<활성화 함수의 변형을 통한 딥러닝의 정확도 유지 및 향상 가능성 탐구>

Ⅰ. 서론

1. 탐구 동기

작년 정보과학 R&E 활동에서 딥러닝을 다루면서 딥러닝을 이해하기 위해선 많은 수학적 개념에 대한 이해가 필요하다는 사실을 알게 되었다. 그러면서 딥러닝 속 수학적 개념에 대해 깊게 알아봤었는데, 딥러닝은 기본적으로 인공신경망이 데이터를 입력받고 계산하여 출력하는 작업의 반복으로 이루어진다. 이때 데이터의 출력 값이 생성될 때 계산에 적용되는 함수를 "활성화 함수"라고 하며, 이는 기계 학습에서 매우 중요하다. 그러나 이런 활성화 함수들의 비선형성 때문에 종류를 바꾸는 방법 이외에는 정확도 향상을 위한 후처리를 활성화 함수를 통해 진행하기는 어려우며 동형암호와 같은 선형 시스템에 딥러닝을 적용하기는 어렵다. 그래서 여러 활성화 함수들을 미분을 이용한 근사 방법 중 하나인 테일러 근사를 활용하여 선형으로 나타낸 후 함수를 조정하면 정확도를 쉽게 향상시킬 수도 있고, 선형 시스템에도 활용이 가능할 것 같아 이와 같은 주제를 선정하였다.

2. 탐구 목표

활성화 함수를 테일러 근사를 통해 변형시킨 후 학습을 시켜도 원래의 결과보다 정확도가 유지되거나 향상되는지를 알아본다.

3. 관련 단원

2.9 Linear Approximations and Differentials / 11.10 Taylor and Maclaurin Series

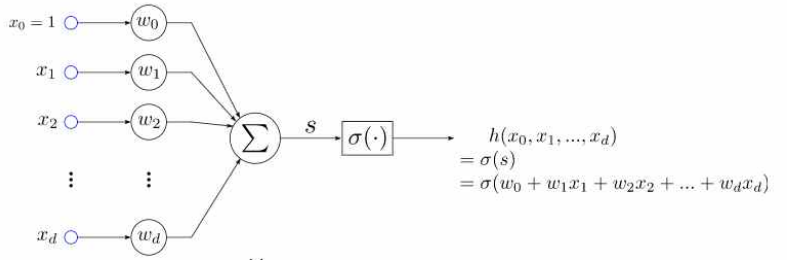
Ⅱ. 서론

1. 딥러닝

딥러닝은 인공신경망(Artificial Neuron Network)을 이용한 학습 모델을 통해 머신러닝을 진행하는 것으로 머신러닝 기법의 일종이라고 할 수 있다. 이때 머신러닝이란 Input data와 Output data 사이의 관계를 컴퓨터가 스스로 학습하여 알아내는 과정을 의미한다.

2. 퍼셉트론

인공신경망은 여러 개의 퍼셉트론(perceptron)의 연결체로 생각할 수 있는데, 퍼셉트론이란 n개의 입력 벡터에 대하여 각 벡터에 대해 가중치(weight)를 곱한 값들의 합에 활성화 함수(activation function)을 적용하여 최종 출력 값으로 출력하는 선형 모델을 말한다. 즉, 다수의 신호를 입력받아 처리 과정을 거쳐 하나의 신호로 출력하는 모델이다. 아래 그림이 퍼셉트론의 모식도인데, n-dimensional 데이터 $x=(x_0,x_1,...,x_n)$이 입력되면 각 성분에 가중치 $(w_0,w_1,...,w_n)$을 곱하여 모두 합한 값을 $s=\sum_{k=1}^{n} w_k x_k$라고 하면, 퍼셉트론 $h(x)=\sigma(s)$로 나타낼 수 있다.

