AIC Advanced Machine Learning Project
AdaNet: Adaptive Structural Learning of Artificial Neural Networks

Qixiang PENG [1] **Xudong ZHANG** [1] **Zizhao LI** [1]

Abstract

This paper provides a new algorithms for adaptively learning artificial neural networks. The algorithms (ADANET) adaptively learn both the structure of the network and its weights. The additional units that will be added are carefully selected and penalized according to rigorous estimates from the theory of statistical learning. Author also did some experiments with one of our algorithms on several binary classification tasks extracted from the CIFAR-10 dataset. The results demonstrate that ADANET can automatically learn network structures with very competitive performance accuracies when compared with those achieved for neural networks found by standard approaches.

1. Introduction

Deep neural networks form a powerful framework for machine learning and have achieved a remarkable performance in severals areas such as natural language processing, image captioning, speech recognition and many others in recent yesrs. However, training these models usually requires domain knowledge and costs lots of time, space and resources.

Also, although neural networks have got excellent performance in several domains, behind them, we can't find a theorical support. So in this paper, authors propose attempt to remedy some of these issues. They give an algorithm which learns the architecture of neural networks and its weights in an adaptive fashion by starting from a simple linear model and then adding more units and additional layers as needed.

Our report is organized as follows. Section 2 describes the broad network architecture, the hypothesis set and the generalization guarantees which guide the design of the algorithm described in Section 3. Section 4 provides the experimental results. We conclude in Section 5 and give our future work in Section 6.

2. Theory Background

2.1. Definition of the Neural Network

In order to treat the problem of Neural Network, firstly, we need to define mathematically the structure of neural network. In the article, the author uses a set of function families to define the node function in different layers. The definition is shown below:

$$\mathcal{H}_1 = \{x \mapsto u \cdot \Phi(x) : u \in \mathbb{R}^{n_0}, ||u||_p \le \Lambda_{1,0}\}$$

$$\mathcal{H}_k = \{x \mapsto \sum_{s=1}^{k-1} u_s \cdot (\phi_s \circ h_s)(x) : \\ u_s \in \mathbb{R}^{n_s}, ||u_s||_p \le \Lambda_{k,s}, h_{k,s} \in \mathcal{H}_s\}$$

These set of function families in fact define a very general form of neural network. We can see from the definition of the function families that for every node at layer k, it has the possibilities to connect to all the nodes in layers below it.

After we define the node functions, we need to define the output function. The definition is shown below:

$$f = \sum_{k=1}^{l} \sum_{j=1}^{n_k} w_{k,j} h_{k,j} = \sum_{k=1}^{l} \mathbf{w_k} \cdot \mathbf{h_k}$$

It can be seen as the weighted sum over all the nodes. Here the weight $\mathbf{w}$ also defines the structure of the neural network. If the weight is zero, there is no connection between the node and the output. While it is not zero, there exists one connection. One example of this general form network can be shown as the image below: In the next part of the article, the author defins a more standard form of neural network. For the node function of $\mathcal{H}_k$, we add one restriction: for $s = 1$ $(k-2)$, $u_s = 0$. In this way, nodes in layer k and only connects to nodes in layer $k-1$. However, the output function is still defined as the weighted sum over all the nodes. In this way, our frame of neural network is also a general form. The conclusion can be used directly to the standard neural network.

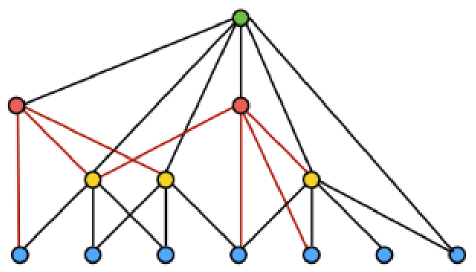

Figure 1. Neural Network Example

2.2. Generalization Bounds

After we have well defined the neural network, the next problem is to generate the bound for the generalization error. The goal is quite simple, we want to minimize the generalization error. Howeverr, we can not minimize that directly. So we calculate the upper bound of this error and we want to minimize the upper bound of generalization error. In this way, we can decrease the value of the generalization error to some extend. Following this idea, the author developed the theorem 1. The result of theorem 1 gives us one elegant but abstract bound on the generalization error:

$$R(f) \le \hat{R}_{S,\rho}(f) + \frac{4}{\rho}\sum_{k=1}^{l} ||\mathbf{w_k}||_\mathbf{1} \mathcal{R}_\mathbf{m}(\tilde{\mathcal{H}}_\mathbf{k}) + \frac{\mathbf{2}}{\rho}\sqrt{\frac{\log \mathbf{l}}{\mathbf{m}}} + C(\rho, l, m, \delta)$$

