Hyperparallel Machine Learning and Big Data Mining Project

Hongyi Guo 516030910306

June 26, 2019

Abstract

This is the project of course 'Hyperparallel Machine Learning and Big Data Mining'. Our task is to process EEG data and decide which emotions they belong to. To achieve that, I implemented four kinds of machine learning methods: Fully-Connected Network, Min-Max Modular Network, CNN and Bi-RNN. In the following parts, I will introduce these models, the dataset we use, the model implementation, my experiments, conclusion and finally, references.

1 Models

In this part, I will introduce four models I used in brief.

1.1 Fully-Connected Network

A fully-connected neural network consists of a series of fully connected layers. A fully connected layer is a function from $\mathcal{R}^m$ to $\mathcal{R}^n$. Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron neural network (MLP).

A multi-layer perceptron (MLP) is a class of feed-forward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back-propagation for training.Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Let $x \in \mathcal{R}^m$ represents the input to a fully connected layer. Let $y_i \in \mathcal{R}$ be the i-th output from the fully connected layer. Then $y_i \in \mathcal{R}$ is computed as follows:

$$y_i = \sigma(w_1x_1 + w_2x_2 + \cdots + w_mx_m)$$

is a nonlinear function called activation function. With activation function, we can fit more complex models.

1.2 Min-Max Modular Network

The Min-Max Modular Network is first introduced by [3]. It is used to overcome the linear inseparable Two Spirals Problem.

1.2.1 Two-class Min-Max Modular Neural Network

Let's consider the two-class problem first, which can be further divided into $N_i \times N_j$ smaller and simpler two-class problems (called atom problems) where N_i and N_j are the numbers of training subsets belonging to C_i(class i) and C_j(class j) respectively. For each atom problem, we train a fully-connected neural network to solve it just like in section **??** except for the output dimension (changed from 4 to 2).

Next, we use Min-Max method to combine these atom problems and get the solution for the two-class problem as illustrated in 1.

- The modules, which were trained on the data sets which have the same training inputs corresponding to desired output 1, should be integrated by the MIN unit, whose transfer function is

$$q = Minimize\{p_1, \cdots, p_n\}$$

- The modules, which were trained on the data sets which have the same training inputs corresponding to desired output 0, should be integrated by the MAX unit, whose transfer function is

$$q = Maximize\{p_l, \cdots, p_n\}$$

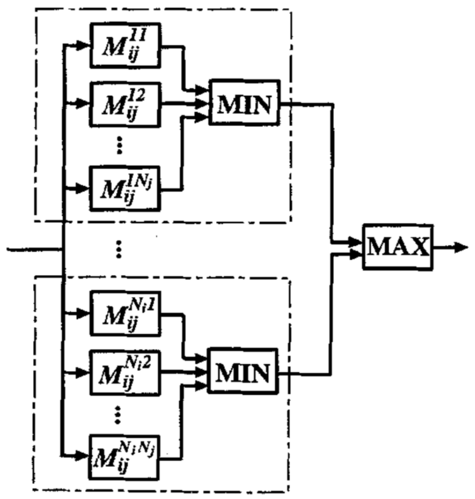


Figure 1: The organization of the $N_i \cdot N_j$ modules by using the MIN and MAX units

Note that all the atom problems are independent, so they can be solved in parallel.

After the MAX module, we get the prediction of the two-class problem. In the next part, I will introduce how to decompose a multi-class problem into two-class problems.

1.2.2 Task Decomposition in a Random Manner

For a k-class ($k \geq 3$) classification problem, we can decompose it into k two-class sub-problems. That is to say, for the i-th sub-problem, input positive samples are from C_k, and input negative samples are from all other classes. This approach is also called 'One v.s. Rest'.

So now, we should have $k \times n \times n$ models (classifiers) assuming all samples from one class are divided into n parts. Note that all these models are fully independent to each other, which means we can train them in parallel to accelerate.

