# Machine Learning Homework 3 Solution

Yang Rongfeng 20644943
HKUST MSBD 5012 Machine Learning

*Update: November 11, 2019*

## 1 Question 1

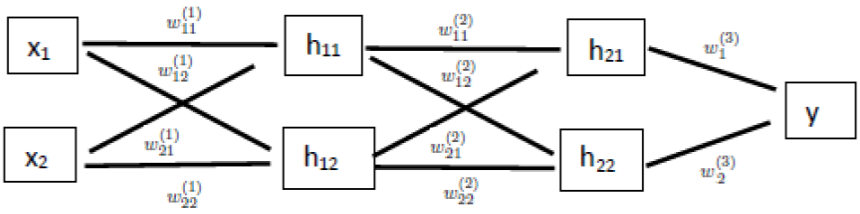



**Figure 1:** A feedforward neural network

(a) According to the initail values that the question is given. When feeding $(x_1, x_2) = (1, 2)$ to the network. The outputs of the hidden units

$$
\begin{aligned}
h_{11} &= x_1 w_{11}^{(1)} + x_2 w_{21}^{(1)} = 1 \times 1 + 2 \times (-1) = -1 \\
\mu_{11} &= ReLU(h_{11}) = 0 \\
h_{12} &= x_1 w_{12}^{(1)} + x_2 w_{22}^{(1)} = 1 \times (-1) + 2 \times 1 = 1 \\
\mu_{12} &= ReLU(h_{12}) = 1 \\
h_{21} &= \mu_{11} w_{11}^{(2)} + \mu_{21} w_{21}^{(2)} = 0 \times (-1) + 1 \times 1 = 1 \\
\mu_{21} &= ReLU(h_{21}) = 1 \\
h_{22} &= \mu_{11} w_{12}^{(2)} + \mu_{12} w_{22}^{(2)} = 0 \times (-1) + 1 \times 1 = 1 \\
\mu_{22} &= ReLU(h_{22}) = 1
\end{aligned}
$$

Therefore, the logit is

$$
\sigma(z) = \sigma(\mu_{21} w_1^{(3)} + \mu_{22} w_2^{(3)}) = \sigma(1 \times 1 + 1 \times 1) = \sigma(2) = \frac{e^2}{e^2 + 1} = 0.88
$$

The output distribution of the output units

$$P(y \mid x_1 = 1, x_2 = 2, \theta) = P(y \mid z = 2) = \begin{cases} \sigma(2), y = 1 \\ \sigma(-2), y = 0 \end{cases} = \sigma((2y-1)\cdot 2) = \sigma(4y-2)$$

(b) The given loss function can be written to

$$L = -logP(y=0|z) = -log\sigma(-z) = -log(\frac{1}{1+e^z}) = log(1+e^z)$$

So the derivative of error of the output unit

$$\frac{\partial L}{\partial z} = \frac{\partial}{\partial z} log(1+e^z) = \frac{e^z}{1+e^z} = \sigma(z)$$

Then the derivative error of the hidden units

$$\delta_{21} = \frac{\partial L}{\partial h_{21}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial \mu_{21}}\frac{\partial \mu_{21}}{\partial h_{21}} = w_1^{(3)} = \sigma(2) = 0.88$$

$$\delta_{22} = \frac{\partial L}{\partial h_{22}} = \frac{\partial L}{\partial \mu_{22}}\frac{\partial \mu_{22}}{\partial h_{22}} = w_2^{(3)} = \sigma(2) = 0.88$$

$$\delta_{11} = \frac{\partial L}{\partial h_{11}} = \delta_{21}\frac{\partial h_{21}}{\partial \mu_{11}}\frac{\partial \mu_{11}}{\partial h_{11}} + \delta_{22}\frac{\partial h_{22}}{\partial \mu_{11}}\frac{\partial \mu_{11}}{\partial h_{11}}$$

Because $\frac{\partial \mu_{11}}{\partial h_{11}} = 0$ then $\delta_{11} = 0$

$$\delta_{12} = \frac{\partial L}{\partial h_{12}} = \delta_{21}\frac{\partial h_{21}}{\partial \mu_{11}}\frac{\partial \mu_{11}}{\partial h_{11}} + \delta_{22}\frac{\partial h_{22}}{\partial \mu_{12}}\frac{\partial \mu_{12}}{\partial h_{12}} = 2\sigma(2) = 1.76$$

