Deep Learning and Applications

DSA 5204 • Lecture 6

Dr Low Yi Rui (Aaron)
Department of Mathematics

NUS
National University of Singapore

First Half of this Course

Basic deep learning

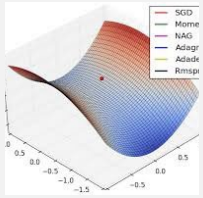
- **Architectures**
  - Shallow/deep fully connected neural networks (FCNN)
  - Convolutional neural networks (CNN)
  - Recurrent neural networks (RNN)
- **Training algorithms**
  - GD
  - SGD
  - SGD with momentum

Second Half of this Course

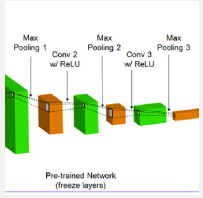
We will learn

- How to improve performance
- Interesting applications outside of supervised learning

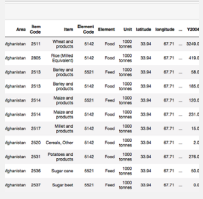
Ways to Improve Performance



Training Methods



Model Architectures



Data

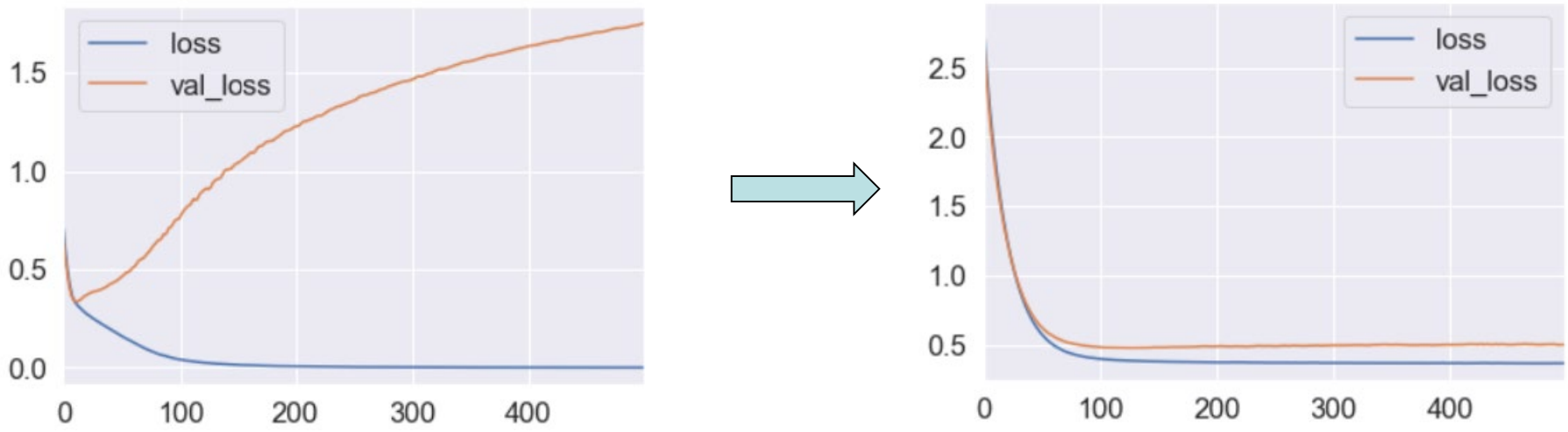
Parameter Norm Regularization (Training)

Regularization

Regularization is a general technique:

"any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

In other words, we trade some bias to lower variance



Parameter Norm Penalties

Empirical risk minimization

$$\widehat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} R(\boldsymbol{\theta}; X, \boldsymbol{y})$$

A parameter norm penalty modifies this by minimizing instead

$$\tilde{R}(\boldsymbol{\theta}; X, \boldsymbol{y}) = R(\boldsymbol{\theta}; X, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

where

- Ω is called a **regularizer**
- $\alpha \geq 0$ is the **strength** or coefficient of regularization

L^2 Regularization

The simplest type of parameter norm regularization is the L^2 (or ℓ^2) regularization

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|^2 = \frac{1}{2}\sum_i \theta_i^2$$

so that

$$\tilde{R}(\boldsymbol{\theta}; X, \boldsymbol{y}) = R(\boldsymbol{\theta}; X, \boldsymbol{y}) + \frac{1}{2}\alpha\|\boldsymbol{\theta}\|^2$$

