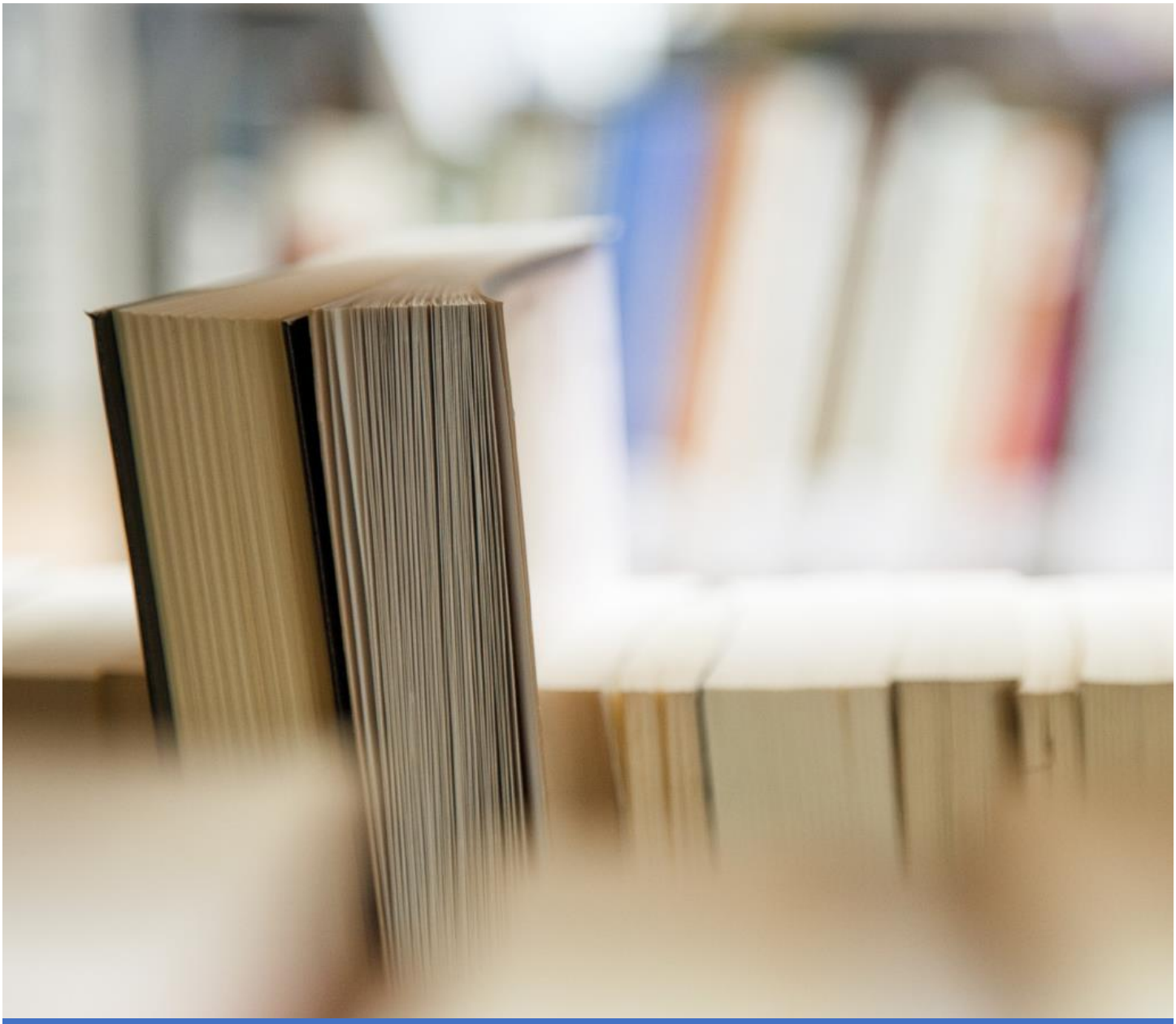


DEEP LEARNING USING LINEAR SUPPORT VECTOR MACHINE

Paper Reproduction

2020021327 조경선[대학원석사과정재학 / 산업경영공학과]