1.2.3 Task Decomposition based on classification prior

We denote the training data for a K-class problem is

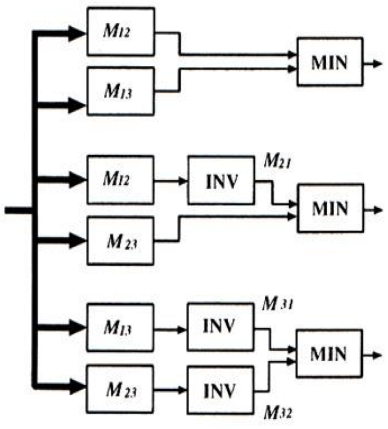
$$T = \{(X_l, Y_l)\}_{l=1}^{L}$$

And then, we can decompose a K-class problem into $K(K-1)/2$ two-class problem

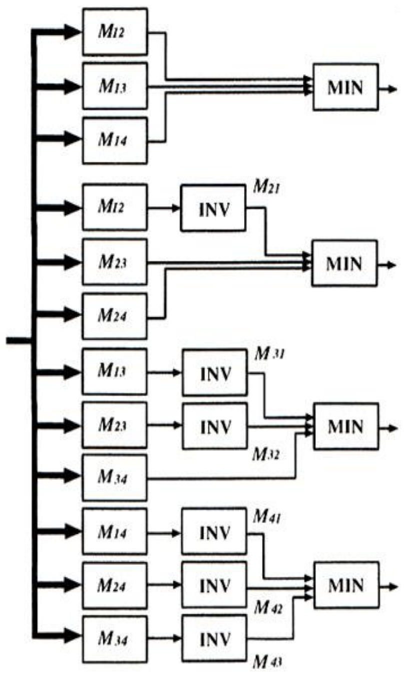
$$X_i = \{X_l^{(i)}\}_{l=1}^{L_i}, \text{ for } i = 1, 2, \cdots, K$$

$$T_{ij} = \{(X_l^{(i)}, 1-\epsilon)\}_{l=1}^{L_i} \cup \{(X_l^{(j)}, \epsilon)\}_{l=1}^{L_j}, \text{ for } i = 1, \cdots, K \text{ and } j = i+1$$

So now, we should have $\frac{k\times(k-1)}{2} \times n \times n$ models (classifiers) assuming all samples from one class are divided into n parts. Note that all these models are fully independent to each other, which means we can train them in parallel to accelerate. Here we put the decomposition of three-class and four-class problem as in Fig.2a and Fig.2b.



(a) three-class decomposition



(b) four-class decomposition

Figure 2: One v.s. One Decomposition

1.3 CNN

CNN is applied to image recognization first by Yann LeCun[2]. A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, ReLU layer i.e. activation function, pooling layers, fully connected layers and normalization layers. We do not give full details about this common kind of neural network.

1.4 RNN

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs.

In our case, the 5 groups of EEG data are not time series. So traditional RNN does not seem reasonable enough. Thus, we apply bidirectional RNN[4] to this task. The memory ability of bidirectional RNN will help us to discover the relationship between the 5 groups of data.

2 Dataset

Our dataset is part of SEED-4 (SJTU Emotion EEG Dataset), including 13 subjects, each with one experiment. Thus 13 sets of brain electricity data, whose feature uses differential entropy (DE). Feature dimensions are $62 \times 5 = 310$. 62 is the number of probes and 5 is the number of frequency fields we take when performing Fourier Transform. The labels are 'neutral', 'sad', 'fear', 'happy', corresponding to 0, 1, 2, 3.

3 Implementation

3.1 Fully-Connected Network

In my implementation, I use a three layer MLP. The hidden layer's dimension is 128. Batchsize is 64. I use sigmoid as the activation function rather than softmax. Experiments show that sigmoid does a better job than softmax when the logits are needed to vote a final results in later sections. For our project, we use sigmoid as the default activation function.

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

3.2 Min-Max Network One v.s. Rest

In our case, we have $4 \times 3 \times 3$ models, each deals with a two-class classification atom problem. To accelerate training, I used python `multiprocessing` package. The `Pool` object offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism). Providing all training data and training labels to processes is not realistic. So I decomposed all samples from each class in advance and saved them in different csv's. Thus, the parallel processes only need to read different data files and don't have to care about how to decompose them.

The parallel processing makes it impossible to test our models on test set during training. I think about it a lot and write a framework where I save these models and test and reload models to continue the training, but it seems to suffer from a memory leak problem which is very common when saving and restoring a lot of models. I haven't fixed this problem yet. So I can only run a few tests during my training.