And

$$\widehat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}}\, \tilde{R}(\boldsymbol{\theta}; X, \boldsymbol{y})$$

Example: Linear Regression

L^2-regularized linear model

$$R(\boldsymbol{w}; X, \boldsymbol{y}) = \frac{1}{2}\|X\boldsymbol{w} - \boldsymbol{y}\|^2 + \underbrace{\frac{1}{2}\alpha\|\boldsymbol{w}\|^2}_{\alpha\Omega(\boldsymbol{w})}$$

This is called ridge regression or Tikhonov regularization

Solution:

$$\widehat{\boldsymbol{w}} = (X^T X + \alpha I)^{-1} X^T \boldsymbol{y}$$

Compare with unregularized version:

$$H = X^T X \rightarrow \widetilde{H} = H + \alpha I = X^T X + \alpha I$$

What is the effect of the regularization?

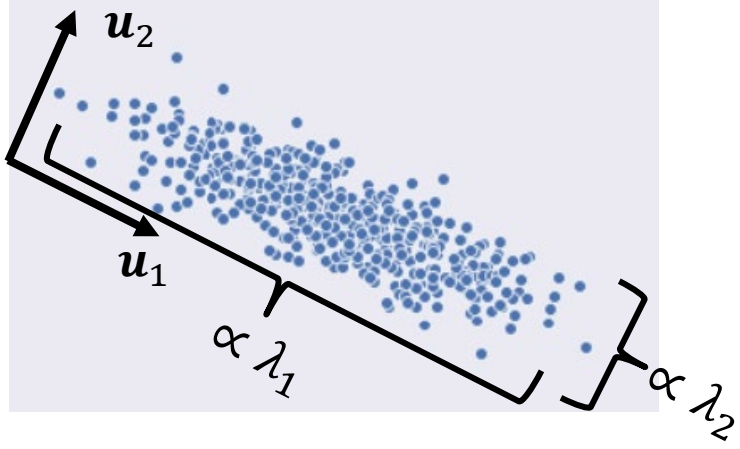
Recall that

$$\frac{1}{m}H_{ij} = \frac{1}{m}(X^T X)_{ij} = \frac{1}{m}\sum_k x_i^{(k)} x_j^{(k)}$$

is the covariance matrix of the data.

Then, the principal directions are the eigenvectors $\{\boldsymbol{u}_i\}$ of H, whose eigenvalues $\{\lambda_i\}$ are the scaled variances along those directions

(Review PCA if this does not make sense)



Now, let $\{u_1, \dots, u_m\}$ orthonormal eigenvectors of H with eigenvalues $\{\lambda_1, \dots, \lambda_m\}$

$$H\boldsymbol{u}_i = \lambda_i \boldsymbol{u}_i$$

Then, we can expand

$$X^T \boldsymbol{y} = \sum_i \beta_i \boldsymbol{u}_i$$

Then for the unregularized problem:

$$\hat{\boldsymbol{w}} = H^{-1} X^T \boldsymbol{y} = \sum_i \beta_i H^{-1} \boldsymbol{u}_i = \sum_i \frac{\beta_i}{\lambda_i} \boldsymbol{u}_i$$

But, for the regularized problem:

$$\hat{\boldsymbol{w}} = \tilde{H}^{-1} X^T \boldsymbol{y} = \sum_i \beta_i \tilde{H}^{-1} \boldsymbol{u}_i = \sum_i \frac{\beta_i}{\lambda_i + \alpha} \boldsymbol{u}_i$$

Unregularized

$$\hat{\boldsymbol{w}} = \sum_i \frac{\beta_i}{\lambda_i} \boldsymbol{u}_i$$

Regularized

$$\hat{\boldsymbol{w}} = \sum_i \frac{\beta_i}{\lambda_i + \alpha} \boldsymbol{u}_i$$

