



M2177.003100

Deep Learning

[10: Regularization]

Electrical and Computer Engineering
Seoul National University

© 2018 Sungroh Yoon. this material is for educational uses only. some contents are based on the material provided by other paper/book authors and may be copyrighted by them.

(last compiled at 14:46:00 on 2018/10/28)

Outline

Introduction

Adversarial Training

Theory of Regularization

Summary

Regularization Techniques

Appendix

References

- *Deep Learning* by Goodfellow, Bengio and Courville [▶ Link](#)
 - ▶ Chapter 7
- online resources:
 - ▶ *Deep Learning Specialization (coursera)* [▶ Link](#)
 - ▶ *Stanford CS231n: CNN for Visual Recognition* [▶ Link](#)
 - ▶ *Machine Learning Yearning* [▶ Link](#)

Outline

Introduction

Adversarial Training

Theory of Regularization

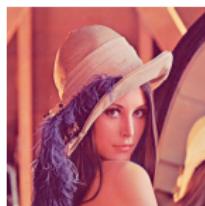
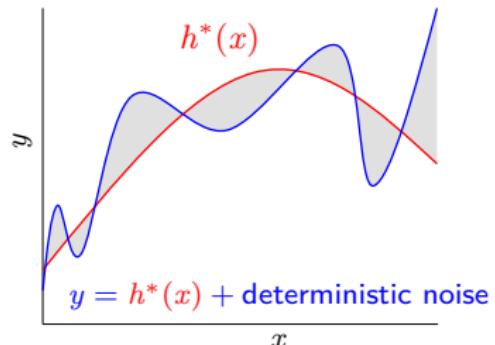
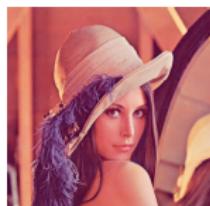
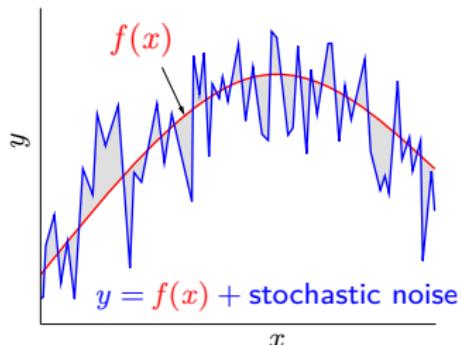
Summary

Regularization Techniques

Appendix

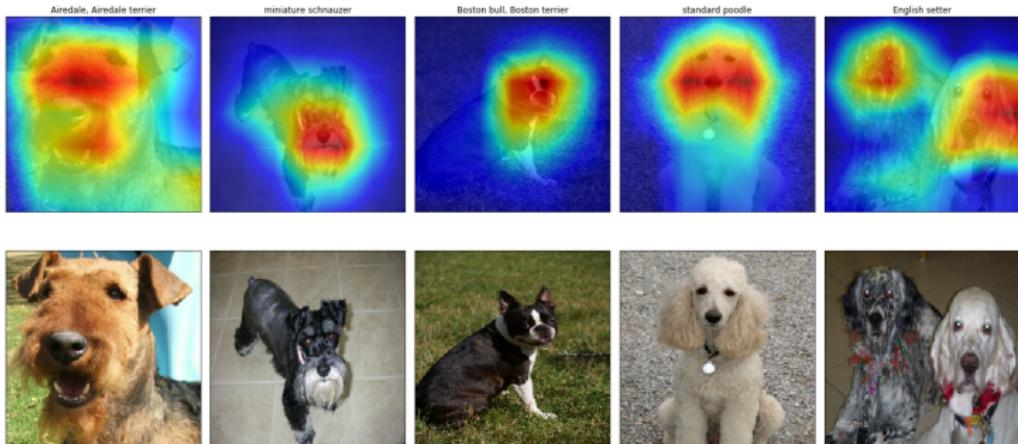
Noise: part of y we cannot model

- stochastic/deterministic noise hurts learning by leading to overfitting



- human: good at extracting **simple** patterns
 - ▶ ignoring noise and complications
- computers: pay equal attention to all pixels
 - ▶ need help for simplification
 - ▶ e.g. feature extraction, regularization, attention

=> 사람은 모든 pixel에 똑같은 attention을 주지 않음



Overfitting

warranted => 필요한 정도

- literally: fitting the data more than is warranted
- consequences
 - ▶ fitting observation (small E_{train}) no longer indicates decent E_{test}
⇒ Train Error alone is no longer good guide for learning
- observed when model의 capacity가 너무 커서 noise까지 learn하는 경우
overfitting 발생
 - ▶ a model is more complex than is necessary to represent target
 - ▶ the model uses its additional DOF to fit noise in data
 - ⇒ inferior final model
- ability to deal with overfitting
 - ▶ what separates professionals from amateurs

Regularization

=> test error를 낮출 수 있는 모든 방법을 다 regularization이라고 할 수 있음

- what is it?

- ▶ a cure for tendency to fit noise, hence improving E_{test}
- ▶ effective especially when noise is present

- how does it work?

- ▶ constrain a model so that it can avoid fitting noise

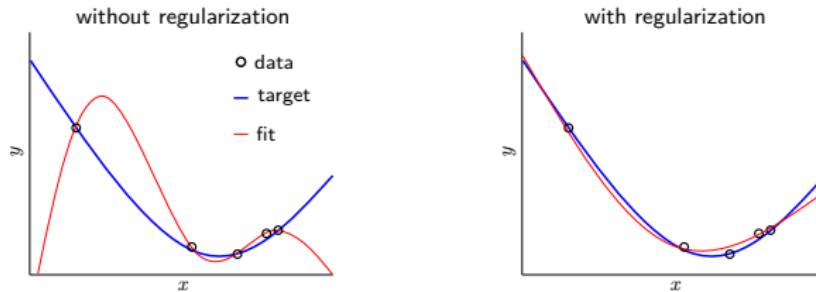
regularization =>

- any side effects?

많이 쓰면 under fitting

적게 쓰면 over fitting

- ▶ we may become incapable of fitting f (signal) faithfully



Regularization strategies

1. put extra constraints on a model

e.g. add restrictions on parameter values

2. add extra terms in objective function

- ▶ can be viewed as a soft constraint on parameter values

3. combine multiple hypotheses that explain training data

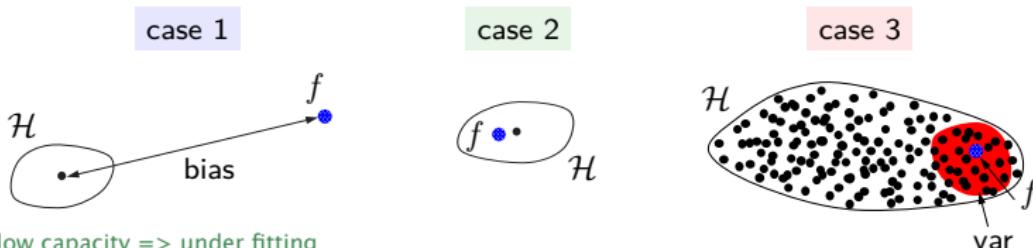
- ▶ called ensemble methods

- extra constraints and penalties encode: => 사전지식을 이용하는 것도 regularization이라고 할 수 있음

- ▶ specific kinds of prior knowledge, or
 - ▶ generic preference for a simpler model to promote generalization

Goal of regularization

- three situations during training: the model either



case1: low capacity => under fitting

case	model	main error	phenomenon
1	exclude f (true data generating process)	bias	underfitting
2	match f		
3	include f but also many others	variance	overfitting

- goal of regularization:

- to take a model from regime #3 → regime #2

In the context of deep learning

- deep learning:
 - ▶ applied to extremely complicated domains (images/audio/text)
 - ⇒ true generation process involves simulating the entire universe
 - ⇒ true data generating process: almost certainly outside the model family
- this means: controlling the complexity of the model is
 - ▶ not a simple matter of finding the model of right size, # of parameters
- instead (almost always in practical deep learning)
 - ▶ best¹ model
 - = a large model that has been **regularized** appropriately

