165B
Machine Learning
Model Evaluation & Regularization

Lei Li (leili@cs)
UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

Resume in-person instruction

- starting on Jan 31, 2022

Recap

- Compute the gradient through Back-propagation algorithm
  - with forward pass and backward pass
  - backward pass is application of chain rule

Forward "Pass"

- Input: D dimensional vector $\mathbf{x} = [x_j,\ \ j = 1 \ldots D]$
- Set:
  - $D_0 = D,$ is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j,\ \ j = 1 \ldots D;\qquad y_0^{(k=1 \ldots N)} = \ x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$

    D_k is the size of the kth layer

    - $$z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$$
    - $$y_j^{(k)} = f_k\left(z_j^{(k)}\right)$$
- Output:
  - $Y = y_j^{(N)},\ \ j = 1..D_N$

Backward Pass

- Output layer (N) :
  - For $i = 1 \ldots D_N$
    - $$\frac{\partial \ell}{\partial z_i^{(N)}} = f_N'(z_i^{(N)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$
    - $$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}} \text{ for each j}$$

Called "Backpropagation" because the derivative of the loss is propagated "backwards" through the network

- For layer $k = N - 1$ $downto$

Very analogous to the forward pass:

  - For $i = 1 \ldots D_k$
    - $$\frac{\partial \ell}{\partial y_i^{(k-1)}} = \sum_j w_{ij}^{(k)} \frac{\partial \ell}{\partial z_j^{(k)}}$$

Backward weighted combination of next layer

    - $$\frac{\partial \ell}{\partial z_i^{(k)}} = f_k'(z_i^{(k)}) \frac{\partial \ell}{\partial y_i^{(k)}}$$

Backward equivalent of activation

    - $$\frac{\partial \ell}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial \ell}{\partial z_j^{(k)}} \text{ for each j}$$

Gradient Descent for FFN

```
learning rate eta.
1.set initial parameter θ ← θ_0
2.for epoch = 1 to maxEpoch or until
  converge:
3.   for each data (x, y) in D:
4.      compute forward y_hat = f(x; θ)
5.      compute gradient g = ∂err(y_hat, y)/∂θ using
  backpropagation
6.      total_g += g
7.   update θ = θ − eta ∗ total_g / num_sample
```

Step 1: $\theta \leftarrow \theta_0$

Step 5: $g = \dfrac{\partial \mathrm{err}(y_{hat}, y)}{\partial \theta}$

Model Evaluation

Training and Generalization

- Training error (=empirical risk): model prediction error on the training data
- Generalization error (= expected risk): model error on new unseen data over full population
- Example: practice a GRE exam with past exams
  - Doing well on past exams (training error) doesn't guarantee a good score on the future exam (generalization error)
  - Student A gets 0 error on past exams by rote learning
  - Student B understands the reasons for given answers

Validation Dataset and Test Dataset

- Validation dataset: a dataset used to evaluate the model performance
  - E.g. Take out 50% of the training data
  - Should not be mixed with the training data (#1 mistake)
- Test dataset: a dataset can be used once, e.g.
  - A future exam
  - The house sale price I bided
  - Dataset used in private leaderboard in Kaggle

Model Inference

- After train a model
- Given an input data x
- to compute the prediction for output y
- For regression:
  – just model output
- For classification:
  – $\hat{y} = \arg\max_i f(x)_i$
- Need to do inference for validation and testing

K-fold Cross-Validation

- Useful when insufficient data
- Algorithm:
  - Partition the training data into K parts
  - For i = 1, …, K
    - Use the i-th part as the validation set, the rest for training
    - Train the model using training set, and evaluate the performance on validation set.
  - Report the averaged the K validation errors
- Popular choices: K = 5 or 10

Underfitting
Overfitting

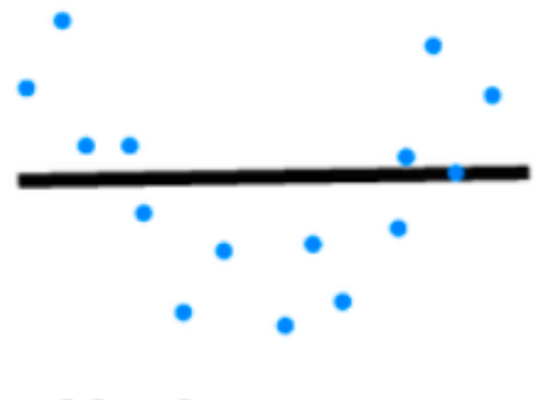
Image credit: hackernoon.com

Underfitting and Overfitting

		Data complexity	
		Simple	Complex
Model capacity	Low	ok	Underfitting
	High	Overfitting	ok