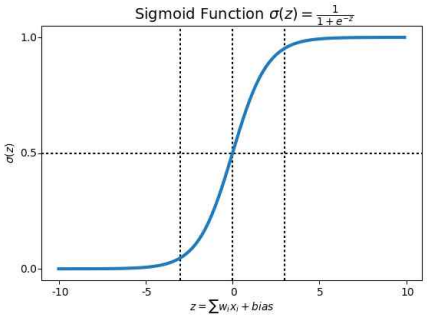


이 퍼셉트론의 반복을 통하여 다층의 인공신경망을 구현할 수 있으며, 이때 입력 벡터가 자리 잡는 층을 입력층, 출력 값이 자리 잡는 층을 출력층, 그 사이의 모든 층을 은닉층이라고 한다.

3. 활성화 함수

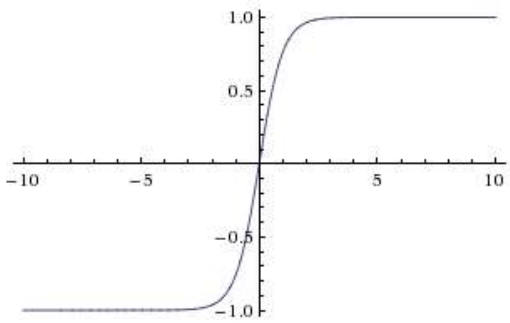
위의 퍼셉트론 설명에서 출력값을 최종적으로 결정하는 함수 $\sigma(s)$를 활성화 함수라고 한다. 즉, 활성화 함수는 어떠한 신호가 입력되었을 때 이를 적절한 처리를 하여 출력해주는 함수이다. 모든 활성화 함수는 **<u>비선형성</u>**을 띠는데, 만약 활성화 함수가 $y = cx$ 꼴의 선형이라면 multiple layer를 쌓아도 그 형태가 $y = c^n x = c'x$이기 때문에 layer를 쌓는 이유가 없기 때문이다.
대표적인 활성화 함수에는 다음과 같은 함수들이 있다.

1) Sigmoid function (시그모이드 함수)



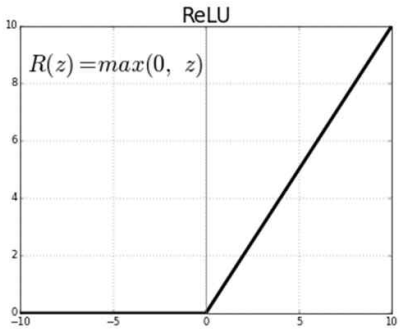
시그모이드 함수는 과거 신경망 학습에서 가장 많이 쓰였던 대표적 활성화 함수 중 하나이며, 그 수식은 $sigmoid(x) = \frac{1}{1+e^{-x}}$이다. 입력 신호를 (0,1) 사이의 값으로 normalize 해주지만, gradient vanishing 문제가 발생한다. 여기서 gradient vanishing이란 우리가 신경망을 통해 학습을 시킬 때는 신경망의 layer를 많이 쌓는 것이 중요한데, sigmoid function의 경우 $x = 0$일 때 미분값이 $\frac{1}{4}$로 최대가 된다. 이러면 신경망에 더 deep하게 들어갈수록 1보다 작은 기울기를 점점 곱하게 되면 그 값이 0에 수렴하게 되어 학습에 실패하게 된다는 것이다. 하지만 과거 많이 사용되었던 것은 step function 보다는 매끄러우며 변화가 연속적이어서 출력값의 다양성을 높일 수 있기 때문이다.

2) $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



입력값을 (-1,1)로 normalize 해주며 대부분의 경우에서 sigmoid보다 성능이 좋다. 하지만 역시 $x = 0$일 때 미분 값이 1로 최대여서 gradient vanishing 문제가 발생하나 sigmoid보다는 덜하다.

3) ReLU(Rectified Linear Unit) function



최근 신경망 학습에서 가장 많이 사용되는 활성화 함수로, 수식은 $f(x) = \max(0, x)$이다. 미분값이 0 또는 1이어서 gradient vanishing 문제가 없고, $x \geq 0$일 때는 입력값을 그대로 출력하므로 출력의 다양성도 어느 정도 보장된다. 특히 학습 속도에 매우 이점이 있는데, sigmoid보다 약 6배 빠르게 학습이 가능하다고 한다.