¹in the sense of minimizing generalization error

Outline

Introduction

Adversarial Training

Theory of Regularization

Summary

Regularization Techniques

Appendix

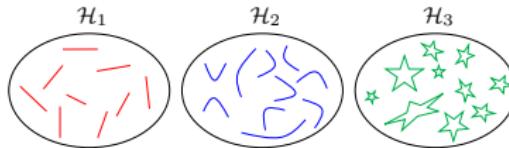
Regularization

- a process of introducing additional information, in order to
 - ▶ solve an ill-posed problem² or
 - ▶ prevent overfitting
- additional information: usually a penalty for complexity, *e.g.*,
 - ▶ restrictions for smoothness
 - ▶ bounds on the vector space norm
- two approaches
 1. mathematical: function approximation for ill-posed problems
 2. heuristic: handicapping the minimization of E_{train}
- regularization is as much an art as it is a science

²a well-posed problem (defined by Jacques Hadamard): (i) a solution exists (ii) the solution is unique (iii) the solution's behavior changes continuously with the initial conditions

VC generalization bound revisited

- f : unknown target function (objective of learning)
 - ▶ g : our (best) model learned from data (one of $h \in \mathcal{H}$)
 - ▶ \mathcal{H} : hypothesis set from which we choose g



- recall **VC generalization bound** (d_{VC} determined by which \mathcal{H} is chosen):

$$\mathbb{P}\left[\underbrace{|E_{\text{train}}(f) - E_{\text{test}}(f)|}_{\text{bad event}} > \epsilon\right] \leq \underbrace{4 \cdot (2N)^{\overbrace{d_{VC}}^{\text{capacity}}} \cdot e^{-\frac{1}{8}\epsilon^2 N}}_{\text{VC bound}}$$

- from this we can derive: $E_{\text{test}}(h) \leq E_{\text{train}}(h) + \Omega(\mathcal{H})$ for all $h \in \mathcal{H}$
i.e. E_{test} : bounded by penalty $\Omega(\mathcal{H})$ on model complexity

Core concept

- VC bound: $E_{\text{test}}(h) \leq E_{\text{train}}(h) + \Omega(\mathcal{H})$ for all $h \in \mathcal{H}$
 - ▶ good: we can fit data using a simple \mathcal{H}
 - regularization: takes one step further
 - ▶ even better: we can fit using a simple $h \in \mathcal{H}$
 - essence of regularization³
 - ▶ concoct $\Omega(h)$ for individual hypothesis
 - ▶ minimize combination of $E_{\text{train}}(h)$ and $\Omega(h)$ (not $E_{\text{train}}(h)$ alone)
- e.g. “weight decay” regularizer: minimize $E_{\text{train}}(\mathbf{w}) + \frac{\lambda}{N} \mathbf{w}^\top \mathbf{w}$
- ▶ $\Omega(h)$: now part of optimization objective
 - ▷ we can thus avoid overfitting by constraining learning process
- a member of even a large model family can be appropriately regularized

=> regularization을 잘하면 복잡한 model이라
도 effective capacity를 낮출 수 있다

³we use $\Omega(h) \leftrightarrow \Omega(w)$ and $E(h) \leftrightarrow E(w)$ interchangeably

Augmented error

- combination of E_{train} and Ω (penalty on model complexity)

E_{train} 을 줄이면 overfitting이지만
 E_{aug} 을 줄이면 overfitting에 fight 가능

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{train}}(\mathbf{w}) + \underbrace{\frac{\lambda}{N} \Omega(\mathbf{w})}_{\text{penalty term}} \quad (1)$$

regularization parameter regularizer

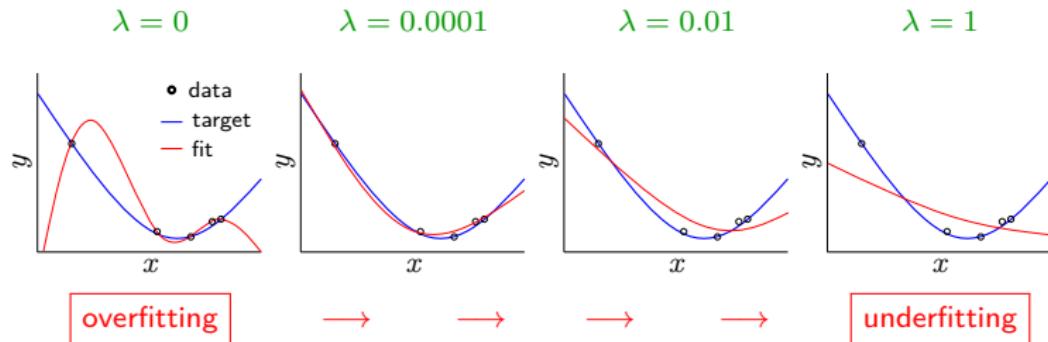
- ▶ better proxy for E_{test} than E_{train} (see [Appendix](#) for additional theory)
- example: weight decay regularizer (more on this soon):

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{train}}(\mathbf{w}) + \frac{\lambda}{N} \mathbf{w}^\top \mathbf{w}$$

- need for regularization goes down as $N \uparrow$
 - ▶ we thus factor out $\frac{1}{N}$
 - ▶ this allows optimal λ to be less sensitive to N

How to select regularizer Ω and parameter λ

- regularizer Ω ← heuristic
 - ▶ typically fixed ahead of time, before seeing data
 - ▶ sometimes problem itself dictates an appropriate regularizer
- choice of optimal λ ← principled
 - ▶ typically depends on data
 - ▶ overdose leads to underfitting ⇒ validation will tell us (see Appendix)



Considerations in neural nets

parameter	weight	bias
role	specifies how two variables interact	controls only a single variable
accurate fitting needs	more data (to observe both variables in various conditions)	less data than weights

- ▶ unregularized bias → not much variance
- ▶ regularizing bias → can give **significant underfitting** => bias regularization은 안한다!
(weight만 regularization함)
- we thus use penalty Ω that leaves **biases unregularized**
 - ▶ penalize *only weights* of affine transformation at each layer
- convention in textbook:
 - w all the weights that should be affected by Ω
 - θ all the parameters including both w and unregularized parameters

Outline

Introduction

Dropout
Other Techniques

Theory of Regularization

Adversarial Training

Regularization Techniques

Norm Penalties

Summary

Early Stopping

Appendix

Ensemble Methods

Most famous: l_1 and l_2 regularizers

- l_1 regularizer: aka lasso (statistics), *basis pursuit* (signal processing)
 - ▶ convex but not differentiable everywhere
 - ▶ variable shrinkage + selection: \Rightarrow sparse solution L1 regularization => sparse
(대부분 0이고 소수가 살아남음)

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_1 = \sum_q |w_q|$$

- l_2 regularizer: aka ridge (statistics), *weight decay* (neural nets)
 - ▶ math friendly (convex/differentiable) => 작게는 만들 수 있지만 sparse는 힘듬
 - ▶ variable shrinkage only: shrinks w 's of correlated x 's

$$\Omega(\mathbf{w}) = \|\mathbf{w}\|_2 = \mathbf{w}^\top \mathbf{w} = \sum_q w_q^2$$

- see [Appendix](#) for additional comparison

- why “weight decay”? $\text{AugError} = \text{Etrain} + |\mathbf{W}|$

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla \text{E}_{\text{aug}}(\mathbf{w})$$

$$= \mathbf{w} - \epsilon \nabla \text{E}_{\text{train}}(\mathbf{w}) - 2\epsilon \frac{\lambda}{N} \mathbf{w}$$

$$= \underbrace{\left(1 - 2\epsilon \frac{\lambda}{N}\right)}_{\text{decay}} \mathbf{w} - \epsilon \nabla \text{E}_{\text{train}}(\mathbf{w})$$

