Learning Lambda Calculus with Recurrent Neural Networks

Abstract

We present a neural networks which can take in $(input, ouput)$ pairs and generate $program$ where $input$, $output$, and $program$ are representations of pure untyped lambda calculus.

1 Introduction

Lambda calculus is a universal language of computation[1]. Below is the syntax for lambda calculus:

$$\begin{aligned} term &\Rightarrow \alpha\ term\ term \\ term &\Rightarrow \ell\ term \\ term &\Rightarrow v \\ v &\Rightarrow |\ v \\ v &\Rightarrow \sim \end{aligned}$$

To many, this will seem as an unnatural construction of lambda calculus. The construction is based upon John Tromp's 2004 interpretation of binary lambda calculus[2], where instead of mapping lambda calculus to a domain of size two $\{0\ 1\}$, we instead map lambda calculus to an unambiguous domain of four $\{\alpha\ \ell\ |\ \sim\}$. We continue an elaboration of some terms in our new language:

$$\mathrm{false} = \ell\ \ell\ |\ \sim \tag{1}$$
$$\mathrm{true} = \ell\ \ell\ \|\ \sim \tag{2}$$
$$\mathrm{and} = \ell\ \ell\ \alpha\ \alpha\ |\ \sim\ \ell\ \ell\ |\ \sim\ \|\ \sim \tag{3}$$
$$\mathrm{or} = \ell\ \ell\ \alpha\ \alpha\ |\ \sim\ \|\ \sim\ \ell\ \ell\ \|\ \sim \tag{4}$$
$$\mathrm{xor} = \ell\ \ell\ \alpha\ \alpha\ |\ \sim\ \ell\ \ell\ \|\ \sim\ \alpha\ \ell\ \alpha\ \alpha\ |\ \sim\ \ell\ \ell\ \|\ \sim\ \ell\ \ell\ |\ \sim\ \|\ \sim \tag{5}$$
$$\tag{6}$$

Here is an example calculation:

$$\begin{aligned} \mathrm{and\ true\ true} &= \ell\ \ell\ \alpha\ \alpha\ |\ \sim\ \ell\ \ell\ |\ \sim\ \|\ \sim\ \ell\ \ell\ \|\ \sim\ \ell\ \ell\ \|\ \sim \\ &\rightarrow \ell\ \alpha\ \alpha\ \ell\ \ell\ \|\ \sim\ \ell\ \ell\ |\ \sim\ |\ \sim\ \ell\ \ell\ \|\ \sim \\ &\rightarrow \alpha\ \alpha\ \ell\ \ell\ \|\ \sim\ \ell\ \ell\ |\ \sim\ \ell\ \ell\ \|\ \sim \\ &\rightarrow \alpha\ \ell\ |\ \sim\ \ell\ \ell\ \|\ \sim \\ &\rightarrow \ell\ \ell\ \|\ \sim \\ &= \mathrm{true} \end{aligned}$$

[1] **Turing**.
[2] **Tromp**.

$$zero = \ell\ \ell\ |\ \sim \tag{7}$$

$$suc = \ell\ \ell\ \ell\ \alpha\ |||\ \sim\ |\ \sim \tag{8}$$

$$one = \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ |\ \sim$$

$$two = \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ |\ \sim$$

$$three = \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ \alpha\ ||\ \sim \ell\ \ell\ |\ \sim$$

$$\dots$$

$$\begin{aligned} add = {} & \ell\ \ell\ \ell\ \alpha\ \alpha\ ||\ \sim \alpha\ \alpha\ \alpha\ \ell\ \ell\ \ell\ |||\ \sim\ |\ \sim\ ||\ \sim\ |||\ \sim \alpha\ \alpha\ \alpha \\ & \ell\ \ell\ \ell\ \ell\ \alpha\ \alpha\ |\ \sim\ ||||\ \sim \alpha\ \ell\ \ell\ \ell\ \alpha\ |||\ \sim\ |\ \sim\ |||\ \sim\ |\ \sim\ ||\ \sim\ |||\ \sim \end{aligned} \tag{9}$$

2 Existing Work

2.1 Arithmetic Classifiers

Franco and Cannas showed in 1997 the ability for neural networks to learn arithmetic operations [3]

[3] http://www.lcc.uma.es/ lfranco/A1-Franco+Cannas-1998.pdf