11/1/20

BUSINESS ANALYTICS

Paper Summary

합성곱 신경망(Convolutional neural network) 은 음성 인식, 이미지 분류, 자연어 처리 및 생물정보학과 같은 다양한 작업에서 최고의 성능을 달성하도록 훈련한다. 분류(classification) 문제에서 예측과 cross-entropy 최소화를 위해 대부분의 Deep Learning 모델들은 softmax activation function 을 사용한다. 이 논문에서는 softmax layer 를 linear support vector 로 변환하여 작지만 일관되는 장점이 있는 모델을 설명하고 있다. 학습은 Cross-entropy loss 대신에 margin-based loss 를 최소화한다. 모델에 대한 실험으로 MNIST, CIFAR-10 등의 Data 를 사용한다.

The Model

(1) Softmax

Deep Learning 을 사용하는 분류(Classification) 문제에서 softmax 나 1-of-K encoding 방식을 사용하는 것이 일반적이다. Softmax 는 다음과 같은 수식을 갖는다.

$$a_i = \sum_k h_k W_{ki}$$

$a: total\ input\ into\ a\ softmax\ layer$

$h: penultimate\ layer\ node\ 의\ activation$

$W: penulimate\ layer\ 에서\ the\ softmax\ layer\ 로\ 연결되는\ 가중치$

이 때 discrete probability distribution 은

$$p_i = \frac{\exp(a_i)}{\sum_j^N \exp(a_{ij})}$$

이고, 예측 class $\hat{\imath}$ 는 다음과 같다.

$$\hat{\imath} = \arg\max p_i = \arg\max a_i$$

Softmax 에 대한 기법을 코드로 구현하면 다음과 같다.

```
def softmax():
    loss =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=output,
labels=input))
```

(2) Support Vector Machines

Linear Support Vector Machines(SVM) 은 binary 분류를 위한 공식이다. SVM 은 다음과 같이 모델화 된다.

$$\min \frac{1}{2} w^T w + C \sum_{n=1}^{N} \xi_n$$
$$s.t. w^T x_n t_n \geq 1 - \xi_n \quad \forall_n$$
$$\xi_n > 0 \quad \forall_n$$

$\xi_n: slack\ variables\ which\ penalizes\ data\ points\ which\ violate\ the\ margin\ requirements$

모든 data vector x_n 는 1 값으로 bias 를 포함시킬 수 있는데, 이를 나타내면 다음과 같다.

$$l(w) = \min \frac{1}{2} w^T w + C \sum_{n=1}^{N} \max(1 - w^T x_n t_n, 0) \quad (Equation\ 1)$$

이를 standard hinge loss 를 가진 L1-SVM 이라고 한다. 하지만 이는 미분할 수 없기 때문에 squared hinge loss 를 최소화 하는 L2-SVM 를 변형하여 사용한다.

$$l(w) = \min \frac{1}{2} w^T w + C \sum_{n=1}^{N} \max(1 - w^T x_n t_n, 0)^2 \quad (Equation\ 2)$$

이는 미분 가능하고,margin 을 위반하는 포인트에 대해 더 큰 손실을 부과할 수 있다.
테스트 데이터 x 의 클래스를 예측하려면, 다음과 같다.

$$\arg\max (w^T x) t$$

이를 구현하면 다음과 같이 정의할 수 있다.

```
def svm(C):
    """
    :param C: The SVM penalty parameter.
    """
    reg_loss = tf.reduce_mean(tf.square(weight))
    hinge_loss = tf.reduce_mean(
        tf.square(
            tf.maximum(tf.zeros([batch_size, class_cnt]), 1 - input * output)
        )
    )
    total_loss = reg_loss + C * hinge_loss
```

(3) Deep Learning with Support Vector Machines

하위 계층 가중치는 상위 계층 linear SVM 에서 기울기를 역 전파하여 학습한다. 이 때문에 penultimate layer activation 와 관련하여 SVM 목표를 미분한다.

