# Course Project

徐薛楠，周之恺，马达，吕波尔

December 2019

## 1 Brief Introduction

This report is to show the results of brain-like intelligence course project including 7 topics in total. Our group takes 4 of them, topic *A, B, C* and *D*. The former 3 researches on the effect of some basic hyperparameters and the later one pays attention to the bi-directional learning to the smoothness of the loss function. Finally, we show our conclusion according to the experiments. Also, the code is public on the github[1].

## 2 Topic A B C

### 2.1 Main Ideas

These topics are about how hyperparameters affect the smoothness of loss functions when we use neural networks[3]. A, B and C study sample size, depth of network and number of iterations respectively.

To better visualize the experimental results, we decide to do some experiments on a simple binary classification problem. But these problems are sometimes easy for neural network to learn. In this case, we choose the binary spiral classification problem as the final problem in which the training data can be easy to generate and the sample mixing is more complicated.

In this project, we use the Feedforward Neural Network(FNN)[9] with a RELU activation function and change the number of these hyperparameters

[1]Our code is available at https://github.com/311dada/brain-like-AI-Proj

individually to control variables. We draw the boundaries of the classification and the curves of the loss function to observe the change and make the conclusion.

## 2.2 Methods

The binary spiral line can be generated with the help of mathematical formula easily[10]. We simply apply neural network to classify it. The smoothness of the loss function can be evaluated through visualization.

### 2.2.1 Dataset

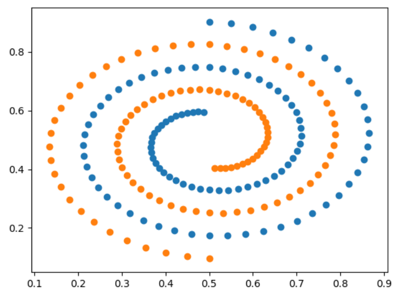


Figure. 1: The generated binary spiral data

Figure 1 shows the generated samples according to the following formula.

Denote

- $(x_{0,i}, y_{0,i})$: the $i$-th point of the first spiral
- $(x_{1,i}, y_{1,i})$: the $i$-th point of the second spiral
- $n$: the points number of each spiral
- $d$: the half distance of the start points of the two spirals
- $r$: the radius of two spirals
- $s$: the start position of the two spirals

and $c$ is the circle number. Then

$$\alpha = \frac{\pi \cdot i}{\frac{n}{2c}}$$

$$r_i = r * \frac{n+d-i}{n+d-1}$$

$$x_{0,i} = r_i * \sin\alpha + s$$

$$y_{0,i} = r_i * \cos\alpha + s$$

$$x_{1,i} = -r_i * \sin\alpha + s$$

$$y_{1,i} = -r_i * \cos\alpha + s$$

### 2.2.2 The Evaluation Criteria

We can visualize the relation between loss and iteration number. However, the loss cannot be sampled randomly. If we employ the average loss of each batch, the fluctuation is serious due to the large differences among the batch distributions. So we sample the average loss of a certain number of batches. To guarantee the fairness, the sampling number for each condition must be the same.

## 2.3 Algorithms

To evaluate the difficulty of jumping out of the local optimum, the learning rate should be fixed. Because the adjustment of learning rate for different models is different. If the model jumps out of the local optimum, we cannot judge if it happens due to the learning rate. So we use *stochastic gradient descent* (SGD)[1] algorithm to train our model.

SGD selects a few samples called a batch from the training set randomly to update the gradient of the parameters of the model. It accelerates the training speed and optimize the model at the same time. Also, it meets our demand of fixing the learning rate.

## 2.4 Experimental Settings

For the data, we set $s = 0.5, d = 100, c = 2, r = 0.8$ for all the experiments. For the model, we set the neuron number of hidden layer 100 and learning rate 0.1 for all the experiments. For the loss, we sample once 100 batches.