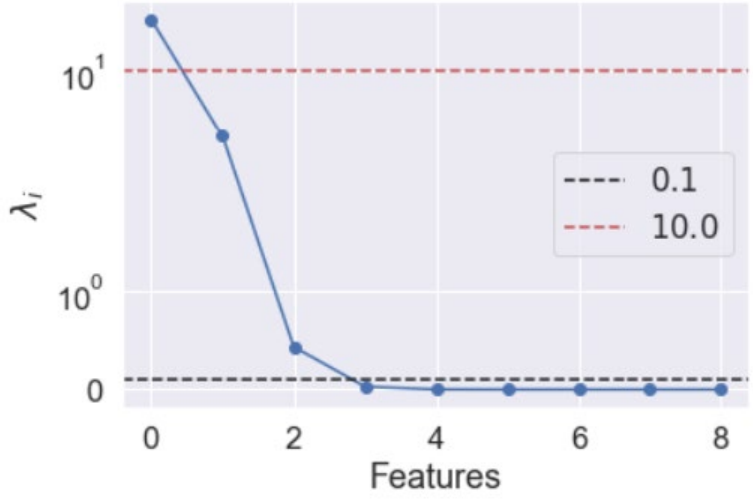
Effect of L^2 regularization

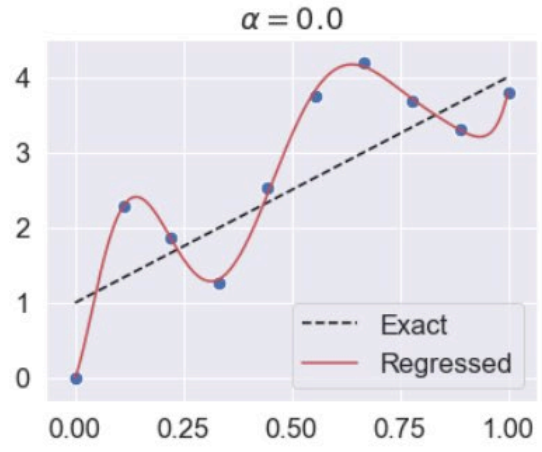
- If $\lambda_i \gg \alpha$, then $\lambda_i + \alpha \approx \lambda_i$, almost no change
- If $\lambda_i \ll \alpha$, then $\lambda_i + \alpha \approx \alpha$, the effect of the data is removed
- In other words, **the regularization removes the influence of the data in the direction of small variance!**

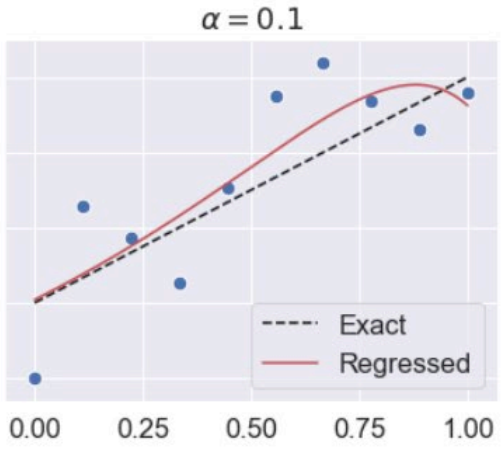
Numerical Example

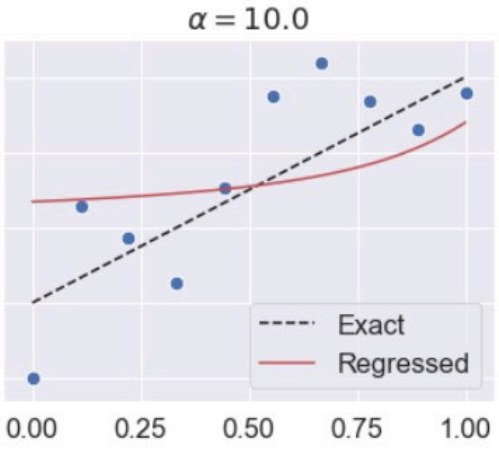
Polynomial Regression: 10 datapoints, fit with degree 8 polynomial features.

