Reminder: Linear Classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

\[
 activation_{\theta}(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)
\]

- If the activation is:
  - Positive, output +1
  - Negative, output -1

Canonical Optimization

- Optimization
  - i.e., how do we solve:

\[
 \max_w l(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)
\]

Hill Climbing

- Recall from CSPs lecture: simple, general idea
  - Start wherever
  - Repeat: move to the best neighboring state
  - If no neighbors better than current, quit

- What’s particularly tricky when hill-climbing for multiclass logistic regression?
  - Optimization over a continuous space
    - Infinitely many neighbors!
  - How to do this efficiently?

Gradient Ascent

- Perform update in uphill direction for each coordinate
- The steeper the slope (i.e., the higher the derivative) the bigger the step for that coordinate
- E.g., consider: \( g(w_1, w_2) \)

  - Updates: 
    \[
    \begin{align*}
    w_1 &\leftarrow w_1 + \alpha \cdot \frac{\partial g}{\partial w_1}(w_1, w_2) \\
    w_2 &\leftarrow w_2 + \alpha \cdot \frac{\partial g}{\partial w_2}(w_1, w_2)
    \end{align*}
    \]
  - Updates in vector notation:
    \[
    w \leftarrow w + \alpha \cdot \nabla_w g(w)
    \]

  with: \( \nabla_w g(w) = \nabla x g(x) = \) gradient

Gradient Ascent

- Idea:
  - Start somewhere
  - Repeat: Take a step in the gradient direction

Figure source: Mathworks
Mini-Batch Gradient Ascent on the Log Likelihood Objective

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

**Observation:** gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

* init $w$
* for iter = 1, 2, ...
  * pick random subset of training examples $J$

$$w \leftarrow w + \alpha \sum_{j \in J} \nabla \log P(y^{(j)} | x^{(j)}; w)$$

Deep Neural Network = Also learn the features!

$$P(z^{(1)} | x) = \frac{1}{1 + e^{-x}}$$

$$P(z^{(2)} | x) = \frac{1}{1 + e^{-z^{(1)}}}$$

$$P(z^{(3)} | x) = \frac{1}{1 + e^{-z^{(2)}}}$$

Common Activation Functions

<table>
<thead>
<tr>
<th>Sigmoid Function</th>
<th>Hyperbolic Tangent</th>
<th>Rectified Linear Unit (ReLU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g(x) = \frac{1}{1 + e^{-x}}$</td>
<td>$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$</td>
<td>$g(x) = \max(0, x)$</td>
</tr>
<tr>
<td>$g'(x) = g(x)(1 - g(x))$</td>
<td>$g'(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$</td>
<td>$g'(x) = 1, \quad x &gt; 0$</td>
</tr>
</tbody>
</table>

Deep Neural Network: Also Learn the Features!

- Training the deep neural network is just like logistic regression:

$$\max_w ll(w) = \max_w \sum_i \log P(y^{(i)} | x^{(i)}; w)$$

just $w$ tends to be a much, much larger vector 😏

- just run gradient ascent
- + stop when log likelihood of hold-out data starts to decrease
Neural Networks Properties

- **Theorem (Universal Function Approximators).** A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

- **Practical considerations**
  - Can be seen as learning the features
  - Large number of neurons
    - Danger for overfitting
    - (hence early stopping!)

Universal Function Approximation Theorem*

- In words: Given any continuous function f(x), if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate f(x).

Universal Function Approximation Theorem*

- How about computing all the derivatives?
  - Derivatives tables:

Fun Neural Net Demo Site

- Demo-site:
  - http://playground.tensorflow.org/

How about computing all the derivatives?

- But neural net f is never one of those?
  - No problem: CHAIN RULE:
    
    \[
    \begin{align*}
    f(x) &= g(h(x)) \\
    f'(x) &= g'(h(x))h'(x)
    \end{align*}
    \]

  ➔ Derivatives can be computed by following well-defined procedures
Automatic Differentiation

- Automatic differentiation software
  - e.g. Theano, TensorFlow, PyTorch, Chainer
  - Only need to program the function $g(x,y,w)$
  - Can automatically compute all derivatives w.r.t. all entries in $w$
  - This is typically done by caching info during forward computation pass of $f$, and then doing a backward pass = “backpropagation”
  - Autodiff / Backpropagation can often be done at computational cost comparable to the forward pass
  - Need to know this exists
  - How this is done? -- outside of scope of CS188

Summary of Key Ideas

- Optimize probability of label given input
  \[ \max_w \log P(y_i | x_i) ; w \]

- Continuous optimization
  - Gradient ascent:
    - Compute steepest uphill direction = gradient (= just vector of partial derivatives)
    - Take step in the gradient direction
  - Keep train/test data accuracy close to drop = “early stopping”

- Deep neural nets
  - Last layer = still logistic regression
  - Now also many more layers before this last layer
  - Computing the features = not by hand but learned
  - Universal function approximation theorem
    - If neural net is large enough
      - Then neural net can represent any continuous mapping from input to output with arbitrary accuracy
    - But remember: need to avoid overfitting / memorizing the training data = “early stopping”
    - Automatic differentiation gives the derivatives efficiently (how? = outside of scope of CS188)

How well does it work?

Computer Vision

Object Detection

Manual Feature Design
Features and Generalization

Image

HoG

Performance

ImageNet Error Rate 2010-2014

Performance

ImageNet Error Rate 2010-2014

Performance

ImageNet Error Rate 2010-2014

Performance

ImageNet Error Rate 2010-2014
Performance

ImageNet Error Rate 2010-2014

MS COCO Image Captioning Challenge

Karpathy & Fei-Fei, 2015; Donahue et al., 2015; Xu et al., 2015; many more

Visual QA Challenge

Stanislaw Antol, Aditya Agrawal, Aishwarya Ailawad, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, Devi Parikh

Speech Recognition

TIMIT Speech Recognition

Machine Translation

Google neural machine translation (in production)

What’s still missing? – correlation ≠ causation

[Ribeiro et al.]

Table 1: Raw data and reproduction of a bad model's prediction in the “Sharks vs. Whales” task.

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Mislabeled</th>
<th>Guess</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharks</td>
<td>50 out of 25</td>
<td>2 out of 25</td>
<td>25 out of 25</td>
</tr>
<tr>
<td>Whales</td>
<td>25 out of 25</td>
<td>25 out of 25</td>
<td>25 out of 25</td>
</tr>
</tbody>
</table>
What’s still missing? – covariate shift

[Carroll et al.]

37

What’s still missing? – covariate shift

[Carroll et al.]

38