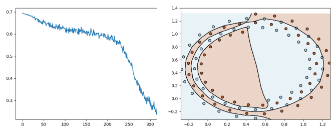
For topic A, we should fix all the hyperparameters except the data number. The data number varies in $\{100, 200, 500, 1000, 10000, 100000\}$. The iteration number is fixed to 30000.

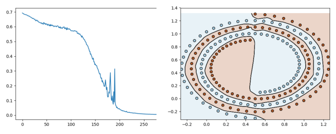
For topic B, we should fix all the hyperparameters except the depth. The depth varies in $\{2, 3, 4, 5\}$. The iteration number is fixed to 95000 and the data number to 1000.

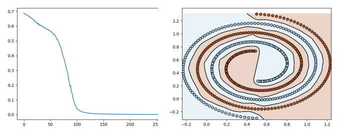
For topic C, we should fix all the hyperparameters except the iteration number. The epoch number varies in $\{10, 100, 500, 1500\}$. The depth is fixed to 3 and the data number to 1000.

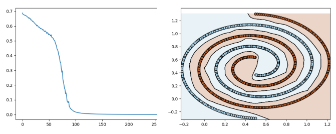
## 2.5 Results

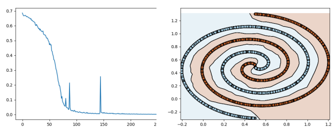
### 2.5.1 topic A


(a) 100


(b) 200


(c) 500


(d) 1000


(e) 10000

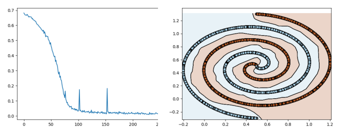
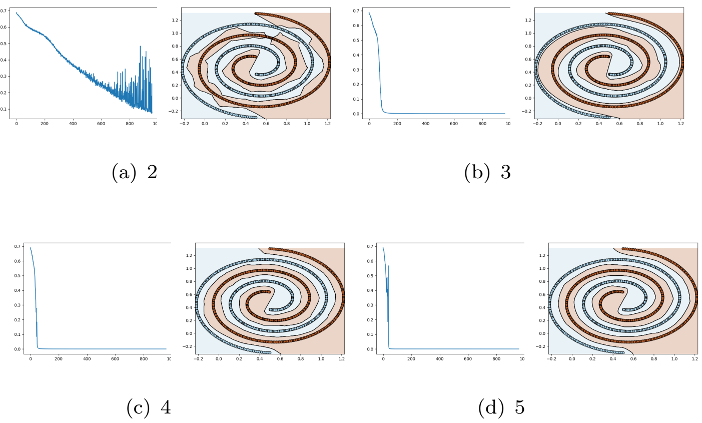

(f) 100000

Figure. 2: The average loss and decision boundary for different data numbers

Figure 2 shows the results. The average loss fluctuates seriously when the sample size is small. As the sample size increases, the model converges earlier. However, some more local optimums appear if the sample size is too large. When the sample size increases but not so large, the distribution of the train dataset is more accurate. So the average loss curve is more stable and the model reaches the global optimum earlier. However, the distribution is stable when the dataset is large enough. But the samples who can lead the model to the local optimum are always increasing. Due to the fixed batch size, the model meets the batch including these samples more often which leads to the fluctuation. So if the sample size is too large, there will be more local optimums.

#### 2.5.2 topic B


(a) 2

(b) 3

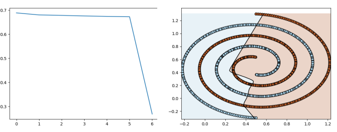
(c) 4

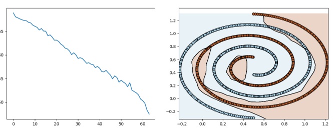
(d) 5

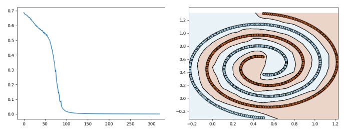
Figure. 3: The average loss and decision boundary for different depths

Figure 3 shows the results. The average loss fluctuates seriously near the global optimum with shallow depth. Similarly, the average is more smooth as the depth increase but with small fluctuation when the depth is too deep. Obviously, the deeper the model is, the stronger the fitting ability of the model is. But too many parameters may cover some dirty parameters who take the responsibility of fitting the data which can cause local optimum. So some small fluctuations appear when the depth of the model is too deep.