We explain different terms in the inequality here. The generalization error $R(f) = \mathbb{E}_{(x,y)\ D}[1_{yf(x)\le 0}]$, the empirical margin error is defined by $\hat{R}_{S,\rho}(f) = \frac{1}{m}\sum_{i=1}^{m} 1_{y_i f(x_i \le \rho)}$. $\mathcal{R}_m(\tilde{\mathcal{H}}_k)$ is the Rademacher Complexity of $\tilde{\mathcal{H}}_k$. And finally $C(\rho, l, m, \delta) = \tilde{O}(\frac{1}{\rho}\sqrt{\frac{\log l}{m}})$.

This theorem tell us that the generalization error can be bounded by the right part of the inequality: the empirical margin error, the second term which is the weighted sum over the Rademacher Complexity, the rest terms which are determined by the depth of the neural network. Here we focus on the second item, we can found that this term has relation to the weight. In fact, as has been said before, it is the weighted sum over the Rademacher complexity. According to the discussion before, the weight defines to some extend the complexity of the network. So this term reflects to some extend the structural complexity of the neural network. But this is not enough because the Rademacher complexity is too abstract to see clearly the property of the second term. So the author uses some relaxation tricks and get the following inequality:

$$R(f) \le \hat{R}_{S,\rho}(f) + \frac{4}{\rho}\sum_{k=1}^{l} ||\mathbf{w_k}||_\mathbf{1}[\bar{\mathbf{r}}_{\inf}\mathbf{\Lambda_k}\mathbf{N_k}^{\frac{1}{q}}\sqrt{\frac{\mathbf{2}\log(\mathbf{2n_0})}{\mathbf{m}}}] + \frac{2}{\rho}\sqrt{\frac{\log l}{m}} + C(\rho, l, m, \delta)$$

where $\Lambda_k = \prod_{s=1}^{k} 4\Lambda_{s,s-1}$ and $N_k = \prod_{s=1}^{k} n_{s-1}$.

So now we see clearly that the second item is a weighted sum over a complex variable. The value of this variable will grow up with the depth of the neural network and most importantly, it has no relationship with the weight. So it is not hard to imagine that if we want to decrease the value of the second term, one simple method is to use small weight value in deeper layers.

Now we can clearly define our goal according to the previous inequality. Our goal is to minimize the right part of the inequality. There are many parameters that can be changed. In this article, we focus on the weight $\mathbf{w}$. We try to tune the value of the weight to minimize the right part. It is easy to find that only the first and second terms have relationship with the weight. So in the next algorithm part, our goal is to tune the weight to minimize the first two terms of the inequality before.

3. Algorithm

In this section, we will give some details about ADANET, for adaptive learning of neural networks. This is to say that ADANET adaptively grows the structure of a neural network, balancing model complexity with empirical risk minimization.

3.1. Objective function

ADANET aims to find out a function $f = \sum_{j=1}^{N} w_j h_j \in \mathscr{F}^*$(or neural network) that directly minimizes the datadependent generalization bound. So the objective function is like:

$$F(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{1}\Phi(1 - y_i\sum_{j=1}^{N}\omega_j h_j) + \sum_{j=1}^{N}\Gamma_j\,|\omega_j|$$

where the first term represents the empirical margin error and second term is the regularization term by using empircal Rademacher complexity. It is not hard to find out that they correspond to the first and second terms in the right part of corollary 1.

3.2. Description

The algorithm is under the GradientBoosting framework. We denote the ADANET model after $t1$ rounds by f_{t1}, which is parameterized by w_{t1}. Let $h_{k,t1}$ denote the vector of outputs of units in the k-th layer of the ADANET model, l_{t1} be the depth of the ADANET architecture, $n_{k,t1}$ be the number of units in k-th layer after $t1$ rounds. At round t, we select descent coordinates by considering two candidate subnetworks $h \in \widetilde{\mathscr{H}}^*_{l_{t-1}}$ and $h^{'} \in \widetilde{\mathscr{H}}^*_{l_{t-1}+1}$ that are generated by a weak learning algorithm WEAKLEARNER.

