Neural Networks Report #8

Introduction

The final homework is about the transformer model this time more inclined to the calculation part, nonlinear independent component analysis and variational autoencoder. A transformer is a deep learning model that adopts the mechanism of attention, weighing the influence of different parts of the input data and uses large number of adaptive elements for this. Independent component analysis is a computational method for separating a multivariate signal into additive subcomponents. VAEs are directed probabilistic graphical models whose posterior is approximated by a neural network, forming an autoencoder-like architecture, in other words autoencoder that employs probabilistic model.

Problem1

a)

a) $X = (X_1 \ldots X_{N_{voc}})$

$X : N_{mod} \times N_{voc}$

$\therefore \quad (N_{que} = N_v = N_{head})$ given to model

$N_{mod} = N_v \times N_h$

$Q, K, V : N_{mod} \times N_{head}$

$W_h^j : N_{mod} \times N_{q|v}$

$A_h = \text{soft}[\, Q^T W_h^Q (W_h^K)^T K / \sqrt{N_q} \,] : N_{pos} \times N_{pos}$

$B = A_h V_h^T : N_{val} \times N_{pos}$

$C = W^o B$ where $W^o : N_{mod} \times (N_{val} \times N_{head})$

$C : N_{mod} \times N_{pos}$, $y = MLP(C) : N_{mod} \times N_{hed}$

As a result we need to train

W_h^q, W_h^k, W_v^h and W^o

So the total number is

$N_1 = 3 \cdot N_{mod} \times N_{qvk} + N_{mod} \times N_{val} \times N_{head}$

$N_1 = N_{val} \cdot N_h (3N_{qvk} + N_{val} \cdot N_{head})$

where $N_{qvk} = N_{que} = N_{key} = N_{val}$

as they should be equal by constraint

as we also have hidden layer

there is other matrix $W: N_{hd} \times N_{mod}$

As there are N_{enc} blocks we have

$N_1 = N_{enc} N_{val} \cdot N_h (3N_{qvk} + N_{val} \cdot N_{head} + N_{hd})$

b)

b) for input it is evident that

$X: N_{mod} \times N_{voc} = N_{qvk} \times N_{head} \times N_{voc}$

Similarly the Output size should be same as input as it is only the shifted version. as we also have ecoder/decoder boxes

$N_2 = (N_{enc} + N_{dec}) \times N_{qvk} \times N_{head} \times N_{voc}$

c)

c) The first module has same step number, so $N_3 = N_1 \frac{N_{dec}}{N_{enc}}$

the Masked Multihead takes the values of dimension $N_{mod} \times N_{hid}$

which equals to $N_{mod} \times N_{head}$'

Therefore

$N_4 =$

$$N_{mul} N_{val} \cdot N_h (3 N_{qvk} + N_{val} \times N_{head} + N_{hid})$$

where $N_{mul} = \frac{N_{hid}}{N_{head}}$

$$N = \sum_i N_i$$

$$\approx (N_{enc} + N_{dec}) \times N_{qvk} \times N_{head} \times N_{voc}$$

$$+ N_{val} \cdot N_h (3 N_{qvk} + N_{val} \times N_{head} + N_{hid})$$

$$\times [N_{enc} + N_{dec} + \frac{N_{hidden}}{N_{head}}]$$

d)

d) With numerical values

$N_1 = 6 \cdot 64 \cdot 8 (3 \cdot 64 + 64 \cdot 8 + 1064)$

$N_1 = 5431296$

$N_2 = (6+6) \cdot 64 \cdot 8 \cdot 50\,000$

$N_2 = 307200000$

$N_3 = N_1 = 5431296$

$N_4 = 115867648$

so $N = 423067649$

Problem2

a)

For scalar u & X:

a) $u = f(X)$ we have $p(u)$

$F_U(u) = P(U \leq u) = P(f(X) \leq u)$
$= P(X \leq f^{-1}(u)) = F_X(f^{-1}(u))$

$F(X) = \int_0^x p(x)\,dx \rightarrow p(x) = \frac{dF(X)}{dx}$

$p(u) = \frac{dF_x}{du}(f^{-1}(u))$

$p(x) = \frac{dF_x(X)}{dX}$

let $x = f^{-1}(u)$

then $dx = \frac{du}{f'(f^{-1}(u))} = \frac{du}{f'(x)}$

as $x = f^{-1}(u)$

$p(u) = \frac{dF_x(x)}{dx\,f'(x)} = \frac{p(x)}{f'(x)} = \frac{p(x)}{df/dx}$

So generally we can say

$p(u) = p(f^{-1}(y)) \left| \frac{d}{dy}(f^{-1}(y)) \right|$

We know that vector is

$f_X(x) = f_{X_1 \sim X_n}(x_1, \ldots, x_n)$

if we follow same rules but using vector definitions we get

$$p(u) = p(f^{-1}(w)) \left| \det \left[\frac{d f^{-1}(\vec{z})}{d \vec{z}} \right] \right|$$

