수합본

Introduction

Pure Learning Algorithm

: a function taking training data as input and producing
as output a function

- 하지만, 실전 Learning Algorithm에는 Hyper-parameters가 포함된다.

Hyper-Parameter

: 학습 알고리즘 외부에서 설정하는 변수로, 알고리즘의 학습 과정 및 성능에 영향을 미치는 요소.

Hyper-Parameter Selection

⇒ 하이퍼 파라미터 선정은 알고리즘의 성능과 결과에 큰 영향을 미친다.

⇒ 하이퍼 파라미터는 학습 알고리즘을 적용하기 전에 설정한다.

⇒ model selection과 유사한 기능과 의미를 가진다.

- 사람이 직접 정하거나 알고리즘을 활용해 조정할 수 있지만, value 자체는 명확히 선정되어야 한다.
- 일반적으로, *validation set* performance/error와 같은 out-of-sample data에 기반하여 선정된다.
    - 사용된 out-of-sample data는 재사용하지 않고 generalization을 평가하기 위해 별도의 *test set*을 활용한다.

딥러닝에서는 하이퍼파라미터의 수가 많아 조정이 어려울 수 있다. 따라서 효과적인 하이퍼파라미터 최적화는 모델 성능을 극대화하는 데 필수적이다. 이 논문은 하이퍼파라미터의 정의, 중요성 및 최적화 방법에 대해 논의하고 있다.

3.1 Neural Network Hyper-Parameters

Different learning algorithms → Different sets of hyper-parameters

3.1.1 Hyper-Parameters of the Approximate Optimization

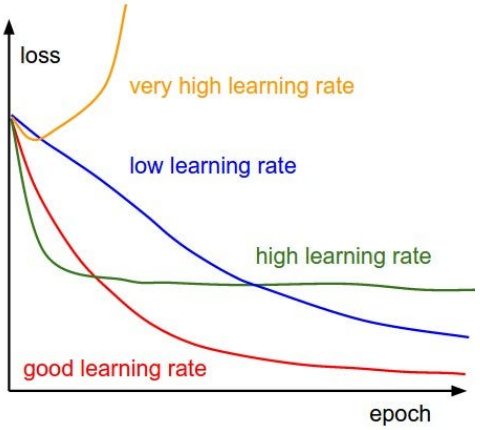
- **Learning algorithms** → training criterion & model + optimizing this criterion
- *optimizer related hyper-parameters*와 *model itself related hyper-parameters*로 구분 가능하다.

- Optimizer related hyper-parameters

  : initial learning rate, learning rate schedule, mini-batch size, momentum 등
- Model itself related hyper-parameter

  : 일반적으로 함수 클래스, 정규화자 및 손실 함수와 관련된 하이퍼파라미터

1. **Initial learning rate**



   - 학습 과정에서 Loss를 최소화하는 데 중요한 역할을 하며, 적절히 tuning해야 한다.
   - 표준 다층 신경망의 경우 0.01이 default value로 사용될 수 있지만, 상황에 따라 tuning이 필요하다.

2. **Learning rate schedule**

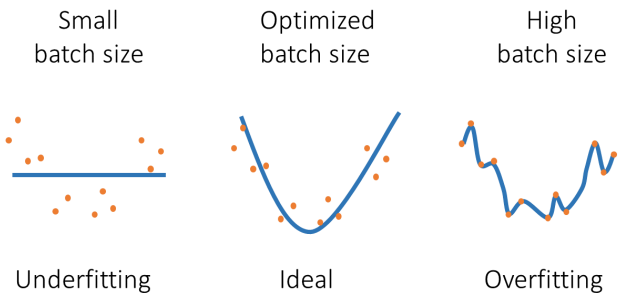
   : 학습률을 일정하게 유지하거나 점진적으로 감소시키는 전략

$$\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$$

  - O(1/t) learning rate schedule → keep lr constant for first few steps and decrease in O(1/t)
  - training criterion stops decreasing significantly 할 때부터 (threshold 활용) lr 감소시킨다.
- O(1/t) 학습률 스케줄이 일반적으로 사용된다.
- 학습 과정의 효율성을 높인다.

