Report: Realistic Evaluation of Deep Semi-Supervised Learning Algorithms

Bouizaguen, Suliman
Kaina Mohamed, Abdellah
NGREMMADJI, Mbaimou Auxence

13 Janvier 2020

Introduction

This work aims to understand some methods of evaluation of semi-supervised learning models proposed by **Olivier and al** in [1]. For numerical implementation, [2] served us as guide.
The outline of this report is as followed :
In the first section, we will give an overview of paper [1]. The second paragraph will dedicated to our numerical implementation.

I Overview of paper

The main idea developped in [1] is about how to implement realistically evaluation of semi-supervised methods used in deep learning.
When a given dataset misses some targets, supervised learning approach can not be used for classification. Indeed, in supervised learning framework, we always need dataset which contains features x and target y, where $(x, y) \in \mathcal{D}$. How can we deal with the dataset which has a few number of labeled features $(x, y) \in \mathcal{D}$ and a great number of unlabeled features $x2 \in \mathcal{D}2$?
In [1], several methods are applied with interesting recommendations. We can list these recommandations in the following points :

1) Vary both the amount of labeled and unlabeled data when reporting performance method.
2) Using of the exact underlying model when comparing semi-supervised approaches.
3) As Semi-Supervised Learning methods studied in [1] suffer when the unlabeled data came from different classes than the labeled data, it is recommanded to report result when the class distribution mismatch systematically varies.
4) Take care not to over-tweak hyperparameters on an unrealistically large validation set.

The methods mentionned in [1] are : π-Model, Mean teacher, Virtual Adversarial Training (VAT), Entropy Minimization (EntMin) and Pseudo-Labeling.

a) Pi-Model : the prediction function $f(\theta, x)$ may give differents value (f is stochastic function). To minimize this variation, loss term is added to encourage the distance between a network's output for different passes of $x \in \mathcal{D}2$ through the network to be small.
b) As Pi-model can make unstable target for $x \in \mathcal{D}2$ (unlabeled feature), Mean teacher method is used to stabilize it by setting the target to predictions made using an exponential moving average of parameters.
c) Virtual Adversarial Training (VAT) method approximates a perturbation to add to $x \in \mathcal{D}2$ which affect the output the predicted.
d) Entropy Minimization (EntMin) method adds a loss term that encourages the network to make confident predictions for all unlabeled data.
e) Pseudo-Labeling method consists in producing "pseudo-labels" of unla-

beled feature. Then, Pseudo-labels which have a corresponding class probability which is larger than a predefined threshold are used as targets for a standard supervised loss function applied to unlabel features.

We will apply $Pi-$ model to mnist dataset by using only ten labels. Pi-model is clearly detailed by **Samuli and al** in [2]. We used carefully this paper for our numerical implementation. The π-model pseudo-code is given as followed :

Require: x_i = training stimuli
Require: L = set of training input indices with known labels
Require: y_i = labels for labeled inputs $i \in L$
Require: $w(t)$ = unsupervised weight ramp-up function
Require: $f_\theta(x)$ = stochastic neural network with trainable parameters θ
Require: $g(x)$ = stochastic input augmentation function
for t in $[1, num\_epochs]$ **do**
  for each minibatch B **do**
    $z_{i\in B} \leftarrow f_\theta(g(x_{i\in B}))$ ▷ evaluate network outputs for augmented inputs
    $\tilde{z}_{i\in B} \leftarrow f_\theta(g(x_{i\in B}))$ ▷ again, with different dropout and augmentation
    $loss \leftarrow -\frac{1}{|B|}\sum_{i\in(B\cap L)} \log z_i[y_i]$ ▷ supervised loss component
      $+ w(t)\frac{1}{C|B|}\sum_{i\in B} ||z_i - \tilde{z}_i||^2$ ▷ unsupervised loss component
    update θ using, e.g., ADAM ▷ update network parameters
  end for
end for
return θ

FIGURE 1 – Π-Model pseudo-code

II Application of a semi-supervised method : Π-Model

In this section, we will focus on a semi-supervised learning algorithm known as Π-Model. The purpose of this method is to use labeled data as well as unlabeled data to train the network. Thus, the model is supposed to perform better than when trained only on labeled data. The dataset we will be considering is MNIST digits, with only 100 labeled samples and 58900 unlabeled samples. The validation set is composed of 1000 labeled samples.

II.1 Overview of Π-Model

Independently of the underlying neural network architecture, the main idea of the training pass of Π-Model can be summed up in the following figure extracted from [2].