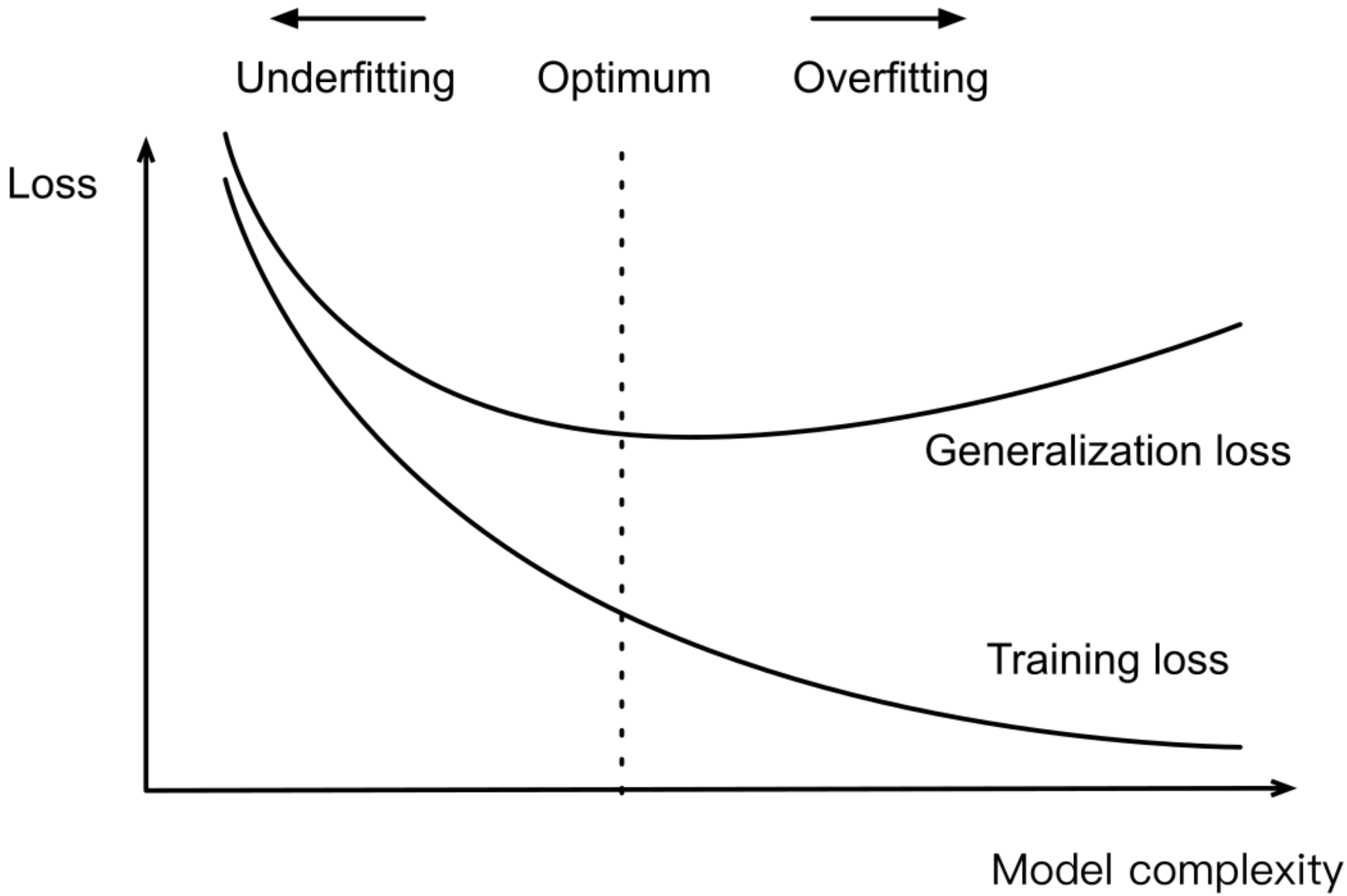
Model Capacity

- The ability to fit variety of functions
- Low capacity models struggles to fit training set
  - Underfitting



- High capacity models can memorize the training set
  - Overfitting

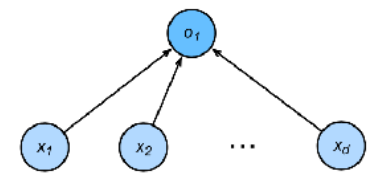
Influence of Model Complexity



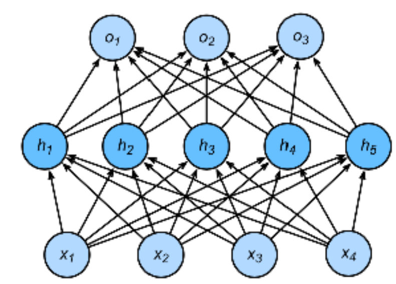
Estimate Model Capacity

- It's hard to compare complexity between different algorithms
  - e.g. tree vs neural network
- Given an algorithm family, two main factors matter:
  - The number of parameters
  - The values taken by each parameter

$$d + 1$$



$$(d + 1)m + (m + 1)k$$



VC Dimension

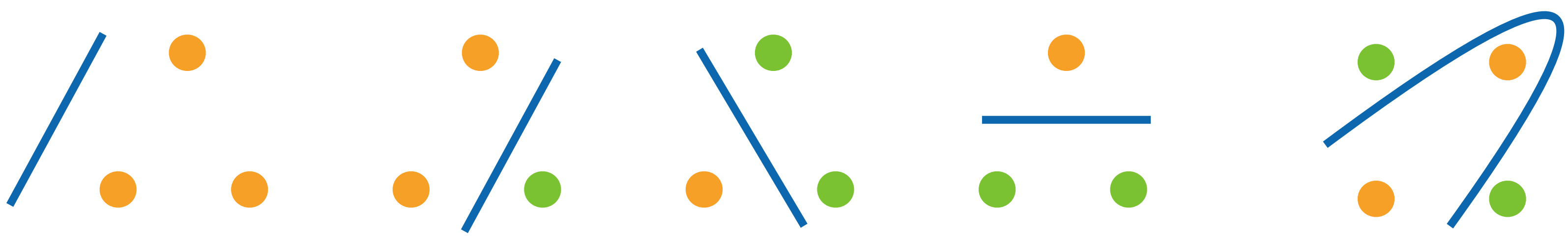
- A center topic in Statistic Learning Theory
- For a classification model, it's the size of the largest dataset, no matter how we assign labels, there exist a model to classify them perfectly

Vladimir **V**apnik

Alexey **C**hervonenkis

VC-Dimension for Classifiers

- 2-D perceptron: VCdim = 3
  - Can classify any 3 points, but not 4 points (xor)



