Abstract

A completely new learning rule, is applicable to convolutional neural networks, can do long-term accumulative learning, make neuron’s meaning explicit. Similar to the learning rule in the brain, completely different with gradient descent. This learning rule, is the foundation and the key of whole memory, will open a huge growth potential for Artificial Intelligence.

1 New Learning Rule

True Artificial Intelligence, needs lots of memories, and the thinkings based on memory. Making neural networks can get knowledge through visual perception, for example: cherry is red, banana is yellow, is the best choice for the first step. Then, making memory’s all kinds of characteristics.

Getting knowledge through perception system, is extremely important, and irreplaceable.

But, current learning rule: gradient descent, can’t brace these. Whether neural networks getting knowledge through visual perception, nor memory’s other characteristics, for example: association, language attaching, all can’t be braced by gradient descent.

I propose a completely new learning rule, to brace the realization of whole memory. This learning rule, is applicable to convolutional neural networks, can do long-term accumulative learning, make neuron’s meaning explicit. Similar to the learning rule in the brain, completely different with gradient descent. Really great!

Without this learning rule, "neural networks getting knowledge through visual perception", most memory’s characteristics, and so on, can’t be accomplished. Lots of important things can’t start.

With this learning rule, whole memory part can start to do. Lots of important things can start to do. This learning rule, is the foundation and the key of whole memory, will open a huge growth potential for Artificial Intelligence.

New learning rule, consists of four parts, they are: convolution layer part, neuron part, long-term accumulative learning.

Preliminarily validation has been done, MNIST handwritten digits "0", "1" recognition (memory generation, memory recall).

Code and paper all will be released.

In Memory1,2 (has done), have made "new learning rule".
In Memory3, will do "improving new learning rule futher".
In Memory4, will do "neural networks learn basic knowledge through visual perception", for example: cherry is red, banana is yellow.

2 Experiment AI-Memory1

MNIST handwritten digits "0" recognition.
AI-Memory1 code: https://github.com/jiurl/AI-Memory1
all details are in code.
AI-Memory1 focuses on conv1 learning, remove those parts which belong to Memory2.

### 2.1 Network Architecture

Entire network consists of two convolution layers.
- **conv1.** kernel size: 7, stride: 2. number of features: 300. learning steps: 1500,
- **conv2.** kernel size: 11, stride: 1. number of features: 100. learning steps: 1000.

### 2.2 Training Data

MNIST dataset, 500 images of digit "0".
New learning rule, will not need data labeling in the future.

### 2.3 The Features Learned by Conv1

![Figure 2-1](image)

As shown in Figure 2-1.
In weight figure, pink color is positive weight, blue color is negative weight.

### 2.4 Code Introduction

AI-Memory1 is a pure C++ program.
The parts of "Tensor" and "convolution layer computing input sum", are based on libtorch library. All rest parts are written by myself.
Main subject of code:
Implementations of things about neural networks.
The observation tool: TensorObserver.
All kinds of observations of neural networks.
The observation of all kinds of internal procedures, in neural networks running, is crucial and indispensable.
So, I design and implement a simple, stand-alone, observation tool: TensorObserver, which included in AI-Memory1, in source code form.
All kinds of observations, are all based on TensorObserver.
Most figures in this paper, are generated by TensorObserver too.
3 Learning Rule - Convolution Layer Part

Input is 1x28x28 image. conv1, kernel_size:7, stride:2, number of features:300.
so, conv1 has 300*11*11 neurons.

The neuron in convolution layer, is named as "cell".
A "cell input", is a sub input which size is input_channels*kernel_size*kernel_size.
Cell index: (channel, y, x).

For example, some conv1 cell inputs:
(0,0,0) cell input, is a y=0, x=0 (0*stride), 7*7 area of input image.
(0,0,1) cell input, is a y=0, x=2 (1*stride), 7*7 area of input image.
(0,1,0) cell input, is a y=2, x=0, 7*7 area of input image.

3.1 New Feature Generation

One input, the sub areas which not make neurons active, will directly become new features.
So, to generate new features, only one input(e.g. a image) is enough.
As shown in Figure 3-1-1, 3-1-2, 3-1-3.