새로운 \mathbf{W} 가 이전 \mathbf{W} 의 곱을 이용하여 정의됨

- l_2 (weight decay) regularized cost function in neural nets:

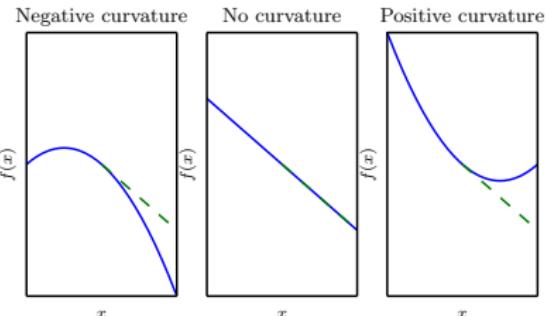
$$\underbrace{J(\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \dots, \mathbf{W}^{[L]}, \mathbf{b}^{[L]})}_{\text{E}_{\text{aug}}(\mathbf{W}, \mathbf{b})} = \underbrace{\frac{1}{m} \sum_{i=1}^m L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)})}_{\text{E}_{\text{train}}(\mathbf{W}, \mathbf{b})} + \underbrace{\frac{\lambda}{m} \sum_{l=1}^L \|\mathbf{W}^{[l]}\|_F^2}_{\Omega(\mathbf{W}): \text{regularizer}}$$

- ▶ no regularization for bias
- ▶ Frobenius norm of a matrix $\Leftrightarrow l_2$ norm of a vector

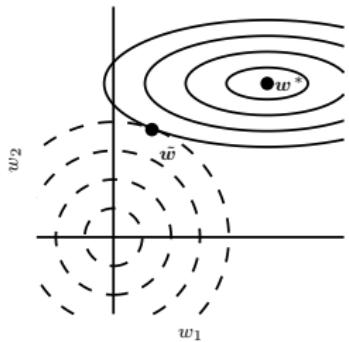
$$\|\mathbf{A}\|_F^2 = \sum_{i,j} A_{i,j}^2$$

Effect of weight decay

- Hessian \mathbf{H} of J gives curvature
 - ▶ the higher eigenval(\mathbf{H}) \uparrow
 - ⇒ the more curvature \uparrow



- rescale w^* along the axes defined by the eigenvectors of \mathbf{H}
 - ▶ scale factor for i -th eigenvector: $\frac{\text{eigenval}(\mathbf{H})_i}{\text{eigenval}(\mathbf{H})_i + \lambda}$



- w_1 direction (\leftrightarrow): eigenval($\mathbf{H})_1$ small
 - ▶ J not increase much when moving away from w^* ⇒ J has no strong preference
 - ▶ regularizer has a strong effect
- w_2 direction (\Downarrow): eigenval($\mathbf{H})_1$ large
 - ▶ regularizer affects this direction little

Other (generalized) regularizers

- Tikhonov regularizer
 - ▶ generalization of weight decay

$$\Omega(\mathbf{w}) = \mathbf{w}^\top \Gamma^\top \Gamma \mathbf{w} = \sum_p \sum_q w_p w_q \gamma_p \gamma_q$$

- elastic-net penalty => L1과 L2의 합집합
 - ▶ compromise between l_1 lasso ($\alpha = 1$) and l_2 ridge ($\alpha = 0$)

$$\Omega(\mathbf{w}) = \sum_q \left\{ \alpha |w_q| + \frac{1}{2}(1 - \alpha) w_q^2 \right\}$$

Comparison

- regularization path

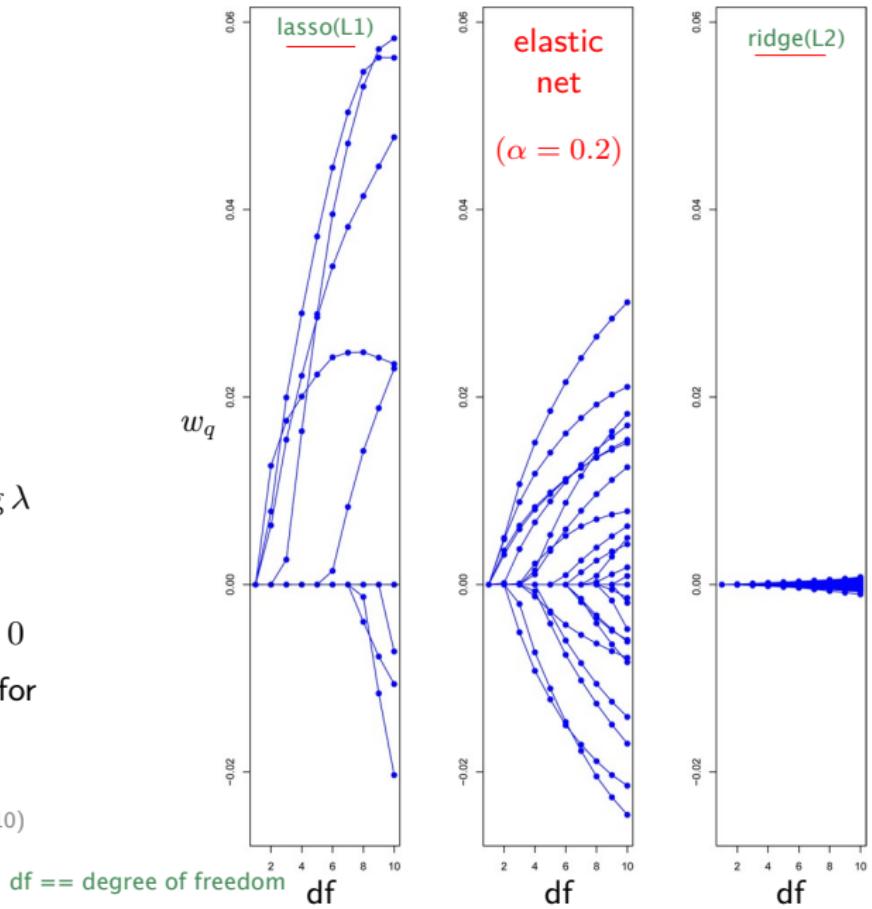
- ▶ l_1 (lasso)
- ▶ elastic net
- ▶ l_2 (ridge)

- degree of freedom:

$$\sim C \sim \frac{1}{\lambda} \sim -\log \lambda$$

- ▶ $df \rightarrow 0$ as
 $\lambda \rightarrow \infty, C \rightarrow 0$
- ▶ see [Appendix](#) for
concept of C

(source: Friedman *et al.*, 2010)



Outline

Introduction

Dropout
Other Techniques

Theory of Regularization

Adversarial Training

Regularization Techniques

Norm Penalties

Summary

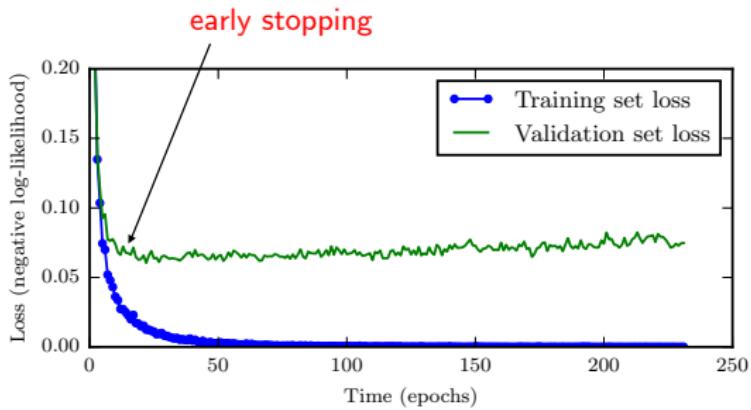
Early Stopping

Appendix

Ensemble Methods

Early stopping

- overfitting: $E_{\text{train}} \searrow$ but $E_{\text{dev}} \nearrow$
 - ▶ natural idea: how about stopping when E_{dev} gets low?
- early stopping: keep track of both E_{train} and E_{dev}
 - ▶ stop training with lowest E_{dev} \Rightarrow potentially better E_{test}