- Perceptron with *N* parameters: VCdim = N
- Some Multilayer Perceptrons: VCdim = $O(N \log_2(N))$

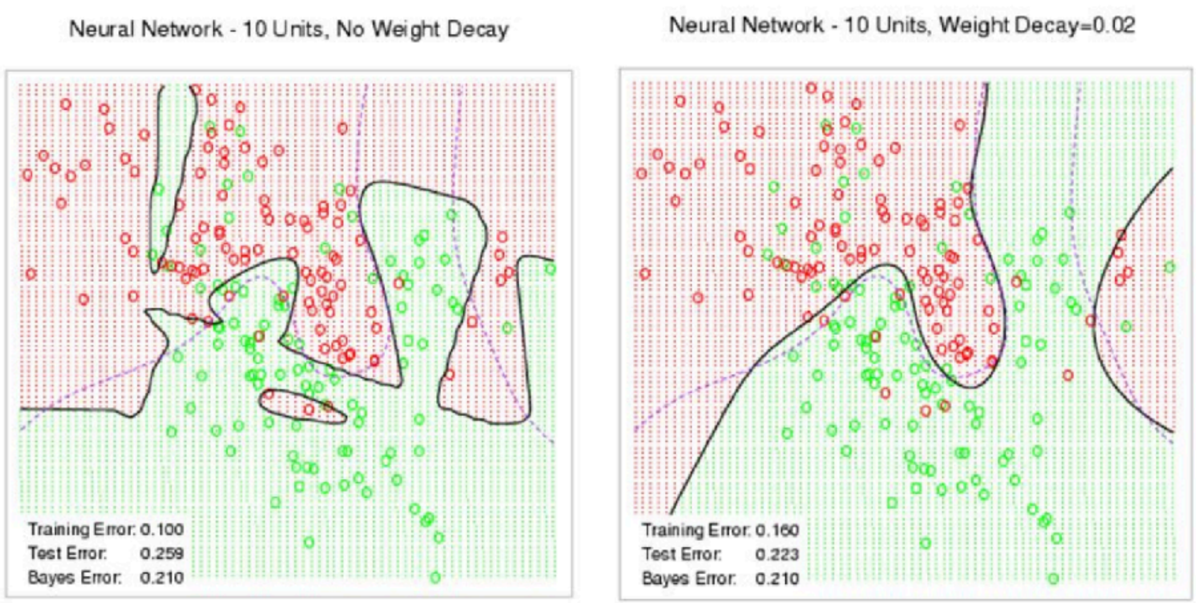
Usefulness of VC-Dimension

- Provides theoretical insights why a model works
  - Bound the gap between training error and generalization error
- Rarely used in practice with deep learning
  - The bounds are too loose
  - Difficulty to compute VC-dimension for deep neural networks
- Same for other statistic learning theory tools

Data Complexity

- Multiple factors matters
  - # of examples
  - # of features in each example
  - temporal/spacial structure
  - diversity/coverage

Regularization

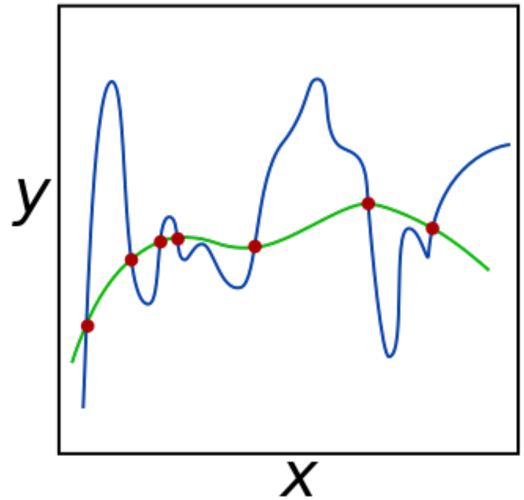


L_2 Regularization as Hard Constraint

- Reduce model complexity by limiting value range

$$\min \ \ell(\theta) \quad \text{subject to} \quad \|\theta\|^2 \leq \lambda$$

  - Often do not regularize bias *b*
    - Doing or not doing has little difference in practice
  - A small λ means more regularization



L_2 Regularization as Soft Constraint

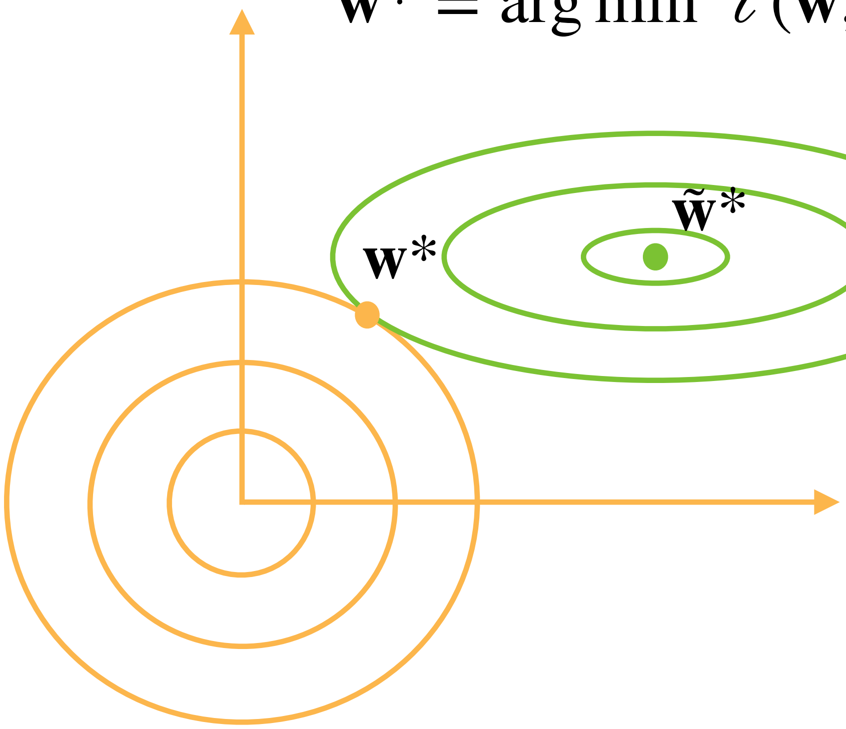
- Using Lagrangian multiplier method
- Minimizing the loss plus additional penalty