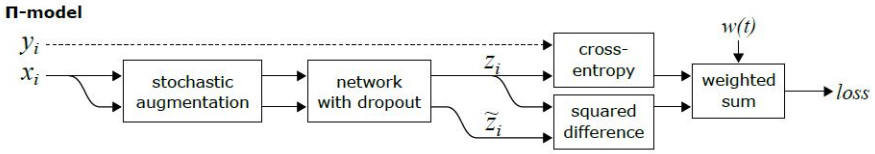

FIGURE 2 – Π-Model training pass

As one can infer from the FIGURE 2, there is two evaluation of the network during training for each input x_i. Indeed, the two output predictions are noted z_i and $\tilde{}z_i$. The loss function has two components, one "supervised" component i.e. cross-entropy which is evaluated only for the labeled inputs and an "unsupervised" component i.e. mean square difference between z_i and $\tilde{z}_i$. The total loss is the sum of the supervised component and the unsupervised component scaled by a time dependent factor w(t).

This factor is crucial insofar as at the beginning of the training the total loss and the learning gradients must be dominated by the supervised loss component, i.e., the labeled data only. The rampup of the factor must be slow enough—otherwise, the network gets easily stuck in a degenerate solution where no meaningful classification of the data is obtained.
In the articles [2], the unsupervised loss weighting function w(t) ramps up, starting from zero, along a Gaussian curve during the first 80 training epochs and then decreases along a Gaussian curve as well.
However, it seemed that for our implementation, considering an increasing factor w(t) along with a Gaussian curve during the whole 30 epochs (number of epochs is 30) yields better results.
The factor considered is $w(t) = \exp(-5 \times (1 - \frac{t}{80})^2)$

II.2 Underlying convolutionnal neural network (CNN)

We use the underlying CNN architercture. The figure below is about this architecture.

```
layer1 = tf.nn.conv2d(input=x,filter=weights['conv1w1'],strides=[1, 1, 1, 1], padding='SAME')
layer1 = tf.layers.batch_normalization(layer1,training=is_training)
layer1 = tf.nn.dropout(layer1, keep_prob)
layer1 = tf.nn.conv2d(input=layer1,filter=weights['conv1w2'],strides=[1, 2, 2, 1], padding='SAME')
layer1 = tf.layers.batch_normalization(layer1,training=is_training)
layer1 = tf.nn.leaky_relu(layer1,alpha=0.1)

layer2 = tf.nn.conv2d(input=layer1,filter=weights['conv2w1'],strides=[1, 1, 1, 1], padding='SAME')
layer2 = tf.layers.batch_normalization(layer2,training=is_training)
layer2 = tf.nn.dropout(layer2, keep_prob)
layer2 = tf.nn.conv2d(input=layer2,filter=weights['conv2w2'],strides=[1, 2, 2, 1], padding='SAME')
layer2 = tf.layers.batch_normalization(layer2,training=is_training)
layer2 = tf.nn.leaky_relu(layer2,alpha=0.1)

layer_shape = layer2.get_shape()
num_features = layer_shape[1:4].num_elements()
layer_flat = tf.reshape(layer2, [-1, num_features])

fc1 = tf.add(tf.matmul(layer_flat, weights['fc1_w']), biases['fc1_b'])
fc1 = tf.layers.batch_normalization(fc1,training=is_training)
fc1 = tf.nn.leaky_relu(fc1,alpha=0.1, name='fc1')
fc1 = tf.nn.dropout(fc1, keep_prob)

#Output layer with a linear activation, don't put the softmax here
output_layer = tf.add(tf.matmul(fc1, weights['out_w']), biases['out_b'],name='output')
```

FIGURE 3 – CNN architecture

CNN supervised baseline with 100 labels give the following result :

```
| Epoch: 1000/1000 | Train: Loss 0.000018 Accuracy : 1.000000 | Test: Loss 1.539110 Accuracy : 0.761000

<matplotlib.legend.Legend at 0x7f58155d0828>
```