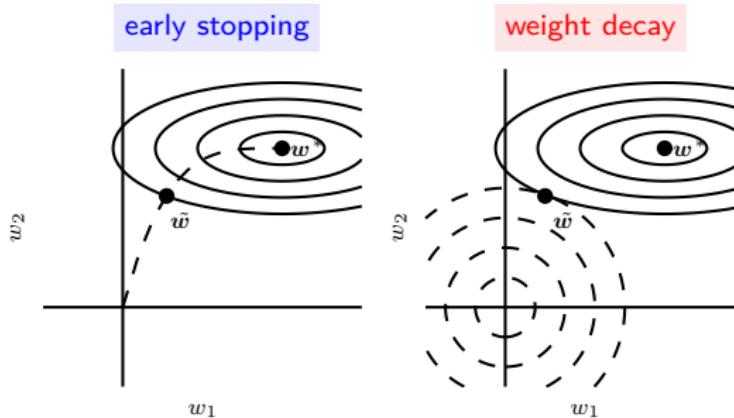


- ▶ effective/simple \rightarrow very popular also in deep learning

- how it works (algorithm 7.1):
 - ▶ every time the error on validation set improves
 - ▷ store a copy of parameters (\leftarrow returned when training terminates)
 - ▶ algorithm terminates when no improvement in E_{dev}
- advantages:
 1. very unobtrusive (unnoticeable) [c.f. weight decay]
 - ⇒ no change in learning dynamics (e.g. no need for adding Ω to J , etc)
 2. early stop \Rightarrow fewer epochs \Rightarrow computational savings
 3. leave extra data for additional training (algorithms 7.2 & 7.3)
early stopping해서 아직 쓰지 않은 train set을
이용하여 추가 train 가능(data가 늘어난 것 같은
효과)
- disadvantages:
 - ▶ must compute E_{dev} periodically during training
 - ▷ mitigated by separate GPUs, small dev set, infrequent validation
 - ▶ additional memory to store best parameters (but negligible)

Comparison

early stopping과 weight decay가 똑같
은게 아님(그림이 우연히 그렇게 나온것임)



- **early stopping:** more than mere restriction of trajectory length
 - ▶ monitors E_{dev} to stop trajectory
 - ⇒ auto-determines correct amount of regularization (a big plus)
- **weight decay**
 - ▶ requires many training experiments with different hyperparameters

Outline

Introduction

Dropout
Other Techniques

Theory of Regularization

Adversarial Training

Regularization Techniques

Norm Penalties

Summary

Early Stopping

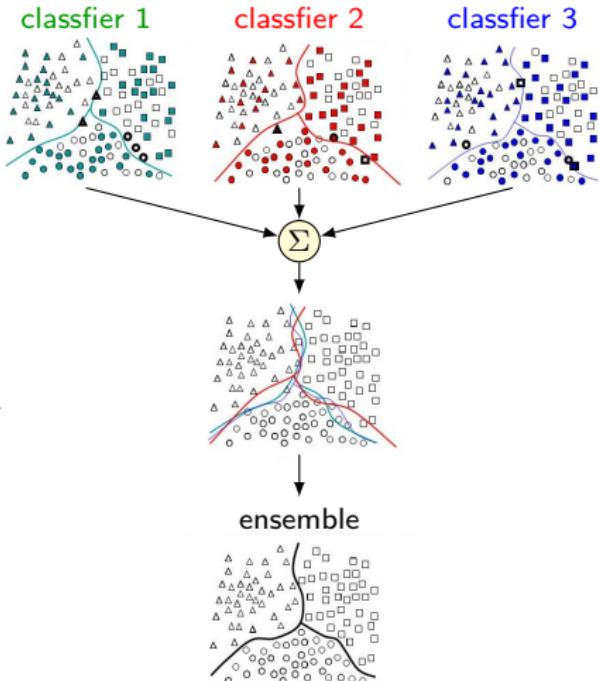
Appendix

Ensemble Methods

regularization => Etest를 낮추는 모든 방법을 통칭

Ensemble learning

- idea: make a strong model
 - ▶ by combine weak models
 - ▶ “model averaging”
- strong?
 - ▶ better bias/variance/accuracy...
- assumption
 - ▶ different models → different mistakes
 - ▶ averaging noise → zero
- most famous
 - ▶ (weighted) voting
 - ▶ bagging
 - ▶ boosting



(source: scholarpedia)

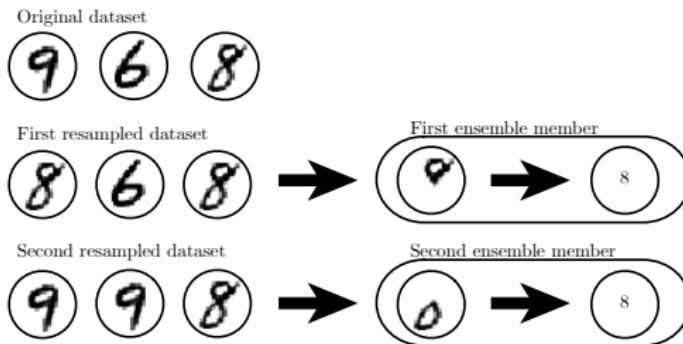
Bagging (bootstrap aggregating)

=> parallel함

- combines several models to **reduce generalization error** (see Appendix)
 1. **train different models** separately
 2. then have all models vote on output for test examples
- how to “**train different models**”?
 - ▶ construct k different datasets by random sampling
 - ▶ reuse same model/training algorithm/objective function

⇒ equivalent to training different models

=> 똑같은 구조여도 data가 다르기 때문에 다른 'model'임



Boosting

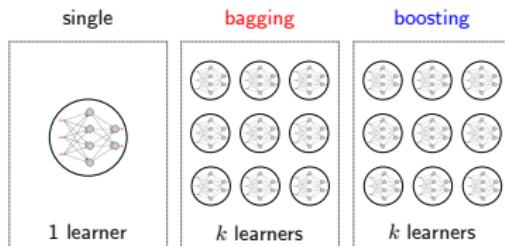
=> sequential합

- constructs an ensemble with **higher capacity** than individual models
 - ▶ meta-algorithm for primarily reducing bias, and also variance
 - ▶ most famous: AdaBoost
- train multiple weak learners **sequentially** to get a strong learner
 - ▶ future learners focus more on the examples previous learners misclassified
 - ▶ how? reweight training examples wrt previous learning results

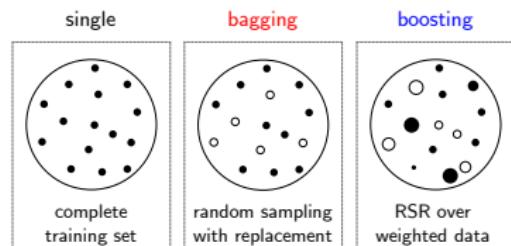
=> 틀린애한테 더 높은 weight를 줌(어려운 문제를 다시 푸는 느낌)
- boosting examples in neural nets:
 - ▶ incrementally add neural nets to the ensemble
 - ▶ incrementally add hidden units to the neural net

Comparison

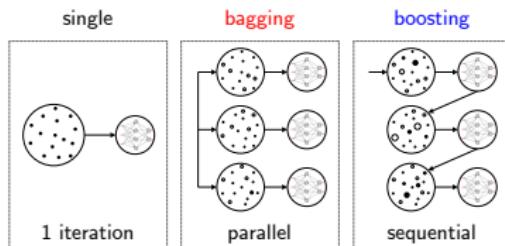
- # of learner(s)



