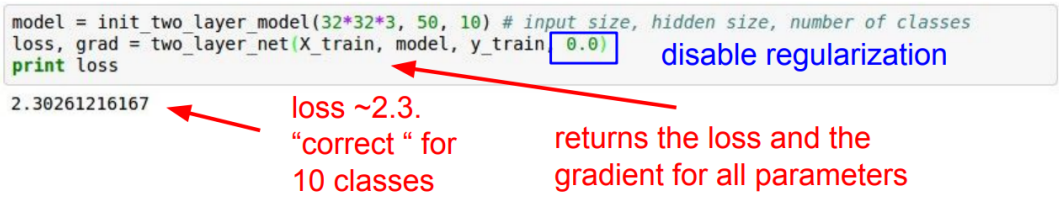
Lecture 6-5

Title	Training Neural Networks I
slide	http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

Babysitting the Learning Process

지금까지 네트워크 설계를 배웠다. 이제부터는 학습 과정을 어떻게 모니터링하고 하이퍼파라미터를 조절할 것인지 배워보자. 가장 첫 단계는 데이터 전처리이다. 앞서 배운 것처럼 zero-mean을 사용하여 데이터 전처리를 해준다. 두 번째 단계는 아키텍쳐를 선택하는 것이다. 예를 들어 하나의 Hidden Layer와 50개의 뉴런을 가진 모델이 있다고 해보자.



그 다음에 할 일은 네트워크를 초기화시키는 것이다. Forward pass를 하고 난 후에 Loss가 그럴듯해야 한다. 가령 softmax를 사용하고자 한다면, 가중치가 작은 값일 때 Softmax classifier의 Loss는 negative log likelihood가 되어서, 만약 10개의 클래스라면 Loss는 -log(1/10)이 될 것이다. 위의 결과를 보면 약 2.3 정도 되는 것을 확인 할 수 있다. 이를 통해 Loss가 원하는 대로 동작한다는 것을 알 수 있으며, 이 방법은 언제든 사용할 수 있는 유용한 방법(Good sanity check)이다. 위의 결과는 regularization을 0으로 설정한 것이다. 여기에 regularization을 추가하면 손실 함수에 regularization term이 추가되기 때문에 Loss가 증가한다. 이것 또한 유용한 sanity check이다.

이제 학습을 진행할 준비가 끝났다. 학습을 처음 시작할 때 좋은 방법은 데이터의 일부만 우선 학습시켜 보는 것이다. 데이터가 적으면 Overfit이 생길 것이고 Loss가 많이 줄어들 것이고, (이때는 regularization를 사용하지 않는다) Loss가 Epoch마다 내려 가는지 확인하면 된다. 데이터가 작은 경우라면 모델이 완벽하게 데이터를 overfit 할 수 있어야 한다. 여기까지의 sanity checks가 잘 끝났다면 이제부터는 정말로 학습을 시작할 차례이다.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