$$\min \quad \ell(\theta) + \frac{\lambda}{2}||\theta||^2$$

  - Hyper-parameter λ controls regularization importance
  - $\lambda = 0$: no effect
  - $\lambda \to \infty, \theta^* \to \mathbf{0}$

Illustrate the Effect on Optimal Solutions

$$\mathbf{w}^* = \arg\min\ \ell(\mathbf{w}, b) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



$$\tilde{\mathbf{w}}^* = \arg\min\ \ell(\mathbf{w}, b)$$

Update Rule - Weight Decay

- Compute the gradient

$$\frac{\partial}{\partial \theta}\left(\ell(\theta) + \frac{\lambda}{2}\|\theta\|^2\right) = \frac{\partial \ell(\theta)}{\partial \theta} + \lambda\theta$$

- Update weight at step *t*

$$\theta_{t+1} = (1 - \eta\lambda)\theta_t - \eta\frac{\partial \ell(\theta_t)}{\partial \theta_t}$$

backprop

  – Often $\eta\lambda < 1$, so also called weight decay in deep learning

Weight Decay in Pytorch

```
import torch

learning_rate = 1e-3
weight_decay = 1.0
optimizer =
torch.optim.SGD(model.parameters()
, lr=learning_rate,
weight_decay=weight_decay)
```

General Penalty

- Minimizing the loss plus additional penalty

$$\min \quad \ell(\theta) + R(\theta)$$

  - $\ell(\theta)$ is the original loss
  - $R(\theta)$ is penalty (or regularization term), not necessary smooth

L1 Regularization

- Minimizing the loss plus additional penalty

$$\min \ \ell(\theta) + \lambda |\theta|$$

  - $\ell(\theta)$ is the original loss
  - using L1 norm as penalty

L1 Update Rule - Soft Thresholding

- $\ell(\theta) + \lambda|\theta|$ is not always differentiable!
- Soft-threshold (Proximal operator):

$$S_\lambda(x) = \text{sign}(x)\max(0, |x| - \lambda) = \text{sign}(x)\text{Relu}(|x| - \lambda)$$

- Update weight at step *t*

$$\tilde{\theta}_t = \theta_t - \eta \frac{\partial \ell(\theta_t)}{\partial \theta_t}$$

$$\theta_{t+1} = S_\lambda(\tilde{\theta})$$

- Also known as Proximal Gradient Descent

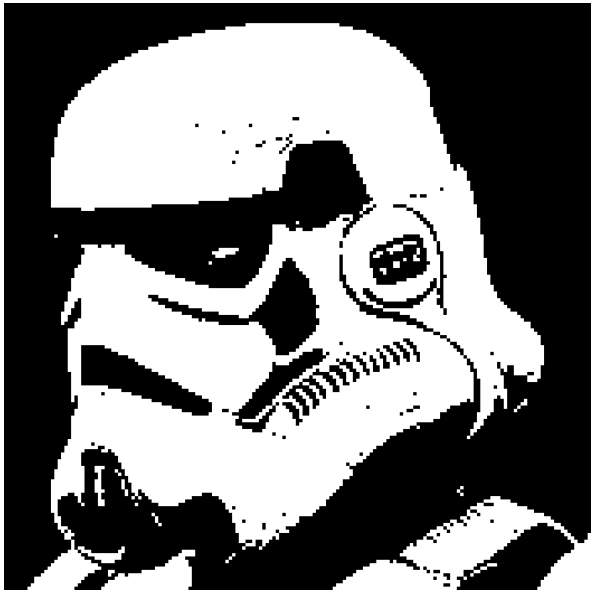
Effects of L1 and L2 Regularization

- L1 Regularization
  - will make parameters sparse (many parameters will be zeros)
  - could be useful for model pruning
- L2 Regularization
  - will make the parameter shrink towards 0, but not necessary 0.

Dropout

Motivation

- A good model should be robust under modest changes in the input
  - Dropout: inject noises into internal layers (simulating the noise)