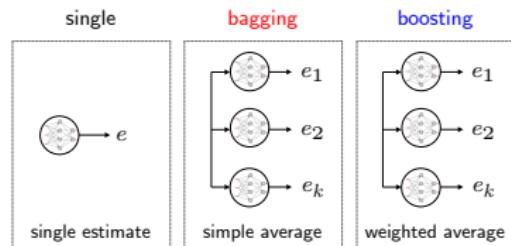
- data



- training



- aggregation⁴



(source: QuatDare [▶ Link](#))

⁴ simple average: $e = \frac{1}{k} \sum_{i=1}^k e_i$, weighted average: $e = \frac{1}{k} \sum_{i=1}^k w_i e_i$

Neural networks

- reach a wide enough variety of solution points
 - ▶ that they can often benefit from model averaging
 - ▶ even if all the models are trained on the **same dataset**
 - differences in
 - ▶ random initialization
 - ▶ random selection of minibatches
 - ▶ hyperparameters
 - ▶ non-deterministic implementations
- ⇒ often enough to cause
- ▶ different members of the ensemble to make **partially independent errors**

Final remarks

- model ensembles: extremely powerful and reliable to reduce generalization error
 1. train multiple independent models
 2. at test time: average their results

⇒ typically gives about 2% extra performance (price: computation/memory)
- ML contests:
 - ▶ won by methods using model averaging over dozens of models
 - e.g. Netflix Grand Prize, Kaggle, and many more
- c.f. scientific papers: ensembles discouraged when benchmarking algorithms
 - ▶ benchmark comparisons: usually made using a single model

Outline

Introduction

Theory of Regularization

Regularization Techniques

Norm Penalties

Early Stopping

Ensemble Methods

Dropout

Other Techniques

Adversarial Training

Summary

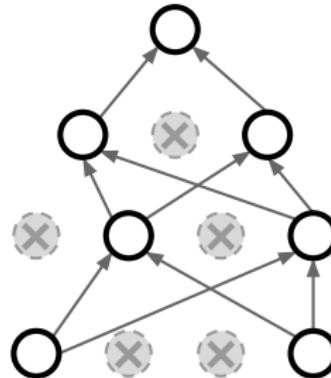
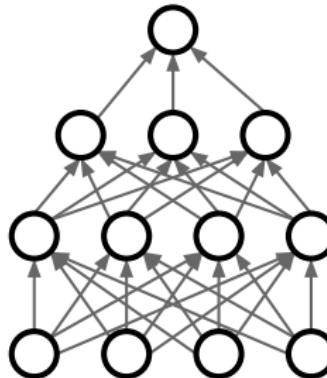
Appendix

Dropout => bagging을 뽑시 저렴하게 할 수 있는 방법

- computationally inexpensive but powerful regularization
 - ▶ makes bagging practical for large neural nets
- recall: bagging
 - ▶ trains multiple models and
 - ▶ evaluates multiple models on each test example
 - ⇒ impractical when each model is a large neural net
- common: use ensembles of five to ten neural nets
 - e.g. Szegedy *et al.* (2014a) used six to win ILSVRC
 - ▶ but more than this rapidly becomes unwieldy
- dropout provides an inexpensive approximation to
 - ▶ training/evaluating a bagged ensemble of **exponentially many** neural nets

Idea

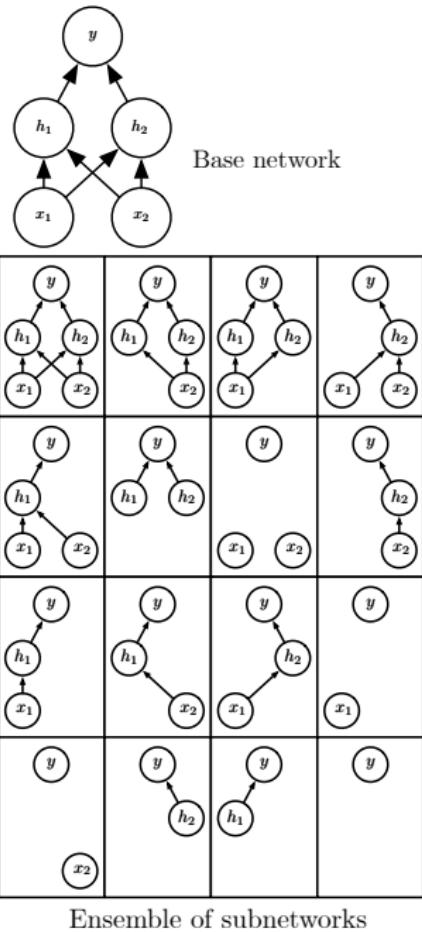
- in each forward pass
 - ▶ randomly set some neurons to zero
- probability of inclusion (or dropping out)
 - ▶ a hyperparameter (commonly: 0.5 for hidden units)



Example

4variable => $2^4 = 16$ 가지 경우

- given an underlying base network
 - e.g. two input units and two hidden units
 - ⇒ sixteen subnetworks
- dropout trains the ensemble of all subnetworks
- subnetworks: formed by
 - ▶ removing **non-output units** from base net
- how to remove a unit?
 - ▶ multiply its output value by zero

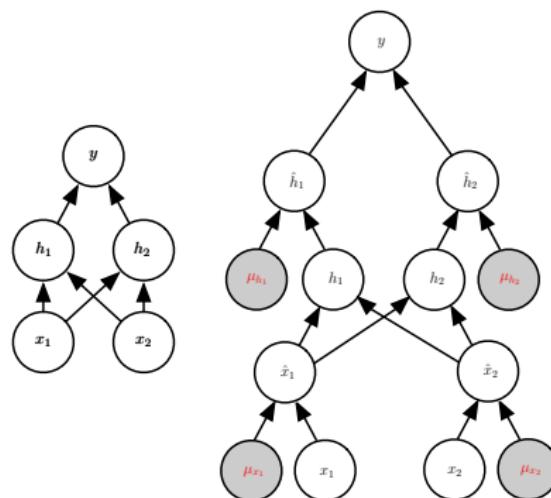


Dropout: training time

매 minibatch마다 random mask vector를 만 들어서 곱해주면 됨

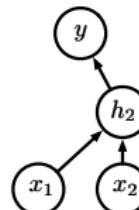
- each time we load an example into a minibatch
 - ▶ randomly sample a binary mask⁵ to apply to all input/hidden units
- then run as usual: forward prop → backprop → parameter updates

e.g.



- let μ : mask vector
 - ▶ specifies which units to include
- top right subnet on page 39

$$\begin{aligned}\mu &= (\mu_{x_1}, \mu_{x_2}, \mu_{h_1}, \mu_{h_2}) \\ &= (1, 1, 0, 1)\end{aligned}$$



⁵typically, $\text{prob}\{\text{mask value} = 1\}$: 0.5 (hidden unit), 0.8 (input unit)

Dropout: inference time

- ensemble prediction (bagging):

$$p_{\text{ensemble}}(y | \mathbf{x}) = \frac{1}{k} \sum_{i=1}^k \underbrace{p^{(i)}(y | \mathbf{x})}_{\text{from model } i} \quad (2)$$

- ensemble prediction (dropout):

$$p_{\text{ensemble}}(y | \mathbf{x}) = \sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) \underbrace{p(y | \mathbf{x}, \boldsymbol{\mu})}_{\substack{\text{submodel defined by} \\ \text{mask vector } \boldsymbol{\mu}}} \quad (3)$$

- $p(\boldsymbol{\mu})$: probability distribution used to sample $\boldsymbol{\mu}$ at training time
- eq (3): intractable (\leftarrow exponential number of terms in \sum) $=> 2^n$ 이므로 계산 불가
 - solution: just average over many masks (e.g. 10–20) or
 - approximate $p_{\text{ensemble}}(y | \mathbf{x})$ by evaluating $p(y | \mathbf{x})$ in one model (e.g. weight scaling inference rule)

Weight scaling inference rule (Hinton *et al.*, 2012)

