Hyperparameter Tuning, Batch Normalization, and Programming Frameworks

Hyperparameter Tuning process:

최적의 파라미터를 찾는 가이드라인

1. 우선 순위

1 순위: learning rate α

2 순위: momentum β, mini-batch size, # hidden units

3 순위: # layers, learning rate decay

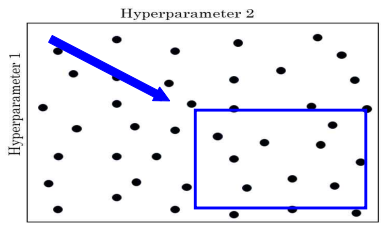
마지막: Adam optimizer의 hyperparameter($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$)

2. Try random values and Don't use a grid.

hyperparameter의 개수가 작을 때는 grid search로 일일이 확인해 볼 수 있지만 hyperparameter의 개수가 많으면 어렵고 확인하는 시간이 너무 길기 때문에 포인트의 개수는 같지만 랜덤하게 선택하여 확인해 보는 것이 좋다. 어떤 문제에서 어떤 hyperparameter가 가장 중요할지 미리 알 수 없기 때문이다.

3. Coarse to fine

성능이 좋은 hyperparameter 포인트 주변으로 줌인해서 이 공간에 대해 조금 더 밀도 있게 샘플링을 진행한다. 그리고 이 공간에 대해 랜덤 샘플링을 다시 진행할 수도 있다.



4. Using an appropriate scale to pick hyperparameters

hyperparameter를 랜담하게 골랐을 때 scale의 문제가 생긴다.

ex) α가 0.0001부터 1 사이에서 랜덤하게 골라졌을 때 대략 0.1~1에 90% / 0.0001~0.1에 10%로 분포되어 있을 것이다.

하지만 적절한 분포는 0.0001~0.001 / 0.001~0.01 / 0.01~0.1 / 0.1~1 에 비슷하게 나눠져 있어야 한다.

1. 10^a에 대해 $a = \log_{10}10^a$
2. 10^b에 대해 $b = \log_{10}10^b$
3. $r = (a-b) \times np.random.randn()$, $r \in [a, b]$, $a < b$
4. $\alpha = 10^r$, $\alpha \in [10^a, 10^b]$

여기서 β가 1에 가깝다면 조금 변하더라도 결과는 크게 변하므로 선형 scale로 샘플링하지 않는 것이다.

ex) $\beta = 0.9, \ldots, 0.999$일 때 $1-\beta = 0.1, \ldots, 0.0001$ 이므로

샘플링 개수 $\frac{1}{1-\beta} = 10, \ldots, 1000$ 이다.

β가 1에 가까울수록 혹은 $(1-\beta)$가 0에 가까울수록 샘플링을 밀도있게 진행한다.

β: 0.9000 → 0.9500 } 10개

β: 0.9999 → 0.9995 } 1000개

1. $r \in [a, b]$ 2. $1-\beta = 10^r$ 3. $\beta = 1-10^r$, $\beta \in [1-10^b, 1-10^a]$	r = np.random.rand() beta = 1-10**(- r - 1)

5. Re-test hyperparameters occasionally

데이터의 변화 혹은 추가, 하드웨어 업그레이드 등으로 상황이 변했다면 hyperparameters를 다시 찾아볼 필요가 있다.

6. Hyperparameters tuning in practive: Pandas vs. Caviar

Babysitting one model	Training many models in prarallel
산출 자원이 많이 없는 경우, 한번에 여러 모델을 트레이닝 시킬 여력이 없을 때 사용하는 방법 하루하루 혹은 매시간 직접 확인해가면서 hyperparameter tuning ⇒ Panda approach	parallel 방식으로 여러 모델을 트레이닝 시키는 방법 동시에 여러 모델를 다양한 hyperparameter 값으로 혼자 작동하게 놨둔다. 마지막에 가장 잘 작동하는 것을 찾을 수 있다. ⇒ caviar approach

Batch Normalization

batch norm은 hyperparameter 서치 문제를 훨씬 더 쉽고 신경망을 더욱 robust하게 만들어 준다. hyperparameter 서치를 도와줘 dnn에서도 더 쉽게 트레이닝할 수 있도록 해준다.

단순 logistic regression에서 noramlization하는 경우

$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$X = X - \mu$$

$$\sigma^2 = \frac{1}{m}\sum_i (x^{(i)})^2$$

$$X = \frac{X}{\sigma}$$

Can we normalize $A^{[l-1]}$, so as train $W^{[l]}$ and $b^{[l]}$ faster?