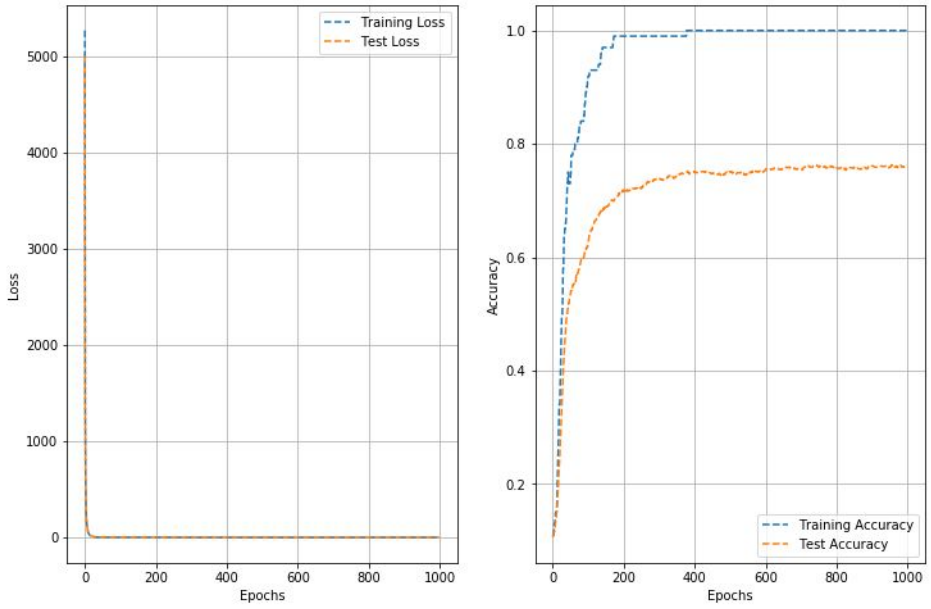

FIGURE 4 – CNN results

This classic CNN used with one hundred labels gives an accuracy around 76.10%.

II.3 Results of Π-Model for 100 labels

We trained our model for 30 epochs with only 100 labels and 58900 unlabeled samples. An issue that we faced was to impose a minimal number of labeled samples in each batch. Indeed, the total absence of labeled data in a batch degenerates the training and prevents the algorithm to converge.

The results of the evaluation of Π-Model are displayed on the following figure.

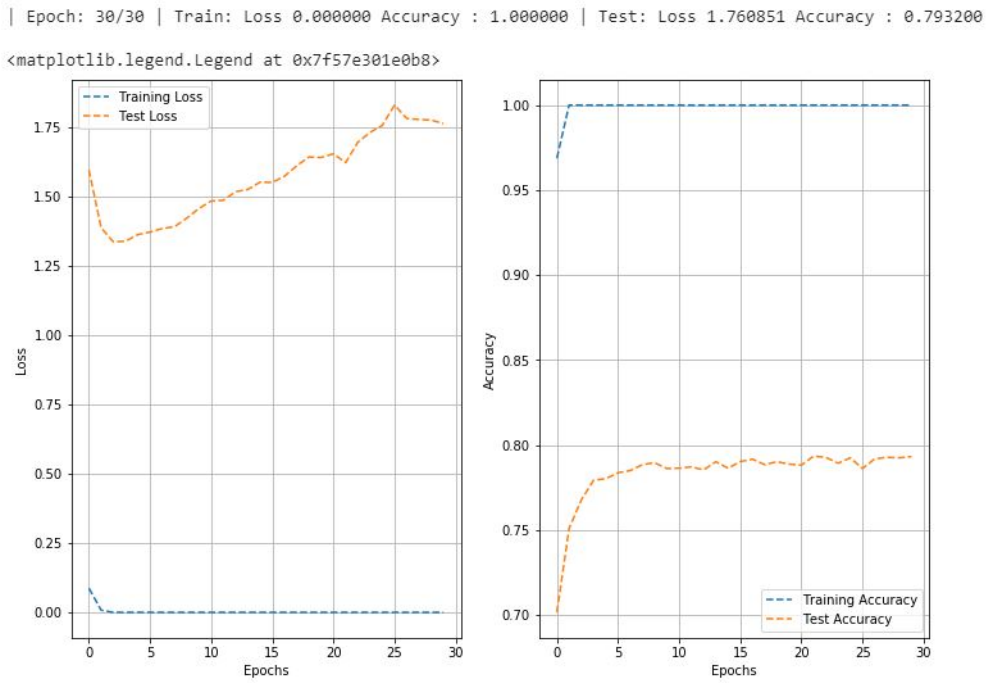

FIGURE 5 – CNN architecture

REMARKS : We notice that the model reached an accuracy of 79.32%, which is a little bit more than 3% better than the supervised approach with the underlying convolutionnal neural network.

III Conclusion

Although the π-model gives better result in our case, the selection of hyperparameters need some efforts (optimization of hyperparameters). As mentionned in [1], one can state that semi-supervised learning methods may lead to unsatisfactory result when taking into account the cost of optimization of the hyperparameters.

Références

[1] Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, Ian J. Goodfellow Zhixun Su, and Xi. Realistic evaluation of deep semi-supervised learning algorithms.

[2] Samuli Laine and Timo Aila. Temporal ensembling for semi-supervised earning.