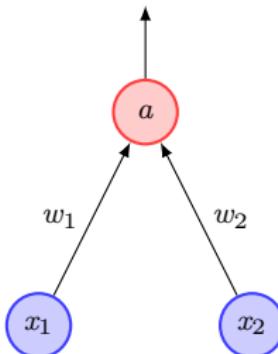
- for each unit:
 - ▶ replace outgoing weight W \longrightarrow aW
 - ▶ α : probability of including (not dropping out) this unit
- motivation
 - ▶ at test time: all neurons are always active
 - ⇒ we must scale activations so that for each neuron:

output at **test time** = expected output at **training time**

- example: a simple model ($\alpha = 0.5$)

► test time:

$$a = w_1 x_1 + w_2 x_2$$



► training time:

$$\begin{aligned}\mathbb{E}[a] &= \frac{1}{4}(w_1 x_1 + w_2 x_2) + \frac{1}{4}(w_1 x_1 + 0) \\ &\quad + \frac{1}{4}(0 + w_2 x_2) + \frac{1}{4}(0 + 0) \\ &= \frac{1}{2}(w_1 x_1 + w_2 x_2)\end{aligned}$$

► thus $\underbrace{\frac{1}{2}a}_{\text{adjusted output at test time}} = \underbrace{\mathbb{E}[a]}_{\text{expected output at training time}}$

- bottom line: at test time, multiply by inclusion probability α

- weight scaling rule with usual inclusion probability (0.5)
 1. training time: dropout with prob 0.5
 2. test time: divide activations by 2
- inverted dropout: more common way to achieve the same result
 1. training time: dropout with prob 0.5 and then multiply activations by 2
 2. test time: no change (\rightarrow computationally more efficient)

```
"""
Vanilla Dropout: Not recommended implementation (see notes below)
"""

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

    def predict(X):
        # ensembled forward pass
        H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
        H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
        out = np.dot(W3, H2) + b3

    return out, H1, H2, U1, U2
```

drop in fwd pass

scale at test time

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)

