CSC321 Lecture 9: Generalization

Roger Grosse

테크닉의 장 : 일반화

Overview

Lec 8 : 있는 데이터로 얼마나 Loss function 최소점을 잘 찾아갈 것인가?

- We've focused so far on how to *optimize* neural nets — how to get them to make good predictions on the training set.
- How do we make sure they generalize to data they haven't seen before?
- Even though the topic is well studied, it's still poorly understood.

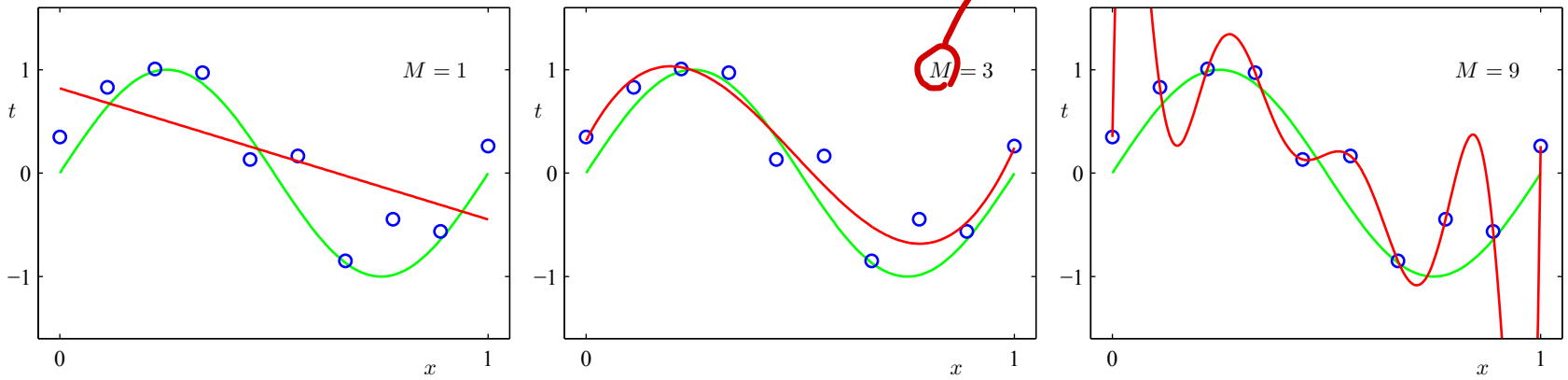
Lec 9 : "optimizing"을 잘하면 처음보는 데이터 (학습에 쓰이지 않은) 에 대해서도 성능이 좋은가?

⇒ X, "일반화"는 또다른 얘기

Generalization

Recall: overfitting and underfitting

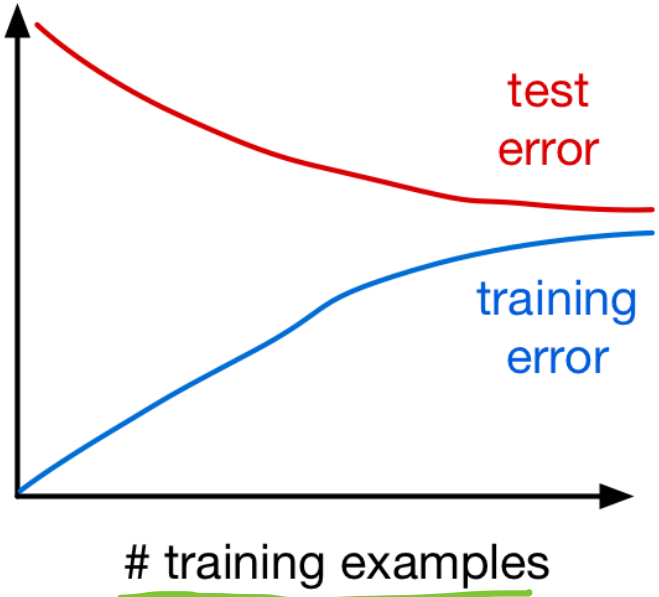
degree



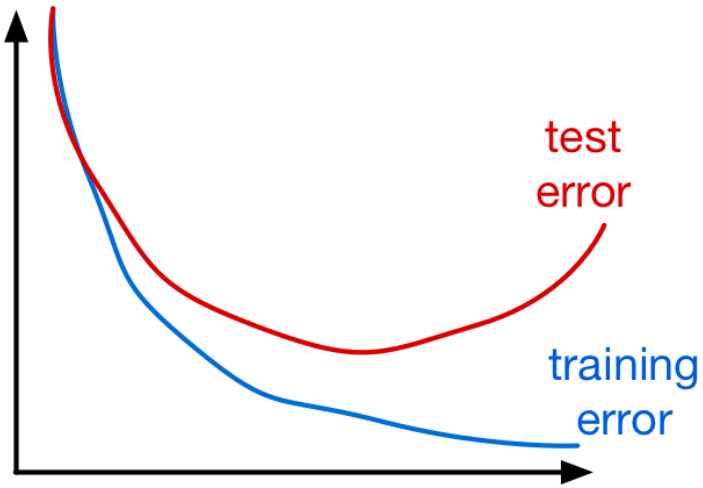
We'd like to minimize the generalization error, i.e. error on novel examples.

Generalization

- Training and test error as a function of data and model capacity.
- Note: capacity isn't a formal term, or a quantity we can measure.