Eigenvalues









Some Additional Remarks on L^2 Regularization

L^2 regularization is sometimes also called weight decay

This is because of the gradient descent algorithm:

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \boldsymbol{\theta}_k - \epsilon\nabla\tilde{R}(\boldsymbol{\theta}_k) \\ &= \boldsymbol{\theta}_k - \epsilon\nabla R(\boldsymbol{\theta}_k) - \underbrace{\epsilon\alpha\boldsymbol{\theta}_k}_{\text{decay}}\end{aligned}$$

For example, if $R = \text{Constant}$ then the weight simply decays

$$\boldsymbol{\theta}_{k+1} = (1 - \epsilon\alpha)\boldsymbol{\theta}_k$$

L^1 Regularization

Another type of parameter norm regularization is the L^1 regularization

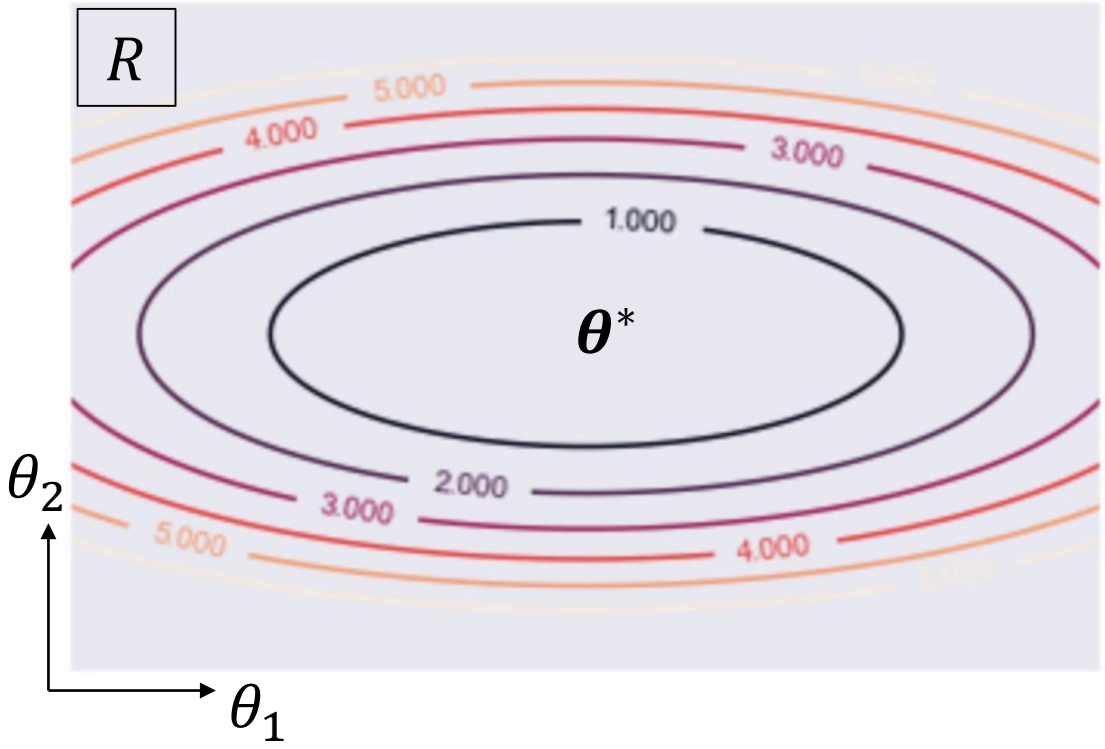
$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i|$$

So, what is the difference between L^1 and L^2 regularizations?

Example: L^1 vs L^2 Regularization

Consider minimizing a diagonal quadratic loss function

$$R(\boldsymbol{\theta}) = \frac{1}{2}\sum_i \lambda_i(\theta_i - \theta_i^*)^2 \quad \boldsymbol{\theta}^* \in \mathbb{R}^m \text{ (fixed)}$$



Solution for L^2 regularized case:

$$\nabla\tilde{R}(\hat{\theta}) = 0 \Rightarrow \lambda_i(\hat{\theta}_i - \theta_i^*) + \alpha\hat{\theta}_i = 0 \Rightarrow \hat{\theta}_i = \frac{\lambda_i}{\lambda_i + \alpha}\theta_i^*$$

Solution for L^1 regularized case:

$$\tilde{R}(\boldsymbol{\theta}) = \sum_i R_i(\theta_i) \quad \text{with} \quad R_i(\theta_i) = \frac{1}{2}\lambda_i(\theta_i - \theta_i^*)^2 + \alpha|\theta_i|$$