If, one input, all sub areas make neurons active, then there is no new feature generation.
Detail in code: ConvolutionLayer::ForwardLearnNewFeature()

3.1.1 Select Cell Input Which Will Become New Feature

Iterate every (11*11 individual) cell input.
Cell input which make active, is forbidden to select.
Cell input which have too much overlap with cell input which make active, is forbidden to select.
Cell input which have too much overlap with cell input which has been selected, is forbidden to select.
Figure 3-1-2: new features generated in step 1

Figure 3-1-3: new features generated in step 7
So, if a cell input, not make active, have no overlap or little overlap (hyper parameter) with forbidden cell inputs, then this cell input will be selected, to become new feature.

Figure 3-1-1, shows selected cell inputs in step 0. These 7 selected cell inputs become 7 features.

At step 0, there are no features in conv1. So, all cell input not make active.
Two types selected cell input, are not drawn out on figure, they are:
If selected cell input’s inputs are all 0, then no feature generate.
If selected cell input’s positive inputs are too few (hyper parameter), then no feature generate.

Figure 3-1-2, shows selected cell inputs in step 1.
At step 1, there are 7 features in conv1. So, many sub areas of input, make active, these parts will not be selected. The sub areas which not make active, will be selected, to generate new features.

3.1.2 The Feature Generated From Cell Input

With a specified cell input, a corresponding feature could be generated, which make:
Similar cell input, will make the feature neuron active.
Distinct cell input, will not make the feature neuron active.
More introduction in Neuron Part.

3.2 Short-Term Memory, Long-Term Memory, Forget

Newly generated feature, is in a short-term memory state.
The short-term memory which frequently be activated, will finally turn to long-term memory.
The short-term memory which rarely be activated, will be forgotten.

The feature in a short-term memory state, can continue to learn, weights can be changed.
The feature in a long-term memory state, wouldn’t learn anymore, weights wouldn’t be changed.
That means, newly generated feature, will continue to learn.

Figure 3-2: a example of a feature being forgotten
3.3 Forget - Remove Unwanted Feature

Using forgetting, remove unwanted feature.

There are some unwanted features in newly generated features. These unwanted features, will be removed by forgetting.

As shown in Figure 3-2. w7 is a newly generated feature in step 1 (see Figure 3-1-2). It is a unwanted feature for digit "0" recognition. It is only being activated 1 time, in 100 steps. Because too few activations, at step 100, it is being forgotten, being removed.

3.4 A Design of Short-Term Memory, Long-Term Memory, Forget

By now, a simple design of short-term memory, long-term memory, forget, is: Just do some simple statistics. When a feature being activated, a count increase. Then, after a period of time, feature with a high activation rate, turn to long-term memory state, feature with a low activation rate, is forgotten.

3.5 Compete to Be Active

Compete to be active: In convolution layer neurons, in one area (across features), only one neuron is allowed to be active.

Detail in code: ConvolutionLayer::NmsInAllNeurons()
Competing to be active, can be considered as, neurons compete for some resources, or there are inhibitions between neurons. It is a very important part for a convolution layer activation routine. As shown in Figure 3-4.

[Figure 3-4]

Neuron (0,3,4) inhibit, neuron (0,2,5), (0,5,3), (1,3,6), (3,5,5) activation. Neuron (9,10,6) inhibit, neuron (1,10,5), (3,8,7), (5,9,4), (6,9,6), (7,10,4), (7,10,5), (9,9,7) activation. Neuron (3,7,8) inhibit, neuron (3,6,8), (4,6,8), (9,8,8) activation.

3.6 Continuing Learning of Short-Term Memory Feature

Newly generated feature (short-term memory state), will continue to learn. Continuing learning of short-term memory feature, is very important, will play an important role in the future.

3.6.1 Positive Learning

When a feature (short-term memory state) being activated by a cell input, the feature will do a positive learning to this cell input. The feature will change towards this cell input direction.

As shown in Figure 3-5.

