Backpropagation

CS 510
Lecture #20
March 23\textsuperscript{th}, 2018
Backpropagation

- Backpropagation is the algorithm that describes how we update weights in a network, given
  - Training samples
  - Training labels
  - A cost function
- It's used for (almost) all networks
- Network nodes may be
  - Non-linear perceptrons (the most common)
  - Convolutional units
  - Pooling units
  - Batch normalization units
  - ...
Goals

• Walk you through the math of backpropagation
  – Complicated, but just calculus
  – Almost universal: modifiable for different node types (see previous slide)

• Today’s derivation assumes multi-layer perceptrons
  – $z(x) = wx + b$
  – $a(x) = \sigma(z(x)) = \sigma(wx + b)$

• Remember the chain rule from calculus:
  – $f(x) = g(h(x)) \rightarrow f'(x) = g'(h(x))h'(x)$
Simple Neural Network

Layer 1

Layer 2

Layer 3

Notation: superscripts are layers, subscripts are node numbers
Adding a Cost Function

Layer 1  Layer 2  Layer 3  Cost

N^1_1  N^2_1  N^3_1  Y_1
N^1_2  N^2_2  N^3_2  Y_2
N^1_3  N^2_1  N^3_1
N^1_4
Cost Functions

• Cost functions measure the gap between the network output and the ideal output

• Two necessary properties
  1. An average over samples: \( C = \frac{1}{n} \sum_x C_x \)
  2. Function of output activations: \( C = C(a_l) \)

• Example: mean squared error
  • \( C = \frac{1}{2n} \sum_x \| y(x) - a^L(x) \|^2 \)

Allows us to optimize per sample

Allows us to initialize the partial derivative computations
δs : local derivatives as error measures

\[ \Delta C = \frac{\partial C}{\partial z_j^l} \Delta z_j^l \]
Partial derivatives as error measures

• Imagine you want to change the output $z^l_j$, by $\Delta z^l_j$

• Then $\Delta C = \frac{\partial C}{\partial z^l_j} \Delta z^l_j$

• If $\left| \frac{\partial C}{\partial z^l_j} \right|$ is large, then C becomes smaller by giving $\Delta z^l_j$ the opposite sign

• But if $\left| \frac{\partial C}{\partial z^l_j} \right|$ is near zero, then $\Delta z^l_j$ doesn’t matter.
  
  – $\frac{\partial C}{\partial z^l_j}$ is already optimal!

  – $\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$
So where are we?

- We can optimize on a per-sample basis
  - Because the cost function is an average
- Minimizing the $\delta$s optimizes the net
  - The $\delta$s depend on the data samples
- But how do we minimize the $\delta$s?

- We will assume that nodes have non-linear functions, so $a_j^l = \sigma(z_j^l)$
Output Layer

- \( \delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \) by the chain rule

- \( \frac{\partial C}{\partial a_j^L} \) is the partial derivative of C with respect to the activation of output unit j
  - If C is LMS (slide #6)
    - \( \frac{\partial C}{\partial a_j^L} = a_j^L(x) - y(x) \)
    - The difference between the output & desired output
      - Explains the \( \frac{1}{2} \) in LMS
Output Layer (cont.)

- $\sigma'(z_j^L)$ is the derivative of the non-linear transfer function at $z_j^L$
- If $\sigma(x) = \tanh(x)$, $\sigma'(x) = 1 - \tanh^2(x)$
- If $\sigma(x) = (1 + e^{-x})^{-1}$,
  \[
  \sigma'(x) = \sigma(x)(1 - \sigma(x))
  \]
- $\delta_j^L = \left( a_j^L(x) - y(x) \right) \left( 1 - \tanh^2 \left( z_j^L(x) \right) \right)$ or
- $\delta_j^L = \left( a_j^L(x) - y(x) \right) \left( a_j^L(x) \left( 1 - a_j^L(x) \right) \right)$
\( \delta^L \) given \( \delta^{L+1} \)

- \( \delta^l_j = \sigma'(z^l_j) \sum_k w^l_{kj} \delta^{l+1}_k \)
- \( \sigma' \) is computed as on previous slide
- The RHS is just the sum of the impacts
- This is where \textit{backpropagation} comes from
  - Calculate \( \delta \)'s for output layer
  - Then recursively compute \( \delta \)'s for previous layers
Computing $\delta s$...

Layer 1

Layer 2

Layer 3

Cost

Compute intermediate $\delta s$ (slide #12)

Compute output layer $\delta s$ (slide #11)
So...

• Given an input $x$ and output $y$:
  – We can compute $\delta^l_j$ for every node $j$ at every level $l$
  – Minimizing the $\delta$s will optimize the network
    • Relative to this sample
  – So we need to adjust the weights $w_i$ and $b$ to reduce the $\delta$s
    • But just a little for each input/output pair
    • So we can optimize across all samples
Adjusting b

• Remember that $\delta_j^l \equiv \frac{\partial c}{\partial z_j^l}$ (slide #8)

• And that $z_j^l = w_j^l x + b$

• So $\frac{\partial c}{\partial b_j^l} = \delta_j^l$

• So $b_j^l \leftarrow (1 - \alpha)b_j^l - \alpha \delta_j^l$
  
  – Where $\alpha$ is a learning rate
  
  – Regulates how much you react to each sample
Adjusting w’s

• \( \frac{\partial c}{\partial w_{jk}^l} = a_{k}^{l-1} \delta_{j}^{l} \)

• So \( w_{jk}^l \leftarrow (1 - \alpha)w_{jk}^l - \alpha a_{k}^{l-1} \delta_{j}^{l} \)
  
  – Where \( \alpha \) is the same learning rate as before
  
  – We are collectively minimizing the deltas by heading downhill in the \( k+1 \) dimensional space defined by \( w \) & \( b \)
Backpropagation (redux)

• Backpropagation updates weights in a network, given
  – Training samples
  – Training labels
  – An cost function

• Network nodes may be
  – Non-linear perceptrons (the most common)
  – Convolutional units
  – Pooling units
  – Batch normalization units
  – …