Thus, we can separately minimize each component of θ, yielding

$$\hat{\theta}_i = \begin{cases} \theta_i^* - \text{Sign}(\theta_i^*)\dfrac{\alpha}{\lambda_i} & |\theta_i^*| > \dfrac{\alpha}{\lambda_i} \\ 0 & |\theta_i^*| \leq \dfrac{\alpha}{\lambda_i} \end{cases}$$

Unregularized	L^2-Regularized	L^1-Regularized
$\hat{\theta}_i = \theta_i^*$	$\hat{\theta}_i = \frac{\lambda_i}{\lambda_i + \alpha}\theta_i^*$	$\hat{\theta}_i = \begin{cases} \theta_i^* - \text{Sign}(\theta_i^*)\frac{\alpha}{\lambda_i} & \lvert\theta_i^*\rvert > \frac{\alpha}{\lambda_i} \\ 0 & \lvert\theta_i^*\rvert \leq \frac{\alpha}{\lambda_i} \end{cases}$

Comparison between L^2 and L^1 regularization

- Common to both
  - **As α increases from 0 to ∞, the solution $\widehat{\theta}$ is pushed from θ^* to 0**
  - **Higher variance components of the data (large λ_i) are less affected than low variance components (small λ_i)**
- Differences
  - **L^1 is "hard": if λ_i is small enough, the corresponding weight is set to exactly 0.**
  - **In other words, L^1 penalty induces sparsity**

Remarks on L^1 Regularization

- **The sparsity inducing property can be used for feature selection. This is the underlying principle behind LASSO (least absolute shrinkage and selection operator)**
- **Regularization and Priors: from a Bayesian MAP viewpoint, parameter-norm regularization is equivalent to placing a prior on the distribution**
  - L^2: Gaussian Prior [PDF $\propto \exp(-\alpha\|\boldsymbol{\theta}\|^2/2)$]
  - L^1: Laplace Prior [PDF $\propto \exp(-\alpha\|\boldsymbol{\theta}\|_1)$]

Remarks on Regularizing Neural Networks

So far, our examples has been linear models, but regularization behaves the same way on nonlinear neural networks

Some special things to take note

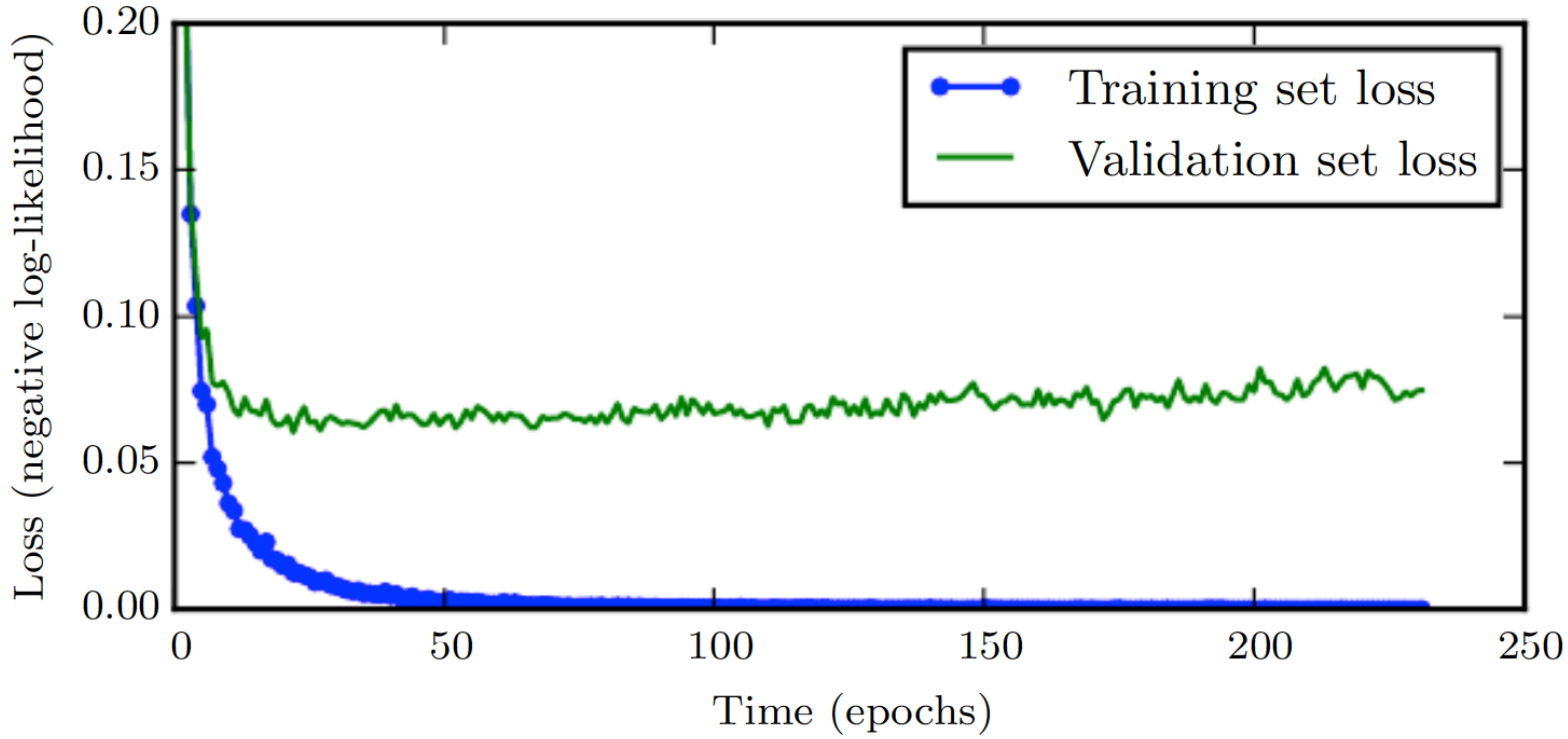
- NN layer: $\sigma(W\boldsymbol{x} + \boldsymbol{b})$
  Usually, we only regularize the weights W but not the bias $\boldsymbol{b}$.
  (bias adjusts the effect of nonlinearity, need not be small)
- We may use a different strength of regularization for each layer