At step 1, the input make feature 0, 2, 3, 5, 6 (newly generated in step 0) active. Cell input (1,6), (3,3) make feature 0 active. Feature 0 will do a positive learning to cell input (1,6), (3,3).
Cell input (2,9), make feature 2 active. Feature 2 will do a positive learning to cell input (2,9).
Cell input (6,3), make feature 3 active. Feature 3 will do a positive learning to cell input (6,3).
Cell input (9,3), make feature 5 active. Feature 5 will do a positive learning to cell input (9,3).
Cell input (9,6), make feature 6 active. Feature 6 will do a positive learning to cell input (9,6).
3.6.2 Negative Learning

When a cell input, simultaneously make multiple features active, the features except the max activation one, will do a negative learning to this cell input. The features will change towards distinction this cell input direction.

As shown in Figure 3-6.

At step 25, the cell input (6,1) simultaneously make feature 38, 40, 50 active.
Input sum of feature 38 is max, 0.920.
Feature 38 will do a positive learning to cell input (6,1).
Feature 40 will do a negative learning to cell input (6,1).
Feature 50 will do a negative learning to cell input (6,1).
3.7 Internal Procedure of Learning

```
torch::Tensor
ConvolutionLayer::Forward(const torch::Tensor& input, bool is_learning)
{
    torch::Tensor input_sum = ComputeInputSum(input);

    // compete to be active
    torch::Tensor output;
    torch::Tensor active_of_neurons;
    vector<vector<CellInfo>> suppress_list;
    std::tie(output, active_of_neurons, suppress_list) =
        ForwardActivation(input_sum);

    // learning
    if (is_learning == true)
    {
        ForwardNegativeLearn(input, suppress_list);
        ForwardLearnNewFeature(input, active_of_neurons);
        ForwardPositiveLearn(input, active_of_neurons);
        MemoryStateTransition();
    }

    return output;
}
```
4 Learning Rule - Neuron Part

4.1 The Design of Neuron

As shown in Figure 4-1-1, 4-1-2, 4-1-3.
4.1.1 Output

Output of neuron set as 0-1.
Output of neuron is no negative value. Inhibition is achieved by negative weights.
This setting, imply a hypothesis, from the perspective of output, every neuron is equal.

4.1.2 Activation

As shown in Figure 4-1-1, 4-1-2, 4-1-3.
There are two activation threshold, min_activation_threshold, max_activation_threshold.
When input sum of neuron is less than min_activation_threshold, neuron is not active, output is 0.
When input sum of neuron is greater than min_activation_threshold, neuron is active, output is greater
than 0.
When input sum of neuron is greater than max_activation_threshold, output is 1.
When input sum of neuron is between min_activation_threshold and max_activation_threshold, output
is k*input_sum+b, a value in 0-1.

pseudo code:

```java
if (input_sum <= min_activation_threshold) output = 0;
else if (input_sum >= max_activation_threshold) output = 1;
else output = k * input_sum + b;
```

4.1.3 The Intention of Weight Design

As shown in Figure 4-2-1, 4-2-2, 4-2-3. (hide "0.", e.g. "0.027" display "027")
There are three inputs: a1, a2, b1. a1, a2 are similar inputs. a1, b1 are very different inputs.

There’s a neuron, the intention is:
The weights of neuron, is generated from the input a1. Then it make:
when input is a1, the neuron is active.
when input is a2, the neuron is active.
when input is b1, the neuron is not active.

4.1.4 Weight Design
The weights of a neuron, have positive value, have negative value. The total amount of positive weights is 1. Any neuron is same. The total amount of negative weights, is different for different neuron.

The negative weights is very important, indispensable, for prevent wrong activations.

Figure 4-2-4, shows weights generated from input a1. Figure 4-2-5, shows weights generated from input b1.

4.2 Weights Generation - Weight Dispatch
Generating weights by weight dispatch.
Detail in code: Neuron::LearnNewWeight()
With a specified input:

Positive weights generation:
Every connection which stimulus is greater than 0, will get some positive weight, from the total amount of positive weight.
The stimulus is stronger, the positive weight being dispatched is more.
The stimulus is 0, then no positive weight being dispatched.
The total amount of positive weight will all be dispatched to connections, by the stimulus intensities of connections.

Negative weights generation:
Every connection which stimulus is 0, will get some negative weight. The dispatch value are all same.
The negative weight dispatch value, is in proportion to average positive weight value of one connection.

Figure 4-3, shows internal procedure of generating weights from input a1.