샘플수 ↑ = 과적합 확률 ↓
(underfitting 은 해결 x)



차원, epoch 수 와 같은
"수치"로 표현되지 않는 용어

Bias/Variance Decomposition ⇒ over/underfitting 에 대한 다른 표현 (from MSE)

- There's an interesting decomposition of generalization error in the particular case of squared error loss.
- It's often convenient to suppose our training and test data are sampled from a data generating distribution $p_{\mathcal{D}}(\mathbf{x}, t)$.
- Suppose we're given an input $\mathbf{x}$. We'd like to minimize the expected loss:

MSE 손실은 다음과 같이 표현 가능 : $\underline{\mathbb{E}_{p_{\mathcal{D}}}[(y-t)^2 \,|\, \mathbf{x}]}$

- The best possible prediction we can make is the conditional expectation

$$y_\star = \mathbb{E}_{p_{\mathcal{D}}}[t \,|\, \mathbf{x}].$$

- Proof:

다음 페이지

$$\begin{aligned}
\mathbb{E}[(y-t)^2 \,|\, \mathbf{x}] &= \mathbb{E}[y^2 - 2yt + t^2 \,|\, \mathbf{x}] \\
&= y^2 - 2y\mathbb{E}[t \,|\, \mathbf{x}] + \mathbb{E}[t^2 \,|\, \mathbf{x}] \\
&= y^2 - 2y\mathbb{E}[t \,|\, \mathbf{x}] + \mathbb{E}[t \,|\, \mathbf{x}]^2 + \mathrm{Var}[t \,|\, \mathbf{x}] \\
&= (y - y_\star)^2 + \mathrm{Var}[t \,|\, \mathbf{x}]
\end{aligned}$$

- The term $\mathrm{Var}[t \,|\, \mathbf{x}]$, called the Bayes error, is the best risk we can hope to achieve.

$$\mathbb{E}[(y-t)^2 \mid \mathbf{x}] = \mathbb{E}[y^2 - 2yt + t^2 \mid \mathbf{x}]$$

① $$= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t^2 \mid \mathbf{x}]$$

$$= y^2 - 2y\mathbb{E}[t \mid \mathbf{x}] + \mathbb{E}[t \mid \mathbf{x}]^2 + \mathrm{Var}[t \mid \mathbf{x}]$$

② $$= (y - y_\star)^2 + \mathrm{Var}[t \mid \mathbf{x}]$$

① 확률 분야 에서 평균, 분산 표기 및 응용

x의 평균 : $E(x)$

x의 분산 : $E((x-m)^2)$, $m = E(x) =$ 평균

↳ 전개 해보면,

$$E((x-m)^2) = E(x^2 - 2xm + m^2)$$
$$= E(x^2) - 2mE(x) + E(m^2)$$
$$= E(x^2) - 2m^2 + m^2 = E(x^2) - E(x)^2$$

즉, $E(x^2) = Var(x) + E(x)^2$

위 예제에서는 $E(t^2|x) = Var(t|x) + E(t|x)^2$

② $y_* = E(t|x)$ 라 두고 식을 묶으면 $(y-y_*)^2 + Var(t|x)$ 가 된다.

$(y-y_*)^2 + Var(t|x)$ 식이 가지는 의미

베이즈 에러 : 달성할 수 있는 가장 최소 오차

데이터가 가우시안 분포를 따를때 학습데이터에 없는 처음보는 입력에 대하여 이론적으로

Bias/Variance Decomposition

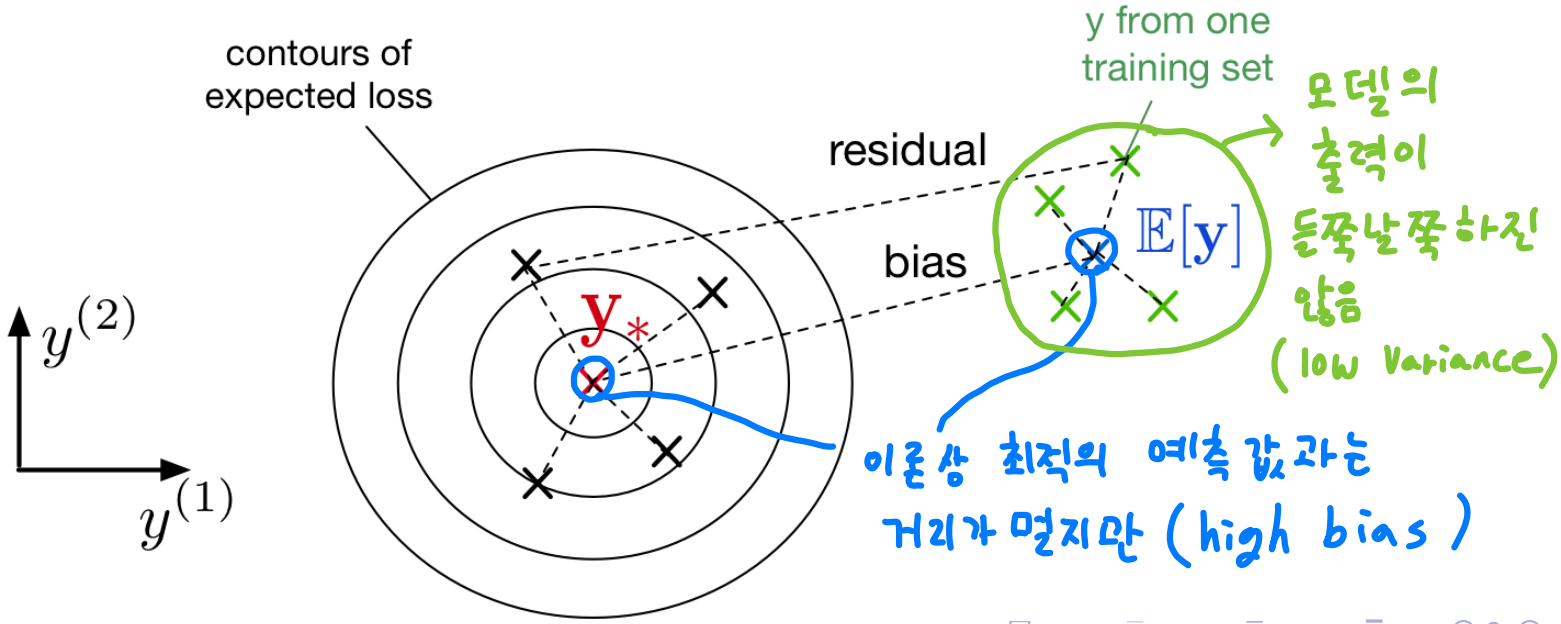
- Now suppose we sample a training set from the data generating distribution, train a model on it, and use that model to make a prediction y on test example $\mathbf{x}$.
- Here, y is a random variable, and we get a new value each time we sample a new training set.
- We'd like to minimize the risk, or expected loss $\mathbb{E}[(y-t)^2]$. We can decompose this into bias, variance, and Bayes error. (We suppress the conditioning on $\mathbf{x}$ for clarity.)