$A^{[l-1]}$ is the input to the next layer, that therefore affects training off $W^{[l]}$ and $b^{[l]}$.

이것이 batch nom이 하는 역할이다. 엄밀히 말하면 $A^{[l-1]}$가 아닌 $Z^{[l-1]}$를 정규화시키는 것이다. 학계에서는 $A^{[l-1]}$, 즉 activation function을 적용한 다음에 정규화를 해야 하는지에 대한 토론이 있다. 실제로는 $Z^{[l-1]}$를 정규화하는 것이 훨씬 더 자주 있다. 따라서 $Z^{[l-1]}$를 정규화하는 것으로 정리할 것이고 $Z^{[l-1]}$를 기본으로 하도록 추천한다.

Given some intermediate values in NN

$$\mu = \frac{1}{m}\sum_i z^{[l](i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i (z^{[l](i)} - \mu)^2$$

$$z^{[l](i)}_{norm} = \frac{z^{[l](i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{[l](i)} = \gamma z^{[l](i)}_{norm} + \beta$$

use $\tilde{z}^{[l](i)}$ instead of $z^{[l](i)}$

mean이 0, variance가 standard unit(즉 1) 갖도록 정규화를 했으나 hiddne units은 아마도 다른 분포도를 갖을 것이다.

이 때문에 이후에는 $\tilde{z}^{[l](i)}$로 학습시킨다.

γ와 β는 trainable variable 혹은 learnable vaiable로 $\tilde{z}^{[l](i)}$의 mean이 0, variance가 1이 되도록 한다.

만약 γ가 $\sqrt{\sigma^2+\epsilon}$이고 β가 μ이면 $\tilde{z}^{[l](i)} = z^{[l](i)}$ 일 것이다. γ와 β의 적합한 설정으로 정규화 단계에서 여기 4개의 식을 identity function을 계산한다. 하지만 gamma와 beta를 다른 값으로 고르면 hidden units이 다른 mean과 variance를 갖게 해준다.

Adding batch Norm to a network

$$A^{[l-1]} \xrightarrow{W^{[l]}, b^{[l]}} Z^{[l]} \underset{batch norm}{\xrightarrow{\gamma^{[l]}, \beta^{[l]}}} \tilde{Z}^{[l]} \rightarrow A^{[l]} = g^{[l]}(\tilde{Z}^{[l]})$$

parameters in layer l: $W^{[l]}$, $b^{[l]}$, $\beta^{[l]}$, $\gamma^{[l]}$ → $d\beta^{[l]}$ → $\beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$

※batch norm에서 β와 Adam에서 β와는 다르다.

Working with mini-batch

전체 데이터셋에 대해 업데이트하는 것이 아니라 mini-batch로 Z를 정규화한다.

$$A^{[l-1]\{t\}} \xrightarrow{W^{[l]}, b^{[l]}} Z^{[l]\{t\}} \underset{batch norm}{\xrightarrow{\gamma^{[l]}, \beta^{[l]}}} \tilde{Z}^{[l]\{t\}} \rightarrow A^{[l]\{t\}} = g^{[l]}(\tilde{Z}^{[l]\{t\}})$$

parameters in layer l: $W^{[l]}$, $\beta^{[l]}$, $\gamma^{[l]}$ → $d\beta^{[l]}$ → $\beta^{[l]} = \beta^{[l]} - \alpha\, d\beta^{[l]}$

※ batch norm에서 $b^{[l]}$은 어떤 값을 갖더라도 $b^{[l]}$은 없어질 것이다. batch norm에서 $Z^{[l]}$의 mean을 구하고 $Z^{[l]} - \mu$할 것이기 때문이다. 따라서 batch norm에서는 파라미터 $b^{[l]}$를 제거할 수 있다. 아니면 0으로 설정할 수도 있다.

Implementing gradient descent

for $t = 1,...,\#mini-batches$

compute forward prop on $X^{\{t\}}$

in each hidden layer, use batch norm to replace $Z^{[l]}$ to $\tilde{Z}^{[l]}$

use back prop to compute $dW^{[l]}$, $d\beta^{[l]} d\gamma^{[l]}$

update parameters

$W^{[l]} := W^{[l]} - \alpha\, dW^{[l]}$, $\beta^{[l]} := \beta^{[l]} - \alpha\, d\beta^{[l]}$, $\gamma^{[l]} := \gamma^{[l]} - \alpha\, d\gamma^{[l]}$

or Work with Momentum, RMSprop, or Adam optimizer, ...