3. **Mini-batch size**

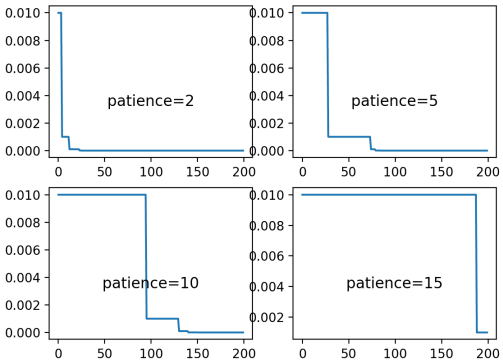
   : 각 업데이트에서 사용하는 데이터 샘플의 크기



- 일반적으로 1에서 몇 백 사이의 값을 사용한다. → 계산 효율성에 영향.
- 32가 좋은 default value이다.
- training time에 영향을 미치지만 test perfomance에는 큰 영향을 미치지 않는다.

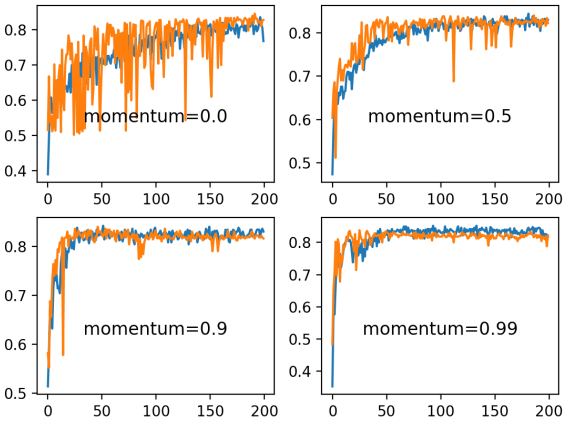
  → 다른 하이퍼파라미터들을 선택한 후에 별도로 최적화 가능
- 배치 사이즈를 선택하면, 고정할 수 있으며 다른 하이퍼파라미터들을 추가로 최적화할 수 있다.

4. **# of training iterations T**



- *Early Stopping*을 사용하여 최적의 반복 횟수를 설정한다.
  - Early Stopping → overfitting을 방지할 수 있는 가장 효율적인 방법이다.
  - 하이퍼파라미터의 overfitting effect를 숨긴다.
    - 따라서, 개별 하이퍼파라미터의 효과를 분석할 때는 비활성화해야 한다.
- 실질적으로는, T 이후에도 학습을 계속하여 validation error가 더 낮아질 가능성이 없는지 확인해야 한다.
- Overfitting을 방지하고 모델의 Generalization performance를 향상시킴.
- 학습이 진행되면서 새로운 최소 T가 관찰되면, patience (선택된 T를 최종 답으로 결정하기 전에 확인해야 할 최소 학습 예제 수) 파라미터가 덧셈 또는 곱셈으로 증가한다.
- ***Patience***
  - 학습이 진행되면서 validation error의 새로운 minimum # (T^)이 관찰되면, 현재 최적 모델을 저장하고 T^를 update한다.
  - 동시에, patience 파라미터를 증가시켜 학습 중단 조건을 조정한다.
  - patience 파라미터는 덧셈 또는 곱셈 방식으로 증가한다.

5. **Momentum**



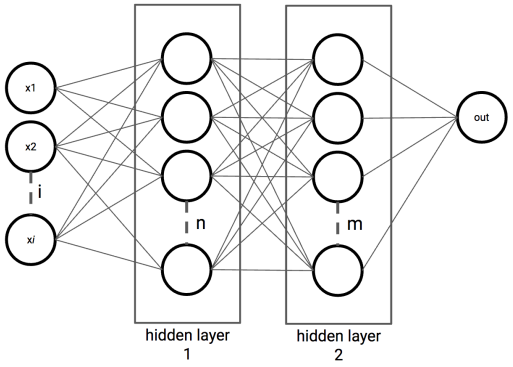
- gradient의 이동 평균을 계산하여 학습 과정의 진동을 줄이고 convergence 속도를 높인다.