$$
\begin{aligned}
\mathbb{E}[(y-t)^2] &= \mathbb{E}[(y-y_\star)^2] + \mathrm{Var}(t) \\
&= \mathbb{E}[y_\star^2 - 2y_\star y + y^2] + \mathrm{Var}(t) \\
&= y_\star^2 - 2y_\star \mathbb{E}[y] + \mathbb{E}[y^2] + \mathrm{Var}(t) \\
&= y_\star^2 - 2y_\star \mathbb{E}[y] + \mathbb{E}[y]^2 + \mathrm{Var}(y) + \mathrm{Var}(t) \\
&= \underbrace{(y_\star - \mathbb{E}[y])^2}_{\text{bias}} + \underbrace{\mathrm{Var}(y)}_{\text{variance}} + \underbrace{\mathrm{Var}(t)}_{\text{Bayes error}}
\end{aligned}
$$

저/고 적합에 따른
시소 관계

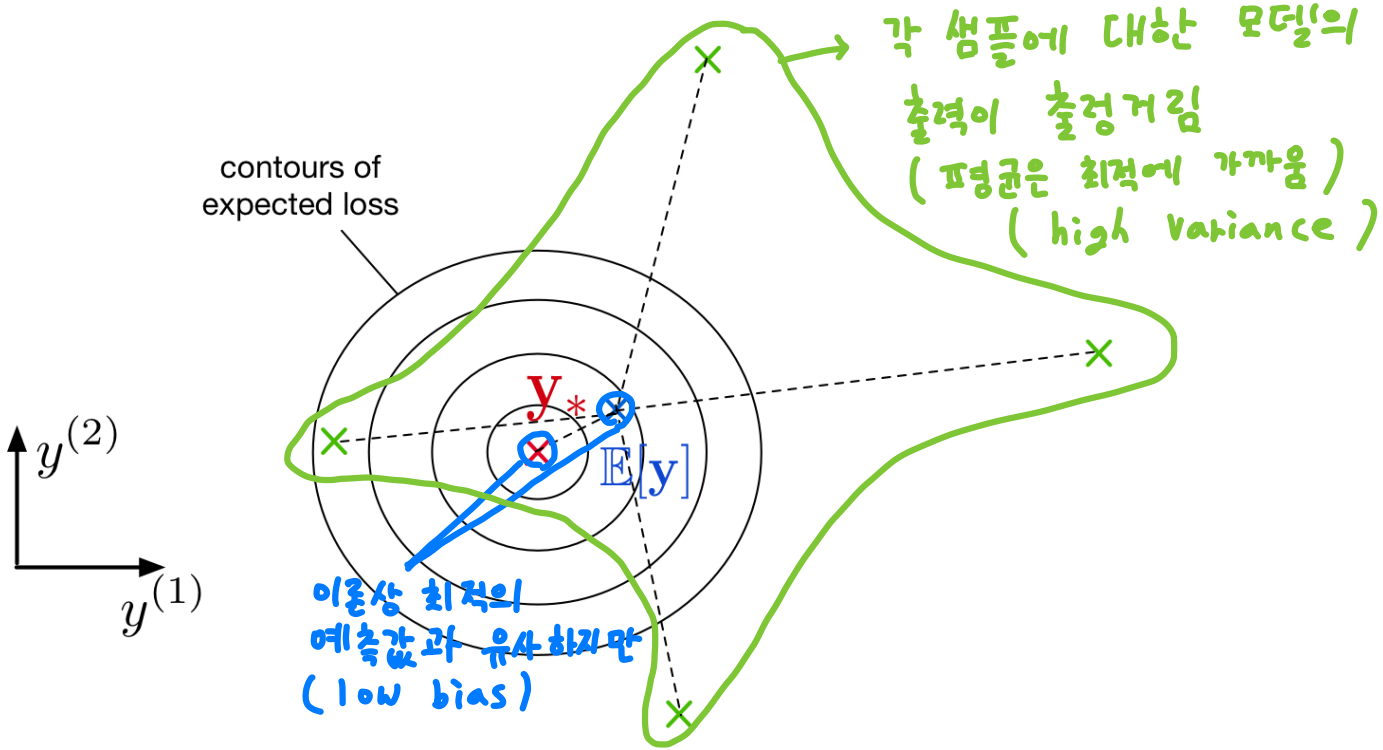
Bias/Variance Decomposition

- We can visualize this decomposition in prediction space, where the axes correspond to predictions on the test examples.
- If we have an overly simple model, it might have
  - high bias (because it's too simplistic to capture the structure in the data)
  - low variance (because there's enough data to estimate the parameters, so you get the same results for any sample of the training data)



Bias/Variance Decomposition

- If you have an overly complex model, it might have
  - low bias (since it learns all the relevant structure)
  - high variance (it fits the idiosyncrasies of the data you happened to sample)



- The bias/variance decomposition holds only for squared error, but it provides a useful intuition even for other loss functions.

Bias : $(Y_* - E[Y])^2$

- low bias : well-trained (could be overfitted)
- high bias : bad-trained (underfitted)

Variance : $Var(Y)$

- low variance : test set result ≈ train set result (Both great or both poor)
- high variance : big diff between train, test (Overfitted)

Our Bag of Tricks

- How can we train a model that's complex enough to model the structure in the data, but prevent it from overfitting? I.e., how to achieve low bias and low variance?
- Our bag of tricks
  - data augmentation
  - reduce the capacity (e.g. number of paramters)
  - weight decay
  - early stopping
  - ensembles (combine predictions of different models)
  - stochastic regularization (e.g. dropout, batch normalization)

6가지

- The best-performing models on most benchmarks use some or all of these tricks.