3.3 Min-Max Network One v.s. One

Here I have $\frac{4\times(4-1)}{2} \times 3 \times 3 = 48$ models. I used the same multiprocessing training technique as in Section 3.2. But then we need to decide what the final predictions are based on the outputs of all 48 models. Like what's previously demonstrated, we analysis probabilities predicted by all relevant models that a sample belongs to each class. Then, we voting the final probabilities by the sum of these probabilities. Note that this is different from the origin algorithm illustrated in Fig.2. But in practice, this method works better. And also, use sigmoid as the activation function rather than softmax brings some improvement.

3.4 CNN

Note that our EEG data have 310 dimensional features. The dimensions can be separated into 5 groups, and each consists of 62 dimensions corresponding to 62 electrode poles.

We can reshape the training data from $[batchsize, 310]$ into $[batchsize, 5, 62, 1]$, where 5, 62, 1 correspond to the images' height, width, and channel like in traditional image recognition tasks. Then we can use CNN to abstract features and get predictions.

Our network architecture is as follows:

1. Input tensor is reshaped into $(batchsize, 5, 62, 1)$.
2. The first layer is a convolutional layer, with relu activation function and 16 kernels of size $(5, 5)$, stride of 1, using padding. So the output is of shape $(batchsize, 5, 62, 16)$.
3. The second layer is a max-pooling layer, whose pool size is $(2, 2)$, stride is 2, with padding. So the output is of shape $(batchsize, 3, 31, 16)$.
4. The third layer is a convolutional layer, with relu activation function and 32 kernels of size $(5, 5)$, stride of 1, using padding. So the output is of shape $(batchsize, 3, 31, 32)$.
5. The fourth layer is a convolutional layer, whose pool size is $(2, 2)$, stride is 2, using padding. So the output is of shape $(batchsize, 2, 16, 32)$.
6. We flatten the fourth layer and put them into the final layer, which is a MLP, whose outputs are of dimension 4 (number of classes). We use softmax to transform them into the probabilities of their belonging to four classes.

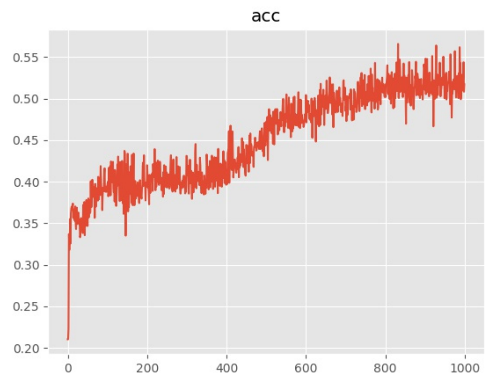
3.5 RNN

Our network architecture is as follows:

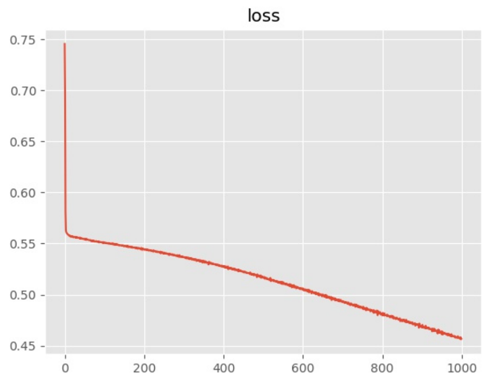
1. Input tensor is reshaped into $(batchsize, 5, 62)$.
2. Unstack input tensor into 5 tensors of shape $(batchsize, 62)$. View them as a time series (5 time steps). And use bidirectional LSTM to process them. Bidirectional LSTM cell has output dimension 32.
3. For now, we have 5 outputs of $(batchsize, 32)$. We stack them and flat them into tensor of shape $(batchsize, 160)$.
4. Finally, we use a MLP to output the final prediction of shape $(batchsize, 4)$.

4 Experiments

In this part, I will give the experiment details, including the training time, predicting time, accuracy, confusion matrix, F1 score of different models.



(a) Accuracy Curve of Fully-Connected Network



(b) Loss Curve of Fully-Connected Network

Figure 3: Training Curves of Fully-Connected Network

By definition a confusion matrix C is such that C_{ij} is equal to the number of observations known to be in group i but predicted to be in group j. And F1 Score can be computed as follows.