or $$p(u) = p(x) \det \left[\left(\frac{d}{dx} f(x) \right)^{-1} \right]^{z=y}$$

where $\frac{df}{dx}$ is jacobian matrix

$$J(f) = \frac{df}{dx} = \begin{bmatrix} \frac{du_1}{dx_1} & \frac{du_2}{dx_1} & \cdots \\ \frac{du_2}{dx_1} & \ddots & \cdots \\ & & \frac{du_m}{dx_n} \end{bmatrix}$$

So $$p(u) = \frac{p(x)}{\det\left(\frac{df}{dx}\right)}$$

b)

b)
$$I(u) = E[\log p_u(f(Wx))] - \sum_j E[\log p_j(u_j)]$$

$$p_u[f(Wx)] = \frac{p(Wx)}{\det(f'(Wx))} = \frac{p(x)/\det(W)}{\det(J(f))}$$

$$I(u) = E[\log p_u(Wx)] - \log|\det(W)|$$
$$- \sum_j E[\log p_j(u_j)] - \log(\det(J(f)))$$
$$= E[\log p_u(x)] - \log|\det(W)|$$
$$- E[\log(\prod_j p_j(u_j))] - \log[\det(J(f))]$$

since $\log a + \log b = \log ab$ & $E(\sum_{i=1}^{n} x) = \sum_{i=1}^{n} E(x)$

$$\frac{dI}{dw_{ik}} = \frac{-1}{\det(W)} \frac{\partial}{\partial w_{ik}} \det(W)$$
$$- \sum_j \frac{1}{p_j(u_j)} \frac{dp_j(u_j)}{du_j} \frac{\partial u_j}{\partial w_{ik}}$$
$$- \frac{1}{\det(J(f))} \cdot \frac{d}{dw_{ik}} \det(J(f(Wx)))$$
$$= -(W^{-T})_{ik} + \varphi(u_i)x_k - (J(f(Wx)))_{ik}$$

where $\varphi(u_i) = \frac{-1}{p_i(u)} \frac{dp_i(u)}{\partial u}$

$$\Delta W = -\eta \frac{\partial I(u)}{\partial w_{ik}} = \eta (W^{-T} - \phi(u) x^T - J(f(Wx))^{-T}$$

$$= \eta (I - \phi(u) u^T) W^{-T} - J(f(Wx))^{-T}$$

making it computationally efficient we can multiply it by $W^T W$

$$\Delta W = \eta (I - \phi(u) u^T) W - J(f(Wx))^{-T} W^T W$$

c)

c) If we have one hidden layer perceptron we can say the function is sigmoid one

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \frac{d\sigma}{dx} = \sigma(x)(1-\sigma(x))$$

so let's use it in our perceptron

Let $y = Wx$. $m \times n$

$$J(\sigma(y)) = \begin{bmatrix} \frac{\sigma(y_1)}{dx_1} & \frac{\sigma(y_2)}{\partial x_1} & \cdots & \\ \vdots & & \ddots & \\ \vdots & \cdots & & \frac{\sigma(y_m)}{dx_n} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma(y_1)(1-\sigma(y_1)) & & 0 \\ & \ddots & \\ 0 & & \sigma_m(1-\sigma_m) \end{bmatrix}$$

$J(\cdots)^T = J(\cdots)$

as J is diagonal matrix

$$D^{-1} = \frac{1}{|D|} \operatorname{adj}(D)$$

$$\det(J^{T}) = \sum_i \sigma(y_i)(1-\sigma(y_i))$$

Let's make further simplifications & now we have 2×2 matrices

$$J^{-T} = \frac{1}{\sigma(y_1)\sigma(y_2)(1-\sigma(y_1))(1-\sigma(y_2))} \begin{bmatrix} \sigma_2(1-\sigma_2) & 0 \\ 0 & \sigma_1(1-\sigma_1) \end{bmatrix}$$

$$= \begin{bmatrix} [\sigma_1(1-\sigma_1)]^{-1} & 0 \\ 0 & [\sigma_2(1-\sigma_2)]^{-1} \end{bmatrix}$$