Data Augmentation 1. 데이터 증식

사실 가능만 하다면
끝판왕, 깡패

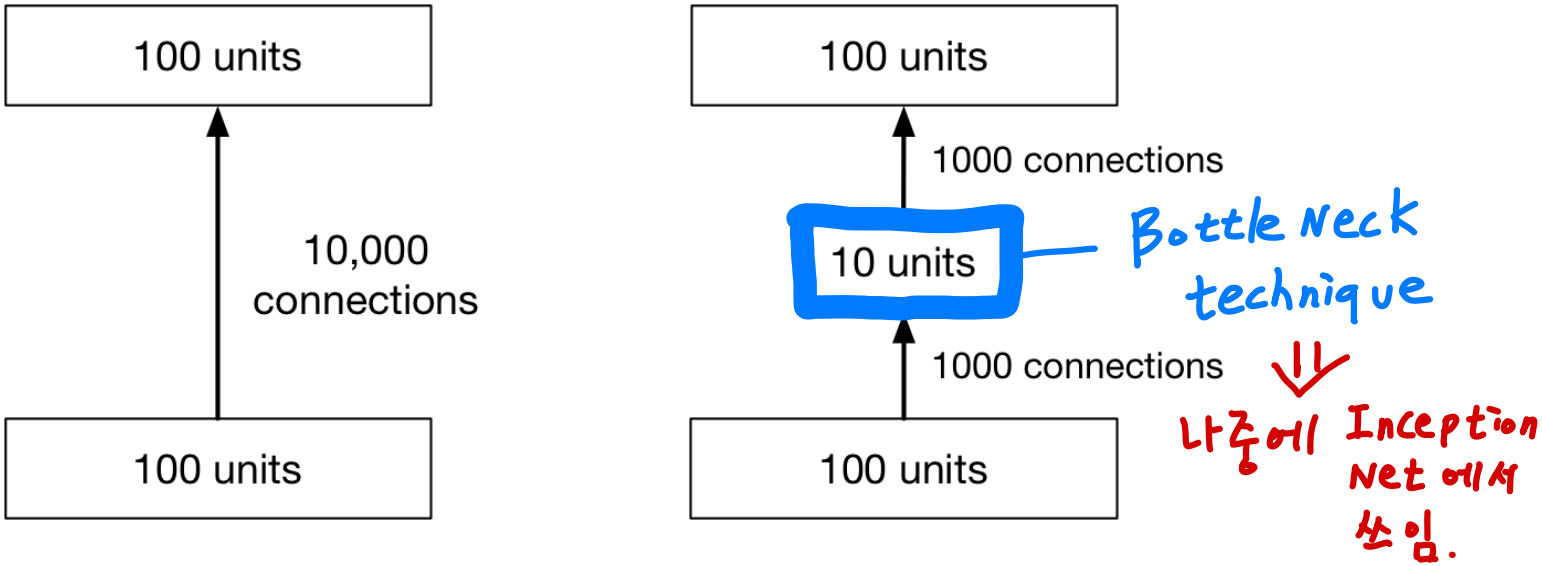
- The best way to improve generalization is to collect more data!
- Suppose we already have all the data we're willing to collect. We can augment the training data by transforming the examples. This is called data augmentation.
- Examples (for visual recognition)
  - translation (위치 이동)
  - horizontal or vertical flip
  - rotation
  - smooth warping
  - noise (e.g. flip random pixels)

이미지 데이터를 예로 들면
다음과 같은 증식 방법들이 있으며,
세부 분야 성격에 맞게 일부
또는 전체 방법들을 활용하여 증식.

- Only warp the training, not the test, examples.
- The choice of transformations depends on the task. (E.g. horizontal flip for object recognition, but not handwritten digit recognition.)

Reducing the Capacity 2. 모델 용량 줄이기

- Roughly speaking, models with more parameters have more capacity. Therefore, we can try to reduce the number of parameters.
- One approach: reduce the number of layers or the number of paramters per layer.
- Adding a linear bottleneck layer is another way to reduce the capacity:



- The first network is strictly more expressive than the second (i.e. it can represent a strictly larger class of functions). (Why?)
- Remember how linear layers don't make a network more expressive? They might still improve generalization.

Weight Decay 3. 가중치 감소 (A.K.A Regularize)

- We've already seen that we can regularize a network by penalizing large weight values, thereby encouraging the weights to be small in magnitude.

$$\mathcal{E}_{\text{reg}} = \mathcal{E} + \lambda \mathcal{R} = \mathcal{E} + \frac{\lambda}{2} \sum_j w_j^2$$

- We saw that the gradient descent update can be interpreted as weight decay:

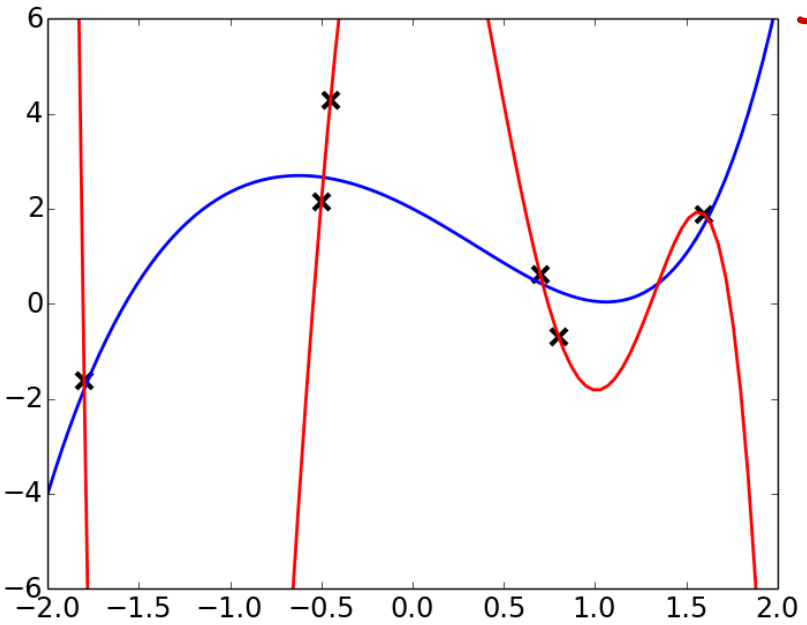
$$\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \alpha \left(\frac{\partial \mathcal{E}}{\partial \mathbf{w}} + \lambda \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \right) \\
&= \mathbf{w} - \alpha \left(\frac{\partial \mathcal{E}}{\partial \mathbf{w}} + \lambda \mathbf{w} \right) \\
&= (1 - \alpha\lambda)\mathbf{w} - \alpha \frac{\partial \mathcal{E}}{\partial \mathbf{w}}
\end{aligned}$$

Weight Decay

상대적 x, 절대적 o

Why we want weights to be small: 가중치가 크면 왜 문제인가?

