Week 3: MLPs and Backprop

NYU DS-GA 3001

Slides by Sam Bowman
Based on lecture notes by Kyunghyun Cho

bowman@nyu.edu

9/20/16

Overview

1. Introductions & Administrativia
2. Multilayer Perceptron
   - Example: Binary classification with a single hidden unit
   - Example: Binary classification with more than one hidden unit
3. Automating Backpropagation
   - What if a Function is *not* Differentiable?
4. Next time

Introductions & Administrativia

- You
- Me (hi again!)
- Changes from the original syllabus. So far:
  - MT (pending)
  - Topics
  - Slides
- Questions?

Overview

1. Introductions & Administrativia
2. Multilayer Perceptron
   - Example: Binary classification with a single hidden unit
   - Example: Binary classification with more than one hidden unit
3. Automating Backpropagation
   - What if a Function is *not* Differentiable?
4. Next time

Multilayer Perceptron

- The *classic* deep artificial neural network (ca. 1962).
- Structure:
  - Input $\mathbf{x}$
  - ...feeds a sequence of one or more hidden layers

$$\phi_0(\mathbf{x}) = g(\mathbf{W}_0\mathbf{x} + \mathbf{b}_0),$$

    Where $g \in$

    - Sigmoid: $\sigma(x) = \frac{1}{1+\exp(-x)}$
    - Hyperbolic function: $\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$
    - Rectified linear unit: $\text{rect}(x) = \max(0, x)$
  - ...which feed an output layer (usually softmax)

Simplest possible MLP: Structure

- Structure:
  - Scalar input $x \in \mathbb{R}$
  - ...feeds one hidden layer

$$\phi(x) = \sigma(ux + c)$$

  - ...which feeds a sigmoid output layer representing a (Bernoulli) conditional probability $p(y|x)$ over a Boolean output variable y.

$$\mu = f(x) = \sigma(w\phi(x) + b)$$

- ...where

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

- Four scalar parameters: u, c, w, b

Simplest possible MLP: Cost Function

- Objective: Minimize KL divergence between
  - true (empirical) conditional distribution $p(y|x)$
  - predicted conditional distribution $\hat{p}(y|x)$

$$\begin{aligned}
\mathrm{KL}(p \| \hat{p}) &= \sum_{y \in \{0,1\}} p(y|x) \log \frac{p(y|x)}{\hat{p}(y|x)} \\
&= \sum_{y \in \{0,1\}} p(y|x) \log p(y|x) - p(y|x) \log \hat{p}(y|x).
\end{aligned}$$

- Equivalent to minimizing:

$$-\sum_{y \in \{0,1\}} p(y|x) \log \hat{p}(y|x).$$

Simplest possible MLP: Cost Function

- When we compute gradients for gradient descent, we'll be computing the *per sample* version of this cost function and averaging

$$\begin{aligned} C_x &= -\log \hat{p}(y|x) \\ &= -\log \mu^y (1-\mu)^{1-y} \\ &= -y \log \mu - (1-y)\log(1-\mu) \end{aligned}$$

- Our task: Define the gradient for each of our four parameters wrt. C_x: $\frac{\partial C_x}{\partial u}, \frac{\partial C_x}{\partial c}, \frac{\partial C_x}{\partial w}, \frac{\partial C_x}{\partial b}$

Simplest possible MLP: Gradient

- Let's start with w and use the chain rule:

$$\frac{\partial C_x}{\partial w} = \frac{\partial C_x}{\partial \mu} \frac{\partial \mu}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial w}$$

($\underline{\mu} = w\phi(x) + b$, which is the input to f)

- Starting from the left...

$$\frac{\partial C_x}{\partial \mu} \underbrace{\frac{\partial \mu}{\partial \underline{\mu}}}_{=\mu'} = -\frac{y}{\mu}\mu' + \frac{1-y}{1-\mu}\mu'$$

$$= \frac{-y + y\mu + \mu - y\mu}{\mu(1-\mu)}\mu' = \frac{\mu - y}{\mu(1-\mu)}\mu' = \mu - y$$

(the derivative of $\sigma(\mu)$ is $\mu(1-\mu)$.)

Simplest possible MLP: Gradient

- Continuing with $\frac{\partial C_x}{\partial w}$, all we need is:

$$\frac{\partial \underline{\mu}}{\partial w} = \phi(x)$$

- So:

$$\frac{\partial C_x}{\partial w} = \frac{\partial C_x}{\partial \mu} \frac{\partial \mu}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial w} = (\mu - y)\phi(x)$$

Simplest possible MLP: Gradient

- Now let's try b and use the chain rule:

$$\frac{\partial C_x}{\partial b} = \frac{\partial C_x}{\partial \mu}\frac{\partial \mu}{\partial \underline{\mu}}\frac{\partial \underline{\mu}}{\partial b}$$

- Starting from the left, we need...