We use the chain rule, we can get

$$\frac{\partial L}{\partial w_{22}^{(2)}} = \delta_{22}\cdot\frac{\partial h_{22}}{\partial w_{22}^{(2)}} = \delta_{22}\cdot\mu_{12} = \sigma(2)\times 1 = 0.88$$

$$\frac{\partial L}{\partial w_{22}^{(1)}} = \delta_{12}\cdot\frac{\partial h_{12}}{\partial w_{22}^{(1)}} = \delta_{22}\cdot x_2 == 2\sigma(2)\times 2 = 4\sigma(2) = 3.52$$

If we want to reduce the loss on the example

$$\frac{\partial L}{\partial w_{22}^{(2)}} = \delta_{22}\cdot\mu_{12} \Rightarrow \delta_{22} > 0, \mu_{12} > 0 \Rightarrow \text{Decrease the parameter of } w_{22}^{(2)}$$

$$\frac{\partial L}{\partial w_{22}^{(1)}} = \delta_{12}\cdot x_2 \Rightarrow \delta_{12} > 0, x_2 > 0 \Rightarrow \text{Decrease the parameter of } w_{22}^{(1)}$$

## 2 Question 2

The sigmoid units are not recommeded for hidden units because they saturates across most of their domains. The function curve becomes flat and easily causes gradient disappearing. However, they are fine as the output units because the negative log-likelihood helps to prevent the problem.

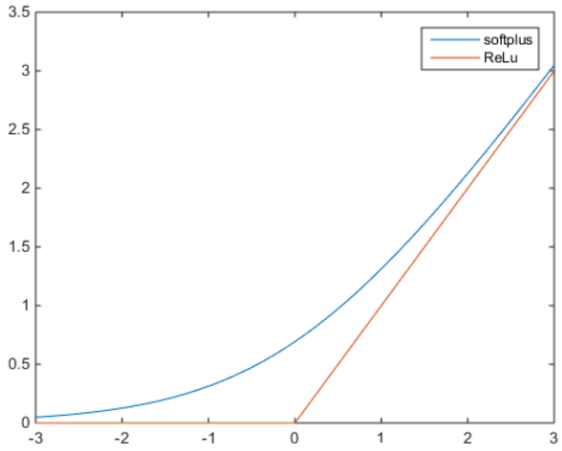



**Figure 2:** Softplus and ReLU

## 3 Question 3

Dropout is used with minibatch-based algorithms such as SGD in deep learning. For each minibatch, dropout first randomly sample values for mask variables, and then carry out one step of gradient descent (i.e. randomly make some units not updated). Actually, Dropout is a kind of **regularization** method. It reduces overfitting by preventing complex co-adaptations of parameters on training data.

## 4 Question 4

Adam algorithm is roughly a combination of RMSProp and momentum, with bias correction.

**SGD** maintains a single learning rate for all weight updates and the learning rate does not change during training. **Momentum method** brorrows the concept of *momentum* in Pysics. By accumulating momentum, it reduces the *Zigzag effect* during gradient descent and accelerates the process of converging to the minimum. **RMSProp** also try to reduce the *Zigzag effect* but using the accumulation of Root Mean Square of the gradient and it converges faster and have less fluctuation than Momentum method. However, this also means that the RMSProp prevents exploring in the direction of vibration.

Therefore, it's the key idea of Adam method that combining both of them should have a better performance. In figure 2, we can see Adam uses Momentum and RMSProp to update the first and the second moment estimate and applies bias correction after that.

**Algorithm 8.7** The Adam algorithm

**Require:** Step size $\epsilon$ (Suggested default: 0.001)
**Require:** Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
**Require:** Small constant $\delta$ used for numerical stabilization. (Suggested default: $10^{-8}$)
**Require:** Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  **while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$ Momentum
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$ RMSProp
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  **end while**

**Figure 3:** The Adam algorithm

# 5 Question 5

According to the question. Set the INPUT $W_2 \times H_1 \times D_1$, $K$ number of $F \times F$ filters, stride= $S$, zero padding= $P$. The OUTPUT $W_2 \times H_2 \times D_2$

The output

$$W_2 = \frac{W_1 + 2P - F}{S} + 1 = 27 - 3 + 1 = 25$$
$$H_2 = \frac{H_1 + 2P - F}{S} + 1 = 27 - 3 + 1 = 25$$
$$D_2 = K = 384$$

Therefore the shape of the output layer is $25 \times 25 \times 384$. And because we have 384 number of $3 \times 3$ filters, the number of parameters

