Improving Generalization

Nan Ye

School of Mathematics and Physics
The University of Queensland

Schedule

A tentative schedule is available on BlackBoard

- Week 1-2: machine learning basics
- Week 3-4: neural network basics
- Week 5-6: deep architectures
- Week 7-8: optimization
- Week 9-10: improving generalization
- Week 10-11: unsupervised learning
- Week 12: reinforcement learning

Chasing After Regularity

- So far, we have already seen two building blocks for crafting a good learning system
  - many different machine/deep learning models
  - many techniques to optimize a given numerical objective
- Often, we minimize the training error

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} L((\mathbf{x}_i, y_i), \mathbf{w}).$$

- Whether a model works well is problem-specific, and we need to choose a model of the right capacity (Lecture 6 Model Selection).

What can go wrong with a poor model

- A model underfits if it has poorer training set and test set performance than another model.
  - The model has a limited capacity, and can't fit the regularity well.
  - Use a model with higher capacity.
- A model overfits if it has better training performance than another model, but poorer test performance.
  - Deep nets often overfit.
- We focus on how to avoid overfitting for high-capacity models.

Approaches for Avoiding Overfitting

Get more data

- almost always beneficial to train on more data
- availability of large datasets is a main driver for deep learning's success

Model selection

- Choose several models of different capacity
- Use model selection techniques to choose one with the right capacity
  - enough to fit regularities in data
  - not enough to also fit noise/irregularities

Model averaging

- Train multiple models, and combine their predictions (instead of keeping only the best model in model selection)
- Some common model averaging methods
  - Bagging: train models on bootstrap samples of the training data
  - Bayesian learning: construct a ensemble of models weighted by their posterior

Regularization

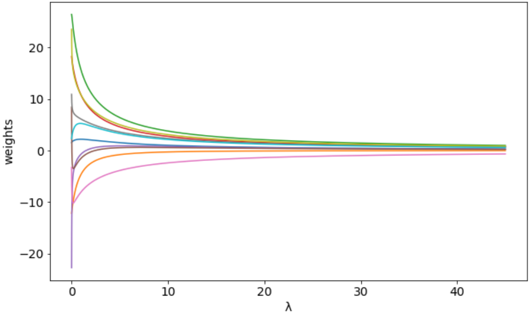
- Regularization techniques aims to minimize the training error and avoid fitting to irregularities at the same time
- A regularization method may
  - explicitly define a modified training objective (such as ℓ_2 regularization), or
  - implicitly induce a regularization effect by modifying the learning algorithm
- Some common regularization methods
  - data augmentation
  - ℓ_1/ℓ_2 regularization
  - early stopping
  - dropout

Data Augmentation

- Data augmentation adds perturbed examples or synthetic examples to the training set
  - e.g. LeNet uses distorted digit images
  - e.g. AlexNet uses left-right reflections of the images
- This prevents the model from overfitting the original training set.
- Perturbed/synthetic examples should still be realistic to avoid creating too much irregularities.

ℓ_2 Regularization

- We have already seen ℓ_2 regularization
  - Ridge regression: $\min_{\mathbf{w}} \left(\frac{1}{n}\sum_i (\mathbf{w}^\top \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_2^2\right)$.
  - Instead of minimizing the training error $L(\mathbf{w})$, minimize $L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$, where $\lambda > 0$ is a user-specified constant.
- ℓ_2 regularization keeps the weights small, preventing one feature from dominating others.
  - If there are two similar inputs, it puts about half the weight on each, rather than putting all the weights on one.



A typical plot of how weights change as λ increases in ridge regression.

Weight decay

- From an optimization perspective, ℓ_2 regularization is often known as weight decay
  - Gradient descent on $L(\mathbf{w})$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla L(\mathbf{w}).$$

  - Gradient descent on $L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$

$$\mathbf{w} \leftarrow (1 - 2\eta\lambda)\mathbf{w} - \eta \nabla L(\mathbf{w}) \qquad \text{(weight decay of } 2\eta\lambda\text{)}$$

  - Thus ℓ_2 regularization shrinks the weight first, and moves along the negative gradient direction.