so

$$\Delta W = \eta\left(I - \phi(u)u^T - \begin{bmatrix} [\sigma_1(1-\sigma_1)] & 0 \\ 0 & [\sigma_2(1-\sigma_2)]^{-1} \end{bmatrix} W^T\right) W$$

$$\sigma_i = \sigma_i(Wx) = u_i$$

& we get

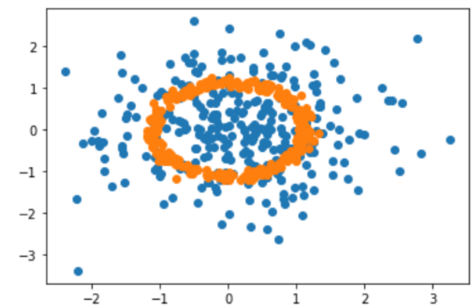
$$W[n+1] = W[n] + \Delta W$$

$$\Delta W = \begin{bmatrix} \Delta W_1 \\ \Delta W_2 \end{bmatrix} = \eta\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} \frac{1}{P_1(u)} \frac{\partial P_1(u_1)}{\partial u_1} \\ \frac{1}{P_2(u)} \frac{\partial P_2(u_2)}{\partial u_2} \end{bmatrix} [u_1, u_2] - \begin{bmatrix} \sigma_1(Wx)(1-\sigma_1(Wx)) & 0 \\ 0 & \sigma_2(Wx)(1-\sigma_2(Wx)) \end{bmatrix}\right) W$$

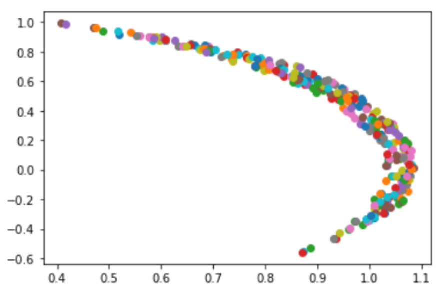
__Problem3__

a) The code is in the notebook provided, it just generates the gaussian and then maps it into the result.

```
z: blue
y: orange
```



b) In this task I used previous code, just modified the perceptron to give two output vector function. However, as previously, it did not work so well and as a result the output is somehow disturbed.



The behavior is similar to the noisy one, and every time it gives different output. So the perceptron does not work properly. Btw, the hidden layer number is set to be 5.

c)

c) $D_{p\|g} = \int p_x(x) \log\left(\frac{p_x(x)}{g_x(x)}\right) dx$

$= E[\log(p_x(X) - \log(g_x(X))]$

We have $N(\mu_0, \Sigma_0)$ & $N(\mu_1, \Sigma_1)$

$N(\mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$

$D = E\left[-\frac{1}{2}\log 2\pi|\Sigma_0| - \frac{1}{2}(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)\right.$
$\left. + \frac{1}{2}\log 2\pi|\Sigma_1| + \frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\right]$

$= E\left[\frac{1}{2}\log \frac{\cancel{2\pi}}{\cancel{2\pi}} \frac{|\Sigma_1|}{|\Sigma_0|}\right] - \frac{E}{2}\left[(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right.$
$+ \frac{E}{2}\left[((x-\mu_1)^T \Sigma_1 (x-\mu_1)\right]$

1 2 3

$(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0) \in R$ so we can write it as

trace: $tr[(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)]$

Using a theorem $tr(ABC) = tr(BCA) = tr(CAB)$

we get a term

$$\frac{1}{2} E \left[tr \{ (x-\mu_0)^T \{ (x-\mu_0)^T \Sigma_0^{-1} \} \right]$$

as expectation is linear function we can interchange:

$$= \frac{1}{2} tr \{ E [(x-\mu_0)(x-\mu_0)^T \Sigma_0^{-1}] \}$$

$$= \frac{1}{2} tr \{ \underbrace{E[(x-\mu_0)(x-\mu_0)^T]}_{\Sigma_0} \Sigma_0^{-1} \}$$

$$= \frac{1}{2} tr \{ \Sigma_0 \Sigma_0^{-1} \}$$

$$= \frac{1}{2} tr \{ I_k \} = \frac{k}{2} \quad (1)$$

Using similar operations for the third term:

$$E[(x-\mu_1)^T \Sigma_1^{-1} (x-\mu_0)] = (\mu_0-\mu_1)^T \Sigma_1^{-1} (\mu_0-\mu_1) + tr \{ \Sigma_1^{-1} \Sigma_0 \} \quad (3)$$