#### 2.5.3 topic C


(a) 10


(b) 100


(c) 500

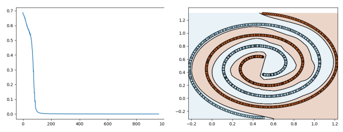

(d) 1500

Figure. 4: The average loss and decision boundary for different iteration numbers

Figure 4 shows the results. When the model converges, the iteration number has no influence. Because the model has reached the global optimum already.

## 3 Topic D

### 3.1 Main Ideas

Topic D is about whether bi-directional learning is helpful to overcome the problem of getting stuck in the local optimum. Owing to our research interest, we do some experiments on acoustic models.Acoustic model is a very important part in speech recognition. Despite the frontier research about the end-to-end automatic speech recognition(ASR) system, acoustic model is always a significant part in many application scenarios. The performance of acoustic model is the key to build an ASR system. In this project, we study how to improve the ability of acoustic model to classify the phoneme.

## 3.2 Methods

We compare the performance of auto-encoder[4] on phoneme classification task with deep neural networks who has the same architecture with the encoder part of auto-encoder in order to study how bi-directional learning is helpful to overcome the problem of getting stuck in the local optimum. Furthermore, we visualize the bottleneck feature of auto-encoder by using principle component analysis to investigate the ability of auto-encoder to classify phonemes.

### 3.2.1 A glimpse of speech recognition

As we can see in figure, speech recognition systems take speech waveform as input and output the words and sentences. The front end processing like feature extraction transforms speech waveform into vectors. And then acoustic models recognize the vectors and output phoneme sequence. At last language model with lexicon turns phoneme into word sequence. The phoneme recognition is a part of automatic speech recognition. In addition, we have about 171476 English words but only 39 phonemes in English pronunciation. So using phoneme to build an acoustic model is much more feasible.

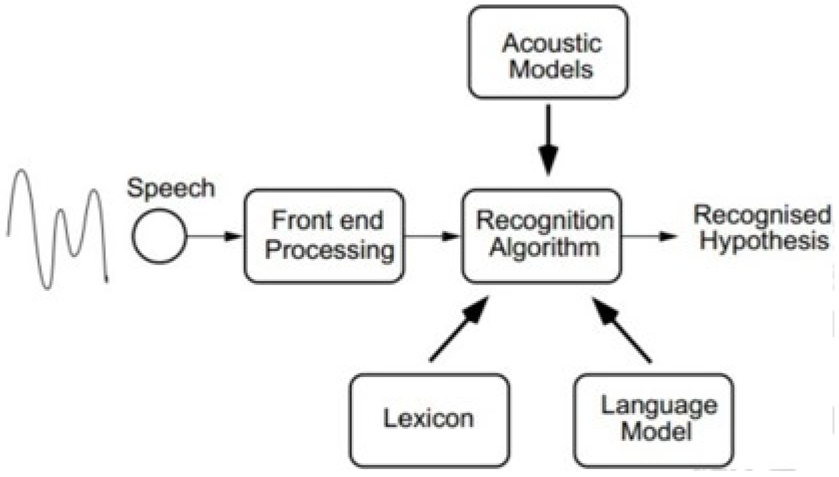
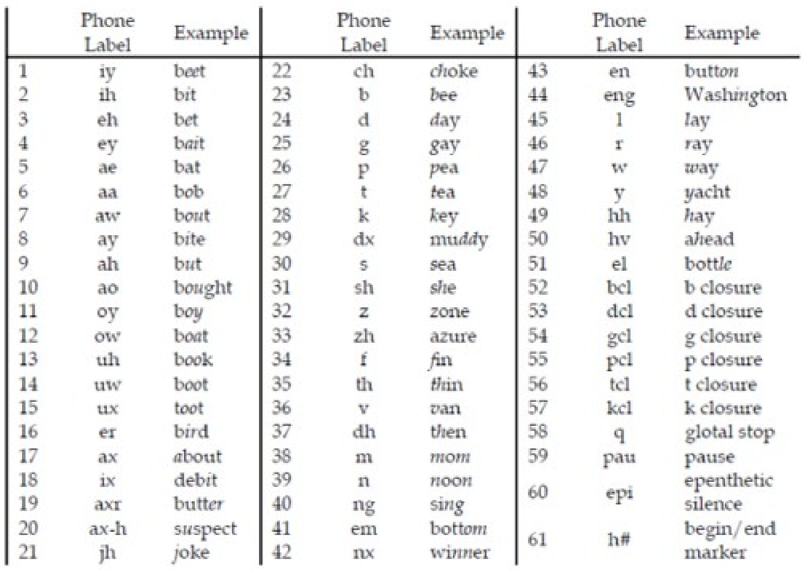