6. **Layer-specific optimization hyper-parameters (Sec. 3.3.4.)**
   - 자주 사용되지는 않음
   - MLP(다층 퍼셉트론)의 각 layer에서 최적화 하이퍼파라미터 값을 다르게 설정할 수 있다.
   - 특히 각 layer의 unit 수의 편차가 클 때 유용하다.

3.1에서는 SGD(확률적 경사 하강법) 설정에서의 신경망에서 활용되는 하이퍼파라미터에 대해서만 다루었다. 다른 최적화 알고리즘에는 다른 하이퍼파라미터가 있을 수 있다.

3.2 Hyper-Parameters of the Model and Training Criterion

1. **Number of hidden units**



- MLP 내 각 layer의 크기는 자유롭게 설정이 가능하고 이를 통해 capacity를 통제한다.
- Early stopping이나 다른 regularizers (예시. weight decay)때문에 충분히 큰 크기를 설정하는게 중요하다.
    - optimal보다 크기가 커져도 generalization performance에 큰 영향이 가지 않기 때문

- hidden layer마다 hidden units 단위수 다르게 하는 옵션도 존재한다.
    - **하지만,** 모든 layer에 같은 단위를 설정한게 더 나은 결과를 보였다.
    - overcomplete(input의 units보다 큰 단위수를 설정)한 첫번째 hidden layer 또한 좋은 결과를 보였다.
- Supervised neural networks에서 unsupervised pre-trainng를 사용한 모델에서는 hidden units의 optimal values가 매우 컸다.
    - → unsupervised pre-training이후에 많은 수의 hidden units은 irrelevant한 정보를 가지고 있고 여기에서 relevant한 정보를 capture하기 위해서는 larger hidden layers가 필요하기 때문.

2. **Weight decay**

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

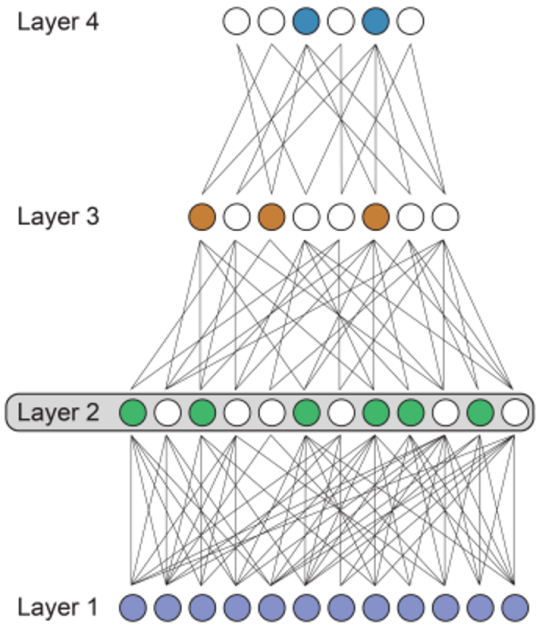
$$\text{Cost} = \sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term