It is done using a theorem (380) in the referenced book

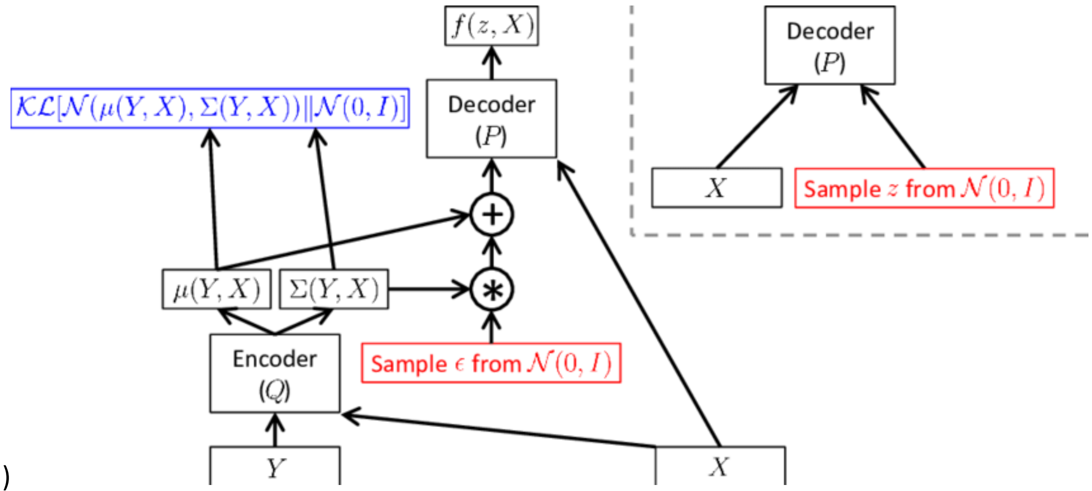
Combining, we get

$$D[N(\mu_0, \Sigma_0) \| N(\mu_1, \Sigma_1)] =$$

$$\frac{1}{2}\left(tr(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - k\right.$$

$$\left. + \log\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right)\right)$$

d)



As always there are two parts, encoder and decoder. Encoder wants to encode Y into its hidden representation while decoder wants to find this hidden representation to decode into the original image. So, both encoder Q and decoder P are neural networks that we aim to train to. Q takes Y and produces some distribution which is gaussian for simplicity, thus Q needs to find parameters average and variance of the input and at the same time wants it to be close to N(0, I) that's why its divergence with the encoded part is measured. After that, the samples are transferred to the decoder that wants to use these hidden features to decode the input. As the transferred sample are hard to train, the reparameterization trick is used instead, instead of samples, the parameters are given to P and using the normal function this data is reproduced again. Than feed forward part for the decoder occurs and result is given. The conditional variational encoder helps to control the learning process by introducing new conditional variable X to both Q and P, after that the distribution produced depends on the condition of X so we have P(z|X) instead of P(z), Q(z|Y,X) instead of Q(z|Y) etc. In other words, let's say, given an input *X*(label of the image) we want our generative model to produce output *Y*(image). So, the process of VAE will be modified as the following: given observation *y, z* is drawn from the prior distribution *P(z|y*), and the output **x** is generated from the distribution *P(x|y, z)*. X is like the label of the image which makes the learning resemble supervising one.