Figure. 5: A sketch of automatic speech recognition system

### 3.2.2 Label

Generally speaking, we only have audio waveform and transcription from audio datasets. In common recipe, a GMM-HMM model will be trained using Baum-Welch algorithm ( A kind of EM). Then the label of each frame (i.e. alignment) can be estimated by viterbi algorithm. In our project, the alignment is obtained in this way by LDA+MLLT recipe. The force alignment in this way is proved to be more accurate and of course much more efficient than human annotation when data and model are both considerable. The model we build in our project is much smaller.



| | Phone Label | Example | | Phone Label | Example | | Phone Label | Example |
|---|---|---|---|---|---|---|---|---|
| 1 | iy | b*ee*t | 22 | ch | *ch*oke | 43 | en | butt*on* |
| 2 | ih | b*i*t | 23 | b | *b*ee | 44 | eng | Wash*ing*ton |
| 3 | eh | b*e*t | 24 | d | *d*ay | 45 | l | *l*ay |
| 4 | ey | b*ai*t | 25 | g | *g*ay | 46 | r | *r*ay |
| 5 | ae | b*a*t | 26 | p | *p*ea | 47 | w | *w*ay |
| 6 | aa | b*o*b | 27 | t | *t*ea | 48 | y | *y*acht |
| 7 | aw | b*ou*t | 28 | k | *k*ey | 49 | hh | *h*ay |
| 8 | ay | b*i*te | 29 | dx | mu*dd*y | 50 | hv | a*h*ead |
| 9 | ah | b*u*t | 30 | s | *s*ea | 51 | el | bott*le* |
| 10 | ao | b*ou*ght | 31 | sh | *sh*e | 52 | bcl | b closure |
| 11 | oy | b*oy* | 32 | z | *z*one | 53 | dcl | d closure |
| 12 | ow | b*oa*t | 33 | zh | a*z*ure | 54 | gcl | g closure |
| 13 | uh | b*oo*k | 34 | f | *f*in | 55 | pcl | p closure |
| 14 | uw | b*oo*t | 35 | th | *th*in | 56 | tcl | t closure |
| 15 | ux | t*oo*t | 36 | v | *v*an | 57 | kcl | k closure |
| 16 | er | b*ir*d | 37 | dh | *th*en | 58 | q | glotal stop |
| 17 | ax | *a*bout | 38 | m | *m*o*m* | 59 | pau | pause |
| 18 | ix | deb*i*t | 39 | n | *n*oo*n* | 60 | epi | epenthetic silence |
| 19 | axr | butt*er* | 40 | ng | si*ng* | | | |
| 20 | ax-h | s*u*spect | 41 | em | bott*om* | 61 | h# | begin/end marker |
| 21 | jh | *j*oke | 42 | nx | wi*nn*er | | | |

Figure. 6: Sixty-one phonemes used in TIMIT records

### 3.2.3 Training Pipeline

**The overall structure** The whole structure of our neural network is depicted in figure. We have two stages. The first stage is to train the auto-encoder. The second stage is to train the classifier with the same structure with encoder.