Demo: Parameter Norm Penalties

Early Stopping (Training)

Training vs Validation Loss

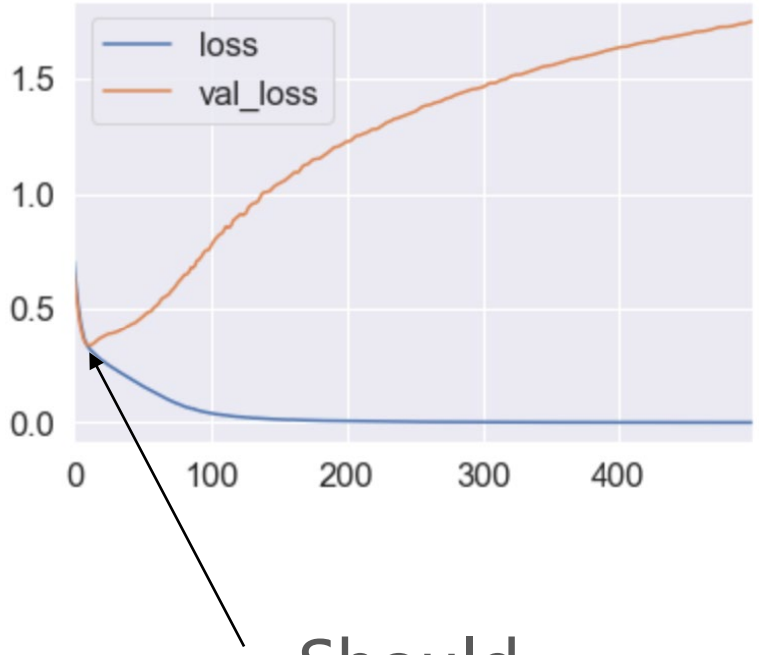
In practice, we often observe the training loss decreasing whereas the validation error stays the same (or increases)



Early Stopping

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*



Should stop here!

Early Stopping Variants

Early stopping requires a validation set, i.e. not all training data is trained.

To resolve this, we can use some variants

- Strategy 1: retrain the dataset
- Strategy 2: continue training with full dataset after early stopping

Variant I

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This returns i^*, the optimal number of steps.
Set $\boldsymbol{\theta}$ to random values again.
Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for i^* steps.

Variant II

Algorithm 7.3 Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.