    def predict(X):
        # ensembled forward pass
        H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
        H2 = np.maximum(0, np.dot(W2, H1) + b2)
        out = np.dot(W3, H2) + b3
```

test time: no change

(source: cs231n)

Dropout advantages

1. more effective than standard computationally inexpensive regularizers
 - e.g. weight decay, filter norm constraints, sparse activity regularization
 - ▶ may also be combined with other forms of regularization
2. very computationally cheap
 - ▶ both training and inference are efficient
3. works well with nearly any model
 - ▶ that uses a _____ representation and can be trained with SGD
 - e.g. feedforward nets, RBM, RNN

Cost and limitations

cost for using dropout:

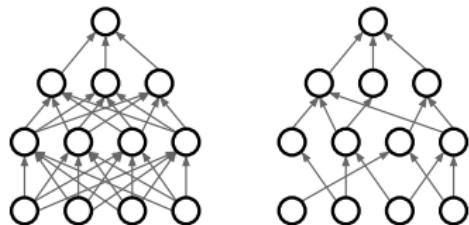
- much $\underbrace{\text{larger model}}$ and many more iterations of training algorithm
dropout = regularizer
 - ⇒ reduces the effective capacity of a model
 - ⇒ to offset this effect, we must increase model size

when is dropout less effective?

1. very large datasets:
 - ▶ regularization confers little reduction in generalization error
 - ▶ _____ cost may outweigh benefit of regularization
2. extremely few labeled training examples are available

Related work

- extensions/improvements
 - ▶ fast dropout
 - ▷ analytical approximation to the sum over all submodels
 - ▶ dropout boosting
 - ▷ dropout : bagging = dropout boosting : boosting
 - ▷ not for regularization but for jointly maximizing likelihood
- implicit ensemble methods (consider exponentially large ensembles)
 - ▶ stochastic pooling: builds ensembles of conv nets
 - ▶ DropConnect: sets a random subset of _____ to zero
 - c.f. dropout: randomly sets **hidden units** to zero



Why does dropout work?

- dropout training \Rightarrow a model cannot rely on any one feature
 \Rightarrow should spread out weights \Rightarrow shrink weights
- dropout trains an ensemble of models sharing hidden units
 \Rightarrow each hidden unit: regularized to encode a feature good in many contexts
- masking noise: applied to _____ units⁶
 - ▶ highly intelligent/adaptive destruction of input information

e.g. if a hidden unit h_i detects a face by finding eyebrows

- ▶ dropping h_i = erasing the info that there are eyebrows
- ▶ the model must learn another h_i that either
 - ▷ redundantly encodes the presence of eyebrows, or
 - ▷ detects the face by another feature (e.g. mouth)



⁶a large portion of the power of dropout arises from this fact

Batch normalization

- reparametrizes a model to introduce
 - ▶ both **additive** and **multiplicative** noise on hidden units
- c.f. dropout: only **multiplicative** noise on hidden units
- primary purpose: better optimization
 - ▶ but noise can have a regularizing effect
 - ⇒ sometimes makes dropout _____



Outline

Introduction

Dropout
Other Techniques

Theory of Regularization

Adversarial Training

Regularization Techniques

Norm Penalties

Summary

Early Stopping

Appendix

Ensemble Methods

Theme: immunizing a model by noise injection



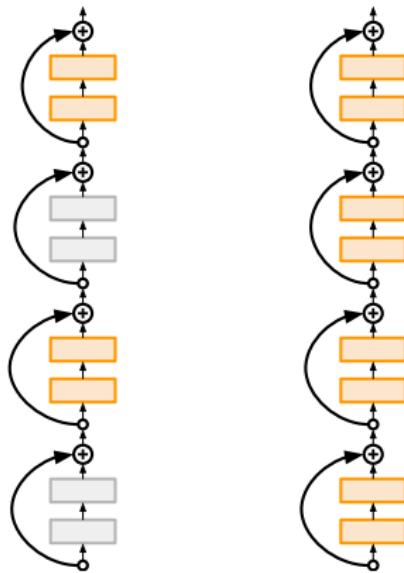
- various injection points exist:

noise added to	technique	reference
input	(1) data augmentation (2) imposing penalty on weight norm	sec 7.4 Bishop (1995a, b)
hidden units	dropout	sec 7.12
weights	(1) Bayesian inference over weights (2) stabilizing learned function	sec 7.5 sec 7.5
output targets	label smoothing	sec 7.5

- **label smoothing**: assume y is correct with prob $1 - \epsilon$ (ϵ : small constant)
 - ▶ regularizes a model based on a softmax with k output values
 - ▷ (hard) label 0 \rightarrow soft label $\frac{\epsilon}{k-1}$
 - ▷ (hard) label 1 \rightarrow soft label $1 - \epsilon$

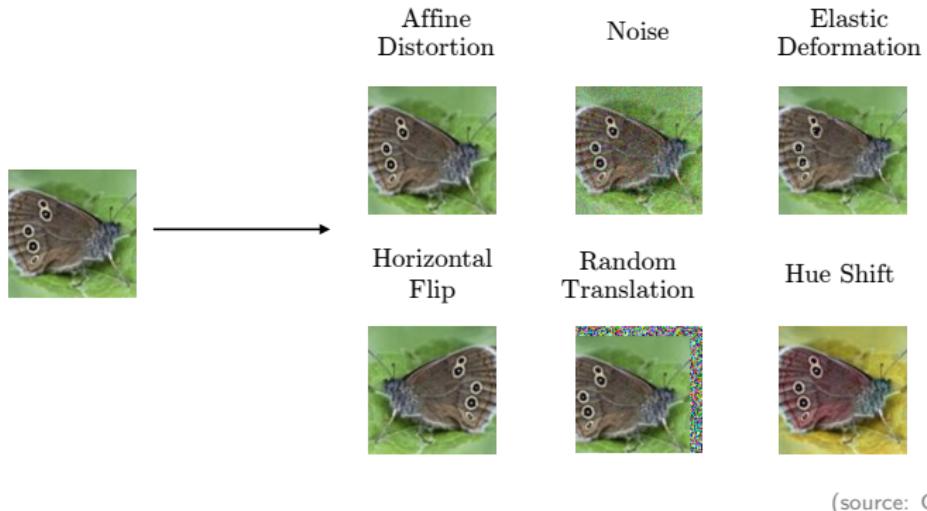
- another common pattern
 - ▶ training: add random noise
 - ▶ testing: _____ over the noise

- examples
 - ▶ dropout
 - ▶ DropConnect
 - ▶ data augmentation
 - ▶ batch norm
 - ▶ stochastic depth



Data augmentation

- create fake data and add it to training set



- difficulty of creating fake data: depends on specific ML tasks
 - ▶ successful cases reported: object recognition, speech recognition

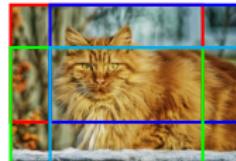
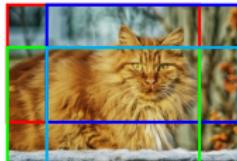
random crops and scales (ResNet):

- **training:** sample random crops/scales

1. pick random $L \in [256, 480]$
2. resize training image (short side = L)
3. sample random 224×224 patch

- **testing:** average results from a fixed set of crops

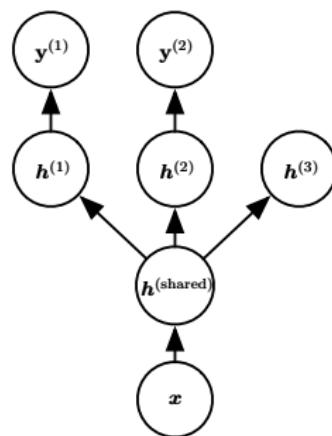
1. resize image at five scales: $\{224, 256, 384, 480, 640\}$
2. for each size, use ____ 224×224 crops: (center + 4 corners) \times flips



(source: cs231n, coursera)

Multi-task learning

- method #1: pool training examples out of several tasks
 - ▶ additional examples \approx soft constraints imposed on parameters
 - \Rightarrow put more pressure on parameters towards values that generalize well
 - \Rightarrow better generalization
 - method #2: share part of a model across tasks
 - ▶ shared part: more constrained towards good values
 - \Rightarrow better generalization
- e.g. $h^{(\text{shared})}$ captures common factors



Parameter tying and parameter sharing

- parameter tying
 - ▶ prior: similar tasks would have similar parameter values
 - ▶ leverage this information through regularization
- e.g. parameter norm penalty (for models A and B):

$$\Omega(\mathbf{w}^{(A)}, \mathbf{w}^{(B)}) = \|\mathbf{w}^{(A)} - \mathbf{w}^{(B)}\|_2^2$$

- parameter sharing (more popular)
 - ▶ make sets of parameters equal
 - ⇒ vast representational (*i.e.* fewer weights)/computational efficiency
- e.g. convolutional neural nets (ch 9)
 - ▷ dramatically lower # of parameters
 - ▷ significantly increase net sizes w/o a corresponding increase in data

Sparse representations

- place penalty on **activations of units** encouraging sparse representations
 - e.g. L^1 penalty on hidden representation: $\Omega(\mathbf{h}) = \|\mathbf{h}\|_1 = \sum_i |h_i|$
 - ⇒ implicitly imposes a complicated penalty on parameters
 - c.f. lasso $\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$: places penalty **directly** on **parameters**
- example: simplified view in linear regression

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix} \quad (\text{sparse } \underline{\hspace{10cm}})$$

$y \in \mathbb{R}^m \qquad \qquad A \in \mathbb{R}^{m \times n} \qquad \qquad x \in \mathbb{R}^n$

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix} \quad (\text{sparse } \underline{\hspace{10cm}})$$

$y \in \mathbb{R}^m \qquad \qquad B \in \mathbb{R}^{m \times n} \qquad \qquad h \in \mathbb{R}^n$

Outline

Introduction

Adversarial Training

Theory of Regularization

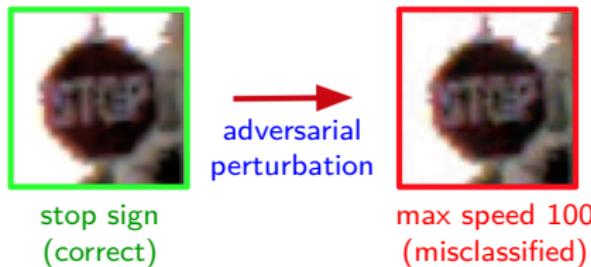
Summary

Regularization Techniques