Batch Norm at test time

Perform the needed normalizations, use μ and σ^2 estimated using an exponentially weighted average across mini-batches seen during training.

Batch Norm의 효과

1. Learning on shifting input distribution

"Covariance Shift": 만약 x에서 y로 가는 매핑을 배운 경우 x의 분포도가 변경되면 러닝 알고리즘을 다시 트레이닝시켜야 할 수도 있다.

Why this is a problem with neural networks?

$W^{[l-1]}$와 $b^{[l-1]}$가 변하면 $A^{[l-1]}$가 변하므로 layer l 입장에서 들어오는 값은 항상 변한다. 그러므로 Covariance Shift 문제의 영향을 받게 되는 것이다.

batch norm는 이렇게 분포가 왔다 갔다 이동하는 정도를 줄여준다.

batch norm가 보장하는 것은 어떻게 변하더라도 $X^{[l-1]}$의 mean과 variance는 똑같을 것이라는 점이다. 구체적으로는 $X^{[l-1]}$ 값이 변하더라도 mean과 variance가 변동없이 그대로 각각 0과 1로 유지할 것이다. mean = 0, variance = 1이 아닐 수 있지만 $\beta^{[l-1]}$와 $\gamma^{[l-1]}$에 따른 값은 유지될 것이다.

이전 층에서 파라미터가 업데이트되면서 변하는 것을 다음 층에 영향을 줄 수 있는 양을 제한시켜 준다. 이 변화량은 다음 층이 새로 배워야 하는 양이기도 하다. 그러므로 batch norm은 입력값이 변해서 생기는 문제를 줄여준다.

이전 layer의 파라미터가 배워야 하는 것들과 다음 layer의 파라미터가 배워야 하는 것을 디커플링해준다. 그래서 각 층이 스스로 러닝할 수 있게 해준다. 다른 층과 조금 더 독립적이 되어 결과적으로 전체 네트워크 속도를 올려준다.

배치 정규화의 의미는 특히 신경망에서 다음 layer의 시각에서 봤을 때 이전 layer의 이동폭은 별로 크지 않는다. 똑같은 mean과 variance를 필요하기 때문이다. 이것은 다른 층의 러닝 과정을 쉽게 해준다.

2. Batch Norm as regularization

- each min-batch is scaled by the mean/variance computed on just that mini-batch
- This adds some noise to the values $Z^{[l]}$ within that

mini-batch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect

dropout은 복수의 noise가 있는데(0과 1로 곱해지기 때문) batch norm도 복수의 noise가 있다. 표준편차에 의한 scaling 때문이기도 하고 mean을 빼면서 생기는 추가적 noise도 있다.(noisy한 mean과 variance)

noise의 양이 작아서 slight이고 만약 훨씬 큰 regularization을 원한다면 batch norm과 dropout을 같이 사용하면 된다.

미니 배치 사이즈가 클수록 noise가 줄고 결과적으로 regularization 효과가 줄어든다.

ng 교수의 경우 batch norm을 일반화로 사용하지는 않는다. batch norm의 본질적 의도가 아니기 때문이다. 따라서 batch norm을 단지 정규화의 방법으로 사용하길 권한다. 이것으로 hidden unit activation이 러닝을 빠르게 할 수 있게 말이다. 그리고 batch norm에게 regularization은 거의 의도하지 않은 부작용이라 생각한다.

Multi-class classification: Softmax regression

Softmax regression generalizes logistic regression to C classes.

$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$

activation function $A^{[L]} = g^{[L]}(Z^{[L]})$:

$t = e^{(Z^{[L]})}$

$A^{[L]} = \frac{e^{(Z^{[L]})}}{\sum_{i=1}^{n_L} t_i}$, $a_i^{[L]} = \frac{t_i}{\sum_{i=1}^{4} t_i}$

softmax의 특이한 부분은 activation 함수가 4x1벡터를 입력값으로 갖고 결과값도 같은 4x1벡터를 출력한다는 것이다.

activation 함수가 single row 입력값을 받는데
sigmoid나 ReLU는 실수를 입력값으로 갖고 실수를 결과값으로 갖는다.
softmax는 다른 결과값들에 정규화를 거쳐서 벡터를 입력값으로 갖고 결과값도 벡터로 갖는다.

Deep Learning Frameworks

Choosing deep learning frameworks

1. Ease of programming(development and deployment)
2. Running speed
3. Truly open(open source with good governance)