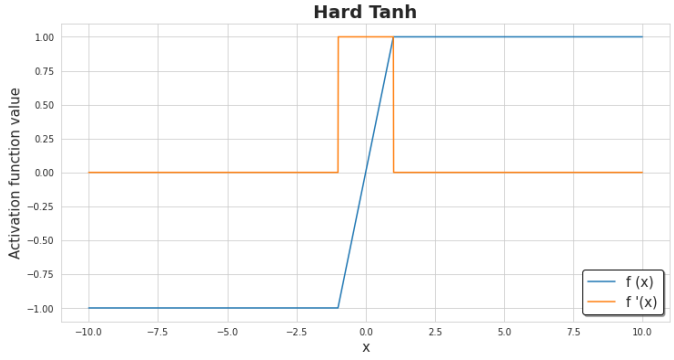
- regularization term을 training criterion (loss function) 에 더함으로써 오버피팅을 억제한다
- regularization term의 종류로는 크게 L2 (제곱 방식), L1 (절대값 방식), Both가 있다.
- L2는 이전까지의 가중치에 제곱을 수행하는 방식이기에 큰 수에 더 큰 페널티를 준다.
    - early stopping과 기본적으로 같은 role 하지만 early stopping이 더 효율적이기 때문에 early stopping이 쓰이면 보통 L2를 drop한다.
- L1는 L2와 약간 다르게 행동한다.
    - feature selection에 주로 쓰인다.
    - 유용하지 않은 parameters는 0으로 만든다.
    - input weights를 더 깨끗하고 쉽게 해석할 수 있도록 한다.

3. **Sparsity of activation**

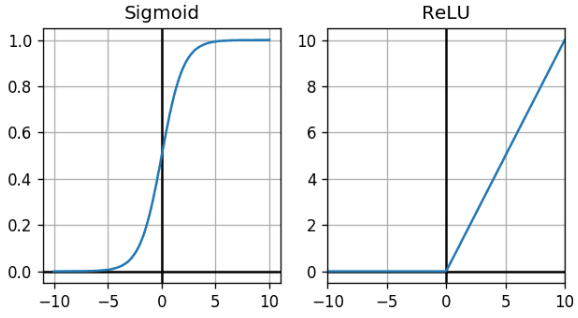


- Penalty term을 training criterion (loss function) 에 더함으로써 hidden units이 sparse한 형태를 띄도록 유도한다.
    - sparse representation이 유리한 이유 → representation의 내재 근본 요인들을 disentangle 되도록 만듦.
    - 또한, learner가 학습할 수 있는 examples의 수를 줄인다는 의미에서 regularize 하는 방법이기도 함.

- Sparse representation (activations이 0에 가까운값을 가지는 representation) 를 유도하기 위해 제안된 여러 접근 방식들이 존재한다.
  - (1) 단순히 L1 norm을 penalty로 적용하는 방식
    - sigmoid같이 결과값이 0이면 통하지만 tanh같이 결과값이 0이 아닌 -1이나 1에 집중되어있으면 통하지 않음.
  - (2) hidden units 내 biases에 penalty를 가하는 방식
    - penalty를 가함으로써 biases를 negative로 만듦.
    - 하지만 weights가 biases를 보상하기 때문에 위험함!
  - (3) activation function을 선택하는 방식
    - regularization penalty 자체보다 activation function의 선택도 sparsity에 강력한 임팩트를 가질수가 있다.
    - ReLU (Sigmoid대신) 나 hard tanh가 매우 성공적인 결과를 보임



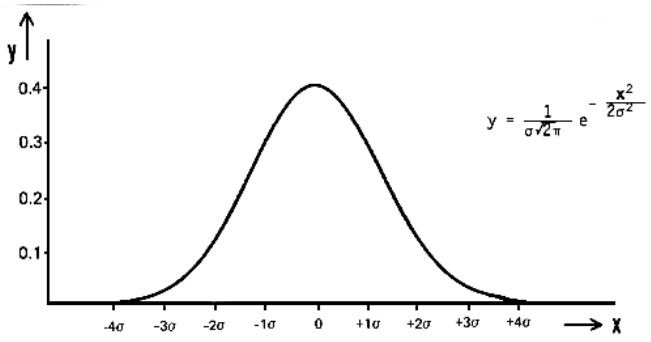
4. **Neuron non-linearity**



- Output layer의 activation function에 ReLU같은 hard neuron non-linearities를 사용할수 없다
    - 이유 : unit이 saturated (ReLU에서는 0 이하) 되고 loss와 합쳐지면 network내부에 기울기가 저전파되지 않는다.
- → 따라서, output units에 sigmoidal non-linearity를 갖는 것이 도움이 된다.