↳ 과적합 모델의 경우 대개 필요 이상의 차원이며 각 가중치들도 큰 값들이다.



$$y = 0.1x^5 + 0.2x^4 + 0.75x^3 - x^2 - 2x + 2$$

$$y = -7.2x^5 + 10.4x^4 + 24.5x^3 - 37.9x^2 - 3.6x + 12$$

The red polynomial overfits. Notice it has really large coefficients.

Weight Decay

이렇게 가중치가 큰 범위를 가지면
학습데이터에 대해서는 비슷한
출력이 나올수있지만, 새로운 데이터에
대해서는 큰 오차가 발생

Why we want weights to be small:

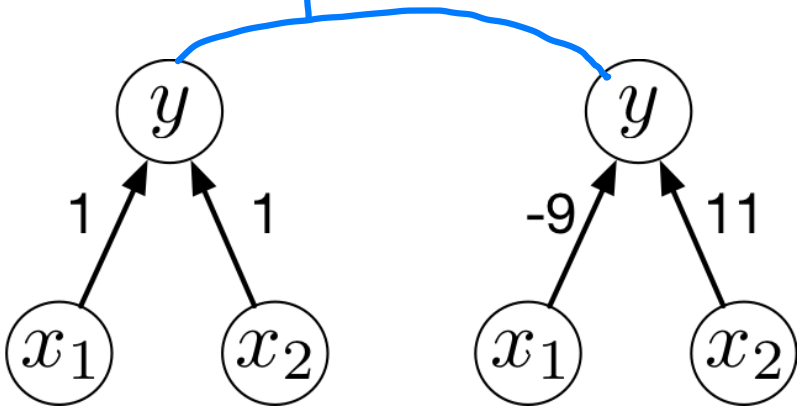
- Suppose inputs x_1 and x_2 are nearly identical. The following two networks make nearly the same predictions:

Ex)

1. $x_1 = 1$, $x_2 = 1$

2. $x_1 = 1$, $x_2 = 2$

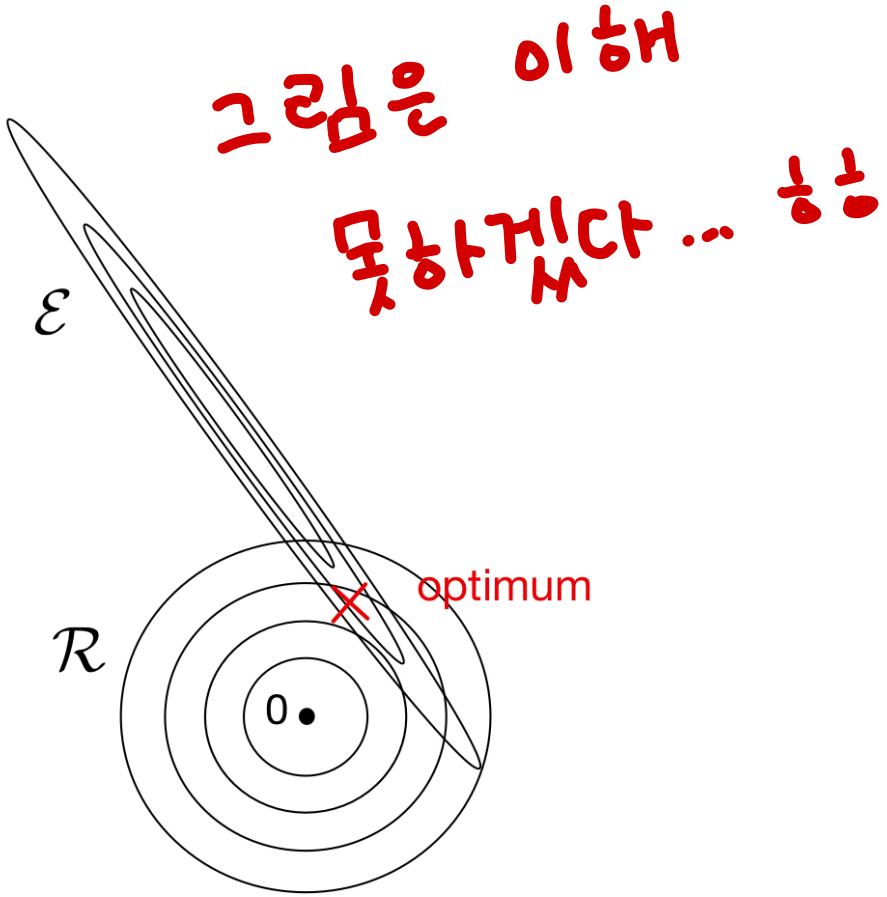
차이가 엄청 남



- But the second network might make weird predictions if the test distribution is slightly different (e.g. x_1 and x_2 match less closely).