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * (Recall * Precision)}{(Recall + Precision}$$

4.1 Fully-Connected Network

The training curves are as in Fig.3. The confusion matrix is as in Table.1. The F1 score is shown in Table.2.

	neutral	sad	fear	happy
neutral	563	108	99	153
sad	107	721	70	350
fear	265	200	162	166
happy	72	113	24	597

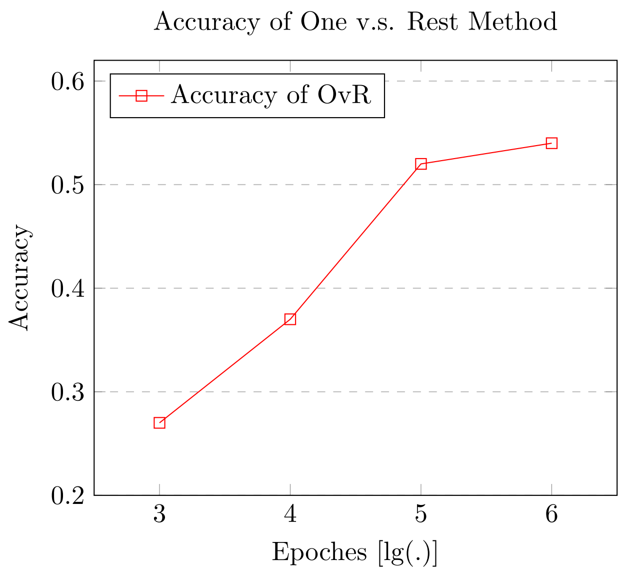
Table 1: Confusion Matrix of Fully-Connected Network

	neutral	sad	fear	happy
F1	0.58	0.60	0.28	0.58

Table 2: F1 Score of Fully-Connected Network

4.2 Min-Max Modular Network One v.s. Rest

The training accuracy is shown below. The confusion matrix and the F1 score are shown in Table.3 and 4.



	neutral	sad	fear	happy
neutral	678	60	28	157
sad	481	673	21	73
fear	278	139	91	285
happy	240	13	0	553

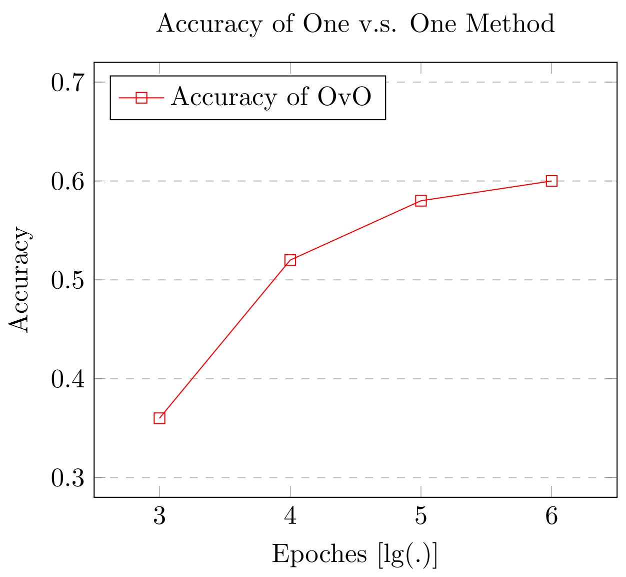
Table 3: Confusion Matrix of Min-Max OvR Method

	neutral	sad	fear	happy
F1	0.52	0.63	0.20	0.59

Table 4: F1 Score of Min-Max OvR Method

4.3 Min-Max Modular Network One v.s. One

The training accuracy is shown below. The confusion matrix and the F1 score are shown in Table.5 6.



	neutral	sad	fear	happy
neutral	370	105	71	377
sad	133	931	86	98
fear	105	157	180	351
happy	11	62	34	699

Table 5: Confusion Matrix of Min-Max OvO Method

	neutral	sad	fear	happy
F1	0.48	0.74	0.31	0.60

Table 6: F1 Score of Min-Max OvO Method

4.4 CNN

The training accuracy curve and loss curve are as in Fig.4.