5. **Weights initialization scaling coefficient**

- bias는 일반적으로 0으로 설정해도 상관없지만 weights는 조심히 initialized되야한다.
    - 이유 : 같은 layer내 hidden units의 symmetry를 없애기 위해서
- 평균 0 및 0.1이나 0.01같은 작은 값의 표준편차의 Gaussian initialization도 좋은 결과를 보인다.



6. **Random seeds**

- Random seed의 선택은 결과에 거의 영향을 안 끼친다.
    - → 대부분의 경우에서 무시 가능.
- Hyper-parameters의 best set에서 seed만 바꾸면 performance가 아주 조금 더 좋아질수 있다.

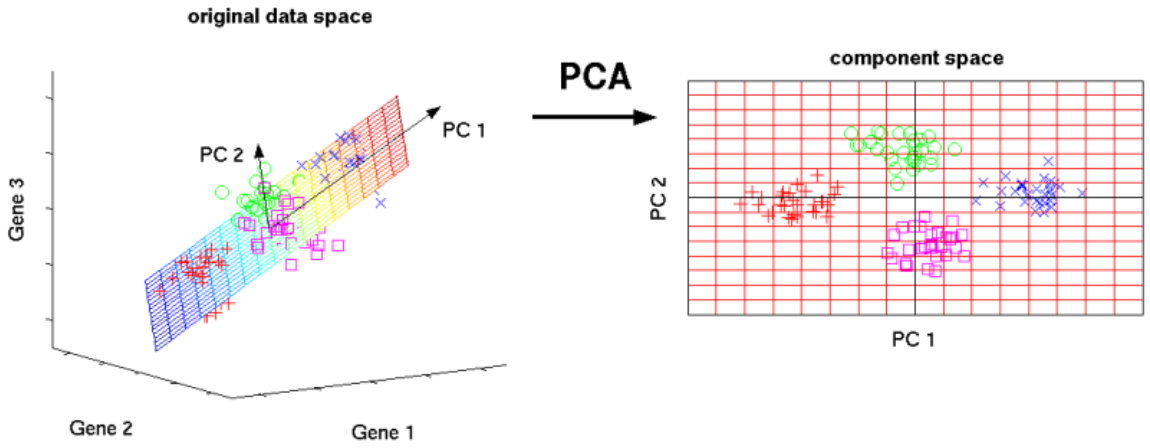
7. **Preprocessing**

- raw data를 적절한 inputs으로 변환하기 위한 수많은 preprocessing 단계들이 제안되어 왔다.
    - Element-wise standardization

        - 평균값을 빼고 표준편차로 나누는 방식

$$z = \frac{x_i - \mu}{\sigma}$$

    - Principal Components Analysis (PCA)
        - 차원축소



    - 각 피처간의 uniformization
        - 편리한 non-linear preprocessing
        - 피처값들을 0에서 1사이로 제한

3.2에서는 model과 training criterion 관련 hyper-parameters 중 일반적인(generic) 옵션들에 관해서만 다루었다. 다른 architectures이나 learning algorithms에 따라서 더 많은 옵션들이 생기기도 한다.