Here, h means augmenting the current neural network with a subnet- work with the same depth as that of the current network, and $h^{'}$ means augmenting the current neural network with a deeper subnetwork.

Figure 2 illustrates this construction and the two options just described. An important aspect of our algorithm is that the units of a subnetwork learned at a previous iteration (say $h_{1,1}$ in Figure 2) can serve as input to deeper subnetwork added later (for example $h_{2,2}$ or $h_{2,3}$ in the Figure). Thus, the deeper subnetworks added later can take advantage of the embeddings that were learned at the previous iterations. The algorithm terminates after T rounds or if the ADANET architecture can no longer be extended to improve the objective function.

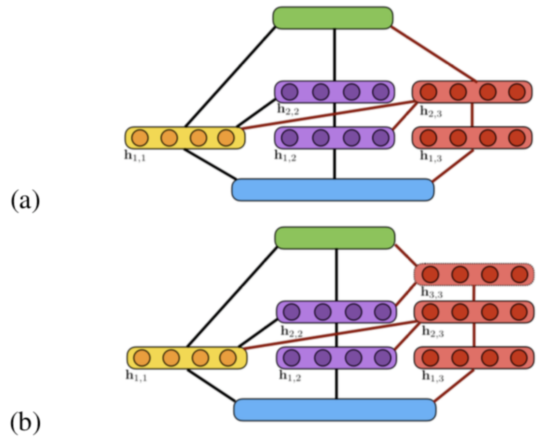

Figure 2. Illustration of the algorithms incremental construction of a neural network. The input layer is indicated in blue, the output layer in green. Units in the yellow block are added at the first iteration while units in purple are added at the second iteration. Two candidate extensions of the architecture are considered at the the third iteration (shown in red): (a) a two-layer extension; (b) a three-layer extension. Here, a line between two blocks of units indicates that these blocks are fully-connected.

Then if $\min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}) \leq \min_{\mathbf{w}} F_t(\mathbf{w}, \mathbf{h}^{'})$, we take:

$$\mathbf{w}^* = arg \min_{\mathbf{w}\in\mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}), \mathbf{h}_t = \mathbf{h}$$

and otherwise

$$\mathbf{w}^* = arg \min_{\mathbf{w}\in\mathbb{R}^B} F_t(\mathbf{w}, \mathbf{h}^{'}), \mathbf{h}_t = \mathbf{h}^{'}$$

If $F(\mathbf{w}_{t-1}+\mathbf{w}^*) < F(\mathbf{w}_{t-1})$ then we set $f_{t-1} = f_t+\mathbf{w}^*\cdot\mathbf{h}_t$ and otherwise we terminate the algorithm.

I give the complete pseudocode of the ADANER algorithm in Figure 3.

```
AdaNet($S = ((x_i, y_i)_{i=1}^m)$)
 1  $f_0 \leftarrow 0$
 2  for $t \leftarrow 1$ to $T$ do
 3      $\mathbf{h}, \mathbf{h}' \leftarrow$ WeakLearner$(S, f_{t-1})$
 4      $\mathbf{w} \leftarrow$ Minimize$(F_t(\mathbf{w}, \mathbf{h}))$
 5      $\mathbf{w}' \leftarrow$ Minimize$(F_t(\mathbf{w}, \mathbf{h}'))$
 6      if $F_t(\mathbf{w}, \mathbf{h}') \leq F_t(\mathbf{w}', \mathbf{h}')$ then
 7          $\mathbf{h}_t \leftarrow \mathbf{h}$
 8      else $\mathbf{h}_t \leftarrow \mathbf{h}'$
 9      if $F(\mathbf{w}_{t-1} + \mathbf{w}^*) < F(\mathbf{w}_{t-1})$ then
10          $f_{t-1} \leftarrow f_t + \mathbf{w}^* \cdot \mathbf{h}_t$
11      else return $f_{t-1}$
12  return $f_T$
```

Figure 3. Pseudocode of the ADANET algorithm. On line 3 two candidate subnetworks are generated. On lines 3 and 4, objective function is solved for each of these candidates. On lines 5-7 the best subnetwork is selected and on lines 9-11 termination condition is checked.