$$\frac{\partial C_x}{\partial \mu}\underbrace{\frac{\partial \mu}{\partial \underline{\mu}}}_{=\mu'} = -\frac{y}{\mu}\mu' + \frac{1-y}{1-\mu}\mu'$$

$$= \frac{-y + y\mu + \mu - y\mu}{\mu(1-\mu)}\mu' = \frac{\mu - y}{\mu(1-\mu)}\mu' = \mu - y$$

Simplest possible MLP: Gradient

- Now let's try b and use the chain rule:

$$\frac{\partial C_x}{\partial b} = \frac{\partial C_x}{\partial \mu}\frac{\partial \mu}{\partial \underline{\mu}}\frac{\partial \underline{\mu}}{\partial b}$$

- Starting from the left, we need...

$$\frac{\partial C_x}{\partial \mu}\underbrace{\frac{\partial \mu}{\partial \underline{\mu}}}_{=\mu'} = -\frac{y}{\mu}\mu' + \frac{1-y}{1-\mu}\mu'$$

$$= \frac{-y + y\mu + \mu - y\mu}{\mu(1-\mu)}\mu' = \frac{\mu - y}{\mu(1-\mu)}\mu' = \mu - y$$

- But we already computed that! [Foreshadowing.]

Simplest possible MLP: Gradient

- On to u and c. Let's use the chain rule again...

$$\frac{\partial C_x}{\partial u} = \frac{\partial C_x}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial \phi} \frac{\partial \phi}{\partial \underline{\phi}} \frac{\partial \underline{\phi}}{\partial u}$$

$$\frac{\partial C_x}{\partial c} = \frac{\partial C_x}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial \phi} \frac{\partial \phi}{\partial \underline{\phi}} \frac{\partial \underline{\phi}}{\partial c}$$

Simplest possible MLP: Gradient

- Continuing with what we still need...

$$\frac{\partial \mu}{\partial \phi} \underbrace{\frac{\partial \phi}{\partial \phi}}_{=\phi'} = w\phi' = w\phi(x)(1-\phi(x))$$

$$\frac{\partial \phi}{\partial u} = x$$

$$\frac{\partial \phi}{\partial c} = 1$$

- And we're done!

$$\frac{\partial C_x}{\partial u} = (\mu - y)w\phi(x)(1-\phi(x))x$$

$$\frac{\partial C_x}{\partial c} = (\mu - y)w\phi(x)(1-\phi(x))$$

A more realistic MLP: Structure

- Structure:
  - Vector input $x \in \mathbb{R}^d$
  - ...feeds one vector-valued hidden layer

    $$\phi(x) = \sigma(Ux + c)$$

    where $U \in \mathbb{R}^{l \times d}$ and $c \in \mathbb{R}^l$
  - ...which feeds a sigmoid output layer representing a (Bernoulli) conditional probability $p(y|x)$ over a Boolean output variable y.

    $$\mu = f(x) = \sigma(w\phi(x) + b)$$

    where $w \in \mathbb{R}^l$ and $b \in \mathbb{R}$
- Four parameters: U, c, w, b
- In all cases we'll be looking at, the nonlinearity (here σ) is applied elementwise.

A more realistic MLP: Gradient

- Keeping the cost as before, let's start with W and use the chain rule:

$$\frac{\partial C_x}{\partial w} = \frac{\partial C_x}{\partial \mu} \frac{\partial \mu}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial w}$$

- w is now a vector, so the function computing $\underline{\mu}$ is now more complex

$$\underline{\mu} = w^\top \phi(x) + b = \sum_{i=1}^{I} w_i \phi_i(x) + b.$$

- Let's start by thinking of $\frac{\partial \underline{\mu}}{\partial w}$ as a vector of scalar partial derivatives

$$\frac{\partial \underline{\mu}}{\partial w} = \left[\frac{\partial \underline{\mu}}{\partial w_1}, \frac{\partial \underline{\mu}}{\partial w_2}, \ldots, \frac{\partial \underline{\mu}}{\partial w_I}\right]^\top = [\phi_1(x), \phi_2(x), \ldots, \phi_I(x)]^\top = \phi(x)$$

A more realistic MLP: Gradient

- So, as before,

$$\frac{\partial C_x}{\partial w} = \frac{\partial C_x}{\partial \mu}\frac{\partial \mu}{\partial \underline{\mu}}\frac{\partial \underline{\mu}}{\partial w}$$
$$= (\mu - y)\phi(x)$$

- Similarly,

$$\frac{\partial C_x}{\partial b} = \frac{\partial C_x}{\partial \mu}\frac{\partial \mu}{\partial \underline{\mu}}\frac{\partial \underline{\mu}}{\partial b}$$
$$= \mu - y$$

A more realistic MLP: Gradient

- Now back to u and c. Let's use the chain rule again...

$$\frac{\partial C_x}{\partial U} = \frac{\partial C_x}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial \phi} \frac{\partial \phi}{\partial \underline{\phi}} \frac{\partial \underline{\phi}}{\partial U}$$

$$\frac{\partial C_x}{\partial c} = \frac{\partial C_x}{\partial \underline{\mu}} \frac{\partial \underline{\mu}}{\partial \phi} \frac{\partial \phi}{\partial \underline{\phi}} \frac{\partial \underline{\phi}}{\partial c}$$