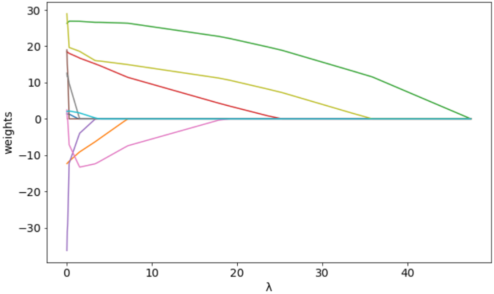
Stability

- A desirable property of a learning algorithm is *stability*, i.e., the learned model should not change drastically if the input is perturbed.
- ℓ_2 regularization increases stability.
  - recall: effect of ℓ_2 regularizer on OLS
- Regularized model has smaller variance but larger bias.

ℓ_1 Regularization

- ℓ_1 regularization is similar to ℓ_2 regularization
  - For a model with parameters $\mathbf{w}$, instead of minimizing the training error $L(\mathbf{w})$, it minimizes $L(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$, where $\lambda > 0$ is a user-specified constant.
- ℓ_1 regularization encourages sparsity — it can make many weights exactly equal to zero.
  - This helps interpretation.

- A classical example is ℓ_1 regularized linear regression (LASSO).



A typical plot of how weights change as λ increases.

Bayesian Interpretation of Regularization

Bayesian MAP (maximum a posterior) hypothesis

- Assume a prior distribution $P(\mathbf{w})$ on the model parameter vector, a likelihood function $P(D \mid \mathbf{w})$, then the posterior probability $P(\mathbf{w} \mid D)$ is given by

$$P(\mathbf{w} \mid D) = P(\mathbf{w})P(D \mid \mathbf{w})/P(D),$$

- $\mathbf{w}_{MAP} = \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w} \mid D)$ is called the MAP hypothesis.

Regularization as Bayesian prior

- Suppose the prior $P(\mathbf{w})$ and the likelihood function $P(D \mid \mathbf{w})$ are

$$P(\mathbf{w}) \propto e^{-r(\mathbf{w})},$$
$$P(D \mid \mathbf{w}) \propto e^{-L(D,\mathbf{w})},$$

  where $L(D, \mathbf{w})$ is the error of $\mathbf{w}$ on D.

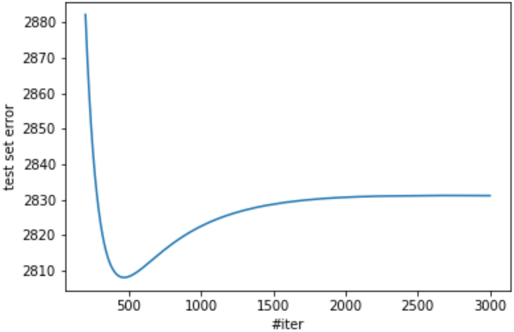
- Then we have

$$\mathbf{w}_{MAP} = \operatorname*{argmax}_{\mathbf{w}} P(\mathbf{w} \mid D) = \operatorname*{argmin}_{\mathbf{w}} \left(L(D, \mathbf{w}) + r(\mathbf{w})\right).$$

  Thus the regularizer can be interpreted as a prior distribution $P(\mathbf{w})$.

Early Stopping

- When we train more, a model's test set error can first decrease, and then increase (see Prac 5).



- We can stop training a model early when its performance becomes poorer on a validation set.

Why early stopping works

- This prevents the model from fitting too well to the training data.
- For deep nets with sigmoid activation, early stopping prevents the network to fully exploit the sigmoid nonlinearity
  - When we start at small initial values, the sigmoid units are at their linear regions, and thus the network is more or less linear.
  - As training goes, some weights become large, and thus the network becomes more nonlinear.
  - Stopping early prevents the network to evolve into a highly nonlinear function.

