Regularization for Deep Learning

Jie Tang

November 5, 2019

Overview

Outline

Background: Overfitting

- In supervised learning settings, overfitting is usually a potential problem.
- Overfitting occurs when a model is excessively complex, such as having too many free parameters.

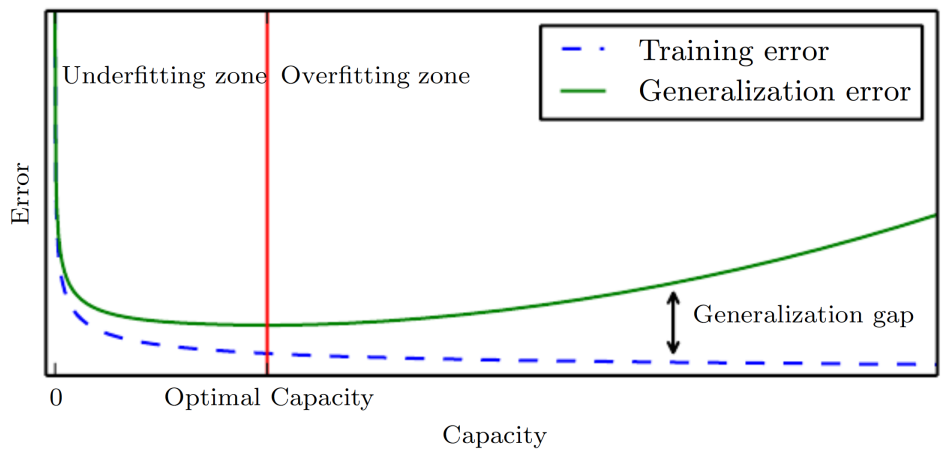


Figure 1: Typical relationship between model capacity and error. More complex models tend to overfit.

Example: Polynomial Regression

- The goal of polynomial regression is to find the best fit polynomial to the observed data points.

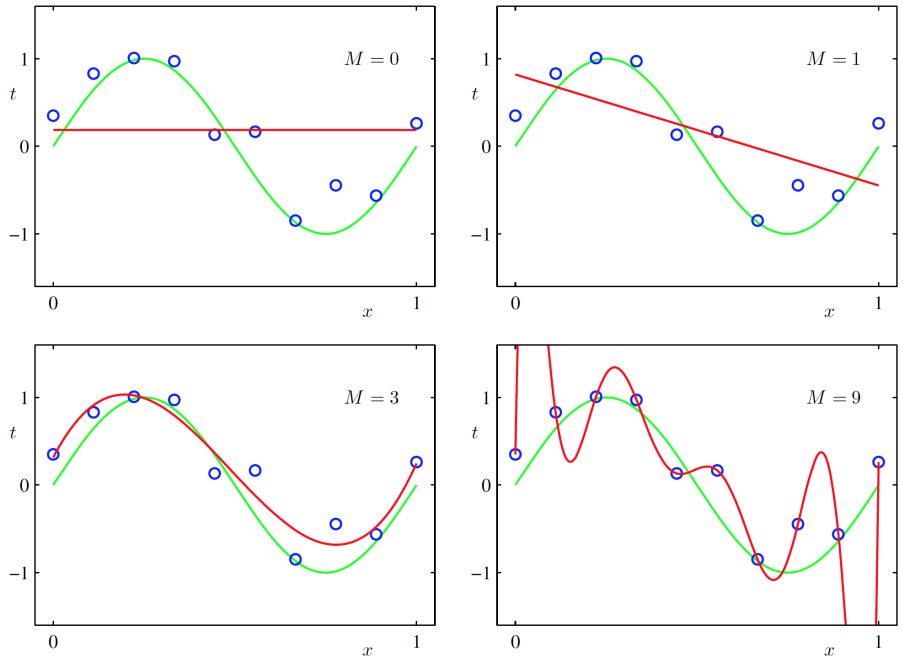


Figure 2: Plots of polynomials having various orders M (red curves) fitted to the data points (blue circles). A high order polynomial (e.g. $M = 9$) can perfectly fit the observed data, but gives a very poor representation of the function used to generate the data (green curves).

Regularization for Deep Learning

- Regularization can be motivated as a technique to improve the **generalization** of a learned model.
  - Many regularization strategies are explicitly designed to reduce the test error, possibly at the expense of increased training error.
- Ideas behind the regularization.
  - Encode specific kinds of prior knowledge.
  - Express preference for simpler model.
  - Make an underdetermined problem determined
- In the context of deep learning, most regularization strategies are based on regularizing estimators.
  - Regularization of an estimator works by trading increased bias for reduced variance.
  - In practical scenarios, we almost always do find that the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately.

Outline

Regularized Objective Function

- The regularized objective function $\tilde{J}$ is introduced by adding a parameter norm penalty Ω to the original objective function J.

$$\tilde{J}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}) \tag{1}$$

  - $\boldsymbol{\theta}$: the set of parameters
  - $\alpha \in [0, \infty)$: a hyperparameter that weights the contribution of the norm penalty term Ω relative to the standard objective function J
- When the training algorithm minimizes the regularized objective function $\tilde{J}$, it will decrease both the original objective J and some measure on the parameters $\boldsymbol{\theta}$ (or some subset of the parameters).
  - Different choices for the parameter norm Ω can result in different solutions being preferred.