Add Noise without Bias

- Add noise into ***x*** to get ***x'***, we hope

$$\mathbf{E}[\mathbf{x}'] = \mathbf{x}$$

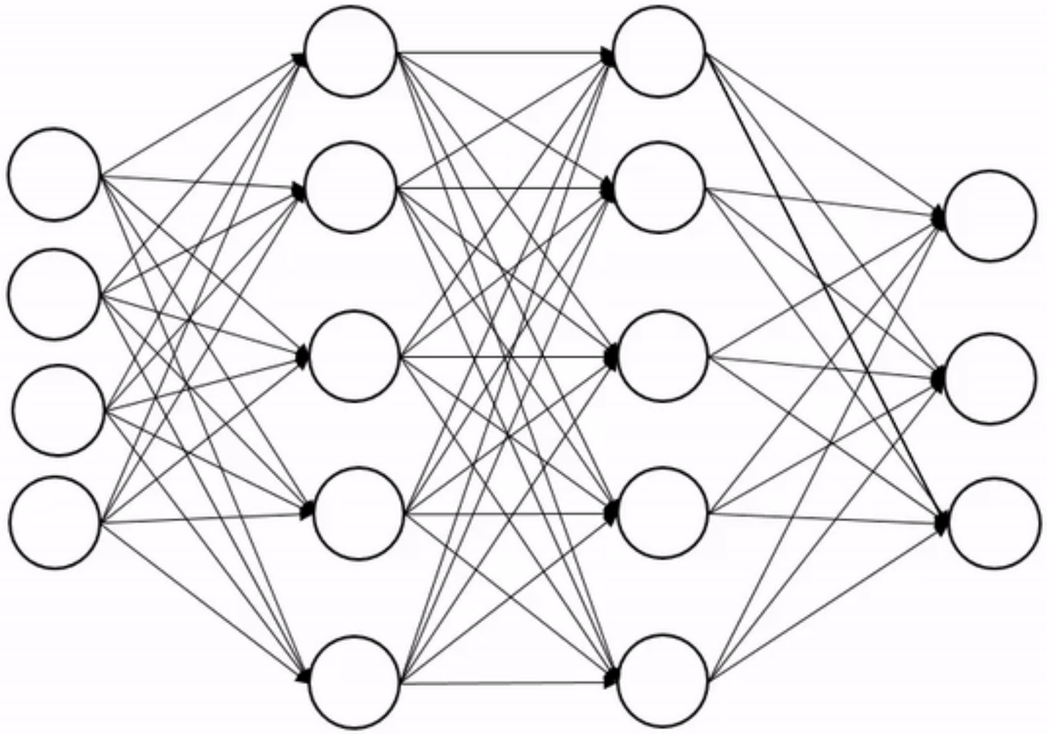
- Dropout perturbs each element by

$$x_i' = \begin{cases} 0 & \text{with probablity } p \\ \dfrac{x_i}{1-p} & \text{otherise} \end{cases}$$

Apply Dropout

- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$



Dropout in Training and Inference

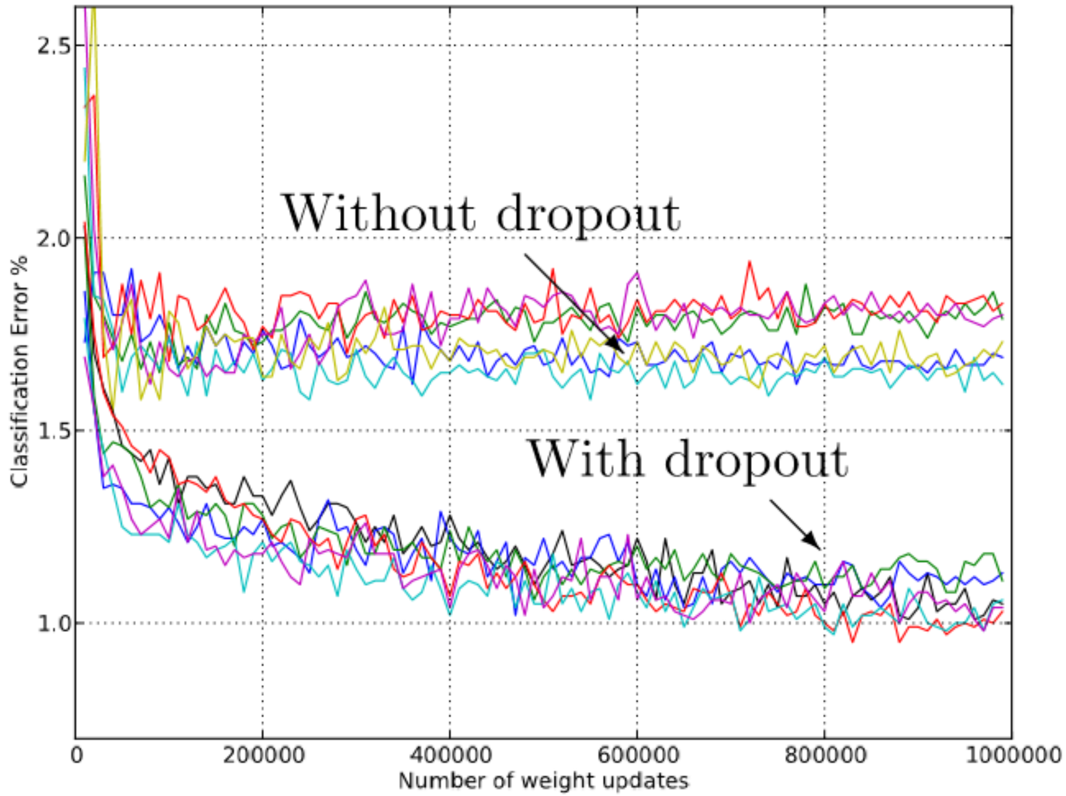
- Dropout is only used in training

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

- No dropout is applied during inference!
- Pytorch Layer:

```
torch.nn.Dropout(p=0.5)
```

Dropout: Typical results



- From Srivastava et al., 2013. Test error for different architectures on MNIST with and without dropout
  - 2-4 hidden layers with 1024-2048 units

Recap

- Generalization error: the expected error on unseen data (general population)
- Minimizing training loss does not always lead to minimizing the generalization error
- Under-fitting: model does not have adequate capacity ==> increase model size, or choose a more complex model
- Over-fitting: validation loss does not decrease while training loss still does
- Regularization
  - L1 ==> more sparse parameters
  - L2/Weight decay ==> shrink parameters
  - Dropout, equivalent to L2, but as a network Layer

