Electronice

Amartya Sanyal

January 6, 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 diagnol 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

procedure ENCODER(X)
  $temp \leftarrow X \cdot A \cdot C \cdot D \cdot C^t$
  for $i = 0$ to $|X|$ **do** ▷ Time = O(n)
    if $temp[i] \leq Tanh(Kx[i])$ **then**
      $out[i] \leftarrow ((temp[i]+1)*(Tanh(Ky[i])+1)/(Tanh(Kx[i])+1)-1)$
    else
      $out[i] \leftarrow ((temp[i]-1)*(Tanh(Ky[i])-1)/(Tanh(Kx[i])-1)+1)$
    end if
  end for
  return out
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

procedure DECODER(X)
  for $i = 0$ to $|X|$ **do** ▷ Time = O(n)
    if $X[i] \leq Tanh(Ky[i])$ **then**
      $temp[i] \leftarrow ((X[i]+1) * (Tanh(Kx[i])+1)/(Tanh(Ky[i])+1)-1)$
    else
      $temp[i] \leftarrow ((X[i]-1) * (Tanh(Kx[i])-1)/(Tanh(Ky[i])-1)+1)$
    end if
  end for
  $out[i] \leftarrow temp \cdot C \cdot D^{-1} \cdot C^t \cdot A^{-1}$
  return out
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 $O(nlogn)$ 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 diagnol matrix and it is unbounded, in the sense that it can be allowed the 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 subscipt below refers to the layer number whereas the j refers to the j^{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 independant.

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

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.

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

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 Determinanat, 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

procedure ACT_JAC(X)
  for $i = 0$ to $|X|$ **do** ▷ Time = O(n)
    if $X[i] \leq Tanh(Ky[i])$ **then**
      $jac[i] \leftarrow (Tanh(Kx[i]) + 1)/(Tanh(Ky[i]) + 1)$
    else
      $jac[i] \leftarrow (Tanh(Kx[i]) - 1)/(Tanh(Ky[i]) - 1)$
    end if
  end for
  return jac
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 containes 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 containes hyper parameters like batch_size and when to view a result.
- **sample.py:** Run *python sample.py* to view a sample.

5 Observations

- The inverse is working fine. I have displayed source and output image and checked that they are same.
- The log-likelihood is increasing after applying the rescaling layer.
- The rescaling layer however appears to however have a uniform distribution. I hope, the dimshuffle I have implemented there isn't working imperfectly.
- The $p_H(h)$ term is still not increasing. It stays the same throughout the training phase. I do not understand this yet.
- There is no overfitting. I have tested on the test_set and it is exactly the same. This is because, the $p_H(h)$ is still the same and the jacobian term is almost independant of the training data except for the non-linearity and somehow it is not mattering.
- The Kx and Ky is getting pushed towards 0. So, effectively it is becoming a linear transformation.
- We tried to overfit the model on a small dataset of only ten images. In this case, the prior term started of with a low value of beow -2000 and then increased to -1086 and stuck there. Note that this is the same value the prioir term usually has.
- I also experimented with putting the prior as the cost function(i.e. removed the log-jacobian Determinant) and even then the prior wouldn't go above -1086. It came to this value and stayed on it. I am guessing the encoder somehow doesn't allow the encoded value to have a better loss. Is there any reason why this might happen ? Laurent, what was the prior log-likelihood term in the original NICE model on mnist ?