4.5 RNN

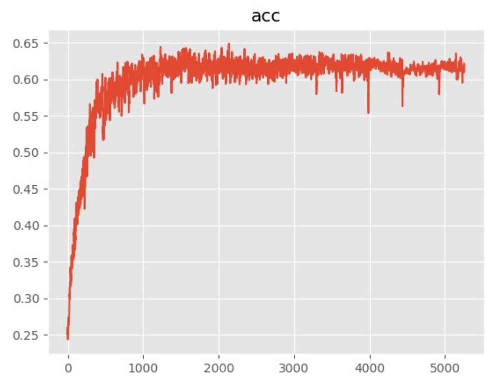
The training accuracy curve and loss curve are as in Fig.5.

4.6 Time Consumed

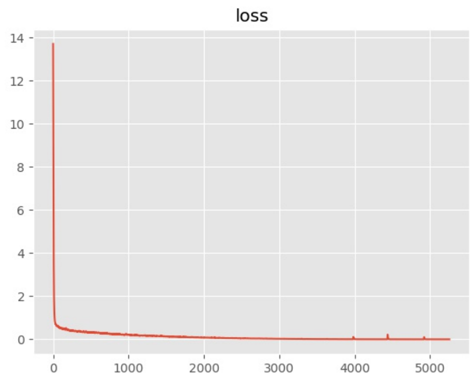
See Table.7 and 8.

fc	ovr	ovo	cnn	rnn
89	2332	2554	374	790

Table 7: Time needed for training 100000 epoches (unit: sec)

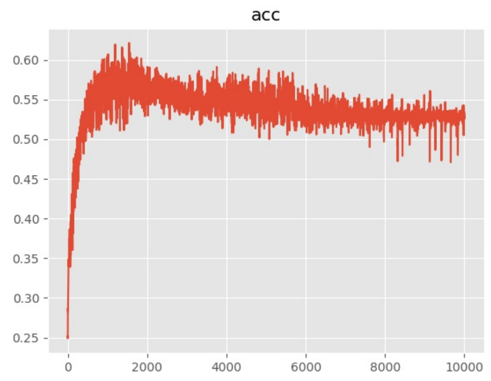


(a) Accuracy Curve of CNN

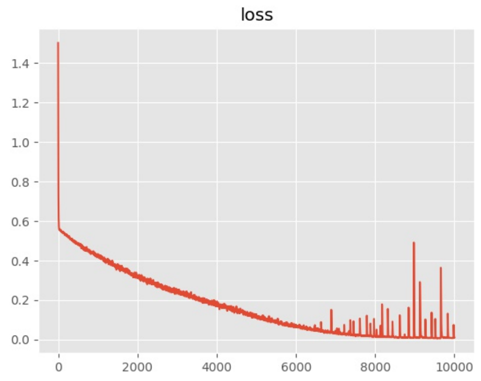


(b) Loss Curve of CNN

Figure 4: Training Curves of CNN



(a) Accuracy Curve of Bi-LSTM



(b) Loss Curve of Bi-LSTM

Figure 5: Training Curves of Bi-LSTM

fc	ovr	ovo	cnn	rnn
0.006	0.019	0.025	0.006	0.006

Table 8: Time needed for predicting all test data (unit: sec)

5 Conclusion

5.1 Experiment Discussion

From the F1 scores, we know that class 2 is the most easy to recognize, and class 3 is the hardest.

From the view of performance, FC gets the poorest accuracy about 50%. For the Min-Max modular network, OvO version (60%) outperforms OvR version (57%). Some of my classmate told me their OvO can even reach 90%. CNN (60%) outperforms RNN (57%).

From the view of training time, FC trains the most fast. CNN and RNN are very close to it. The Min-Max modular network, however, trains very slow even with parallel acceleration and GPU.

From the view of parameter tuning, FC needs almost no tuning. The Min-Max modular networks need large amount of parameter tuning and debugging because of the large number of models and parallel mechanism. CNN and RNN are pretty robust to the hyper-parameters but we need to carefully choose the architecture.

5.2 Further Discussion

Here I want to discuss the difference between nowadays deep neural networks and traditional shadow models from the perspective of the inductive bias.