전체 데이터 셋을 사용하여 regularization을 약간만 주면서 적절한 learning rate를 찾아야 한다. Learning rate는 가장 중요한 하이퍼파라미터 중 하나로, 가장 먼저 정해야 하는 하이퍼파라미터이다. 만약 learning rate를 1e-6로 정한다면, learning rate가 지나치게 작아서 loss가 좀처럼 변하지 않는다. Learning rate가 지나치게 작으면 gradient 업데이트가 충분히 일어나지 않게 되고, cost가 변하지 않게 된다. 여기에서 유심히 살펴봐야 할 점은 loss가 잘 변하지 않음에도 training/validation accuracy가 20%까지 급 상승한다는 것이다. 현재 확률 값들이 아직 멀리 퍼져있고, 때문에 loss는 여전히 비슷하지만 "학습"을 하고 있기 때문에 확률이 조금씩 "옳은" 방향으로 바뀌고 있기 때문이다. 즉, 가중치는 서서히 변하고 있지만 Accuracy는 그저 가장 큰 값만 취하기 때문에 수치가 갑자기 뛸 수 있게 되는 것이다.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

```
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

이제 learning rate를 더 큰 값인 1e6으로 바꿔보면, cost가 NaNs으로 발산(exploded)한 것을 볼 수 있다. 이것은 learning rate가 지나치게 높기 때문이다. 따라서 learning rate는 보통 1e-3 에서 1e-5 사이의 값을 사용하여 cross-validation을 수행해주어야 한다.

Hyperparameter Optimization

하이퍼 파라미터를 최적화시키고, 그 중 가장 좋은 것을 선택하는 방법 중 한가지는 cross-validation이다. Cross-validation은 Training set으로 학습시키고 Validation set으로 평가하는 방법으로, hyperparameter가 잘 동작하는지를 확인할 수 있다.

Cross-validation strategy

coarse -> fine cross-validation in stages

First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
… (repeat as necessary)

hyperparameter를 좀 더 단계적으로 찾아보자. 우선 coarse stage에서는 넓은 범위에서 값을 골라내어, 몇 번의 Epoch만으로도 현재 값이 잘 동작하는지 알 수 있다. Nan이 나타나거나 혹은 Loss가 줄지 않는 것을 보면서, 이에 따라 값을 적절히 조절하면 된다. coarse stage가 끝난 후에, 하이퍼파라미터가 어떤 범위에서 잘 동작할 것인지를 대충 알게 된다.

그 다음 fine stage에서는 좀 더 좁은 범위를 설정하고 학습을 좀 더 길게 시켜보면서 최적의 값을 찾는다. NaNs로 발산하는 징조를 미리 감지할 수도 있다. Train 동안에 Cost가 어떻게 변하는 지를 살펴보면서, 이전의 cost보다 몇 배 더 커진다면 잘못 하고 있다는 것을 알 수 있다. 또한 Cost가 엄청 크고 빠르게 오르고 있다면 loop를 멈춰버리고 다른 하이퍼파라미터를 선택해야 한다.

For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                    model, two_layer_net,
                                    num_epochs=5, reg=reg,
                                    update='momentum', learning_rate_decay=0.9,
                                    sample_batches = True, batch_size = 100,
                                    learning rate=lr, verbose=False)
```

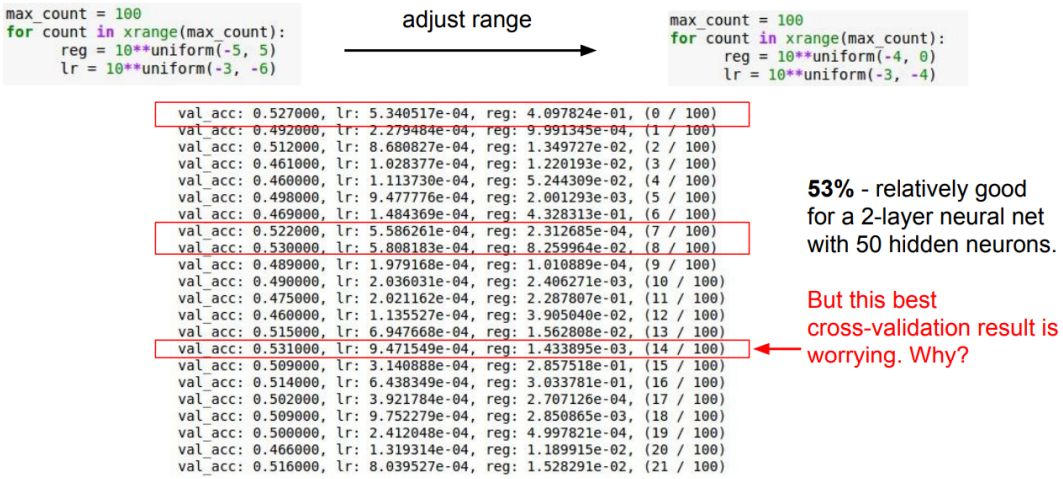
note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