Appendix

Adversarial examples

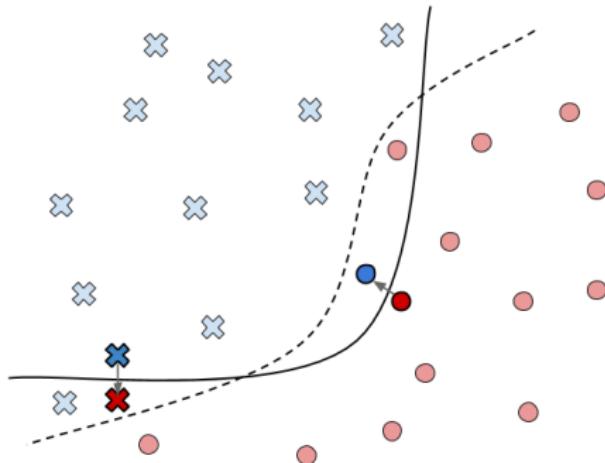
- adversarially perturbed examples
 - ▶ even neural nets with human-level performance make $\sim 100\%$ error



- how to create?
 - ▶ intentionally constructed using optimization
 - ▶ search for input x' near data point x s.t. their labels differ
i.e. $x' \approx x$ but $y(x') \neq y(x)$

- concept:

[▶ Link](#)



- | | | | |
|-------|----------------------------|---|-----------------------------|
| ----- | task decision boundary | ☒ | training points for class 1 |
| ——— | model decision boundary | ● | training points for class 2 |
| ☒ | test point for class 1 | ● | test point for class 2 |
| ✗ | adversarial ex for class 1 | ○ | adversarial ex for class 2 |

(source: Papernot and Goodfellow, 2016)

- adversarial example generation applied to GoogLeNet on ImageNet:

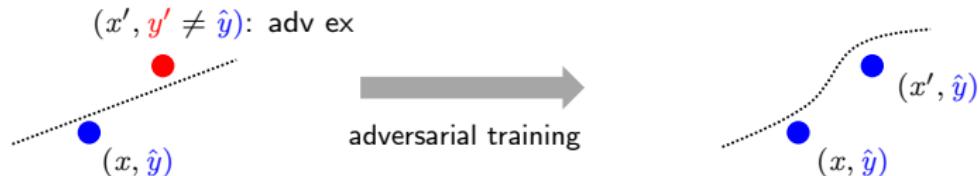
$$\begin{array}{ccc}
 \text{image } \mathbf{x} & + .007 \times & \text{image } \mathbf{x} + \\
 & \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y)) & \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y)) \\
 y = \text{"panda"} & \text{"nematode"} & \text{"gibbon"} \\
 \text{w/ 57.7\%} & \text{w/ 8.2\%} & \text{w/ 99.3 \%} \\
 \text{confidence} & \text{confidence} & \text{confidence}
 \end{array}$$

The diagram illustrates the generation of an adversarial example. It shows a original image of a panda (\mathbf{x}), a small noise matrix ($.007 \times \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$), and the resulting adversarial image ($\mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\theta, \mathbf{x}, y))$). Below the images, the original classification is "panda" with 57.7% confidence, and the adversarial example is classified as "nematode" with 8.2% confidence, while also being classified as "gibbon" with 99.3% confidence.

- change GoogLeNet's classification by adding imperceptibly small vector
 - its elements = _____ of the elements of the gradient of cost function wrt input
 - called “**fast gradient sign method**”

Adversarial training

- training with adversarial examples x'
 1. find inputs x' near the original inputs x
 2. train model to produce $y(x') = y(x)$



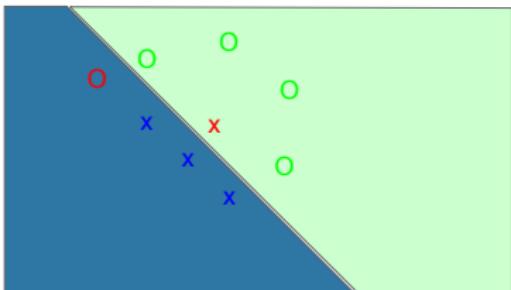
- ▶ based on the _____ assumption
- two desirable effects
 1. can reduce error on original test set
⇒ regularization effect
 2. can give correct labels to data points with incorrect labels
⇒ semi-supervised learning effect

Why do adversarial examples happen?

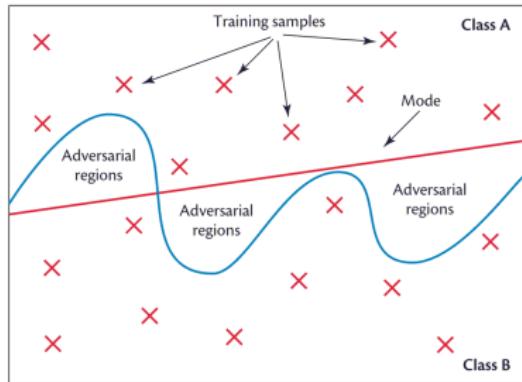
- one of the primary causes:
 - ▶ excessive linearity
- modern neural nets: built out of primarily _____ building blocks
 - ⇒ overall function they implement: sometimes highly linear
- unfortunately, a linear function
 - ▶ can change very rapidly if it has numerous inputs (*i.e.* high dim)
 - i.e.* if we change each input by ϵ
 - ▶ a linear function with weights w can change by as much as $\underbrace{\epsilon ||w||_1}_{\uparrow}$
very large if w is high-dimensional

- Goodfellow et al. (2014b):
 - ▶ adversarial examples are from _____ (rather than from overfitting)

Adversarial Examples from Excessive Linearity



(source: Goodfellow, 2016)



(source: McDaniel et al., 2016)

- adversarial training
 - ▶ discourages this highly sensitive locally linear behavior by
 - ▶ encouraging net to be **locally constant** in neighborhood of training data
 - = explicitly introducing a **local constancy prior** into supervised neural nets

Outline

Introduction

Adversarial Training

Theory of Regularization

Summary

Regularization Techniques

Appendix

Summary

- regularization: constraining a model not to fit noise (a must in practice)
 - ▶ effective when stochastic/deterministic noise is present
- ways to constrain a model (toward simpler/smooth direction)
 - ▶ use augmented error: better proxy for E_{gen} than E_{train}

$$E_{\text{aug}}(\mathbf{w}) = E_{\text{train}}(\mathbf{w}) + \underbrace{\frac{\lambda}{N} \Omega(\mathbf{w})}_{\text{penalty term}}$$

regularization parameter regularizer

- ▶ noise added to input/hidden units (dropout)/weights/output target
- ▶ multi-task learning, parameter sharing, early stopping, ensembling
- choosing $\Omega(\mathbf{w})$: heuristic (popular: l_1 lasso, l_2 ridge ["weight decay"])
- selecting regularization parameter λ : by validation ($\lambda = 0$ for wrong Ω)

Outline

Introduction

Adversarial Training

Theory of Regularization

Summary

Regularization Techniques

Appendix

Regularization and VC theory

regularization by
constrained-minimizing E_{in}

$$\min_{\mathbf{w}} E_{in}(\mathbf{w}) \text{ s.t. } \mathbf{w}^T \mathbf{w} \leq C$$

$$\begin{array}{c} C \downarrow \\ \iff \\ \text{gen} \uparrow \end{array}$$

VC guarantee of
constrained-minimizing E_{in}

$$E_{out}(\mathbf{w}) \leq E_{in}(\mathbf{w}) + \Omega(\mathcal{H}(C))$$

\Updownarrow C equivalent to some λ

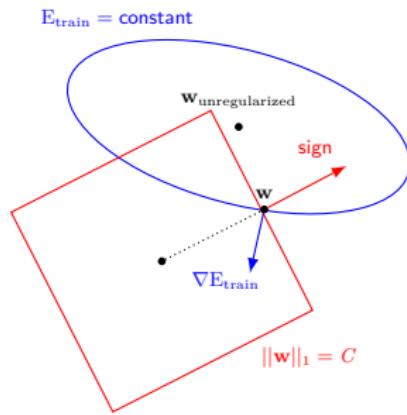
regularization by
minimizing E_{aug}

$$\min_{\mathbf{w}} \left\{ E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \frac{\lambda}{N} \mathbf{w}^T \mathbf{w} \right\}$$

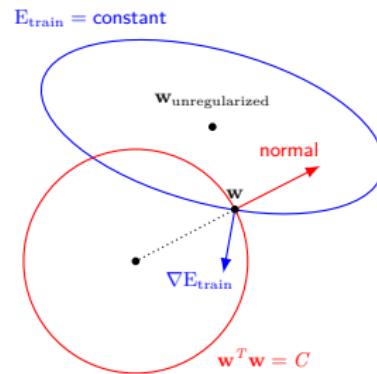
- minimizing E_{aug}
 - ▶ indirectly getting VC guarantee without confining to $\mathcal{H}(C)$

Comparison: l_1 and l_2 regularization

l_1 (lasso)



l_2 (ridge regression)



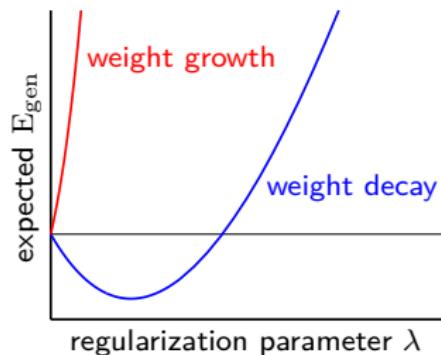
What if we pick a wrong regularizer?

- e.g. weight growth

- ▶ $\Omega(\mathbf{w}) = \sum_{q=0}^Q \frac{1}{w_q^2}$

- don't worry:

- ▶ we still have λ
- ▶ validation will set $\lambda = 0$



Ensemble example: a set of k regression models

- suppose: each model makes an error ϵ_i on each example
 - ▶ errors: drawn from a zero-mean multivariate normal distribution
 - ▷ variances $E[\epsilon_i^2] = v$, covariances $E[\epsilon_i \epsilon_j] = c$
- error (made by average prediction of all the ensemble models):

$$\frac{1}{k} \sum_i \epsilon_i$$

- expected squared error of the ensemble predictor:

$$\begin{aligned}\mathbb{E} \left[\left(\frac{1}{k} \sum_i \epsilon_i \right)^2 \right] &= \frac{1}{k^2} \mathbb{E} \left[\sum_i \left(\epsilon_i^2 + \sum_{j \neq i} \epsilon_i \epsilon_j \right) \right] \\ &= \frac{1}{k} v + \frac{k-1}{k} c\end{aligned}$$

- if errors are perfectly correlated and $c = v$
 - ▶ mean squared error reduces to v
 - ⇒ model averaging does not help at all
- if errors are perfectly uncorrelated and $c = 0$
 - ▶ expected squared error of the ensemble: only $\frac{1}{k}v$
 - ⇒ decreases linearly with the ensemble size
- in other words: on average, the ensemble will perform
 - ▶ at least as well as any of its members and
 - ▶ if the members make independent errors
- ⇒ the ensemble will perform significantly better than its members