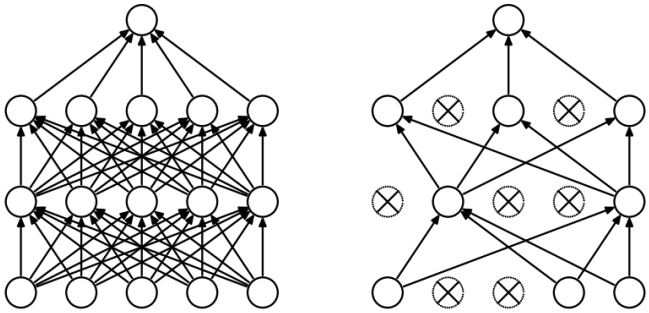
$$n = (FFD_1 + 1) \cdot K = (3 \times 3 \times 256 + 1) \times 384 = 885,120$$

The number of float **multiplication** operations it takes

$$m = FFD_1 \times W_2H_2D_2 = (3 \times 3 \times 256) \times (25 \times 25 \times 384) = 552,960,000$$

# 6 Question 6

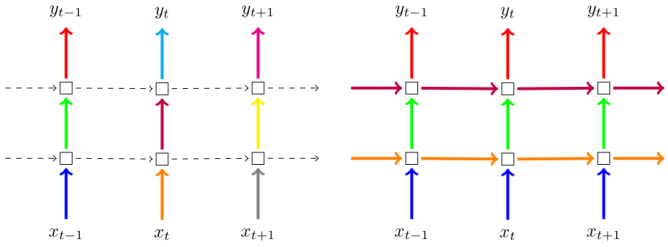
(a) Generally, dropout is a widely used regularization method in deep learning including CNN. The dropout just apply on Fully-connected layer because the number of parameters is large but not on convolutional layer for its much smaller number of parameters.



(a) Standard Neural Net (b) After applying dropout.

**Figure 4:** Apply dropout in fully-connected layer

(b) Talking about dropout in RNN may be a little bit complicated. We look at the figure 4. **The naive dropout RNN** method was proposed by Zaremba et al. (2014) and Pham et al. (2013). They applied dropout only to the non-recurrent connections and thought it's not a good idea to apply dropout to recurrent layers. This is because noise will be amplified for long sequences and drown the signal.




(a) Naive dropout RNN (b) Variational RNN

**Figure 5:** Comparison of naive droupout RNN and variational RNN. Colours representing dropout masks, solid lines representing dropout, dashed lines representing standard connections with no dropout).

However, Gal and Ghahramani (2015) found this approach would cause overfitting after analysing. They proposed **variational dropout** where by repeating the same dropout mask at each time step for both inputs, outputs, and recurrent layers (drop the same network units at each time step) using a bayesian interpretaton. And it performed better than naive dropout method in NLP.

# 7 Question 7

(a) For image classification, the adversarial attack is intentionally adding small and special perturbations to image to fool the model. It makes the classifer completely change its original prediction about the image, with high confidence.

(b) $Z_t(x'), Z_i(x')$ represent the logits of class $t$ and class i respectively. The CW attack is a targeted and strong attack. The optimization problem can be written as

$$min_{x'}[c||x - x'||_2^2 + max\{max_{i \neq t} Z_i(x') - Z_t(x'), -\kappa\}]$$

Focus on the second part, if let $\delta = Z_t(x') - max_{i \neq t} Z_i(x')$

$$max\{max_{i \neq t} Z_i(x') - Z_t(x'), -\kappa\} = max\{-\delta, -\kappa\} = \begin{cases} -\delta, \delta < \kappa \\ -\kappa, \delta \geq \kappa \end{cases}$$

Consider minimizing the $max\{-\delta, -\kappa\}$. When $\delta < \kappa$, it's equivalent to maximize $\delta$ (minimize $-\delta$). This means we maximize the difference between the logits of class $t$ and the second best. In other words, we want the probability of class $t$ be as high as possible relative to other classes. When $\delta \geq \kappa$, we are confident that $x'$ is an adversarial example, and hence turn our attentions to minimize $||x - x'||_2^2$

# 8 Question 8

(a) $f_t$ determines which components of the previous state and how much of them to remember/forget. $i_t$ determines which components of the input from $h^{(t-1)}$ and $x^{(t)}$ and how much of them should go into the current state.

(b) Sigmoid activation function is used for the gates so that their values are often close to 0 or 1. However, tanh is used for the output $h^{(t)}$ and memory cell $c_t$ so as to have strong gradient signal during back propagation.