4. Experiments by author

Authors did several experiments on CIFAR-10 dataset. This dataset consists of 60,000 images evenly categorized in 10 different classes. Because this paper focused on the binary classification problem, authors considered five pairs of classes: deer-truck, deer-horse, automobile-truck, cat-dog, dog-horse. They compared ADANET to standard feedforward neural networks (NN) and logistic regression (LR) models. Like what are said in paper, the goal of these experiments is not to obtain state-of-the-art results for this particular task, but to provide a proof-of-concept illustrating that our structural learning approach can be competitive with traditional approaches for finding efficient architectures and training corresponding networks. But actually, ADANET outperforms other meth- ods on each of the datasets.

In all experiments authors use ReLu activations. NN, NN-GP and LR are trained using stochastic gradient method with batch size of 100 and maximum of 10,000 iterations. T = 30 was used for ADANET in all our experiments although in most cases algorithm terminates after 10 rounds.

Table 1. Experimental results for ADANET, NN, LR and NN-GP for different pairs of labels in CIFAR-10. Boldfaced results are statistically significant at a 5% confidence level.

LABEL PAIR	ADANET	LR	NN	NN-GP
DEER-TRUCK	0.9372 ± 0.0082	0.8997 ± 0.0066	0.9213 ± 0.0065	0.9220 ± 0.0069
DEER-HORSE	0.8480 ± 0.0076	0.7685 ± 0.0119	0.8055 ± 0.0178	0.8060 ± 0.0181
AUTOMOBILE-TRUCK	0.8461 ± 0.0069	0.7976 ± 0.0076	0.8063 ± 0.0064	0.8056 ± 0.0138
CAT-DOG	0.6924 ± 0.0129	0.6664 ± 0.0099	0.6595 ± 0.0141	0.6607 ± 0.0097
DOG-HORSE	0.8350 ± 0.0089	0.7968 ± 0.0128	0.8066 ± 0.0087	0.8087 ± 0.0109

The average architectures for all label pairs are provided in Table 2. Note that NN and NN-GP always selects one layer architecture. The architectures selected by ADANET typically also have just one layer and fewer nodes than those selected by NN and NN-GP. However, on a more challenging problem cat-dog ADANET opts for a more complex model with two layers which results in a better performance. This further illustrates that ADANET allows to learn network architectures in adaptive fashion depending on the complexity of the given problem.

Table 2. Average number of units in each layer.

Label pair	ADANET		NN	NN-GP
	1st layer	2nd layer		
deer-truck	990	0	2048	1050
deer-horse	1475	0	2048	488
automobile-truck	2000	0	2048	1595
cat-dog	1800	25	512	155
dog-horse	1600	0	2048	1273

5. Our work

Note that all our codes can be found at:

https://github.com/DoubleKing0625/AIC_Project/tree/master/Advanced_ML

5.1. Data Preparation

As noted before, we use the CIFAR-10, an image category dataset. It contains 60000 images, 50000 for train-set, 10000 for test-set. And each image has size 32*32*3(32*32 is the shape, 3 is the color channel). The label is the number between [0,9], which represents 10 different class: {airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}, and each class contains 6000 images, 5000 for train-set, 1000 for test-set.

Because we concentrate on the binary classification problem, we reduce 10 classes to 5 pairs of classes: *deer-truck, deer-horse, automobile-truck, cat-dog, dog-horse*. So finally we have 5 data-set, and each one has 10000 samples for train-set, 2000 for test-set. Note that here is the binary classification problem, we map the label into $\{0, 1\}$.

Raw images(32*32*3) have been preprocessed. We compute color histograms and histogram of gradient features[1] for each image, and get the final result: a 154*1 vector with value ranges [0, 1]. This point is different from the paper. We got 155*1(128+27) as the final input data. 128 real values is for the histogram of gradient features($(orient = 8) * ((row\_image/ppc) = 32/8 = 4) * ((col\_image/ppc) = 32/8 = 4) = 128$), and 27 real values for color histograms($3 * 3 * 3$).

5.2. Details of Algorithm Design

In this part, we try to give out the details for realizing the algorithms proposed in this article. In fact, this article only gives out an abstract frame of the algorithm. Here we give out our idea for realizing the simplest version of one AdaNet.

5.2.1. AIM OF THE ALGORITHM

First we have a look at the regularized version of the loss function:

$$\tilde{F}_t(w,h) = \frac{1}{m}\sum_{i=1}^{m}\Phi(1-y_i f_{t-1}(x_i)-y_i w\cdot h(x_i))+R(w,h)$$

Here $R(w,h)$ is a regularization term. The aim is to minimize this F function over w and h. Here for simplicity, we just let $R(w,h) = 0$. So our aim now is to minimize the following function:

$$\tilde{F}_t(w,h) = \frac{1}{m}\sum_{i=1}^{m}\Phi(1-y_i f_{t-1}(x_i)-y_i w\cdot h(x_i))$$

The difficulty here is that the previous function is not a convex function. So we may converge to a local minimal. However, this is not really severe. Anyway, our goal is to minimize the loss function. We just need to let the loss function decrease.