3.3 Manual Search and Grid Search

1. **General guidance for the exploration of hyper-parameters**

- Hyper parameter 종류
    - numerical hyper parameter: 은닉유닛 수, 훈련 반복 횟수 등
    - discrete hyper parameter: 활성화 함수, 최적화 알고리즘 등

- hyper parameter 선택
  - 최적화 문제: 검증 오류가 낮은 구성 찾기
  - 일반화 문제: 검증 성능 최적화 후 exptected generalization에 대한 불확실성

  hyper parameter와 validation error의 관계는 복잡함
  단일 하이퍼파라미터 값에 따른 성능 변화 고려 시 U자형 곡선 기대할 수 있으나 노이즈 포함될 수 있음

Best value on the border

- validation error 고려 시 최적 값이 조사된 구간의 경계 근처에 있는지 주의해야 함
- 경계 근처에 있다면, 그 경계를 넘어 더 나은 값 찾을 수 있기 때문에 더 탐색하는 것이 좋음

Scale of values considered

- hyper-hyper-parameter: numerical hyper-parameter 탐색은 검색할 시작 구간을 선택하는 것을 포함하기 때문
- 충분히 큰 구간 선택 권장

  선형 간격 vs. 로그 도메인 간격

  - 선형 간격
    - hyper-parameter 값을 일정한 간격으로 선택
    - 0.001, 0.002, 0.003 과 같이 선택한다면 absolute difference가 동일
  - 로그 도메인 간격
    - hyper-parameter 값을 로그 스케일에서 일정한 간격으로 선택
    - 0.001, 0.01, 0.1과 같이 선택한다면 값 사이의 ratio가 일정

  ⇒ 로그 도메인이 더 합리적인 이유: 모델이 작은 값일 때와 큰 값일 때 모두 일정한 비율로 변화 → 더 세밀히 탐색 가능

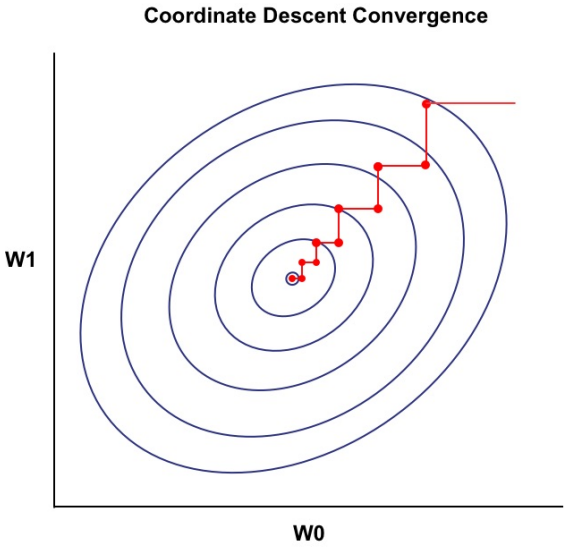
Computational considerations

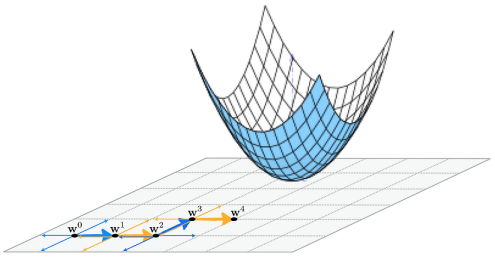
- use computationally cheap estimators of validation error

- convolutional networks의 architecture hyper-parameter 선택 시 하위 레이어에 random weights 사용
  → nosiy and biased estimator of validation error
  → but correlated with expensive validation error
- 고속 컴퓨팅 (ex. 병렬 컴퓨팅)

2. Coordinate Descent and Multi Resolution Search

- **Coordinate Descent**





- 수동 검색 or 단일 컴퓨터 사용 시 한 번에 하나의 hyper-parameter만 변경하며, hyper-parameter의 best 구성에서 변경 시도
- sensitive variable (ex. learning rate) fine-tune

  cf. standard coordinate descent: 모**든 변수가 최적화 순환하며 최적화**
- 미분 후 오차가 줄어드는 방향으로 레버리지를 움직임