Remark

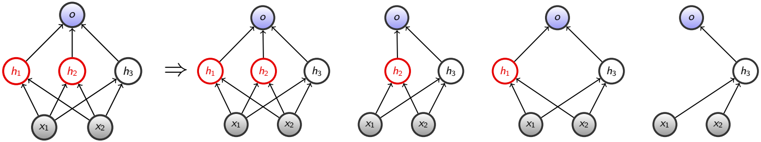
- Early stopping is best used together with a method that is designed to reduce training error at each iteration (such as vanilla gradient descent).
  - Easier to tell when validation error increases.
- It is hard to use early stopping with methods like momentum.
  - They do not attempt to improve performance for each update.
  - The validation error often oscillates with a decreasing trend.

Dropout

- Dropout
  - defines an ensemble of models based on a given architecture
  - trains them simultaneously, and
  - averages them for prediction.
- Specifically, given a neural net, we choose to omit each of a set of h chosen neurons with a probability p.
  - This defines 2^h different architectures.
  - Each is weighted by the probability of obtaining it.
  - All share the same parameters.

Srivastava et al., Dropout: a simple way to prevent neural networks from overfitting, 2014

- A network and the 4 archictures created by dropping out 2 hidden units with probability p.



$(1-p)^2$ $\quad p(1-p) \quad p(1-p) \quad p^2$

- All derived networks share the same parameters as the original network.

Training

- The training objective is to miminize

$$\sum_{i}^{m} p_i \left[\frac{1}{n} \sum_{j}^{n} L(\mathbf{x}_j, y_j, f_{\mathbf{w}}^{(j)}) \right],$$

  where $f_{\mathbf{w}}^{(1)}, \ldots, f_{\mathbf{w}}^{m}$ are the $m = 2^h$ different architectures, and p_i is the weight of $f_{\mathbf{w}}^{(i)}$.
- Dropout can be implemented as applying a random binary mask with values sampled from the Bernoulli distribution $B(1-p)$.
  - Sample a binary vector $(v_1, \ldots, v_h)$ with $v_i \sim B(1-p)$.
  - Let $(a_1, \ldots, a_h)$ be the activation values of the chosen neurons.
  - Change the activation values to $(a_1 v_1, \ldots, a_h v_h)$.

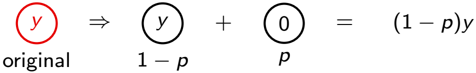
- Dropout can be understood as a way to prevent complex co-adaptation.
    - When hidden units know which other hidden units are present, they can co-adapt with each other to fit irregularities well.
    - If each hidden unit need to work well with different sets of co-workers, it will try to be useful on its own.

Test time

- During test time, we average the outputs of the m models

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_{\mathbf{w}}^{(i)}(\mathbf{x}).$$

  Averaging $m = 2^h$ models is computationally too expensive.
- Can we approximate the above averaging efficiently? Let's look at what happens if we just apply dropout to a single output neuron.



Average of derived networks'outputs $= (1-p)\times$ original network's output

- For the general case, we use the full network, but multiply each chosen neuron's activation by $(1-p)$.
  - Scaling matches the output to its expected value with dropout.

Inverted dropout

- Dropout is typically implemented as inverted dropout.
  - Training: divide the activation of a unit with dropout by $1 - p$.
  - Testing: simply return the output of the full network.
- Inverted dropout is equivalent to dropout, but simpler – it does not require scaling at test time.

DropConnect

- DropConnect drops connections, instead of activations.
- This can be implemented as applying a random binary mask to the weight matrix.

Your Turn

Which of the following statement is correct? (Multiple choice)

(a) Techniques for alleviating overfitting include getting more data, model selection, model averaging, regularization.

(b) Early stopping can be seen as an reguarlization technique that prevents the model from fitting to irregularities in data by stopping early.

(c) ℓ_1 regularization encourages sparse weights.

What You Need to Know

- Many methods for improving generalization performance
  - Get more data
  - Model selection
  - Model averaging
  - Regularization: data augmentation, ℓ_1/ℓ_2, early stopping, dropout
- In practice, we often use a combination of several techniques.