[1]Here we use API of scikit image: http://scikit-image.org/

Here things have become more clearer. For this simplest version, we can find that the parameters in the function: f_{t-1}, w, h just define the new neural network in round t, with parameters in $t-1$ fixed. It is not hard to find that for the new neural network, we try to minimize the loss function by tuning the parameters created by the new nodes added in the network. In the next part, we try to translate the abstract frame into detailed steps for realizing the AdaNet algorithm.

5.2.2. Algorithm–Detailed Version

Now we have data $S = ((x_i, y_i)_{i=1}^{m})$. We listed steps for creating AdaNet:

Step1: We construct a simple neural network in step one. Between the input (a vector representing the image) and the output (a scalar $\in \{-1, 1\}$), we add B nodes as hidden layer 1, where B is fixed as the number of nodes added each time. The input and output is fully connected with the network. We train this neural network with certain iterations and epochs (the stopping criterion may be that the loss does not decrease greatly, or just after maybe 10000 iterations). After the training, we save the loss as F_1 and we fix the parameters already trained.

Step2: Based on the neural network trained in the previous step, we consider adding layers. Here there are two choices, we list them below:

Step2.1: If there are l layers in previous step, we add on each layer B nodes. Each new layer on layer s is fully connected with nodes on layer $s-1$. (Here we need to state one important thing here. As nodes are added step by step, different layers are not fully connected. For example in figure 2, purple nodes in layer two is not fully connected with red nodes in layer one. The aim is to fix values calculated by layers before) After constructing this new network, we train this new network with parameters in previous network fixed. This means we only optimize parameters introduced by the new nodes. The loss function is saved as $F_{t,1}$

Step2.2: If there are l layers in previous step, we add on each layer B nodes. The we add B nodes on layer $l+1$. Each new layer on layer s is fully connected with nodes on layer $s-1$. After constructing this new network, we train this new network with parameters in previous network fixed. The loss function is saved as $F_{t,2}$

Step3: We compare the value of $F(t,1)$ and $F(t,2)$. We note the smaller as $F(t)$.

Step3.1: If $F(t)$ is bigger than $F(t-1)$, the program is ended. We return the network in round $t-1$

Step3.2: If $F(t)$ is smaller than $F(t-1)$, then go to Step2.

To better explain the algorithm before, we give out images to explain step1 and step2: Here the line signifies full con-

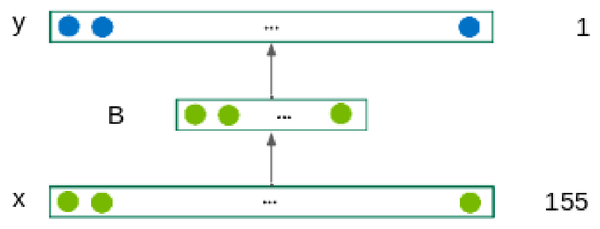

Figure 4. Step 1

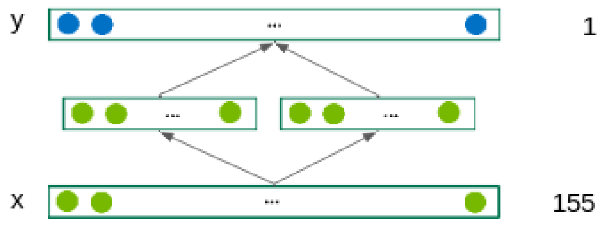

Figure 5. Step2.1

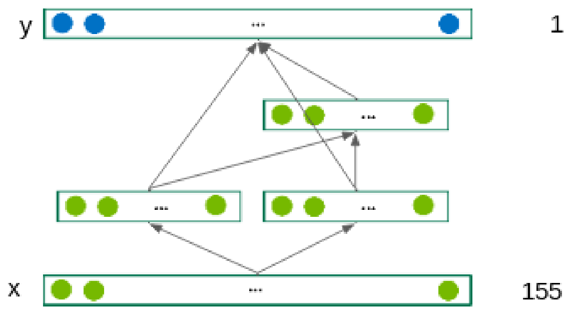

Figure 6. Step2.2

nection. The difference from the illustration image in paper is that we give out all the connections. In the explanation image in paper, it only gives out the connection between nodes. However, every node is directly connected to the output. Here in image of Step2.2, we directly give it out in the image.