    - 참고 자료

- **Multi Resolution Search**
    - 좋은 설정 발견 전에는 fine changes 탐색할 필요 없음

      → 초기에 몇 가지 값만 고려하거나 새로운 값 시도 시 큰 변화 고려하기

      → 최적의 구성 발견 후 locally explore

3. **Automated and Semi-automated Grid Search**

Grid Search

- 각 hyer-parameter에 대해 선택된 값의 간격이나 집합 결정될 경우 병렬 컴퓨팅 하는 방법
- numerical intervals → list로 변환 후 이 값들의 모든 조합 통한 search
- 장점 - fully parallelizable
- 단점 - 실패 시 처음부터 다시 → multiplying the overall computing time

Sequence of grid searches

- single grid search는 충분치 않음
- 이전 결과에 기반하여 ranges of values 조정
- multi-resolution search 이용하여 자동화 가능

With Coordinate descent

- grid search가 소수의 hyper-parameter에만 focus
  ex. 초기 학습률 탐색하며 학습률 descent schedule 고정. 이후 스케줄 형태 선택되면 세밀하게 학습률 조정

4. **Layer-wise optimization of hyper parameters**

- Coordinate Descent + cheap relative validation set performance evaluation

- input 근처의 하위 레이어와 관련된 hyper-parameter 먼저 탐욕적으로 선택한 후, 상위 레이어 훈련

- 하위 레이어 초기 train
  1. 첫 번째 레이어를 다양한 hyper-parameter 값으로 비지도 학습
  2. 상대적인 검증 오류 추정
     a. 최종 네트워크에 첫 번째 레이어, 단일 레이어만 내부 표현으로 사용된다고 가정하고 각 hyper-parameter 설정에서 이게 얼마나 좋은 표현을 학습하는지 평가
     b. 각 hyper-parameter 설정에 대해 validation set에서의 성능 평가
  - 최종 task가 지도학습인 경우 간단한 예측기로 훈련
  - linear predictor(ex. regression / logistic regression)의 경우 비지도 학습이 진행되는 동안에도 수행 가능. → early stopping에도 가능

- greedy evaluation에 따라 찾은 최적의 값을 시작점으로 사용하여 다음 레이어를 동일하게 훈련하고 hyper-optimize
  - 현재까지의 최적의 구성만 유지
  - 각 레이어마다 K개의 최적 구성을 유지하여 탐색을 진행
    ex. layer 1,2의 모든 hyper-parameter 구성 중 최적의 K개 구성을 다음 레이어 탐색의 시작점으로 사용

- 장점 - efficient computationally : 상위 레이어 pre-training 시 하위 레이어 수정 x

Algorithm 1 : Greedy layer-wise hyper-parameter optimization.

input K: number of best configurations to keep at each level.
input $NLEVELS$: number of levels of the deep architecture
input $LEVELSETTINGS$: list of hyper-parameter settings to be considered for unsupervised pre-training of a level
input $SFTSETTINGS$: list of hyper-parameter settings to be considered for supervised fine-tuning

Initialize set of best configurations $S = \emptyset$
for $L = 1$ **to** $NLEVELS$ **do**
  for C **in** $LEVELSETTINGS$ **do**
    for H **in** (S or $\{\emptyset\}$) **do**
      * Pretrain level L using hyper-parameter setting C for level L and the parameters obtained with setting H for lower levels.
      * Evaluate target task performance $\mathcal{L}$ using this depth-L pre-trained architecture (e.g. train a linear classifier on top of these layers and estimate validation error).
      * Push the pair $(C \cup H, \mathcal{L})$ into S if it is among the K best performing of S.
    end for
  end for
end for
for C **in** $SFTSETTINGS$ **do**
  for H **in** S **do**
    * Supervised fine-tuning of the pre-trained architecture associated with H, using supervised fine-tuning hyper-parameter setting C.
    * Evaluate target task performance $\mathcal{L}$ of this fine-tuned predictor (e.g. validation error).
    * Push the pair $(C \cup H, \mathcal{L})$ into S if it is among the K best performing of S.
  end for
end for
output S the set of K best-performing models with their settings and validation performance.