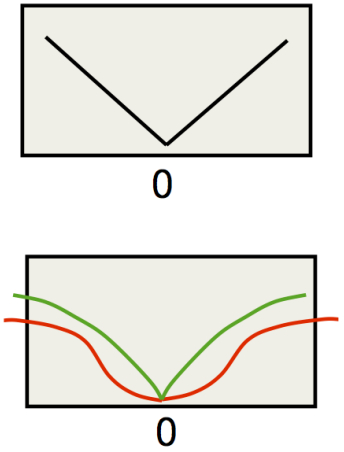
Weight Decay

- The geometric picture:

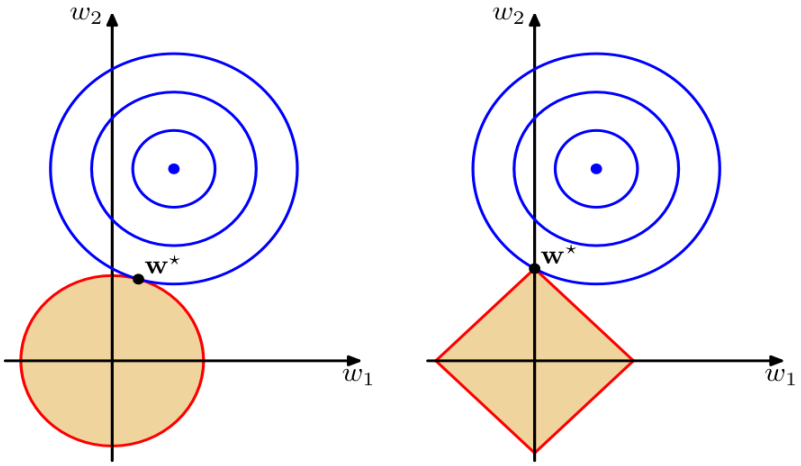


Weight Decay

- There are other kinds of regularizers which encourage weights to be small, e.g. sum of the absolute values.
- These alternative penalties are commonly used in other areas of machine learning, but less commonly for neural nets.
- Regularizers differ by how strongly they prioritize making weights exactly zero, vs. not being very large.



— Hinton, Coursera lectures



L2 regularization

$$\mathcal{R} = \sum_i w_i^2$$

L1 regularization

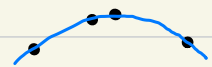
$$\mathcal{R} = \sum_i |w_i|$$

— Bishop, *Pattern Recognition and Machine Learning*

Weight Decay 정리

1. 과적합 모델의 경우 보통 필요 이상의 차원을 가지는 출력 그래프 모양이며 출력이는 높낮이(?) 에 해당하는 가중치 또한 큰숫자들이 많다.

2. Weight Decay 기법은 모델의 차원을 줄이진 못하지만 같은 차원 이라도 출력이는 높낮이가 너무 크지 않도록 가중치 값이 커지는 것을 방지하는 것이 목적. 시각적 표현 ⇒

( , 원래 이렇게 차원을 줄이는게 더 이상적.)

3. "가중치가 커지는 것을 방지" 를 하는 방법은 가중치 업데이트를 결정짓는 '손실함수' 에 'regularization term' 을 추가.

$$Loss_{reg} = Loss + \lambda R$$

lambda, hyper parameter

reg term

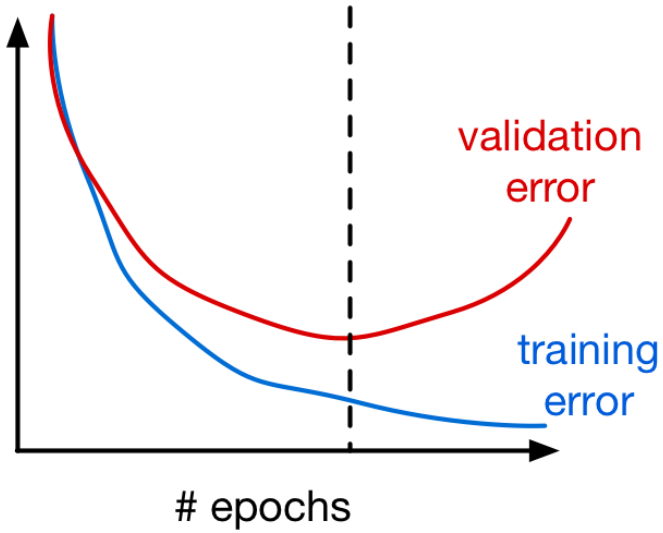
- L_1 reg term : $\sum_i |W_i|$ (아예 0 으로 될 가능성이 있음)
- L_2 reg term : $\sum_i W_i^2$ (0 으로 가까 워지기만 함)

신경망에서는 주로 이것만 있음

Early Stopping 4. 조기 종료

- We don't always want to find a global (or even local) optimum of our cost function. It may be advantageous to stop training early.
- Roughly speaking, training for longer increases the capacity of the model, which might make us overfit.

* 검증 에러를 모니터링 하면서 개선 여지가 없으면 학습

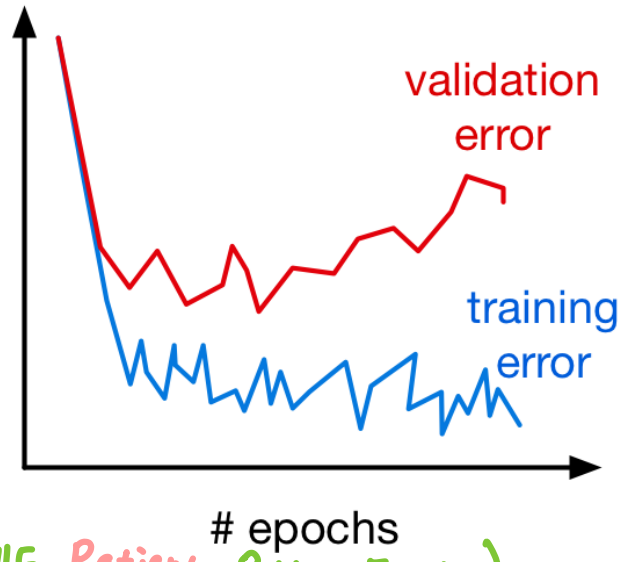


- Early stopping: monitor performance on a validation set, stop training when the validtion error starts going up.

Early Stopping

- A slight catch: validation error fluctuates because of stochasticity in the updates.

단, 실제 데이터는
이전페이지처럼 예쁘게
떨어지는 그림이 아니라
오른쪽 그림처럼 출렁이는 것이
일반적이므로, "개선이 없다"를
판단하기 까다로울 수 있다.
(때문에 keras early stopping 에도 Patient 옵션이 존재)



- Determining when the validation error has actually leveled off can be tricky.