5.2.3. Discussion Over the Algorithm

From the steps stated before, we can find that the function of AdaNet is to train a neural network part by part, the trained parameters are fixed and we try to add more nodes to further minimize the loss function. If there is no need to add layers, we stop the algorithm and return the network already obtained.

Here we need to explain that for the simplest version of the algorithm, normally it will not stop automatically. If there are more layers, the loss will become smaller. This is due to the problem that we set the regularization term $R(w,h)$ to zero. So the loss is only relevant to the accuracy. Normally, if there are more nodes, the loss will become smaller. So in fact, it is important to add regularization term to reflect the complexity of the network and the complexity of the data. These two complexity should be comparable. The trained neural network should well reflect the property of the data set and it should not be too complex to lead to over-fitting.

Another problem is that if we look at the results in the paper, we can find almost all the nodes are in layer 1 and there are few nodes in deeper layers even if we add regularization. That is in the part for constructing the loss function, we only consider minimizing the bounds by tuning the parameter w. However, l the depth is also a parameter which will affect the bounds. So this may lead to the problem that almost all nodes will be put on layer one.

So according to our understanding, in order to have good results with AdaNet, (satisfying accuracy and proper complexity of network compared with the complexity of the data set) It is important to define a proper regularization term to reflect the complexity of the network compared with the complexity of the data set. Rademancher complexity here really gives us an inspiring idea.

Another problem here is that AdaNet train a neural network part by part. As can be seen from the steps before, the trained parameters are fixed and we only train the new parameters introduced by new nodes. However, even if we can get a good structure of neural network, maybe it will have a better performance if we train all the parameters as a whole. Moreover, from step2, we can find that one of the step in step2 will become totally useless. this is one main default of this algorithm, we need much ability in calculation and unfortunately, most of the calculation will become useless. That's why only Google is interested in it. They do not care the cost of calculation and they do have that ability.

However, the theoretical part also inspire us on how to find a economic neural network by harmonize the complexity of the neural network and the complexity of data.

5.2.4. Possible Optimization

One possible optimization that we propose here is that in step2, maybe in step2.2, we can construct the network based on step2.1. We fix the weights obtained in step2.1 and we only tune the parameters introduced by nodes added in layer $l+1$.

5.3. Difficulties

5.3.1. Failure Realization by using Deep Learning Framework

Normally, when we want to construct a neural network model, we think of Tensorflow, Mxnet, Pytorch, etc. However, we need to state out here that it is very difficult to realize this algorithm by DL framework. And we have tired to realized this algorithm though Pytorch and finally we gave up this plan.

The problem is that in the algorithm AdaNet, we need to add layers dynamically. Like what we have written above, we have 100 node in the 1st layers at the m-th iteration, But probably we have 200 ones in the 1st at the m+1-th iteration. And we update the weights according to iteration order, not update the weights layer by layer.

However, in Pytorch, what we can do is to well define a network as a model and test the model. It is not possible to design a quasi-dynamical neural network. We should only re-write many deep source code of Pytorch in order to realize the Adanet. As we known, it's a tough work. Although we fail to realize it with Pytorch, we write it here in case others try to realize it through Pytorch.

5.3.2. Write the neural network by ourselves

After giving up the idea which using pytorch, we decided to write all the things by ourselves. But it's a really tough work to realize the details of the network: back-propagation, batch-size study, etc. And absolutely we have much reference to existing great codes.

5.4. Our Experiments and Results

In this project, we focus on the reproduction of the algorithm proposed by the authors, so we take into account the two pairs ”deer” and ”horse” in CIFAR-10 dataset to do a binary classification. In our experiments, we used the defaut parameters cause we didn't have hardware to do the same experiment as author has done which needs huge computation ability, for example, the number of iterations, 10-folds cross validation, grid search, etc.

With refering to existing codes and developping our own codes, to test on the pairs ”deer” and ”horse” in CIFAR-10, we got an accuracy at around 50% which is not a good result but a similar result to a two hidden layers fully-connected network with around hundred units per layer. We considered that were because that there were maybe some errors in out code but we didn't find it and another reason is that we didn't utilize optimal parameters.