Numerical Stability

Gradients for Neural Networks

- Consider a network with *d* layers

$$\mathbf{h}^t = f_t(\mathbf{h}^{t-1}) \quad \text{and} \quad y = \ell \circ f_d \circ \ldots \circ f_1(\mathbf{x})$$

- Compute the gradient of the loss ℓ w.r.t. $\mathbf{W}_t$

$$\frac{\partial \ell}{\partial \mathbf{W}^t} = \frac{\partial \ell}{\partial \mathbf{h}^d} \underbrace{\frac{\partial \mathbf{h}^d}{\partial \mathbf{h}^{d-1}} \ldots \frac{\partial \mathbf{h}^{t+1}}{\partial \mathbf{h}^t}}_{\text{Multiplication of } d\text{-}t \text{ matrices}} \frac{\partial \mathbf{h}^t}{\partial \mathbf{W}^t}$$

Two Issues for Deep Neural Networks

- Two common issues with $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}}$

Gradient Exploding

$1.5^{100} \approx 4 \times 10^{17}$

Gradient Vanishing

$0.8^{100} \approx 2 \times 10^{-10}$

Example: FFN

- Assume FFN (without bias for simplicity)

$$f_t(\mathbf{h}^{t-1}) = \sigma(\mathbf{W}^t \mathbf{h}^{t-1})$$ σ is the activation function

$$\frac{\partial \mathbf{h}^t}{\partial \mathbf{h}^{t-1}} = \text{diag}\left(\sigma'(\mathbf{W}^t \mathbf{h}^{t-1})\right)(W^t)^T$$ σ' is the gradient function of σ

$$\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$$

Gradient Exploding

- Use ReLU as the activation function

$$\sigma(x) = \max(0,x) \quad \text{and} \quad \sigma'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Elements of $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ may from $\prod_{i=t}^{d-1} (W^i)^T$
  - Leads to large values when *d-t* is large

$$1.5^{100} \approx 4 \times 10^{17}$$

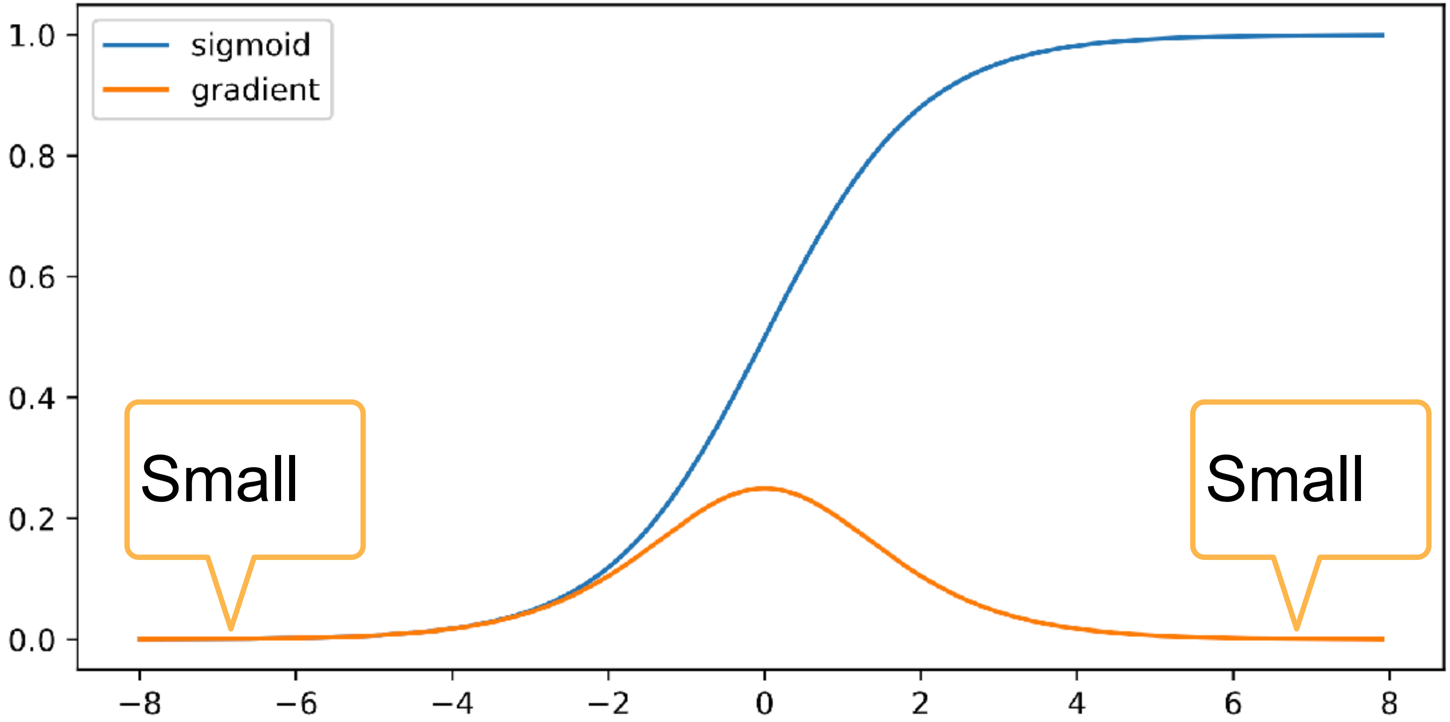
Issues with Gradient Exploding

- Value out of range: infinity value
  - Severe for using 16-bit floating points
    - Range: 6E-5 ~ 6E4
- Sensitive to learning rate (LR)
  - Not small enough LR -> large weights -> larger gradients
  - Too small LR -> No progress
  - May need to change LR dramatically during training

Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \sigma'(x) = \sigma(x)(1-\sigma(x))$$