이외에도 ReLU의 gradient vanishing 문제 해결을 위한 PReLU($f(x) = \max(\alpha x, x)$), Softplus ($f(x) = \ln(1 + e^x)$)와 다중 클래스 분류의 출력층에서 많이 사용되는 Softmax 등이 있다.

4. 테일러 급수와 근사

1) 테일러 급수

매끄러운 함수(무한 번 미분이 가능한 함수) $f : R \rightarrow R$ 과 실수 a에 대하여 다음과 같은 멱급수 $T_f(x)$를 테일러 급수라고 한다.

$$T_f(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \ldots = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

(이때 $f^{(n)}(a)$는 $x = a$에서의 f의 n계 도함수)

이때 $a = 0$인 경우의 테일러 급수를 매클로린 급수라고 한다.

2) 테일러 급수를 통한 근사

테일러 급수를 활용하여 아래와 같이 함수의 근사가 가능하다.

Ex) $e^x = \sum_{n=0}^{\infty} \frac{(e^a)^{(n)}}{n!}(x-a)^n$

$$a = 0 \rightarrow e^x = \sum_{n=0}^{\infty} \frac{(e^0)^{(n)}}{n!}(x-0)^n = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \ldots$$

Ⅲ. 탐구 과정

이번 주제탐구에서는 python 3를 통해 모든 코드를 작성하며, 딥러닝 코드 작성을 위해 python 라이브러리인 keras를 사용한다. keras는 대부분의 딥러닝 모델을 간편하게 만들고 훈련시킬 수 있는 tensorflow 기반의 딥러닝 프레임워크로서 사용하기 쉬운 API를 가지고 있어 딥러닝 모델의 프로토타입을 빠르게 만들 수 있어 연구에서 자주 사용된다. 또한 이번 연구에서는 학습을 위한 데이터셋으로 MNIST 데이터셋을 사용한다. MNIST는 머신러닝 분야 최고 권위자로 불리는 뉴욕대 교수 Yann LeCun이 제공하는 손으로 쓴 숫자 0-9 이미지로 이루어진 대형 데이터베이스로, 기계학습 분야의 학습 및 테스트에 널리 사용된다. 딥러닝 모델로는 CNN(Convolutional Neural Network)을 사용한다.

1) 탐구에서 이용하는 활성화 함수

탐구에서 이용할 활성화 함수의 경우 CNN에서는 주로 이용되는 sigmoid, tanh, ReLU를 사용하기로 한다. 그런데 ReLU 함수의 경우 0의 값에서 미분이 불가능하기 때문에 테일러 근사를 적용할 수가 없다. 따라서 근사 시에는 ReLU 함수의 gradient vanishing 문제 해결을 위해 도입된 Softplus 함수를 사용하기로 한다.