Let $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ be the training set.
Split $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ into $(\boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{X}^{(\text{valid})})$ and $(\boldsymbol{y}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{valid})})$ respectively.
Run early stopping (algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\boldsymbol{X}^{(\text{subtrain})}$ and $\boldsymbol{y}^{(\text{subtrain})}$ for training data and $\boldsymbol{X}^{(\text{valid})}$ and $\boldsymbol{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.
$\epsilon \leftarrow J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{subtrain})}, \boldsymbol{y}^{(\text{subtrain})})$
while $J(\boldsymbol{\theta}, \boldsymbol{X}^{(\text{valid})}, \boldsymbol{y}^{(\text{valid})}) > \epsilon$ **do**
  Train on $\boldsymbol{X}^{(\text{train})}$ and $\boldsymbol{y}^{(\text{train})}$ for n steps.
end while

Example: Early Stopping vs Explicit Regularization

Let us consider the 1D case of the previous example

$$R(\theta) = \frac{1}{2}\lambda(\theta - \theta^*)^2$$

What happens in early stopping?

Gradient descent:

$$\theta_{k+1} = \theta_k - \epsilon\lambda(\theta_k - \theta^*) = (1 - \epsilon\lambda)\theta_k + \epsilon\lambda\theta^*$$

so that

$$\theta_k = (1 - \epsilon\lambda)^k\theta_0 + \left(1 - (1 - \epsilon\lambda)^k\right)\theta^*$$

Let us suppose we stopped at iteration τ, then

$$\hat{\theta} = (1 - \epsilon\lambda)^\tau\theta_0 + (1 - (1 - \epsilon\lambda)^\tau)\theta^*$$

That is, $\hat{\theta}$ is a weighted average of the optimum θ^* and the initial condition θ_0, with weight $(1 - \epsilon\lambda)^\tau$.

Let us now look at a variant of L^2 regularized GD

$$\tilde{R}(\theta) = R(\theta) + \frac{1}{2}\alpha(\theta - \theta_0)^2$$

That is, we apply the regularization centered at θ_0. Then, the solution is

$$\hat{\theta} = \frac{\alpha}{\alpha + \lambda}\theta_0 + \left(1 - \frac{\alpha}{\alpha + \lambda}\right)\theta^*$$

Compare with early stopping

$$\hat{\theta} = (1 - \epsilon\lambda)^\tau \theta_0 + (1 - (1 - \epsilon\lambda)^\tau)\theta^*$$

In other words, early stopping at time τ is equivalent to L^2 regularization with strength

$$\alpha = \frac{\lambda(1 - \epsilon\lambda)^\tau}{1 - (1 - \epsilon\lambda)^\tau}$$

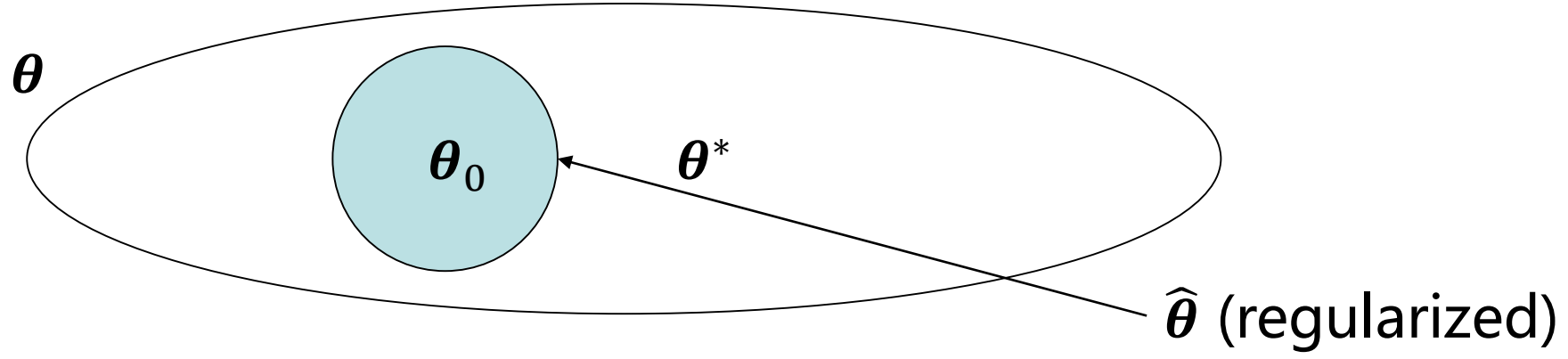
Remarks on Early Stopping

Advantages:

- Very easy to implement
- Effectively a regularization with a penalty depending on the distance from initial condition

$$\frac{1}{2}\alpha(\theta - \theta_0)^2$$

- Implicit regularization, no need to choose α!



