Lecture Notes: 600.475 Neural Networks

Matt Gormley

7 October 2014

Contents

1 Backpropagation **1**
- 1.1 Defining Computational Circuits . . . 1
- 1.2 Chain Rule . . . 2
- 1.3 Small Example . . . 2

2 Neural Network Training **2**
- 2.1 Loss Functions . . . 3
- 2.2 Binary Logistic Regression . . . 4
- 2.3 2-Layer Neural Network . . . 4
- 2.4 Numerical Differentiation . . . 5

3 For Presentation **5**

1 Backpropagation

The backpropagation algorithm is a general method for computing the gradient of a neural network. Here we generalize the concept of a neural network to include any computational circuit. Applying the backpropagation algorithm on these circuits amounts to repeated application of the chain rule.

This general algorithm goes under many other names: automatic differentiation (AD) in the reverse mode, analytic differentiation, module-based AD, autodiff, etc.

Below we define a forward pass which computes the output bottom-up, and a backward pass which computes the derivatives of all intermediate quantities top-down.

1.1 Defining Computational Circuits

The graphical representation of a neural network leaves much to be desired. What non-linear function is used? What does it mean to connect one variable to another? How can we make this more precise?

Computational circuits provide a clearer visual representation of the computation and help provide a clearer intuition for how to apply backpropagation.

1.2 Chain Rule

At the core of the backpropagation algorithm is the chain rule. The chain rule in its familiar form allows us to differentiate a function f defined as a the composition of two functions g and h such that $f = (g \circ h)$.

If the inputs and outputs of f, g, and h are all scalars, then we obtain the familiar form of the chain rule:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{1}$$

or equivalently,

$$\begin{aligned} f'(x) &= (g \circ h)'(x) && (2) \\ &= g'(h(x))h'(x) && (3) \end{aligned}$$

Suppose instead the inputs and outputs of the functions are vector-valued variables, $f : \mathbb{R}^K \to \mathbb{R}^I$. Let $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$ where $\boldsymbol{x} = \{x_1, x_2, \ldots, x_K\}$, $\boldsymbol{y} = \{y_1, y_2, \ldots, y_I\}$, and $\boldsymbol{u} = \{u_1, u_2, \ldots, u_J\}$. Then the chain rule must sum over all the intermediate quantities.

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k \tag{4}$$

1.3 Small Example

This section demonstrates automatic differentiation as applied to a simple example of only one input variable.

The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward	Backward	
$J = cos(u)$	$\frac{dJ}{du} \mathrel{+}= -sin(u)$	
$u = u_1 + u_2$	$\frac{dJ}{du_1} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_1}, \quad \frac{du}{du_1} = 1$	$\frac{dJ}{du_2} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_2}, \quad \frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_1}\frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$	
$u_2 = 3t$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_2}\frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$	
$t = x^2$	$\frac{dJ}{dx} \mathrel{+}= \frac{dJ}{dt}\frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$	

2 Neural Network Training

Choose each of the following:

- **Loss Function:** $J = \ell(\hat{\boldsymbol{y}}, \boldsymbol{y}^*) \in \mathbb{R}$
- **Neural Network:** $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$

Training Find parameters $\boldsymbol{\theta}$ that minimize the objective function over the entire training set $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$: Define the loss / error as function of the parameters for each training instance i.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} E_i(\boldsymbol{\theta})$$
$$\text{where } E_i(\boldsymbol{\theta}) = \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i^*)$$

SGD Stochastic gradient descent minimizes this objective iteratively:

1. Choose a starting point $\boldsymbol{\theta}$.
2. While not converged:
   - Choose a step size $\eta_t > 0$.
   - Sample a training instance i.
   - Take a small step following the gradient down.

   $$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla E_i(\boldsymbol{\theta}) \tag{5}$$

   What is the gradient? It's just a vector of derivatives. For shorthand we can define $J = E_i(\boldsymbol{\theta})$, then the gradient is $\nabla E_i(\boldsymbol{\theta}) = [\frac{dJ}{d\theta_1}, \frac{dJ}{d\theta_2}, \ldots, \frac{dJ}{d\theta_K}]$.

