Training End-to-End Analog Neural Networks with Equilibrium Propagation

2020 年 9 月 2 日

Content

Introduction

在使用电路，而不是软件对神经网络进行retrain的时候需要算出反向传播的导数。对于线性的激活函数，比如在图像识别中用的比较多的ReLU，LRelu等，反向传播的斜率计算比较方便，激活层只需要乘一个系数（LRelu小于0部分）或者对于激活前的导数就是对于激活后的值的导数（ReLU大于0部分）。

但是在NLP，LSTM中，很多用的都是非线性激活函数，比如tanh，$\text{swish}\left(\frac{x}{1+\exp(-x)}\right)$。这时候在用电路中直接进行反向传播的计算会需要用到不同的电路，而且需要大量的模数转换器(ADC)来把中间节点的值转换成数字信号来进行计算。

Background

Energy-based Model

Energy-based Model, EBM（基于能量的模型）：与EBM相对应的是概率模型，一般使用MCMC来进行采样，非常耗费时间，而且MCMC的收敛时间较长。EBM用一个标量来评价结果的好坏，这样的的好处是可以不需要对结果标准化，特别是当结果是一个高维的结构。

Background

Equilibrium Propagation

Equilibrium Propagation, EP: 和一般的SGD过程不同的是，这是一个动态连续的过程。假设u是输入节点x，隐藏层节点h，输出节点y的集合，d是目标结果。需要最小化的能量函数

$$F = E(u) + \beta \cdot C(y, d)$$

如果持续的对$s = \{h, y\}$中的节点持续进行类似于反向传播的操作

$$\frac{\mathrm{d}s}{\mathrm{d}t} = -\frac{\partial F}{\partial s},$$

将会使得总的能量函数F随着时间不断变小

$$\frac{\mathrm{d}F}{\mathrm{d}t} = \frac{\partial F}{\partial s} \cdot \frac{\mathrm{d}s}{\mathrm{d}t} = -\left\| \frac{\mathrm{d}s}{\mathrm{d}t} \right\|^2 \leqslant 0.$$

最终系统会达到不动点，F的值不再下降。

Background

Equilibrium Propagation

EP分为两个部分：free phase, nudged phase。

free phase中，β被设置为0。系统将会达到平衡点s^0。
nudged phase中，β被设置为一个小正数。系统将会达到平衡点s^β。这个平衡点将结果的正确性纳入考虑，类似于完成了一次BP过程。

注意到在之前提到的$F = E(u) + \beta \cdot C(y, d)$中，改变了$\beta$一开始只会影响输出单元$y$的梯度。在$y$的值改变后，这的梯度会传导向中间的隐藏层中，这点和反向传播十分相似。

Background

Equilibrium Propagation

这里提出一个定理：
对于在网络中的任意一个参数θ，(区别于s，是网络中节点的值)

$$\frac{\partial C(y,d)}{\partial \theta} = \lim_{\beta \to 0} \frac{1}{\beta} \left(\left. \frac{\partial F}{\partial \theta} \right|_{\beta=\beta, s=s^{\beta}} - \left. \frac{\partial F}{\partial \theta} \right|_{\beta=0, s=s^{0}} \right)$$

特别的，这边的β不再一定是正数，负数也是可以的，只要是接近0的小数，因为这边的符号会抵消，我们只要求对于每个参数的导数。

Background

Equilibrium Propagation

于是整个EP的过程分为以下几步：

一、将β设为0，等电路平衡了以后，测得每个参数的导数 $\frac{\partial F}{\partial \theta}\Big|_{\beta=0,s=s^0}$.

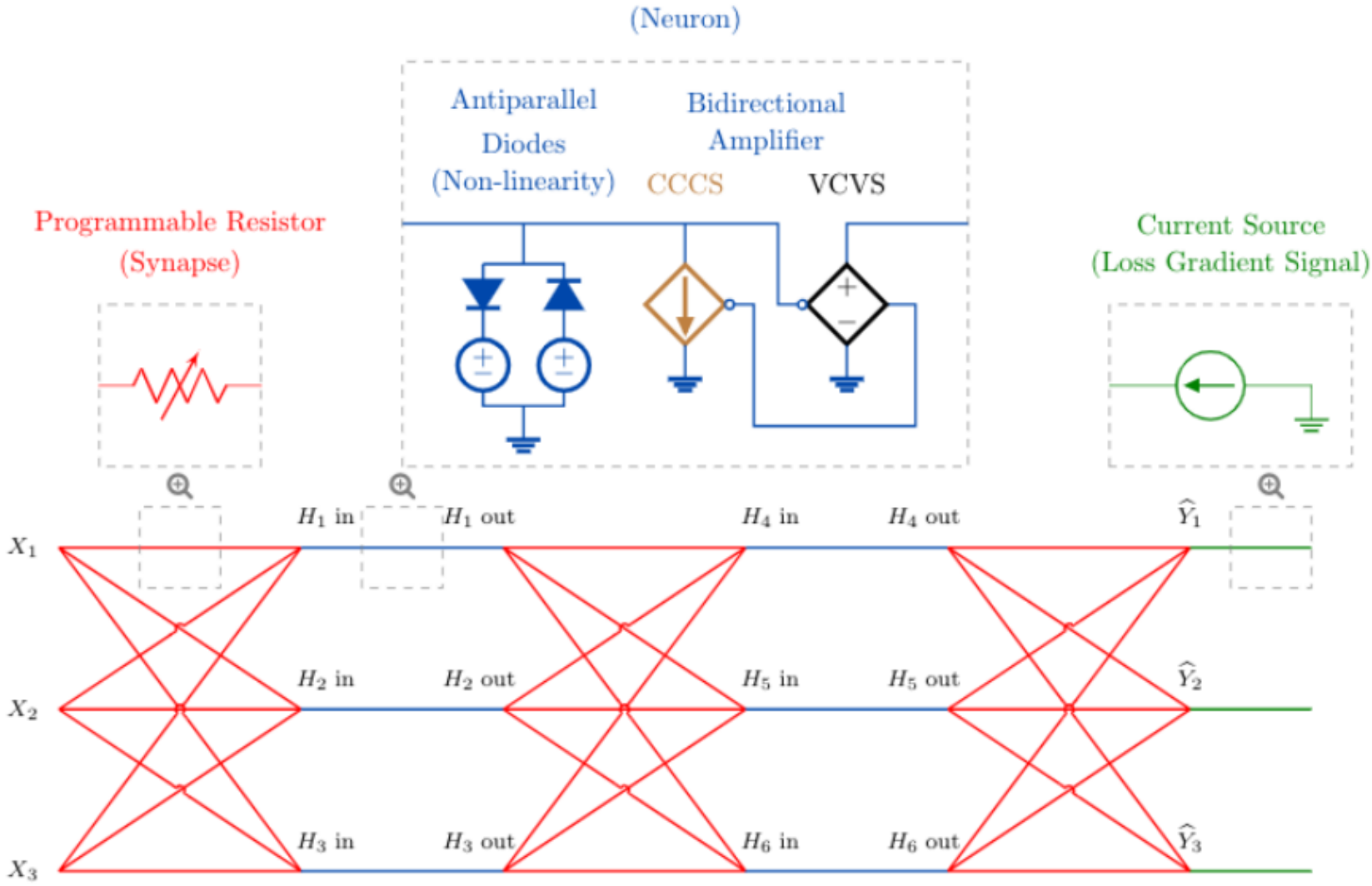
二、将β设为某个小值，可正可负，等待电路平衡后测得每个参数的导数 $\frac{\partial F}{\partial \theta}\Big|_{\beta=\beta,s=s^\beta}$.