Early Stopping

Early Stopping 은 어떻게 보면 Weight decay 와 결과적으로는 비슷한 목적을 가지고 있음

- Why does early stopping work? (가중치를 작은 범위 내에 유지)
  - Weights start out small, so it takes time for them to grow large. Therefore, it has a similar effect to weight decay.
  - If you are using sigmoidal units, and the weights start out small, then the inputs to the activation functions take only a small range of values.
    - Therefore, the network starts out approximately linear, and gradually becomes more nonlinear (and hence more powerful).

???

Ensembles 5. 앙상블

convex 한 손실함수와 어떤게 best인지 모르는 예측값 무더기가 있다면 답은 평균 !

- If a loss function is convex (with respect to the predictions), you have a bunch of predictions, and you don't know which one is best, you are always better off averaging them.

$$\mathcal{L}(\lambda_1 y_1 + \cdots + \lambda_N y_N, t) \leq \lambda_1 \mathcal{L}(y_1, t) + \cdots + \lambda_N \mathcal{L}(y_N, t) \quad \text{for } \lambda_i \geq 0, \sum_i \lambda_i = 1$$

손실(출력의 평균) ≤ λ_1 손실(출력1) + ⋯ + λ_N 손실(출력N)

- This is true no matter where they came from (trained neural net, random guessing, etc.). Note that only the loss function needs to be convex, not the optimization problem.
- Examples: squared error, cross-entropy, hinge loss
- If you have multiple candidate models and don't know which one is the best, maybe you should just average their predictions on the test data. The set of models is called an ensemble.
- Averaging often helps even when the loss is nonconvex (e.g. 0–1 loss).

non-convex 케이스에서도 가끔 성능개선에 도움이 됨

Ensembles

예시
앙상블 방법

1. 가중치 초기화를 서로 다르게 한 NN 끼리
2. 학습데이터에서 서로 다른 subset으로 학습한 NN 끼리
3. 아키텍처, 파라미터, 혹은 아예 전혀 다른 로직의 모델끼리

- Some examples of ensembles:
  - Train networks starting from different random initializations. But this might not give enough diversity to be useful.
  - Train networks on differnet subsets of the training data. This is called bagging.
  - Train networks with different architectures or hyperparameters, or even use other algorithms which aren't neural nets.
- Ensembles can improve generalization quite a bit, and the winning systems for most machine learning benchmarks are ensembles.
- But they are expensive, and the predictions can be hard to interpret.

앙상블의 trade-off : 연산이 무거워진다. 성능 차이가 큰 모델끼리 앙상블 했을때 오히려 더 역효과 가능성

Stochastic Regularization 6. Drop-out

↳ 확률에 기반한 정칙화 기법으로 Lec9. 에서는 drop-out 정도만 다룸

- For a network to overfit, its computations need to be really precise. This suggests regularizing them by injecting noise into the computations, a strategy known as stochastic regularization.

- Dropout is a stochastic regularizer which randomly deactivates a subset of the units (i.e. sets their activations to zero).

  랜덤하게 일부 활성화 함수(ϕ)를 0으로 만듦

$$h_i = \begin{cases} \phi(z_i) & \text{with probability } 1-\rho \\ 0 & \text{with probability } \rho, \end{cases}$$

  where ρ is a hyperparameter.

- Equivalently,

$$h_i = m_i \cdot \phi(z_i),$$

  where m_i is a Bernoulli random variable, independent for each hidden unit.

- Backprop rule:

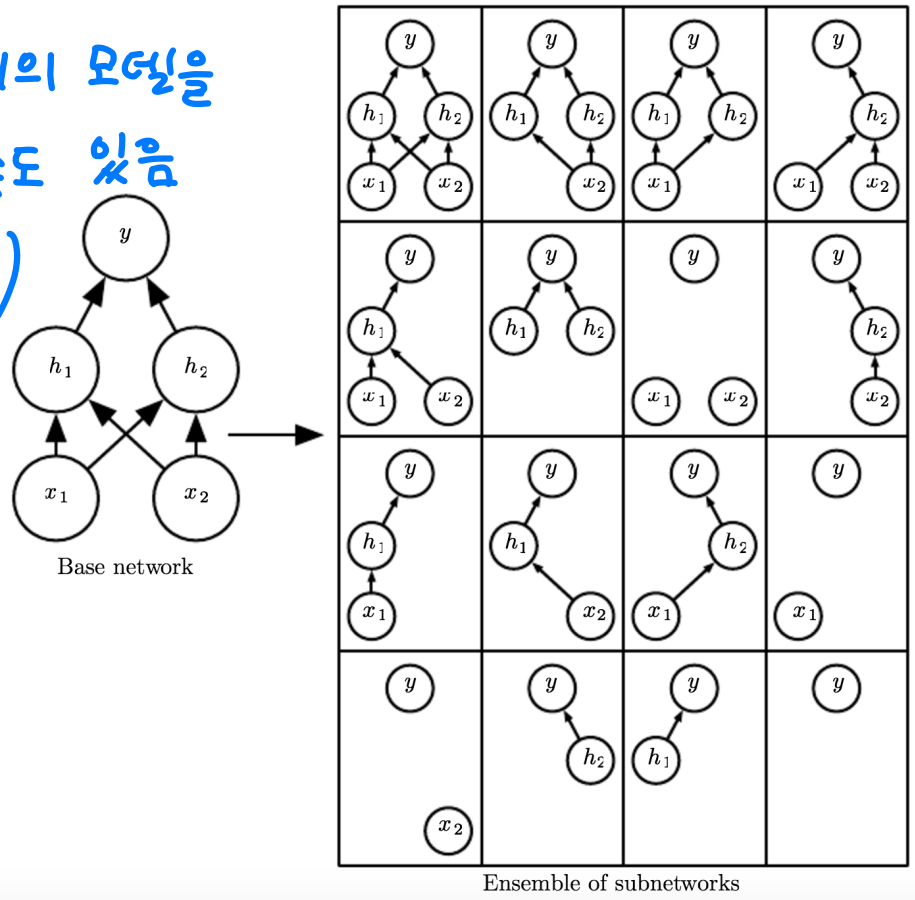
$$\overline{z_i} = \overline{h_i} \cdot m_i \cdot \phi'(z_i)$$