2) 탐구 과정

먼저 keras를 이용해 CNN 모델 기반 MNIST 분류 문제 해결을 위한 python 코드를 작성한다. (코드는 별도 첨부)

```
#CNN
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), padding='same',
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1000, activation='modified_tanh')) #activation function
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

아래 그림의 경우 이번 탐구에서 사용된 CNN 모델의 layer들을 표시한 표인데, 7번째에 있는 dense 층이 활성화 함수가 적용되는 층으로 이 층에서 활성화 함수들을 변화시키며 탐구를 진행했다. 그 이후 sigmoid, ReLU, tanh 세 개의 함수를 활성화 함수로 지정하여 학습을 진행시켜 학습의 정확도 값을 얻어냈다.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 32)	832
max_pooling2d_7 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_8 (Conv2D)	(None, 14, 14, 64)	8256
max_pooling2d_8 (MaxPooling2	(None, 7, 7, 64)	0
dropout_7 (Dropout)	(None, 7, 7, 64)	0
flatten_4 (Flatten)	(None, 3136)	0
dense_7 (Dense)	(None, 1000)	3137000
dropout_8 (Dropout)	(None, 1000)	0
dense_8 (Dense)	(None, 10)	10010

Total params: 3,156,098
Trainable params: 3,156,098
Non-trainable params: 0

1차 학습 이후에는 python의 sympy 라이브러리가 제공하는 series 메소드를 통해 세 함수의 테일러 근사식을 구할 수 있었고, 각 식은 다음과 같다.

$$\tanh(x) = x - \frac{1}{3}x^3 + \frac{2}{15}x^5$$
$$sigmoid(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3 + \frac{1}{480}x^5$$
$$softplus(x) = \ln 2 + \frac{1}{2}x + \frac{1}{8}x^2 + \frac{1}{192}x^4$$

이후 이 식들이 0과 가까이에서 발산하지 않도록 다음과 같이 계수를 조정하여 실제 학습의 활성화 함수

로 사용하였다.

$$\tanh(x) = \frac{23}{25}x - \frac{1}{10}x^3 + \frac{7}{25}x^5$$
$$sigmoid(x) = \frac{1}{2} + \frac{x^5 - 8700x^3 + 540250x}{2700000}$$
$$softplus(x) = 0.66 + \frac{1}{2}x + \frac{2}{17}x^2 + \frac{1}{600}x^4$$

6번의 모든 학습에서 Adam opitimizer를 사용하였으며, Loss Function은 Cross Entropy 함수, epoch는 12, batch size는 128, learning rate는 0.001로 지정하여 학습을 진행하였다.

Ⅳ. 탐구 결과 및 결론

1) 탐구 결과

각 활성화 함수 별 학습 정확도는 다음과 같다.

(m_sigmoid, m_ReLU, m_tanh는 각 함수를 테일러 근사를 통해 변형시킨 경우를 나타낸다.)

활성화 함수	sigmoid	ReLU	tanh	m_sigmoid	m_ReLU	m_tanh
Accuracy	0.9929	0.9927	0.9905	0.9923	0.9913	0.9918

2) 결론

sigmoid, ReLU, tanh의 경우 모두 0.99를 넘는 높은 확률의 정확도를 보여주었으며, 이로 미루어 보아 학습 과정에서의 문제 혹은 모델 자체의 문제는 없었던 것으로 추측할 수 있다.

테일러 근사를 통해 변형한 세 개의 함수 모두 정확도 값이 0.99 이상으로 유지되었으며, tanh의 경우에는 정확도 값이 더욱 향상된 것을 볼 수 있었다. 이를 통해 우리는 활성화 함수의 근사를 통해 학습 정확도의 유지 및 향상이 가능하다는 결론을 낼 수 있다.

3) 기대 효과

테일러 근사를 이용한 활성화 함수의 변형을 통해서 딥러닝에 익숙하지 않은 이들도 쉽게 학습의 정확도를 향상시킬 수 있을 것으로 보이며, 동형암호와 같이 기존에 활성화 함수의 비선형성으로 인해 기계 학습을 적용하지 못했던 선형 시스템 분야에서의 기계 학습 적용이 가능해질 것으로 예상된다.

4) 보완점 및 추가로 연구하고 싶은 점

현재 연구의 경우 학습 시 모델이 학습 데이터에 지나치게 과적합되는 overfitting 현상을 고려하지 않았기에 위와 같이 모두 높은 정확도가 나왔을 가능성이 있다. 추후에는 모델이 overfitting 되지 않았는지를 확인하며 연구를 진행해봐야 할 것 같다.

또한, 위의 세 개의 함수를 제외한 다른 함수들로도 비슷한 과정의 연구를 해보고 싶고, MNIST 분류 문제는 이미 많은 방식으로 여러 사람들에 의해 다루어져 왔기 때문에 난이도가 어려운 편에 속할 수 없기 때문에 난이도가 더 높은 Fashion MNIST나 영상 처리 같은 문제에서도 활성화 함수의 근사와 변형을 통해 정확도를 유지 혹은 높일 수 있는지 알아보고 싶다.

Ⅴ. 참고 자료

http://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/

<밑바닥부터 시작하는 딥러닝>, 사이토 고키, 한빛미디어, 2017