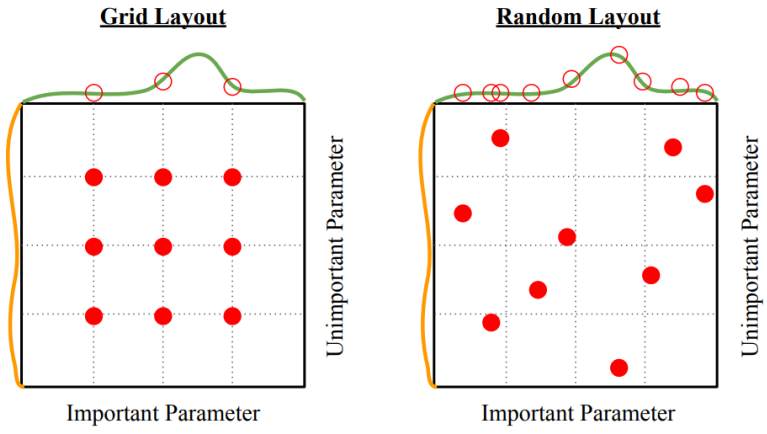
nice

위의 예시는 5 epochs을 돌며 coarse search를 하는 과정이다. 네트워크 구조는 앞서 만든 것과 유사하다. 여기에서 확인 해야 할 것은 validation accuracy이다. 높은 val_acc에는 빨

간색으로 표시되어 있어서, 빨간 색으로 표시해 둔 곳이 fine-stage를 시작할 만한 범위가 된다. 한 가지 주의 해야 할 점은 learning rate는 gradient와 곱해지기 때문에 하이퍼파라미터 최적화 시에는 Log scale로 값을 주는 것이 좋다는 것이다. 파라미터 값을 샘플링할때 10^-3 ~ 10^-6 을 샘플링하는 것이 아니라, -3 ~ -6으로 10의 차수 값만 샘플링하는 것이 좋다.



오른쪽 reg의 경우 범위를 10^-4 에서 10^0 정도로 좁히면 좋을 것 같다. 이 범위에서 학습이 가장 잘 되는 구간으로, 53%의 val_acc를 보이는 것을 확인할 수 있다. 그러나 한가지 문제가 있는데, 화살표를 잘 보면 good learning rates가 전부 10E-4 사이에 존재하고 있다는 것이다. Learning rate의 최적 값들이 우리가 좁혀서 설정한 범위의 경계 부분에 집중되어 있다는 것을 알 수 있다. 이렇게 되면 최적의 Learning rate를 효율적으로 탐색할 수 없을 수도 있기 때문에 좋지 않다. 가령 실제로 최적의 값이 1E-5나 1E-6 근처에 존재할 수도 있기 때문이다. 따라서 최적의 값이 내가 정한 범위의 중앙 쯤에 위치하도록 범위를 잘 설정해 주는 것이 중요하다.

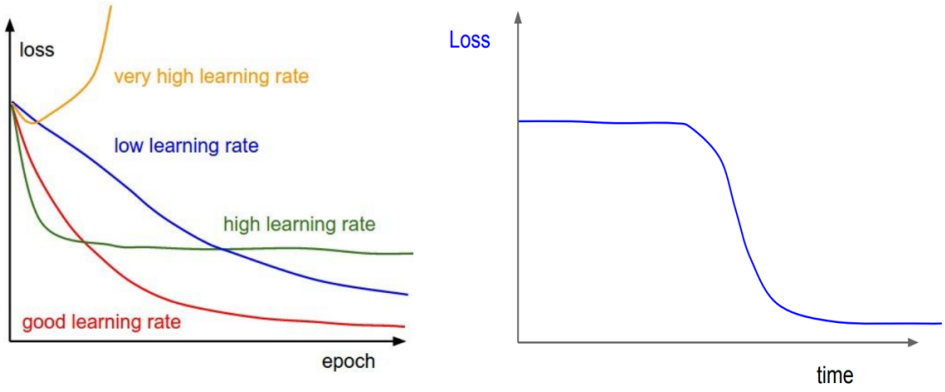


하이퍼파라미터를 찾는 또 다른 방법은 grid search를 이용하여, 고정된 값과 간격으로 샘플링하는 것이다. 그러나 실제로는 grid search보다 기존의 random search를 하는 것이 더