The inductive bias (also known as learning bias) of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.[1]

In machine learning, one aims to construct algorithms that are able to learn to predict a certain target output. To achieve this, the learning algorithm is presented some training examples that demonstrate the intended relation of input and output values. Then the learner is supposed to approximate the correct output, even for examples that have not been shown during training. Without any additional assumptions, this problem cannot be solved exactly since unseen situations might have an arbitrary output value. The kind of necessary assumptions about the nature of the target function are subsumed in the phrase inductive bias. For example:

1. **Maximum margin:** when drawing a boundary between two classes, attempt to maximize the width of the boundary. This is the bias used in support vector machines. The assumption is that distinct classes tend to be separated by wide boundaries.

2. **Nearest neighbors:** assume that most of the cases in a small neighborhood in feature space belong to the same class. Given a case for which the class is unknown, guess that it belongs to the same class as the majority in its immediate neighborhood. This is the bias used in the k-nearest neighbors algorithm. The assumption is that cases that are near each other tend to belong to the same class.

3. **all to all:** For fully connected layers, the inductive bias assumes the relations are all-to-all (all units in layer i can affect all units in layer j). The argument to the rule is the full input signal, there is no reuse, and there is no isolation of information. The implicit relational inductive bias in a fully connected layer is thus very weak: all input units can interact to determine any output units value, independently across outputs.

Above are three inductive bias examples for traditional machine learning methods SVM, KNN, MLP. Now let's talk about other inductive biases of modern neural networks which are shown in Fig.6.

1. For **convolutional layers** in CNN, the inductive bias is **locality** and **translation invariance**. Locality reflects that the arguments to the relational rule are those entities in close proximity with one

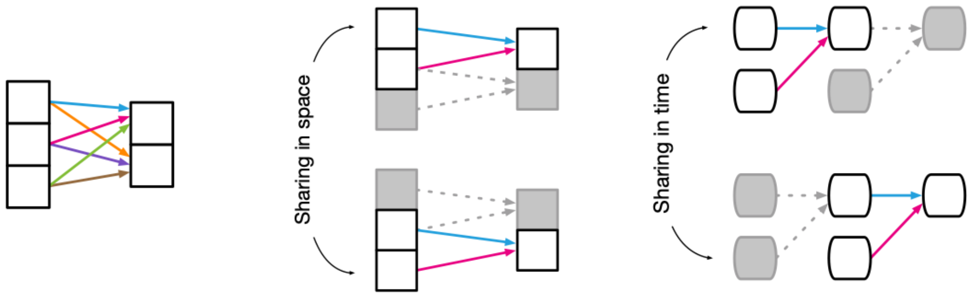


Figure 6: Inductive bias of FC, CNN, RNN

another in the input signals coordinate space, isolated from distal entities. Translation invariance reflects reuse of the same rule across localities in the input. These biases are very effective for processing natural image data because there is high covariance within local neighborhoods, which diminishes with distance, and because the statistics are mostly stationary across an image.

2. For **recurrent layers** in RNN, we can view the inputs and hidden states at each processing step as the entities, and the Markov dependence of one steps hidden state on the previous hidden state and the current input, as the relations. The rule for combining the entities takes a steps inputs and hidden state as arguments to update the hidden state. The rule is reused over each step, which reflects the relational inductive bias of **temporal invariance** (similar to a CNNs translational invariance in space). For example, the outcome of some physical sequence of events should not depend on the time of day. RNNs also carry a bias for **locality** in the sequence via their Markovian structure.

As we can see, the KNNs and SVMs have too strong inductive biases, thus their models are simple and can be trained very fast but cannot gurantee a good prediction. MLPs, however, used a very weak inductive bias, thus usually have a large number of parameters, which makes deep MLPs hard to train and easy to overfit. But for CNNs and RNNs that have promoted the development of Artificial Intelligence in recent years, they accomplished great success due to their reasonable inductive biases for the task related to images and time series respectively.

In conclusion, the true reason for the prosperous of nowadays AI is more advanced inductive biases behind all the fancy network architectures. However, these inductive biases are still not enough. CNN suffer from, for instance, rotation problem and for RNN, gradient vanishing problem. We must leverage more powerful inductive biases to solve the problems fundamentally. That is also the requirement for strong AIs.

References

[1] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. 2018.

[2] LeCun. Backpropagation Applied to handwritten Zip Code Recognition.

[3] Bao-liang Lu and Masami Ito. Task Decomposition Based on Class Relations: A Modular Neural Network Architecture for Pattern Classification.

[4] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.