Demo Continued: Early Stopping

Adding Noise (Training, Data)

Different ways to add Noise

The basic rationale behind adding noise to regularize models

Our model should be robust to noise

Different ways to add noise

- Inputs
- Outputs/labels
- Weights

Adding Noise to Inputs

This is heuristically equivalent to placing the prior:

The output is insensitive to random perturbations of the input

Again, this is a form of implicit regularization.

Increasing noise, no change in label

Example: Linear Regression with Noisy Inputs

Recall: least squares problem minimizes

$$R(\boldsymbol{w}) = \frac{1}{2}\|X\boldsymbol{w} - \boldsymbol{y}\|^2$$

Suppose we add noise to the inputs, which replaces

$$X \mapsto \tilde{X} = X + Z, \qquad \underbrace{\boldsymbol{z}^{(i)}}_{i^{\text{th}} \text{ row of } Z} \sim N(0, \delta I)$$

Then,

$$\tilde{R}(\boldsymbol{w}) = \frac{1}{2}\left\|\tilde{X}\boldsymbol{w} - \boldsymbol{y}\right\|^2 = \frac{1}{2}\|X\boldsymbol{w} - \boldsymbol{y}\|^2 + \frac{1}{2}\|Z\boldsymbol{w}\|^2 + (Z\boldsymbol{w})^T(X\boldsymbol{w} - \boldsymbol{y})$$

And so

$$\mathbb{E}_Z\tilde{R}(\boldsymbol{w}) = R(\boldsymbol{w}) + \frac{1}{2}\delta N\|\boldsymbol{w}\|^2$$

That is, adding noise to the input here is equivalent to L^2 regularization. This is also heuristically true in general.

Adding Noise to the Output

The rationale behind injecting noise to the output varies from application to application.

Examples:

- For classification problems, the given labels may have a small probability of being wrong
- The labels/outputs could be randomly drawn according to some distribution

In these cases, it may be advantageous to model such noise explicitly

Label Smoothing

An oft-used example of output noise injection is label smoothing

For a classification problem with a one-hot label, we can smooth the label by placing $1 \mapsto 1 - \alpha$ **and** $0 \mapsto \alpha/(K-1)$**.**

$$y = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \quad \Longrightarrow \quad y = \begin{pmatrix} \alpha/3 \\ 1-\alpha \\ \alpha/3 \\ \alpha/3 \end{pmatrix}$$

Adding noise to the Weights

Noise can also be added to the weights during training.

Consider one input-output pair x, y

Suppose we have a neural network which makes the prediction

$$\hat{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$$

Now, we add a small perturbation $\delta\boldsymbol{\phi}$ **(**$\delta \ll 1$**) to** $\boldsymbol{\theta}$**,** $\boldsymbol{\phi} \sim N(0, I)$

Then the risk (averaged over $\boldsymbol{\phi}$**) is**

$$\mathbb{E}_{\boldsymbol{\phi}}\tilde{R}(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\boldsymbol{\phi}}(f(\boldsymbol{x}, \boldsymbol{\theta} + \delta\boldsymbol{\phi}) - y)^2$$

$$= \frac{1}{2}(f(\boldsymbol{x}, \boldsymbol{\theta}) - y)^2 + \frac{m\delta^2}{2}\|\nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}, \boldsymbol{\theta})\|^2 + \cdots$$

That is, this penalizes variations of the neural network predictions with respect to the weights.

Demo Continued: Adding Noise

Summary

Today, we introduced a number of regularization strategies

- Parameter norm penalties (Training)
- Early stopping (Training)
- Injecting noise (Training/Data)

In each case, we can use linear models to analyze its effects.

Recurring theme: they are "equivalent" for linear models under different assumptions. In general, they are not equivalent for nonlinear models!