**Neural Network** The calculation process for each layer is in following formula:

$$y = ReLU(BatchNorm(Wx + b))$$

$$ReLU(x) = max(0, x)[7]$$

BatchNorm[5] means batch normalization, which makes data follow normal Gaussian Distribution.

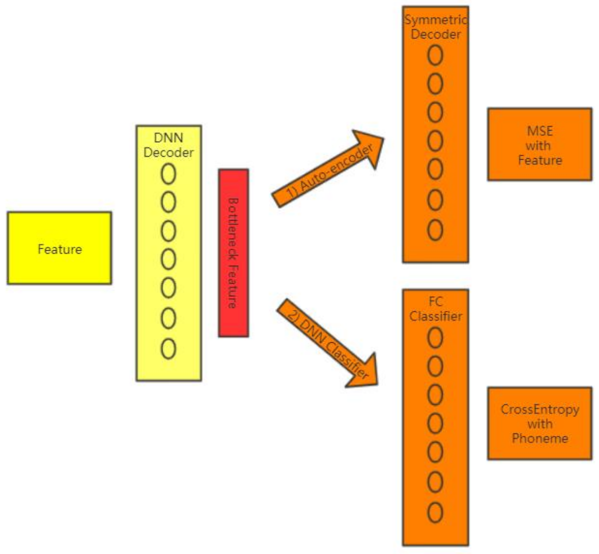


Figure. 7: The overall NN structure

**Auto-encoder** The auto-encoder has two parts: encoder and decoder. They have the symmetric structure and do not have activate function in bottleneck layer. The auto-encoder is trained in mean square error criterion with feature itself.

**DNN Classifier** DNN classifier share the same structure in encoder with the auto-encoder. The full-connected classifier follows the encoder for further inference. The training configuration can be divided into three cases. The first one is a baseline. All of its parameters are randomly initialized.

The second one and the third one are both using transfer learning. Their encoder are initialized by the parameters of encoder of AE. Their difference is whether fixing the parameter of encoder during training stage.

## 3.3 Experimental Settings

### 3.3.1 Dataset Preparation

We utilizes the TIMIT acoustic-phonetic corpus[2, 11] as the experimental dataset. It contains broadband recordings of phonetically-balanced read speech. There are 630 speakers and 6300 utterances in total, with 10 sentences presented for each speaker. Each utterance comes with time-aligned phonetic and word transcriptions. The speech files are all in raw waveform which are downsampled to 16 kHz. We follow Kaldi[8]'s TIMIT recipe to split train/dev/test subsets to obtain 462/50/24 non-overlapping speakers in each subset.

### 3.3.2 Preprocessing

We use Mel-Frequency Cepstrum Coefficients (MFCC) as the audio frame representations. Kaldi is also used to extract features. A 25ms Hanning window with 10ms window shift is applied to compute Short Time Fourier Transformation (STFT), Mel Filtering and Discrete Cosine Transformation (DCT). Finally, we obtain a 39-dimensional feature for each frame. For each utterance, we compute a T * F cepstrum coefficients as the model input. Then a GMM is trained to give the phoneme labels as the ground true labels. There are 51 unique labels in total, including 48 normal phonemes in TIMIT, <sience>, <unk> and <noise> so it is a classification problem with 51 classes in target.

