

# Electronice

Amartya Sanyal

January 5, 2016

## 1 Model Description

The model in a way combines the NICE model and the ACDC layers with a piecewise non-linearity employed at the end of each layer. The reconstruction error is ideally zero as we have an exact inversion of the encoder function in the decoder function.

### 1.1 Layer

Each layer is an ACDC layer with an added non-linearity. The parameters in each layer are  $D$ ,  $A$ ,  $K_x$  and  $K_y$ . Here,  $D$  and  $A$  are diagonal matrices. The dimension of these matrices are  $n \times n$ , where  $n$  is the number of features in an example.  $K_x$  and  $K_y$  are column vectors of  $n$  dimension. Given the input to a layer is  $x$  the output  $o$  is calculated as follows.

### 1.2 Pseudocode

---

**Algorithm 1** Encoder

---

```
1: procedure ENCODER( $X$ )
2:    $temp \leftarrow X \cdot A \cdot C \cdot D \cdot C^t$ 
3:   for  $i = 0$  to  $|X|$  do  $\triangleright$  Time =  $O(n)$ 
4:     if  $temp[i] \leq \text{Tanh}(Kx[i])$  then
5:        $out[i] \leftarrow ((temp[i] + 1) * (\text{Tanh}(Ky[i]) + 1) / (\text{Tanh}(Kx[i]) + 1) - 1)$ 
6:     else
7:        $out[i] \leftarrow ((temp[i] - 1) * (\text{Tanh}(Ky[i]) - 1) / (\text{Tanh}(Kx[i]) - 1) + 1)$ 
8:     end if
9:   end for
10:  return  $out$ 
11: end procedure
```

---

We need to calculate the exact inverse of our encoding function. In this way the *Autoencoder* works as  $out = f^{-1}(f(x)) = x$ . If  $out_{enc} = f(x)$ , then  $f^{-1}(out_{enc})$  is calculated as follows. We use five of these layers one after the

---

**Algorithm 2** Decoder

---

```
1: procedure DECODER( $X$ )
2:   for  $i = 0$  to  $|X|$  do ▷ Time =  $O(n)$ 
3:     if  $X[i] \leq \text{Tanh}(Ky[i])$  then
4:        $\text{temp}[i] \leftarrow ((X[i] + 1) * (\text{Tanh}(Kx[i]) + 1) / (\text{Tanh}(Ky[i]) + 1) - 1)$ 
5:     else
6:        $\text{temp}[i] \leftarrow ((X[i] - 1) * (\text{Tanh}(Kx[i]) - 1) / (\text{Tanh}(Ky[i]) - 1) + 1)$ 
7:     end if
8:   end for
9:    $\text{out}[i] \leftarrow \text{temp} \cdot C \cdot D^{-1} \cdot C^t \cdot A^{-1}$ 
10:  return  $\text{out}$ 
11: end procedure
```

---

other. Each of these layers have only  $O(n)$  parameters. Though my implementation right now has  $O(n^2)$  complexity due to the multiplication with the  $C$  matrix, it can be reduced to  $O(n \log n)$  by using an FFT like algorithm. The multiplication of the vector with  $A$  and  $D$  can be achieved in  $O(n)$  time by element-wise multiplication as  $A$  and  $D$  are diagonal matrices. The piece wise non-linearity can be applied on each element in  $O(1)$  time.

At the end of the encoder, there is rescaling layer. Let this layer be defined by a matrix  $S$ . This is also a diagonal matrix and it is unbounded, in the sense that it can be allowed to expand to infinity at some points. This is similar to the rescaling layer implemented in NICE. The decoder begins with the inverse-rescaling layer and then the decoder functions

## 2 Cost Function

The cost function functions the same way as NICE with a change of variable. We are using logistic distribution as the prior as it has smoother gradients. The  $i$  in the subscript below refers to the layer number whereas the  $j$  refers to the  $j^{\text{th}}$  element in the vector.

$$\log(P_X x) = \log(P_H(f(x))) + \log(|\det(\frac{\partial f(x)}{\partial x})|)$$

The prior distribution is factorial as we want the components to be independent.

$$\log(P_H(f(x))) = \prod_{j=0}^{j=|x|} P_H(f(x[j]))$$

We also impose a logistic distribution on the prior. Hence, the equation expands as follows .

$$\log(P_H(f(x[j]))) = -\log(1 + \exp(x[j])) = \log(1 + \exp(-x[j]))$$

The determinant of the jacobian can be calculated in  $O(n)$  time as follows. The DCT transformation matrices are unitary and hence their jacobian has a unitary determinant. The jacobian for the  $A$ ,  $D$  and the rescaling layer are diagonal matrices as they themselves are diagonal and hence the determinant is simply the product of its diagonal elements.

$$\begin{aligned}
\log(|\det(\frac{\partial f(x)}{\partial x})|) &= \sum_{i=0}^5 \log(\prod_{j=1}^{|A|} A_i[j]) && \text{jacobian for A matrix} \\
&+ \log(\prod_{j=1}^{|A|} D_i[j]) && \text{jacobian for D matrix} \\
&+ \log(\prod_{j=1}^{|A|} \text{act\_jac}_i(x)[j]) && \text{Jacobian for non-linearity} \\
&+ \sum_{j=1}^{|A|} \log(S_i[j]) && \text{Jacobian for rescaling}
\end{aligned}$$

The *act\_jac* functions as follows. Note that the jacobian of the piece-wise linear activation function will be a diagonal matrix. Hence, to calculate the Determinant, we can simply multiply the diagonal elements. The function below return a vector of the diagonal elements given the input vector.

---

**Algorithm 3** Activation\_Jacobian

---

```

1: procedure ACT_JAC( $X$ )
2:   for  $i = 0$  to  $|X|$  do                                      $\triangleright$  Time =  $O(n)$ 
3:     if  $X[i] \leq \text{Tanh}(Ky[i])$  then
4:        $\text{jac}[i] \leftarrow (\text{Tanh}(Kx[i]) + 1) / (\text{Tanh}(Ky[i]) + 1)$ 
5:     else
6:        $\text{jac}[i] \leftarrow (\text{Tanh}(Kx[i]) - 1) / (\text{Tanh}(Ky[i]) - 1)$ 
7:     end if
8:   end for
9:   return  $\text{jac}$ 
10: end procedure

```

---

### 3 Training

We are using ADAM version of stochastic gradient descent for training. The parameters for update are  $A, D, K_x$  and  $K_y$  for each layer and a the final rescaling layer  $S$ . For a five layer network, there are a total of 21 parameters. We are initially training on the mnist training set. The hyper-parameters are as follows. **Learning Rate:** 0.00002, **b1:** 0.1, **b2:** 0.001, **e:** 1e-8. As of now, I have a log-likelihood of around 1000 and it is still increasing. However, sampling is very poor.

## 4 Code

The code is available on <http://github.com/amartya18x/ElectroNice>

- **niceelectro.py:** This describes each ACDC layer with the inversion
- **model.py:** This builds the training network by combining the five ACDC layers and also adds rescaling and calculates cost and updates.
- **optimization.py:** This contains the code to return the ADAM update rules. This is called from model.py.
- **trainmnist.py:** You should run this simply as *python trainmnist.py* to start training. This contains hyper parameters like batch\_size and when to view a result.
- **sample.py:** Run *python sample.py* to view a sample.