이 논문 에서는 L2-SVM 을 사용한다. 그 이유는 실험에 의해서 L1-SVM 보다 L2-SVM 이 더 나은 성능을 보였기 때문이다.

$(Equation\ 2)$ 를 미분하면 다음과 같은 식을 얻을 수 있다.

$$\frac{\partial l(w)}{\partial h_n} = -2Ct_n w(\max(1 - w^T h_n t_n, 0))$$

Experiments

논문에서는 총 3 개의 Data(Facial Expression Recognitio, MNIST, CIFAR-10) 를 활용하여 실험하였다.

이 중에서 MNIST 을 통해서 비교해 보았다.

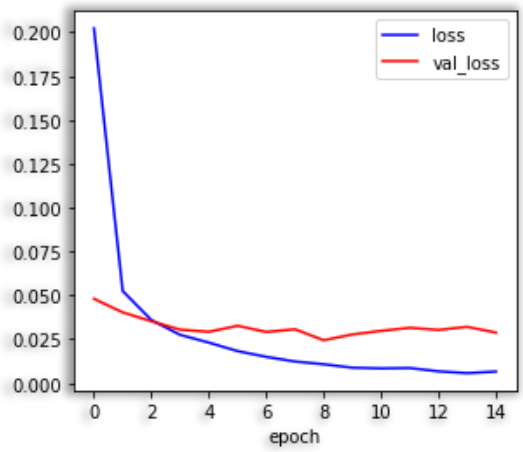
(1) MNIST

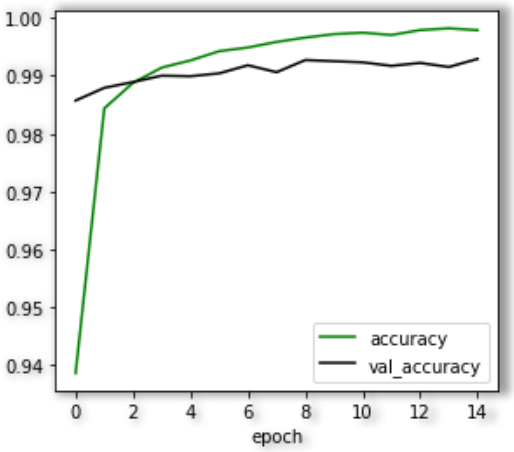
(1.1) SOFTMAX

Softmax 는 아래와 같이 구현하였다.

```
learning_rate = 0.001
training_epochs = 15
batch_size = 100

mnist = keras.datasets.mnist
class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

train_images, test_images = train_images / 255.0, test_images / 255.0

softmax_model = models.Sequential()
softmax_model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)))
softmax_model.add(layers.MaxPooling2D((2, 2)))
softmax_model.add(layers.Conv2D(64, (3, 3), activation='relu'))
softmax_model.add(layers.MaxPooling2D((2, 2)))
softmax_model.add(layers.Conv2D(128, (3, 3), activation='relu'))
softmax_model.add(layers.Flatten())
softmax_model.add(layers.Dense(64, activation='relu'))
softmax_model.add(layers.Dense(10, activation='softmax'))

softmax_model.summary()

softmax_model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

# softmax_model.fit(train_images, train_labels, epochs=training_epochs)
softmax_model.fit(train_images,
                  train_labels,
                  batch_size=batch_size,
                  epochs=training_epochs,
                  verbose=1,
                  validation_data=(test_images, test_labels))

test_loss, test_acc = softmax_model.evaluate(test_images, test_labels,
verbose=2)
```

이 결과는 accuracy 가 0.9929 로 측정되었고, Loss 와 accuracy 를 그래프로 그려보면 다음과 같다.


loss
val_loss
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
2
4
6
8
10
12
14
epoch