<정리>

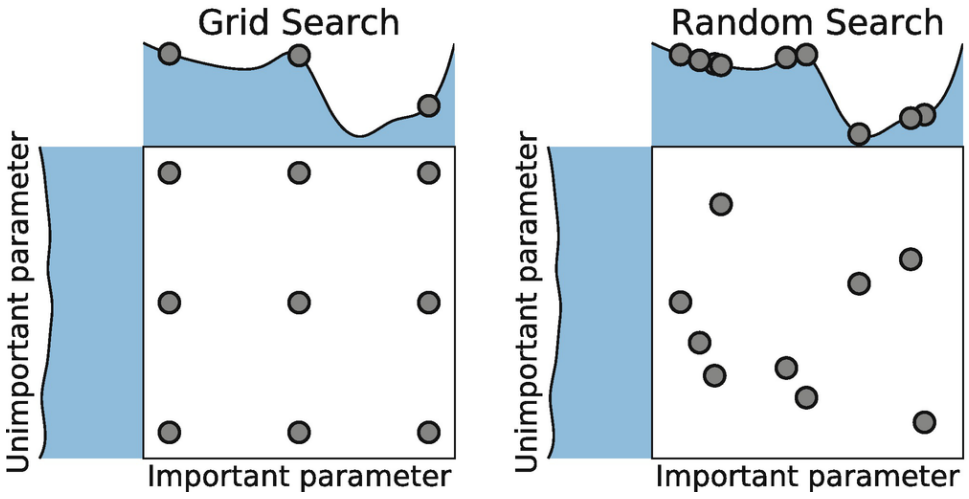
비지도 사전 학습: 관련있는 hyper-parameter 설정

지도 학습 fine tuning: 추가적인 hyper-parameter 선택

최종 지도 학습 fine tuning: label 많을 때 특히 권장됨

3.4 Random Sampling of Hyper-Parameters

- hyper-parameter 수가 많아질수록 Grid Search 비효율적
  → 정규 grid를 random sampling으로 대체: 테스트할 hyper-parameter 구성은 각각 사전 분포에서 독립적으로 샘플링
- discrete hyper-parameters의 경우, 다항분포 사용 가능
  → 최악의 경우는 uniform distribution
- 랜덤 샘플링의 효율성
  - hyper-parameter 수가 2,3개 넘어서면 grid search보다 효율적임.
  - grid search에서는 동일한 hyper-parameter 값이 다른 hyper-parmaeter 구성에서 반복되지만 랜덤 샘플링은 각 hyper-parameter가 더 많은 값을 탐색할 수 있게 해줌
- 영향력이 큰 hyper-parameter
  - 데이터셋, 아키텍처에 대해 중요한 hyper-parameter는 싱아힘
  - 그러나 몇몇 영향력이 큰 hyper-parameter가 있음 ex. learning rate
- random search
  - validation performance 시각화 시 다른 hyper-parameter의 variation 때문에 더 노이즈한 그림 얻지만, 고해상도
  - validation error 시각화 → 최적화가 수렴하는지 보여줌



- 처음 N번 시도에서의 최고 성능 곡선 vs. N 크기의 하위 집합에서의 최고 성능 평균화 곡선

  → 후자가 더 좋은 통계적 대표성을 가짐(여러 번의 시도에서 얻은 데이터를 기반으로 하므로)

  → 시도 횟수 증가의 필요성 암시(전자가 멈춰 보일 때도 후자가 증가하고 있다면 더 많은 시도가 필요할 것)

- random search 장점(빠른 수렴) + gird search 장점(질적 분석이 용이함)

  ⇒ random search로 최적의 솔루션 찾은 후, 그 주변에서 작게 grid search 수행하는 것을 제안

- random search 장점 정리
  - 병렬화 용이(=grid search)
  - 작업 실패하더라도 다시 실행할 필요 X (≠grid search)
  - 초기 작업 후 추가 작업 쉬움(≠ grid search)

  ⇒ random search 유용하지만 반자동 탐색 필요하다!