Gradient Exploding

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})\right)(W^i)^T$ are products of *d-t* small values

$$0.8^{100} \approx 2 \times 10^{-10}$$

Issues with Gradient Vanishing

- Gradients with value 0
  - Severe with 16-bit floating points
- No progress in training
  - No matter how to choose learning rate
- Severe with bottom layers
  - Only top layers are well trained
  - No benefit to make networks deeper

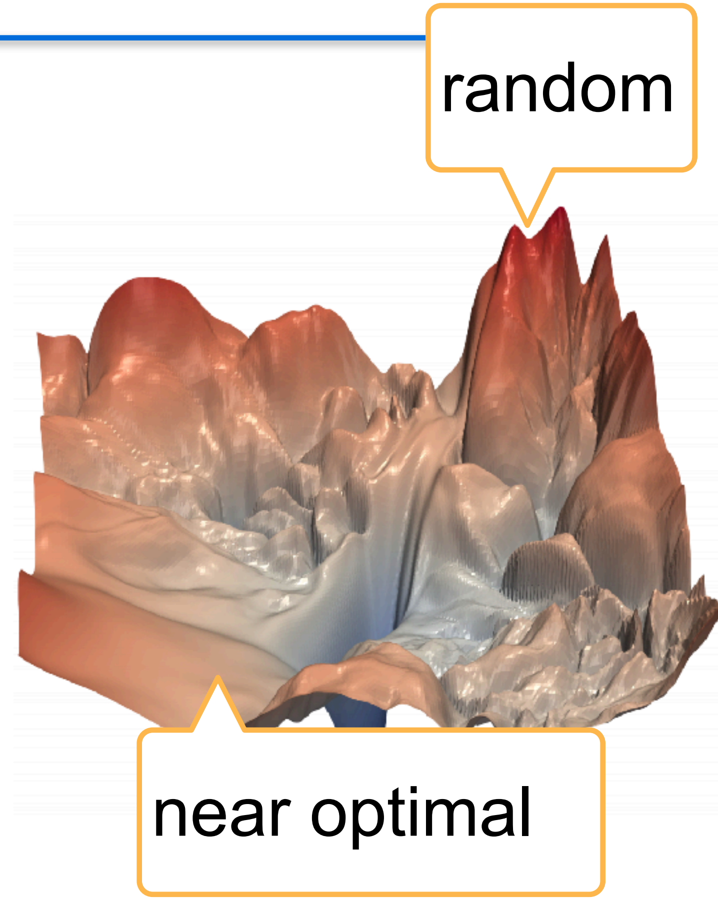
Stabilize Training

Stabilize Training

- Goal: make sure gradient values are in a proper range
  - E.g. in [1e-6, 1e3]
- Multiplication -> plus
  - ResNet, LSTM (later lecture)
- Normalize
  - Gradient clipping
  - Batch Normalization / Layer Normalization (later)
- Proper weight initialization and activation functions

Weight Initialization

- Initialize weights with random values in a proper range
- The beginning of training easily suffers to numerical instability
  - The surface far away from an optimal can be complex
  - Near optimal may be flatter
- Initializing according to $\mathcal{N}(0,\ 0.01)$ works well for small networks, but not guarantee for deep neural networks



Constant Variance for each Layer

- Treat both layer outputs and gradients are random variables
- Make the mean and variance for each layer's output are same, similar for gradients

Forward

$$\mathbb{E}[h_i^t] = 0$$

$$\mathrm{Var}[h_i^t] = a$$

Backward

$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^t}\right] = 0 \quad \mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^t}\right] = b \qquad \forall i, t$$

a and *b* are constants

Example: FFN

- Assumptions $\mathbb{E}[w_{i,j}^t] = 0, \ \mathrm{Var}[w_{i,j}^t] = \gamma_t$
  - i.i.d $w_{i,j}^t$,
  - h_i^{t-1} is independent to $w_{i,j}^t$
  - identity activation: $\mathbf{h}^t = \mathbf{W}^t \mathbf{h}^{t-1}$ with $\mathbf{W}^t \in \mathbb{R}^{n_t \times n_{t-1}}$

$$\mathbb{E}[h_i^t] = \mathbb{E}\left[\sum_j w_{i,j}^t h_j^{t-1}\right] = \sum_j \mathbb{E}[w_{i,j}^t]\mathbb{E}[h_j^{t-1}] = 0$$

Forward Variance

$$\mathrm{Var}[h_i^t] = \mathbb{E}[(h_i^t)^2] - \mathbb{E}[h_i^t]^2 = \mathbb{E}\left[\left(\sum_j w_{i,j}^t h_j^{t-1}\right)^2\right]$$

$$= \mathbb{E}\left[\sum_j \left(w_{i,j}^t\right)^2 \left(h_j^{t-1}\right)^2 + \sum_{j \neq k} w_{i,j}^t w_{i,k}^t h_j^{t-1} h_k^{t-1}\right]$$

$$= \sum_j \mathbb{E}\left[\left(w_{i,j}^t\right)^2\right] \mathbb{E}\left[\left(h_j^{t-1}\right)^2\right]$$

$$= \sum_j \mathrm{Var}[w_{i,j}^t]\mathrm{Var}[h_j^{t-1}] = n_{t-1}\gamma_t \mathrm{Var}[h_j^{t-1}] \Rightarrow n_{t-1}\gamma_t = 1$$

n_{t-1} is the number of units in t-1 layer

Backward Mean and Variance

- Apply forward analysis as well

$$\frac{\partial \ell}{\partial \mathbf{h}^{t-1}} = \frac{\partial \ell}{\partial \mathbf{h}^{t}} \mathbf{W}^{t}$$ leads to $$\left(\frac{\partial \ell}{\partial \mathbf{h}^{t-1}}\right)^T = (W^t)^T \left(\frac{\partial \ell}{\partial \mathbf{h}^{t}}\right)^T$$