4.2.1 Meaning of Activation Threshold
The weight design, make the input_sum reflecting the degree of similarity between input and weight. So, min_activation_threshold, max_activation_threshold can be setted, to your needed.

4.3 Continuing Learning of Weight
After generated, the weight can continue to learn.

4.3.1 Positive Learning
When a neuron is being activated, it can do positive learning to this input.
The intention of positive learning is, according to this input, changing weights, make weight a little more similar with this input.

4.3.2 Negative Learning
When a input simultaneously activate multiple neurons, then except the max activation one, others will do negative learning.
The intention of negative learning is, changing weights, make weight a little more distinction.
4.4 Continuing Learning Design - Weight Recycle, Weight Dispatch, Weight Refresh

4.4.1 Positive Learning

Detail in code: Neuron::PositiveLearn()

As shown in Figure 4-4.

First, recycle some positive weight, from the current weights.
Then, dispatch the recycled positive weight to the weights, according to this input.
So, the weight will change towards this input direction.

After these two steps, the total amount of positive weight, has been changed, usually not 1 anymore.
The total amount of negative weight, has been changed too.
So, by weight refreshing, to refresh the total amount of positive weight to 1 again. And refresh the total amount of negative weight to the origin level.

pseudo code:

```cpp
def positive_weight_recycle(w, positive_recycle_rate):
    w += recycle
    dispatch(w, input, recycle)

def positive_weight_refresh(w, positive_weight_total_amount):
    w -= refresh
```

4.4.2 Negative Learning

Detail in code: Neuron::NegativeLearn()

First, recycle some negative weight, from the current weights.
Then, dispatch the recycled negative weight to the weights, according to this input.
So, the weight will change towards to inhibit this input direction.

Then, by weight refreshing, to refresh the total amount of positive weight to 1 again, and refresh the total amount of negative weight to the origin level.

pseudo code:

```cpp
def negative_weight_recycle(w, negative_recycle_rate):
    w -= recycle
    dispatch(w, input, recycle)

def negative_weight_refresh(w, negative_weight_total_amount):
    w += refresh
```
5 Learning Rule - 4c99011c Part

The content of AI-Memory1 is interesting. And the content is plenty too. Learning rule - 4c99011c (4c99011c4c99011c1c7805508ecb764e0b233c237a12232ac), is a very very important rule, and is a astonishing rule. For keep some suspense, and for easier to understand, learning rule - 4c99011c will be published in AI-Memory2. AI-Memory2 has been done, coming soon.

6 Learning Rule - Long-Term Accumulative Learning

6.1 Long-Term Accumulative Learning

For example,
neural networks can learn digit "0" first, after a while, learn digit "1". learning digit "1", won’t affect the memory of digit "0" learned before.
So, neural networks can keep learning continuously.

6.2 Long-Term Accumulative Learning Rule

For every layer, long-term memory features won’t be changed, won’t be deleted. Once a feature turned to long-term memory, then exist forever.
The features of this layer, only base on long-term features of previous layer, not base on any short-term features of previous layer.
From begin to end, learn layer by layer.
By this way, learning new things, wouldn’t change any features (of any layer) learned before. Just generating new features (of various layers).

6.3 Features Accumulate and Share

Features learned before, will be used in new things learning, will make new things learning a lot easier.
For example,
Some features (of various layers) learned in digit "1", maybe are active and used in digit "7" learning. No need to generate these features again.

7 Contact

Email: jiurl@outlook.com
More Contact Information: https://jiurl.github.io
AI-Memory1 Homepage: https://github.com/jiurl/AI-Memory1

8 Some Chat

8.1 Can True Artificial Intelligence Be Made Out?

Yes. In my opinion, 100% yes.
At present, percent complete is very low. It is faraway and difficult. But it could be done. And it will be done finally.
All kinds of rules of memory, all kinds of rules of thinking, lots of things need to do. In software aspect, lots of things need to do too.
8.2 Some Chat

The long-term objective of Ultimate AI project is, making out True Artificial Intelligence. Wish a lot of people can join this project, we do this together. Wish some people can support this project, by various means.

I’m looking foward to some cooperation, if you are interested, contact me please. Wish people who are interested in AI, can communicate a lot together. That must be fun.

References

[1] https://nba.uth.tmc.edu/neuroscience/
[2] Principles of Neural Science