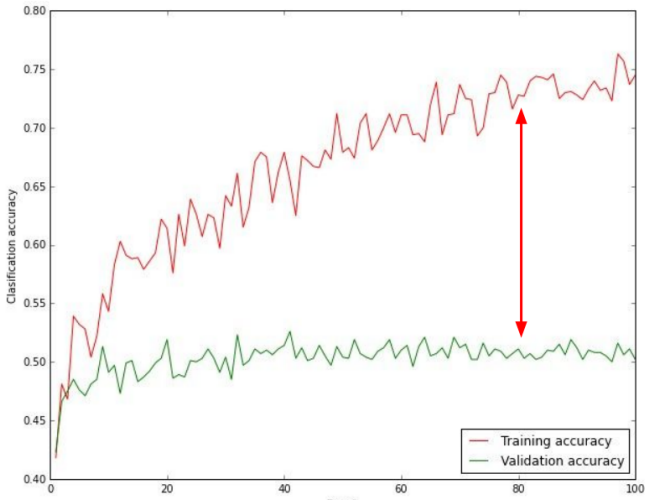
좋다. 만약 내 모델이 어떤 특정 파라미터의 변화에 더 민감하게 반응을 하고 있다고 생각해 보면(여기에서는 노란색보다 초록색에 더 민감하게 반응), Random search는 중요한 파라미터(초록색)에게 더 많은 샘플링이 가능하므로 Random search이 더 좋다. Random search를 사용하면 important variable 에서 더 다양한 값을 샘플링 할 수 있지만, Grid Layout에서는 단지 세 번의 샘플링 밖에 할 수 없으므로 Good region이 어디인지 제대로 찾을 수 없다.

Loss curve

하이퍼파라미터는 우리가 다룬 Learning rate와 그 밖에 decay schedule, update type,regularization, network architecture, hidden unit, depth의 수 등 다양하다. 우리는 이 모든 하이퍼파라미터를 전부 다 최적화 시킬 수 있다. 실제로 하이퍼파라미터 최적화와 Cross-validation 정말 많이 하면서, 많은 하이퍼파라미터를 직접 돌려보고 모니터하면서 값을 계속해서 확인해야 한다. 이는 loss curve를 보면서 좋은 것을 찾아서 시도해보아야 한다. loss curve는 learning rate이 정말 중요하다.



loss curve를 보면 어떤 learning rate가 좋고 안 좋은지 알 수 있다. Loss가 발산하면 learning rate가 높은 것이고 너무 평평(linear)하면 너무 낮은 것이다. 또한 가파르게 내려가다가 어느 순간 정체기가 생기면 여전히 너무 높다는 의미이고, learning step이 너무 크게 점프해서 적절한 local optimum에 도달하지 못하는 경우이다. 최적의 learning rate는 왼쪽의 파란색 그래프처럼 비교적 가파르게 내려가면서도 지속적으로 잘 내려가는 것이다. 만약 오른쪽과 같이 loss가 평평하다가 갑자기 가파르게 내려간다면 이는 초기화의 문제일 수 있다. gradient의 backpropb가 초기에는 잘 되지 않다가 학습이 진행되면서 회복이 되는 경우이다.



big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

accuracy를 모니터링 하다가 train_acc와 val_acc가 큰 차이를 보일 때도 있는데, 이는 overfit가 생긴 것일 수도 있다. 이때는 regularization의 강도를 높여야한다. 만약 두 수치간의 gap이 없다면 아직 overfit하지 않은 것이고, capacity을 높일 수 있는 충분한 여유가 있다는 것을 의미한다.

또한 가중치의 크기 대비 가중치 업데이트 비율을 지켜볼 필요가 있다. 파라미터의 norm을 구해서 가중치의 규모와 업데이트 사이즈를 구하고, 얼마나 크게 업데이트 되는지를 알 수 있다. 이 비율이 대략 0.001 정도 되길 원한다. 이 값은 변동이 커서 정확하지 않을 수 있지만 업데이트가 지나치게 크거나 작은지에 대한 감을 가질 수는 있다. 너무 업데이트가 지나치거나 아무 업데이트도 없으면 안돼서, 문제가 뭔지 디버깅할 때 유용하다.

Summary

Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)

이번 강의에서는 활성 함수와 데이터 전처리, 그리고 가중치 초기화와 Batch Norm, 학습 과정 준비하기, 하이퍼 파라미터 최적화를 배웠다. 요약해보자면 ReLU를 사용하고(활성함수), 평균을 빼고(데이터 전처리), Xavier Initialization을 쓰고(가중치 초기화) Batch Norm을 사용하고 Random Search를 통해 하이퍼파라미터를 찾는다는 것이다.