  $\phi = 0$ 에 해당했던 가중치는 backprop 시 Gradient = 0 (업데이트할게 없)

Stochastic Regularization

- Dropout can be seen as training an ensemble of 2^D different architectures with shared weights (where D is the number of units):

관점에 따라 $2^{유닛수}$ 개의 모델을 앙상블 한것처럼 볼 수도 있음 (단, 가중치는 공유하는)



— Goodfellow et al., *Deep Learning*

Stochastic Regularization

다른 여러 확률기 반 regularizer 가 존재.
Lec 9. 에서 다루진 않음 (dropConnect, Batch Normalization)

- Dropout can help performance quite a bit, even if you're already using weight decay.
- Lots of other stochastic regularizers have been proposed:
  - DropConnect drops connections instead of activations.
  - Batch normalization (mentioned last week for its optimization benefits) also introduces stochasticity, thereby acting as a regularizer.
  - The stochasticity in SGD updates has been observed to act as a regularizer, helping generalization.
    - Increasing the mini-batch size may improve training error at the expense of test error!

Our Bag of Tricks

- Techniques we just covered:
  - data augmentation
  - reduce the capacity (e.g. number of paramters)
  - weight decay
  - early stopping
  - ensembles (combine predictions of different models)
  - stochastic regularization (e.g. dropout, batch normalization)
- The best-performing models on most benchmarks use some or all of these tricks.
- Many of these techniques involve hyperparameters. How do we choose these?

↳ 지금까지 살펴본 6가지 일반화 기법 전체 또는 일부를 활용하려면 그에 맞는 hyperparameter도 잘 튜닝해야 한다.

Choosing Hyperparameters

어떤 것들이 있고 어떻게 베스트를 찾을 것인가?

- Many hyperparameters are relevant to generalization
  - number of layers 층 수
  - number of units per layer 층당 유닛 수
  - L_2 weight cost L_2 에 해당하는 regularizer
  - dropout probability drop-out 확률 (p)
- Ideally, we want to choose hyperparameters which perform the best on a validation set.
- Ways to approximate this
  - manual ("graduate student descent")
  - grid search
  - random search
  - Bayesian optimization (covered later in the course)
  - use Autograd to differentiate through the whole training procedure, and do gradient descent on hyperparameters (only practical for very small problems)

미쳤냐고 ㅠㅠㅠ.... 학생이 죄를 지으면 대학원.....

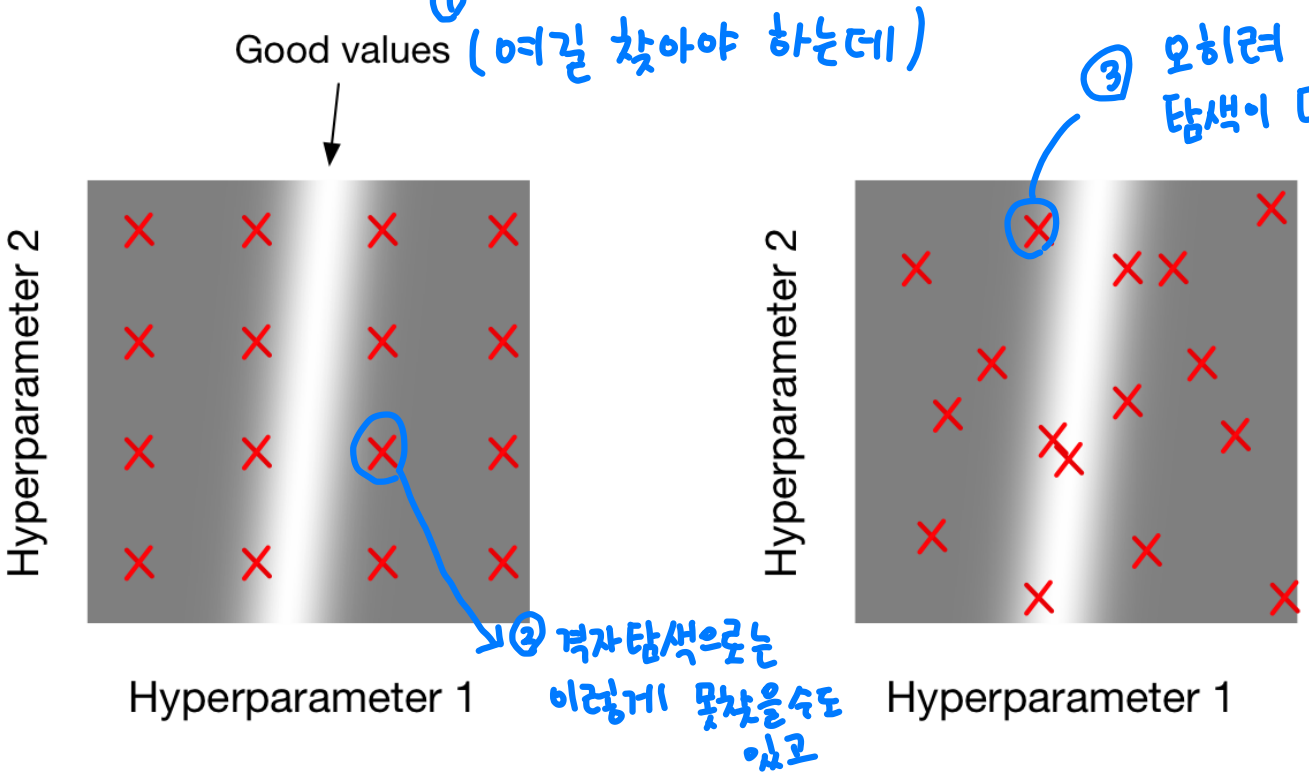
아는것들

모르지만 추후 강의에서 다룰것들

Choosing Hyperparameters

(중요한 내용은 X, 일부 하이퍼파라미터가 영향이 크지 않은 경우라면 무작위 탐색이 격자탐색보다 좋을수도 있다.)

- Random search can be more efficient than grid search when some of the hyperparamters are unimportant:



- But grid search can be more reproducible.

④ 하지만 보통의 경우 grid search가 좋다.