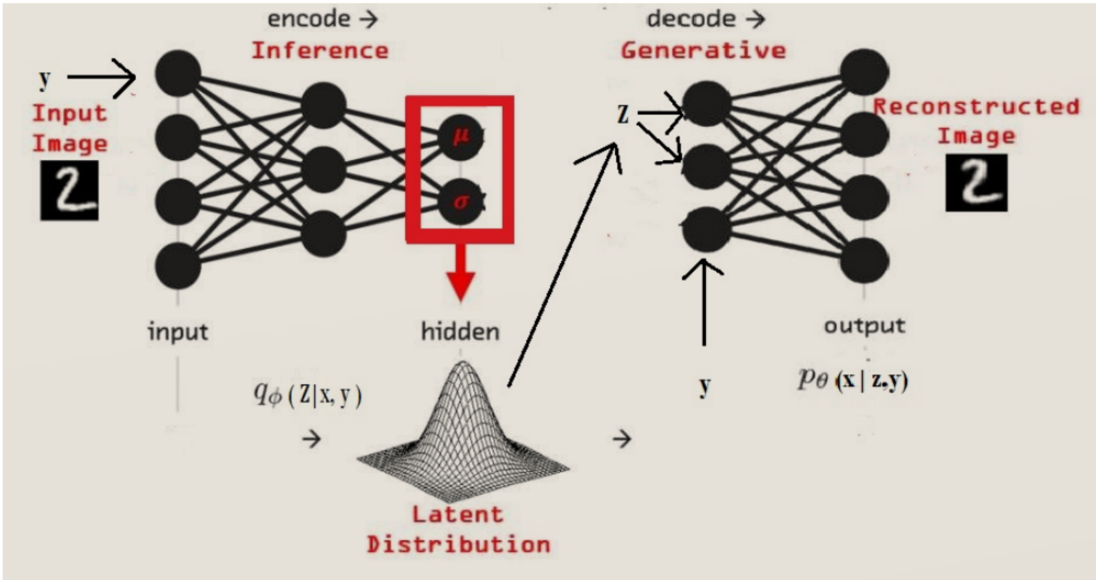
To make the back propagation we need to define the objective function that we would like to minimize/maximize. This function will help to derive the loss function which will be used in the next section for deriving the learning rules. First of all, we want the input and the decoded part to be the same, i.e. we would like to minimize the information is lost during the decoding process. It can be achieved by using KL divergence between Q(z|Y,X) and P(z|Y,X). Another thing is we would like to maximize the distribution of the resultant pdf. The greater the distribution of data, the more information, the larger is capability of the model is for the decoder to decode different data. As a result, objective function is

max E

E = log P(Y|X) – D[Q(z|Y,X) || P(z|Y,X)]

For deriving the learning rule, the next step is make the loss term from this function and feed it to the model do make the update process.

e) The process can be explained by picture below


encode →
Inference
decode →
Generative
y
Input
Image
Z
Reconstructed
Image
μ
σ
input
hidden
output
y
$q_\phi(Z|x, y)$
$p_\theta(x | z,y)$
Latent
Distribution

d) $\log P(Y|X) - D[Q(z|Y,X) \| P(z|Y,X)]$

$= E_{z\sim Q(\cdot|Y,X)}[\log P(Y|z,X)] - D[Q(z|Y,X)$

$\| P(z|X)]$

lower bound

This is the function we would like to maximize, the loss function needs to be minimized, The lower bound of objective function is then

$$\max -E[\log P(Y|z,X)] + D[Q(z|Y,X) \| P(z|X)]$$

As our model wants $P(z|X)$ to be close to $N(0,I)$ we may assume $P(z) \sim N(0,1)$ then $D[\dots]$ has a closed form solution.

Putting it together

$$E_{z\sim Q(z|X,Y)} \log P(Y|z,X) \propto \|Y - f(z,X)\|^2$$

as the perfomance of decoder strongly

depends on the result of neural network
To maximize the function, we would like to minimize its lower bound so now it can have larger range for the following conditions. So we can write loss as

$$\mathcal{L}_e = \|X - f(z|x)\|^2 - \lambda D\,[Q(z|x)\|P(z|x)]$$

where the 1st part is pixel difference between input & output and the second one is regularization term that avoids overfitting & λ is parameter $0 \le \lambda < 1$
This is tricky as we want to minimize D, however when training, this term needs to be high in order to the model to not stick to the parameters provided & be

flexible.

So generally we can write as

nonlinear function

$$Q(z|Y,X) = \sigma(W_n \sigma(\cdots \sigma(W_1 X)))\,]\ldots)$$

$$f(z,X) = \sigma(W_n \sigma(\ldots \sigma(W_1\, Q(z|Y,X)\ldots)$$

$$L_e = \|X - f(z,X)\|^2 - \lambda D\,[Q(z,X)\,\|\,P(z,X)]$$

$$\Delta W_P = -\eta\, L_e \qquad 0<\eta<1$$

$$\Delta W_Q = -\eta\, L_e$$

$$W[n+1] = W[n] + \Delta W$$

The rules defined above cannot explain the learning process well as they are not the algorithm. So I will also provide small pseudocode below[3]:

Given a dataset of examples Y = {Y1, Y2...}

Initialize parameters for Encoder and Decoder

Repeat till convergence:

  Y^M <-- Random minibatch of M examples from Y under condition X

  ε <-- Sample M noise vectors from N(0, I)

Compute L(Y^M , ε, θ, X) (i.e. run a forward pass in the neural network)

Gradient descent on L to updated Encoder and Decoder.

Reference

[1] https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

[2] https://towardsdatascience.com/understanding-conditional-variational-autoencoders-cd62b4f57bf8

[3] https://slazebni.cs.illinois.edu/spring17/lec12_vae.pdf