1 Introduction

So far, we've seen how to train "shallow" models, where the predictions are computed as a linear function of the inputs. We've also observed that deeper models are much more powerful than linear ones, in that they can compute a broader set of functions. Let's put these two together, and see how to train a multilayer neural network. We will do this using backpropagation, the central algorithm of this course. Backpropagation ("backprop" for short) is a way of computing the partial derivatives of a loss function with respect to the parameters of a network; we use these derivatives in gradient descent, exactly the way we did with SVM, and Softmax classification.

If you've taken a multivariate calculus class, you've probably encountered the Chain Rule for partial derivatives, a generalization of the Chain Rule from univariate calculus. In a sense, backprop is "just" the Chain Rule — but with some interesting twists and potential gotchas. This discussion covers the mathematical justification and shows how to implement a backprop routine by hand.

2 The chain rule revisited

Before we get to neural networks, let's start by looking more closely at a simple example: a linear classification model. For simplicity, let's assume we have univariate inputs and a single training example (x, t). The predictions are a linear function followed by a sigmoidal nonlinearity. Finally, we use the squared error loss function. The model and loss function are as follows:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$

Now, to change things up a bit, let's add a regularizer to the cost function. We'll cover regularizers properly in a later discussion, but intuitively, they try to encourage "simpler" explanations. In this example, we'll use the regularizer $\frac{\lambda}{2}w^2$, which encourages w to be close to zero. (λ is a hyperparameter; the larger it is, the more strongly the weights prefer to be close to zero.) The cost function, then, is:

$$\begin{aligned} \mathcal{R} &= \frac{1}{2}w^2 \\ \mathcal{L}_{reg} &= \mathcal{L} + \lambda\mathcal{R} \end{aligned}$$

In order to perform gradient descent, we wish to compute the partial derivatives $\frac{\partial \mathcal{L}_{reg}}{\partial w}$ and $\frac{\partial \mathcal{L}_{reg}}{\partial b}$.

Use chain rule to compute $\frac{\partial \mathcal{L}_{reg}}{\partial w}$ and $\frac{\partial \mathcal{L}_{reg}}{\partial b}$. Comment on the drawbacks of this computation.

As you might have figured out that the above method involves lot of repeated computations and is not modular. The idea behind backpropagation is to share the repeated computations wherever possible. We'll see that the backprop calculations, if done properly, are very clean and modular.

3 Computational graphs and backpropagation

Computational graphs help to visualize and contextualize how we aim to backpropagate derivatives. It is a directed acyclic graph where each node in the graph denote a mathematical operation.

Draw the computational graph for our running example, written again for convenience:

$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
\mathcal{L} &= \frac{1}{2}(y - t)^2 \\
\mathcal{R} &= \frac{1}{2}w^2 \\
\mathcal{L}_{reg} &= \mathcal{L} + \lambda \mathcal{R}
\end{aligned}
$$

The backpropagation algorithm gives us an efficient way of computing the derivatives, necessary for updating the parameters of the model, using the computational graph. The algorithm consists of two main components:

- **Forward pass**: It takes an input and propagates it through the computational graph sequentially until you arrive at your output. With the input as the start and the output as the end, information propagates in a forward manner
- **Backward pass**: It takes the gradients at the output and propagates it backward through the computational graph to the inputs. It enables the calculation of gradients at every stage going back to the inputs.

Backpropagation is not the learning algorithm. It's the method of computing gradients. It is not specific to multilayer neural networks, but a general and modular way to compute derivatives of functions.

Use the computational graph and backpropagation to compute $\frac{\partial \mathcal{L}_{reg}}{\partial w}$ and $\frac{\partial \mathcal{L}_{reg}}{\partial b}$. Comment on the advanatages of this computational mechanism.

4 Practice problems on computational graphs and backpropagation

1. Let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{W} \in \mathbb{R}^{n \times n}$. We define the following loss function

$$\mathcal{L} = \|\mathbf{W}\mathbf{x} - \mathbf{x}\|^2$$

Draw the computational graph and use backpropagation to compute $\nabla_{\mathbf{W}}\mathcal{L}$.

2. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices in $\mathbb{R}^{n\times n}$ and α,β,γ be scalars in $\mathbb{R}$. Let's define the following loss functions:

$$\mathcal{L}_1(\mathbf{A}) = -\frac{\gamma}{2}\log|\alpha\mathbf{A}\mathbf{A}^{\mathbf{T}} + \beta^{-1}\mathbf{I}|$$
$$\mathcal{L}_2(\mathbf{A},\mathbf{B}) = -\frac{1}{2}tr((\alpha\mathbf{A}\mathbf{A}^{\mathbf{T}} + \beta^{-1}\mathbf{I})^{-1}\mathbf{B}\mathbf{B}^{\mathbf{T}})$$

   (a) Draw a computational graph for $\mathcal{L}_1$ and use it to compute $\frac{\partial\mathcal{L}_1}{\partial\mathbf{A}}$.

   (b) Draw a computational graph for $\mathcal{L}_2$ and use it to compute $\frac{\partial\mathcal{L}_2}{\partial\mathbf{A}}$.

   Hint: You may use the following matrix derivative without proof:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{K}} = -\mathbf{K}^{-T}\frac{\partial\mathcal{L}}{\partial\mathbf{K}^{-1}}\mathbf{K}^{-T},$$

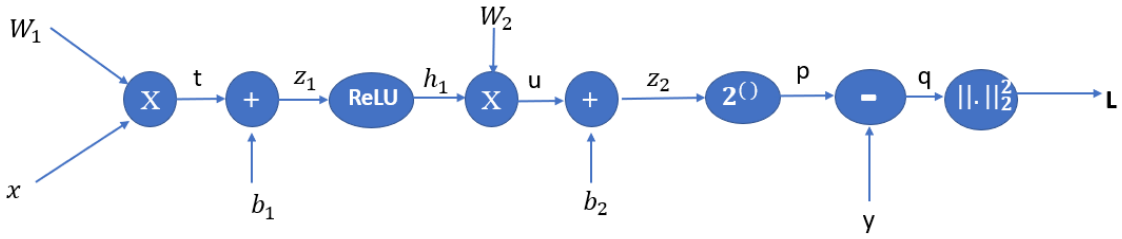
   where

$$\mathbf{K} = \alpha\mathbf{X}\mathbf{X}^T + \beta^{-1}\mathbf{I}.$$

   Also, consider the matrix operation, $\mathbf{Z} = \mathbf{X}\mathbf{Y}$. If we have an upstream derivative, $\partial\mathcal{L}/\partial\mathbf{Z}$, then backpropagate the derivatives in the following way:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{X}} = \frac{\partial\mathcal{L}}{\partial\mathbf{Z}}\mathbf{Y}^T$$
$$\frac{\partial\mathcal{L}}{\partial\mathbf{Y}} = \mathbf{X}^T\frac{\partial\mathcal{L}}{\partial\mathbf{Z}}$$

3. Consider the computational graph of a regression model given below, where $(\mathbf{x},\mathbf{y})$ represent a training sample with $\mathbf{x}\in R^n$, $\mathbf{y}\in R^k$. The learnable parameters of the model are: $W_1\in R^{m\times n}, W_2\in R^{k\times m}, \mathbf{b}_1\in R^m, \mathbf{b}_2\in R^k$. In order to learn the parameters using gradient descent, we need to compute the gradient of the loss function with respect to the learnable parameters.



   (a) Compute $\nabla_{W_2}L$, $\nabla_{b_2}L$.

   (b) Compute $\nabla_{W_1}L$, $\nabla_{b_1}L$.