Another limit of our work is that we don't have a visual view of the architecture of the final network unless we check the weights of each layer.

As mentionned in the previous part, at first we supposed to use one of the existing neural network framework, but they didn't support the design of the quasi-dynamical network like our purpose which made our work more difficult than thinking. This is one thing that we didn't realized. However, by developping our own architecture, we can have a profound impressive understanding of back-propagation. Another remarkable point is that we ignored the complexity of this project which need focuses on many details, such as the definition of variables, the calling program, the size of parameters, etc. Meanwhile, we have got a lot within this experience, like the understanding of statistic machine learning theory and the utility of boosting method.

6. Doubt about Adanet Structure

Each layer in Adanet can be connected to the output layer directly. We think it's a little wired. Normally, the network model is connected layer by layer. We trend to only learn the number of node in each layer and the number of layers adaptively and discard the structure like above. And it's a pity that we have no time to verify this idea.

7. Future work

7.1. Adanet.CVX

From now on, we have realized the basic version Adanet. Authors also presented another version: Adanet.CVX. At each round, ADANET.CVX can design the optimal candidate subnetwork within its searched space in closed form, leading to an extremely efficient update. However, this comes at the cost of a more restrictive search space than the one used in ADANET. The pseudo-code of ADANET.CVX is provided in Figure 7. We would like to realize the Adanet.CVX and figure out which one is really faster to be trained if we have time afterwards.

7.2. Grid search over hyperparameters

We don't fine-tune the hyper parameters. In paper, authors did a lot of work about grid search over hyper parameters. The hyperparamers have been optimized over the following ranges: learning rate $\in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, regularization $\in \{0, 10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}\}$, number of layers $\in \{1, 2, 3\}$, number of nodes to be added $\in \{100, 150, 512, 1024, 2048\}$.

```
ADANET.CVX($S = ((x_i, y_i)_{i=1}^m)$
1  $f_0 \leftarrow 0$
2  for $t \leftarrow 1$ to $T$ do
3      $s^* \leftarrow \text{argmax}_{s \in [l_{t-1}+1]} \Lambda_{s,s-1} \|\boldsymbol{\epsilon}_{t,\mathbf{h}_{s-1,t-1}}\|_q.$
4      $u_i^{(s^*)} \leftarrow \frac{\Lambda_{s^*,s^*-1}}{\|\boldsymbol{\epsilon}_{t,\mathbf{h}_{s^*-1,t-1}}\|_q^{\frac{q}{p}}} |\epsilon_{t,h_{s^*-1,t-1,i}}|^{q-1}$
5      $\mathbf{h}' \leftarrow \mathbf{u}^{(s^*)} \cdot (\phi_{s^*-1} \circ \mathbf{h}_{s^*-1,t-1})$
6      $\eta' \leftarrow \text{MINIMIZE}(\tilde{F}_t(\eta, \mathbf{h}'))$
7      $f_t \leftarrow f_{t-1} + \eta' \cdot \mathbf{h}'$
8  return $f_T$
```

7.3. Code development

Do a double check over the code to make sure that there would be no error in the process of reproduction of algorithm. Then try to find a way to visualize the network after each epoch in order to get the adding units and check their reasonabilities.

8. Conclusion

Authors present a new framework and algorithms AdaNet for adaptively learning artificial neural networks with strong theoretical guarantees. It simultaneously learns a neural network architecture and its parameters by balancing a trade-off between model complexity and empirical risk minimization. They also reported favorable experimental results in the problem of binary classification comparing to Neural Network. Their algorithm is general and can be applied to multi-classification and to other neural network architectures such as CNNs and RNNs.

We have tried our best to realize the Adanet(it's tough but really meaningful) and do some comparison experiments to verify the point of view proposed by authors. According to our experiments result, the algorithm Adanet doesnt works as well as noted in paper. Note here that we are not absolutely sure that our codes are right.

In our opinion, this algorithm is really interesting and practical. The core idea of Adanet is to find the best structure of network in the searched space. Nowadays, computation ability is more and more powerful, and the algorithms like that learning the network structure adaptively is more and more realizable. For example, Google Brain has posted a work: Neural Architecture Search With Reinforcement Learning. We think these both work have the same core idea.

Reference

Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, Scott Yang. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. In ICML 2017.

Barret Zoph, Quoc V. Le. Neural Architecture Search With Reinforcement Learning. In ICLR 2017.