### 3.3.3 Model Architecture

As what a neural network does on datasets like MNIST, the T * F input cepstrum is first stretched to a one-dimensional feature vector. Then the vector is fed to the autoencoder to reconstruct itself and we use the reconstruction loss to train the autoencoder. To evaluate the influence of bidirectional learning, we extract latent representations from the encoder and use another DNN classifier to classify phonemes with the latent features as the input. We incorporate a simple DNN here as the encoder architecture. The decoder utilizes the symmetric structure of the encoder. The hidden unit numbers of each layer in encoder and classifier are 256 -> 128 -> 64 and 64 -> 128 -> 256 -> 512 -> 256 -> 128 -> 64.

#### 3.3.4 Experimental Setups

All training is conducted with Adam[6] optimization algorithm with a batch size of 2048. We set the initial learning rate $\alpha$ as $10^{-3}$ and default hyperparameters $\beta_1$ (0.9) and $\beta_2$ (0.999) provided by PyTorch Framework. Early stop training strategy with patience 10 is adopted, which means the training is terminated the the loss on the development subset does not improve for 10 epochs.

## 3.4 Results

### 3.4.1 Classification Performance

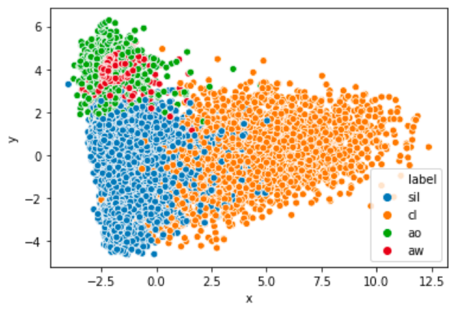
The phoneme classification accuracies on the test set given by different models are listed in Table 1. The three setups refer to random initilizing encoder, initializing encoder with pretrained autoencoder while fixing encoder parameters and adjusting parameters during training. The autoencoder training does not show obvious improvement on classification accuracy. Accuracy on AE initialized with fixed parameters even drops a little. This may attribute to the fixed parameters. The autoencoder training is not supervised by phoneme labels so it may contain information uncorrelated with phonemes. When the encoder parameters are adjusted during following training, the accuracy improves a little.

| Setup | random initialized | AE initialized fixed | AE initialized adjusted |
|---|---|---|---|
| Acc | 68.05% | 67.22% | 68.25% |

Table. 1: Classification Results

### 3.4.2 Visualization

To take a look at what the encoder learns from bidirectional learning, we extract latent features from the encoder and reduce the dimension to 2 for visualization. The features extracted by encoder of the pretrained autoencoder and random initialized dnn are shown in Section 3.4.2. We select four representative phoneme labels, i.e., <silence>, cl, ao and aw to represent corresponding latent representations. The figure indicates that pretrained autoencoder separates ao, aw and <silence> more evidently than random initialized models. This verifies the feature learning effect of the autoencoder, which may be useful in more potential applications like phoneme generation and conversion.



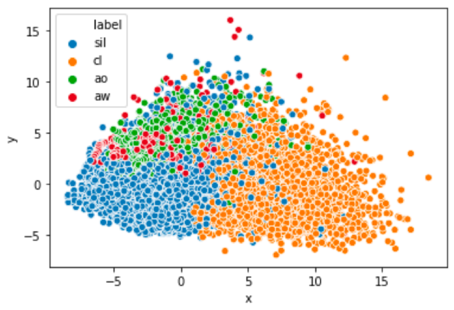


Figure. 8: Autoencoder extracted features

Figure. 9: Random initialized DNN extracted features

## References

[1] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[2] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1. Technical report, Feb 1993.

[3] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[4] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[8] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

[9] Terence D Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks*, 2(6):459–473, 1989.

[10] Wenbo Zhao, De-Shuang Huang, and Ge Yunjian. The structure optimization of radial basis probabilistic neural networks based on genetic algorithms. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No. 02CH37290)*, volume 2, pages 1086–1091. IEEE, 2002.

[11] Victor Zue, Stephanie Seneff, and James Glass. Speech database development at mit: Timit and beyond. *Speech communication*, 9(4):351–356, 1990.