三、更新每个参数θ,

$$\Delta\theta \propto -\frac{1}{\beta}\left(\frac{\partial F}{\partial \theta}\Big|_{\beta=\beta,s=s^\beta} - \frac{\partial F}{\partial \theta}\Big|_{\beta=0,s=s^0}\right)$$

这边的前两步骤的等待电路平衡是时间上连续的，适合在硬件上实现。

Basic Structure



Basic Structure

使用上图的网络结构后，EP可以按照以下步骤执行：

为了简化电路结构，这边的$F = E(u) + \beta \cdot C(y, d)$中的$E(u)$被设置为了0，也就是说不需要对隐藏层的节点施加梯度。

第二步的将β设为一个非0小数，相当于在输出单元上接入大小为$-\beta \frac{\partial C}{\partial \hat{Y}_k}$的电流。

此时对于线性欧姆特性的器件(比如在低电压下的memristor)，其关于损失函数C的导数可以表示为

$$\frac{\partial C}{\partial g_{ij}} = \lim_{\beta \to 0} \frac{1}{2\beta} \left((\Delta V_{ij}^{\beta})^2 - (\Delta V_{ij}^{0})^2 \right)$$

g_{ij}表示i, j两点之间的电导，ΔV_{ij}^{0}，ΔV_{ij}^{β}表示$\beta = 0, \beta$时，两个节点的电压差。

Equilibrium Propagation with Memristor

这个方法的一个好处是不需要ADC开销。利用Sample-and-Hold Amplifier（SHA）保存下每个忆阻器两端的电压差，并计算出梯度后，只需要对忆阻器施加相应大小的脉冲就可以改变他的阻值。现有的文章说明对于一个有16级精度的忆阻器，这种操作的误差可以控制在0.5%以内。

而且，由于改变忆阻器阻值的脉冲操作所需要的时间和能耗很少，所以速度和能耗比一般的GPU操作要高两个数量级。

Result

Table 1: Results on MNIST. SPICE EqProp (our model) is benchmarked against a PyTorch implementation of the original EqProp model [Scellier and Bengio, 2017]. For the PyTorch models, the mean values and 95% confidence intervals of test errors are reported over five runs, after 10 epochs of training and after training completion. 100 and 500 denote the number of neurons in the hidden layer. 'pos. weights' means that weights are constrained to be positive (as explained in section 4.4).

	Error rates after 10 epochs (%)		Final error rates (%)	
	Test	(Train)	Test	(Train)
SPICE EqProp 100 (our model)	**3.43**	**(2.68)**		
PyTorch EqProp 100 (pos. weights)	3.85 ± 0.05	(2.87)	3.44 ± 0.04	(1.38)
PyTorch EqProp 100	3.99 ± 0.19	(2.93)	3.59 ± 0.06	(1.35)
PyTorch EqProp 500 (pos. weights)	2.49 ± 0.01	(1.47)	$\mathbf{2.01 \pm 0.02}$	**(0.26)**
PyTorch EqProp 500	2.92 ± 0.09	(1.82)	2.38 ± 0.11	(0.52)

Discussion

首先，整个方法的好处是将时间上离散的BP操作变成了更容易在硬件上实现的EP的操作。以及，利用EP中导数的特性，可以不使用ADC就算出导数，而且不需要大量的额外电路。

问题之一还是如何让忆阻器的电导稳定变化。现有的16级或者64级分辨率的忆阻器还是不足以表达复杂的网络的，现在模拟都在mnist或者cifar10上进行。对于负数的处理，文章里直接把正负数矩阵分开来处理，让矩阵个数翻倍了，这也是个不小的开销。