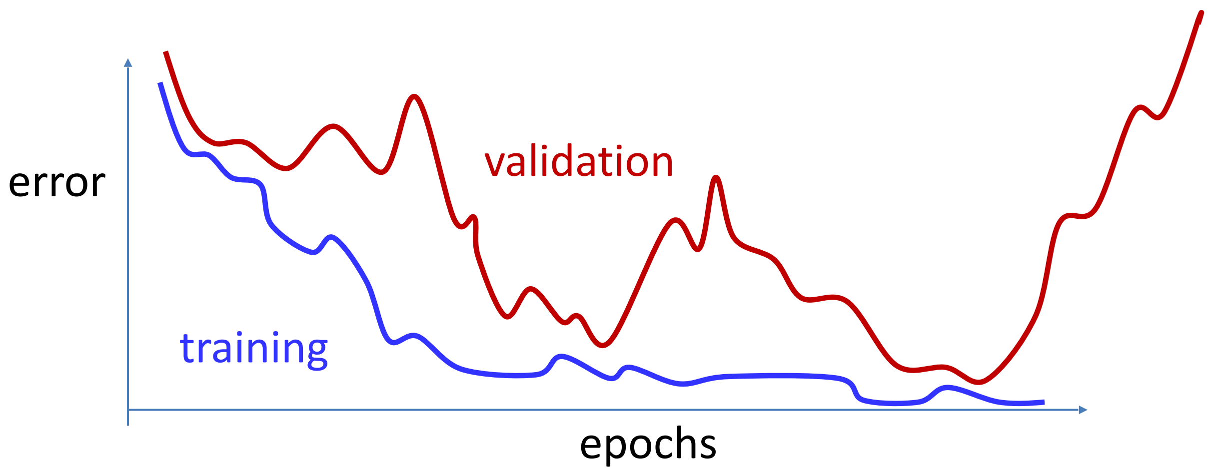
$$\mathbb{E}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = 0$$

$$\mathrm{Var}\left[\frac{\partial \ell}{\partial h_i^{t-1}}\right] = n_t \gamma_t \mathrm{Var}\left[\frac{\partial \ell}{\partial h_j^{t}}\right] \Rightarrow n_t \gamma_t = 1$$

Xavier Initialization

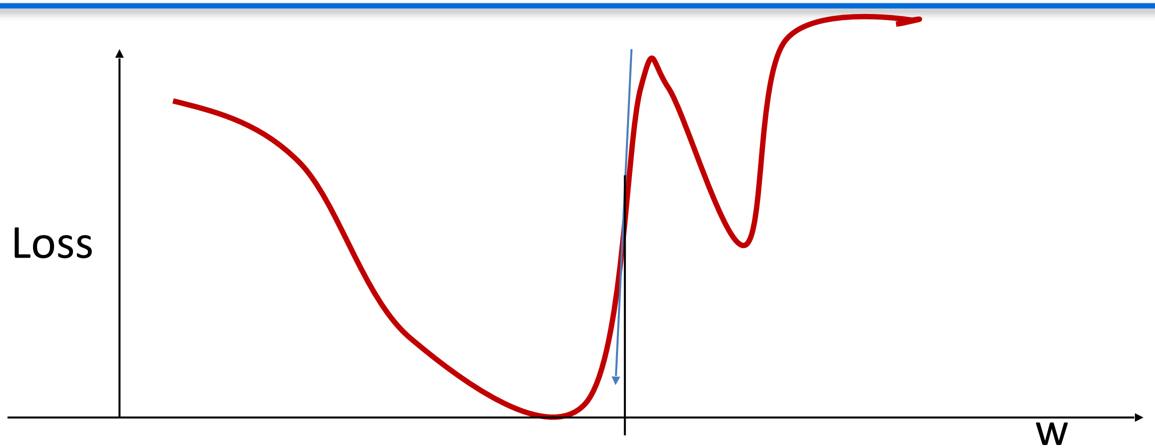
- Conflict goal to satisfies both $n_{t-1}\gamma_t = 1$ and $n_t\gamma_t = 1$
- Xavier $\gamma_t(n_{t-1} + n_t)/2 = 1 \quad \rightarrow \quad \gamma_t = 2/(n_{t-1} + n_t)$
  - Normal distribution $\mathcal{N}\left(0, \sqrt{2/(n_{t-1} + n_t)}\right)$
  - Uniform distribution $\mathcal{U}\left(-\sqrt{6/(n_{t-1} + n_t)}, \sqrt{6/(n_{t-1} + n_t)}\right)$
    - Variance of $\mathcal{U}[-a, a]$ is $a^2/3$
- Adaptive to weight shape, especially when n_t varies

Other heuristics: Early stopping



- Continued training can result in over fitting to training data
  - Track performance on a held-out validation set
  - Apply one of several early-stopping criterion to terminate training when performance on validation set degrades significantly

Additional heuristics: Gradient clipping



- Often the derivative will be too high
  - When the divergence has a steep slope
  - This can result in instability
- **Gradient clipping**: set a ceiling on derivative value

$$if\ \partial_w D > \theta\ then\ \partial_w D = \theta$$

  - Typical θ value is 5

– Can be easily set in pytorch/tensorflow

Recap

- Numerical issues in training
  - gradient explosion
  - gradient vanishing
- Proper initialization of parameters

Next Up

- Convolutional Neural Networks
- Visual perception:
  - Image classification
  - Object recognition
  - Face detection