
  Google and Facebook love it

• Theano http://deeplearning.net/software/theano/
  Compiled from Python. Not as efficient as Torch

• Minerva https://github.com/dmlc/minerva
  Compiler layout of execution on machines

• CXXNet https://github.com/dmlc/cxxnet
  Simpler than Caffe. More efficient

• Parameter Server bindings to https://github.com/dmlc/
  Minerva, Caffe, CXXNet, …