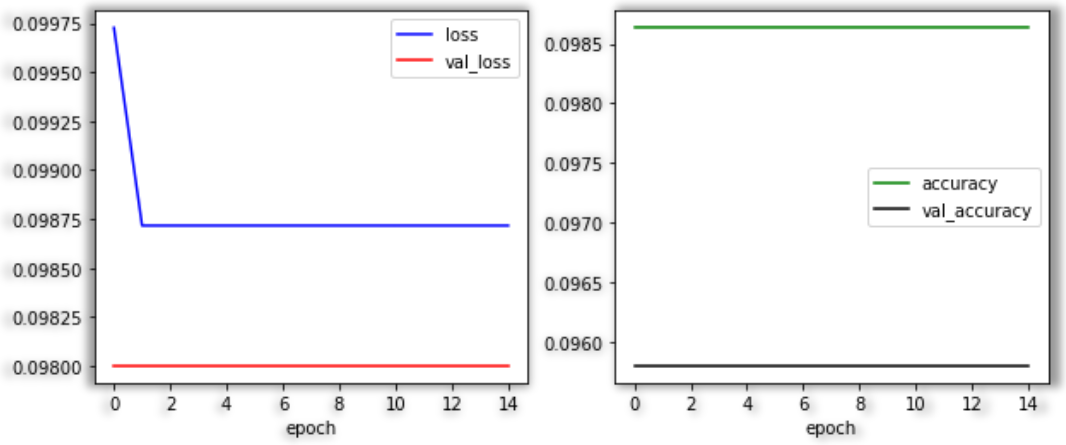

accuracy
val_accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0
2
4
6
8
10
12
14
epoch

(1.2) SVM

SVM 는 아래와 같이 구현하였다.

```
learning_rate = 0.001
training_epochs = 15
batch_size = 100

mnist = keras.datasets.mnist
class_names = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

train_images, test_images = train_images / 255.0, test_images / 255.0

num_classes=train_labels.shape[0]

def getWeight(shape):
    initial = tf.random.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


def getBias(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


def svm(batch_size, num_classes, C):
    weight = getWeight([1024, num_classes])
    bias = getBias([num_classes])
    output = weight + bias

    reg_loss = tf.reduce_mean(tf.square(weight))
    hinge_loss = tf.reduce_mean(
        tf.square(
            tf.maximum(tf.zeros([batch_size, num_classes]), 1 - train_la-
bels * output)
        )
    )
    total_loss = reg_loss + C * hinge_loss
    return tf.fill(1, total_loss)

get_custom_objects().update({'svm': Activation(svm)})
svm_model = models.Sequential()
svm_model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28,
28, 1)))
svm_model.add(layers.MaxPooling2D((2, 2)))
svm_model.add(layers.Conv2D(64, (3, 3), activation='relu'))
svm_model.add(layers.MaxPooling2D((2, 2)))
svm_model.add(layers.Conv2D(128, (3, 3), activation='relu'))
svm_model.add(layers.Flatten())
svm_model.add(layers.Dense(64, activation='relu'))
svm_model.add(layers.Dense(32 , activation=svm(1024,num_classes,1.0)))
svm_model.summary()

# CNN 모델 구조 확정하고 컴파일 진행
svm_model.compile(optimizer='adadelta',
                  loss='squared_hinge',
                  metrics=['accuracy'])
```

이 결과는 accuracy 가 0.0958 로 측정되었고, Loss 와 accuracy 를 그래프로 그려보면 다음과 같다.



Conclusion

Paper 에 따르면, MNIST 같은 경우에는 error 율이 다음과 같았다.

Softmax	DLSVM
0.99%	0.87%

이처럼 softmax 보다도 DLSVM 을 사용할 때 더 낮은 에러율을 보였지만, 실제로 내가 수행한 실험에서는 아주 많이 좋지 않는 결과가 나왔다.
여러가지 이유가 있겠지만 SVM 기법을 사용하기 위한 parameter 를 추정하는 과정이 미흡했던 것으로 생각이 된다.
좀 더 정확한 parameter 추정으로 다시 한번 실험을 해 볼 것을 과제로 남기고자 한다.