   This means that the SGD update can either be written as the vector update in Eq. (5), or as an update of each model parameter as in Eq. (6) below

   $$\theta_j^{(t+1)} = \theta_j^{(t)} - \eta_t \frac{dJ}{d\theta_j} \tag{6}$$

Prediction Given a new instance $\boldsymbol{x}$, predict the output $\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$.

2.1 Loss Functions

Some common loss functions for the case where $y \in \{0, 1\}$.

	Forward	Backward
Regression	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

2.2 Binary Logistic Regression

Binary logistic regression can be interpreted as a computational circuit. To compute the derivative of some loss function (below we use regression) with respect to the model parameters, we can apply backpropagation.

Note that the output y below is the probability that the output label takes on the value 1.

The forward pass computes $J = \frac{1}{2}\left(\left(\frac{1}{1+\exp(\sum_{j=0}^{D}\theta_j x_j)}\right) - y^*\right)^2$. The backward pass computes $\frac{dJ}{d\theta_j}\forall j$.

Forward	Backward	
$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$	
$y = \frac{1}{1+\exp(a)}$	$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \ \frac{dy}{da} = \frac{\exp(a)}{(\exp(a)+1)^2}$	
$a = \sum_{j=0}^{D}\theta_j x_j$	$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \ \frac{da}{d\theta_j} = x_j$	$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \ \frac{da}{dx_j} = \theta_j$

2.3 2-Layer Neural Network

Backpropagation for a 2-layer neural network looks very similar to the logistic regression example above. We have added a hidden layer z corresponding to the latent features of the neural network.

Note that our model parameters θ are defined as the concatentation of the vector β (parameters for the output layer) with the vectorized matrix α (parameters for the hidden layer).

Forward	Backward	
$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$	
$y = \frac{1}{1+\exp(b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \ \frac{dy}{db} = \frac{\exp(b)}{(\exp(b)+1)^2}$	
$b = \sum_{j=0}^{D}\beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \ \frac{db}{d\beta_j} = z_j$	$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \ \frac{db}{dz_j} = \beta_j$
$z_j = \frac{1}{1+\exp(a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \ \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j)+1)^2}$	
$a_j = \sum_{i=0}^{M}\alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \ \frac{da_j}{d\alpha_{ji}} = x_i$	$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \ \frac{da_j}{dx_i} = \sum_{j=0}^{D}\alpha_{ji}$

2.4 Numerical Differentiation

While this document is focused on reverse-mode automatic differentiation, numerical differentiation provides a convenient method for testing gradients computed by autodiff. Unfortunately, in practice, it suffers from issues of floating point precision. Therefore, it is typically only appropriate to use this on small examples with an appropriately chosen epsilon close to machine epsilon.

$$\frac{\partial}{\partial \theta_i} E(\boldsymbol{\theta}) \approx \frac{(E(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}) - E(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}))}{2\epsilon \cdot d_i} \tag{7}$$

3 For Presentation

Recipe for ML

$$\begin{aligned} \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N \\ \hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \\ \ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R} \\ \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) \\ \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i) \end{aligned}$$

Linear, Logistic, NNs

$$\begin{aligned} f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= y \\ y &= h(\boldsymbol{\theta} \cdot \boldsymbol{x}) \\ &\text{where } h(a) = a \\ f_{\boldsymbol{\theta}}(\boldsymbol{x}) &= y = h(\boldsymbol{\theta} \cdot \boldsymbol{x}) \\ &\text{where } h(a) = \frac{1}{1 + \exp(a)} \end{aligned}$$

$$\begin{aligned} y &= f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{\theta} \cdot \boldsymbol{x}) \\ &\text{where } h(a) = \frac{1}{1 + \exp(a)} \end{aligned}$$