- We can use the same vector trick as just before (taking advantage of some symmetry) to show

$$\frac{\partial \underline{\mu}}{\partial \phi} = w$$

A more realistic MLP: Gradient

- Because σ is elementwise, we can simply compute its derivative for each element in $\phi(x)$

$$\frac{\partial \phi}{\partial \underline{\phi}} = \text{diag}\left(\left[\phi_1'(x), \phi_2'(x), \ldots, \phi_l'(x)\right]^\top\right)$$

  where diag returns a diagonal matrix of the input vector

- This gets us

$$\frac{\partial C_x}{\partial \underline{\phi}} = (\mu - y)w^\top \phi'(x) = (\mu - y)(w \odot \text{diag}(\phi'(x))),$$

A more realistic MLP: Gradient

- On to the last step...

$$\frac{\partial \phi}{\partial U} = \frac{\partial U^\top x}{\partial U} = x$$

- So...

$$\frac{\partial C_x}{\partial U} = (\mu - y)(w \odot \mathrm{diag}(\phi'(x)))x^\top \tag{1}$$

- And we can easily generalize to c:

$$\frac{\partial C_x}{\partial U} = (\mu - y)(w \odot \mathrm{diag}(\phi'(x))) \tag{2}$$

- Now all we have to do is hope SGD converges! Yay?

Visualizing Backpropagation

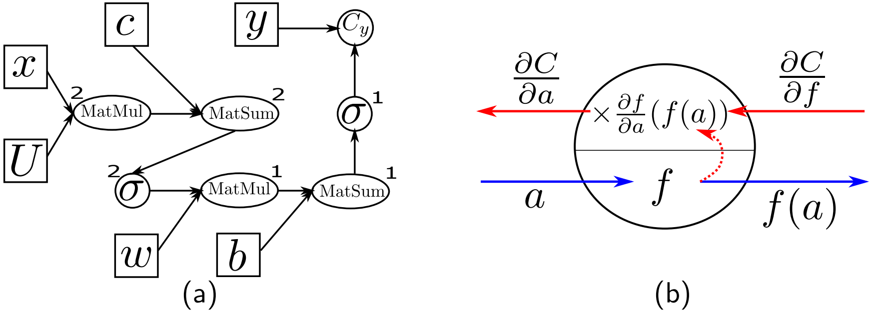
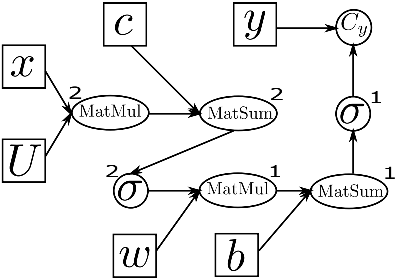


Figure: (a) A graphical representation of the computational graph of the example network. (b) A graphical illustration of a function node ($\rightarrow$: forward pass, $\leftarrow$: backward pass.)

Visualizing Backpropagation



- If you can compute the local gradients for the function nodes, backprop is easy (and easy to automate).
- Inventory of common function nodes is pretty small (dozens), easy to find good library code w/ gradients.
- So, use Tensor-Flow/Theano/Torch/etc. and never apply the chain rule yourself again!

Summing up: Backpropagation

- Nothing more than clever application of the chain rule
- If we compute the gradients we need in the right order (moving backwards through the *computation graph*), each one will:
  - Require only simple functions of local variables *and/or* partial gradients we've already computed
  - Therefore take only about as long as the corresponding computation in the forward pass
- So gradient computation takes about as long as forward computation!

Bonus issue: Gradients as graphs

- Why not represent gradient computation as a second computation graph derived from the first?
- Lets us compute gradients of (gradients of (gradients of)) gradients, visualize gradients with the same tools we use for the rest of the graph, etc.
- Implemented in Torch, Theano

Differentiability

- Backpropagation requires that every function node is differentiable wrt. its inputs and parameters
- All the standard NN building blocks are:
  - Matrix multiplication (normal or elementwise), addition: See above and/or Matrix Cookbook
  - Sigmoid nonlinearity: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
  - tanh nonlinearity: $\tanh'(x) = \left(\frac{2}{\exp(x)+\exp(-x)}\right)^2$

Differentiability

- ReLU is a bit more complicated:

$$\text{rect}'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Exact zero values for x are rare enough to not matter (can stipulatively change $<$ to $\leq$)
- Exact zero values for rect$x$ are common, and block gradients from flowing backwards...
- ...but only for some elements in some examples, so it learns anyway

Differentiability

- What if you want to incorporate a function that just isn't differentiable?
- This includes most functions you might dream up! Examples:
  - Sampling from a multinomial distribution $P(m|x)$
  - Looking up the $int(x)$-th row of a matrix
  - Identifying the argmax index into x
- You can compute gradients and use SGD, but you'll generally have to resort to slower and less reliable techniques (mostly: reinforcement learning). Stay tuned 'til the end of the course.

- Lab tomorrow
- Reading:
- Lecture: RNNs and word vectors!
- HW 1: Due in eight days