L^2 Regularization

- term is formulated as $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{\theta}\|_2^2$.[1]
- The behavior of L^2 regularization:
  - For simplicity, we assume no bias parameters, so $\boldsymbol{\theta}$ is just $\boldsymbol{w}$. The regularized objective function is

$$\tilde{J}(\boldsymbol{w}) = \frac{\alpha}{2}\|\boldsymbol{w}\|_2^2 + J(\boldsymbol{w}) \tag{2}$$

  - Take a single gradient step to update the weights:

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}) \tag{3}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w})) \tag{4}$$

$$\Leftrightarrow \boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}) \tag{5}$$

  - L^2 regularization: shrink the weight by a constant factor on each step.

[1] L^2 regularization is commonly known as *weight decay*. It is also known as *ridge regression* or *Tikhonov regularization*.

Taylor series approximation–review

- Taylor expansion: if $f(x)$ has second-order derivatives, then we have

$$f(x) \approx f(x_0) + (x - x_0)f'(x) + \frac{1}{2}(x - x_0)^2 f''(x) \tag{6}$$

- Gradient:

$$g = \nabla_\theta J(\theta) \tag{7}$$

- Hessian matrix:

$$H_{i,j} = \frac{\partial}{\partial \theta_j} \frac{\partial J(\theta)}{\partial \theta_i} \tag{8}$$

  - H is symmetric, so there exists

$$Q = [v_1, ..., v_n] \tag{9}$$

$$H = Q \Lambda Q^\top \tag{10}$$

  - A second derivative in direction d is given by:

$$d^\top H d \tag{11}$$

L^2 Regularization

- What happens over the entire training step?
  - Make a quadratic approximation to $J(\boldsymbol{w})$ at the optimal weights $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$.[2]

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{12}$$

    where $\boldsymbol{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$. $\boldsymbol{H}$ is positive semidefinite because $\boldsymbol{w}^*$ is a minimum of J.[3]
  - The minimum of $\hat{J}$ occurs where its gradient is equal to $\mathbf{0}$.

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) = \mathbf{0} \tag{13}$$

[2] If the objective function is truly quadratic, then the approximation is perfect.

[3] There is no first-order term in this quadratic approximation, because $\boldsymbol{w}^*$ is defined to be a minimum where the gradient vanishes.

L^2 Regularization

- Adding the L^2 regularization gradient, the regularized version of $\hat{J}$ reaches the minimum at $\tilde{\boldsymbol{w}}$:

$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0 \tag{14}$$

$$\Leftrightarrow \tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^* \tag{15}$$

  - As α approaches 0, the regularized solution $\tilde{\boldsymbol{w}}$ approaches $\boldsymbol{w}^*$.
  - As α grows, decompose $\boldsymbol{H}$ into a diagonal matrix $\boldsymbol{\Lambda}$ and an orthonormal basis of eigenvectors $\boldsymbol{Q}$ such that $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$.[4]

$$\tilde{\boldsymbol{w}} = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{16}$$

$$= (\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^T)^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{17}$$

$$= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{18}$$

[4] $\boldsymbol{H}$ is real and symmetric.

L^2 Regularization

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1} \boldsymbol{\Lambda} \boldsymbol{Q}^T \boldsymbol{w}^*$$

- The effect L^2 Regularization is to rescale $\boldsymbol{w}^*$ along the axes defined by the eigenvectors of $\boldsymbol{H}$.
- The component of $\boldsymbol{w}^*$ that is aligned with the i-th eigenvector of $\boldsymbol{H}$ is rescaled by a factor $\frac{\lambda_i}{\lambda_i + \alpha}$.
- Bayesian prior interpretation: L^2 regularization is equivalent to MAP Bayesian inference with a Gaussian prior on the weights (as we discussed before).

L^2 Regularization

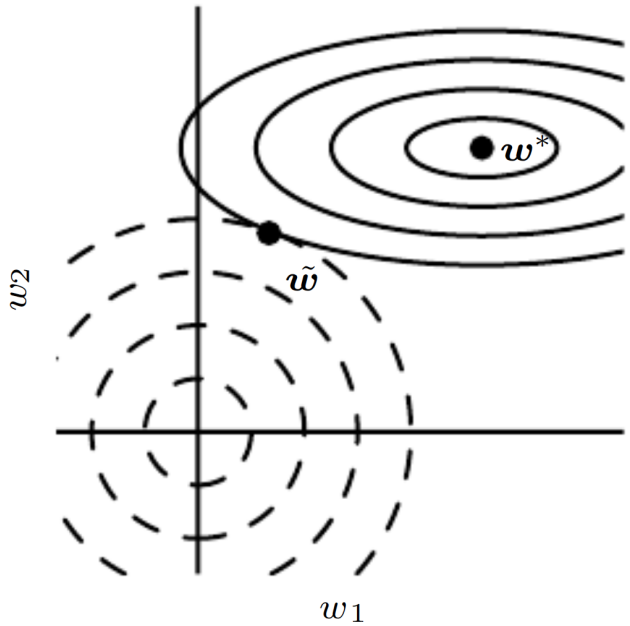


Figure 3: An illustration of the effect of L^2 regularization.The solid ellipses represent contours of equal value of the original objective J. The dotted circles represent contours of equal value of the L^2 regularizer.

- At the point $\tilde{w}$, these competing objectives reach an equilibrium.
- In the first dimension, the objective function does not increase much when moving horizontally away from $\boldsymbol{w}^*$. Thus, the regularizer has a strong effect on this axis.
- In the second dimension, the objective function is very sensitive to movements away from $\boldsymbol{w}^*$. As a result, the regularizer affects the position of w2 relatively little.

L^2 Regularization on Linear Regression

- How does these effects related to machine learning in in particular?
- Take linear regression for example, the objective function is the sum of squared errors:

$$J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) \tag{19}$$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{20}$$

- Add L^2 regularization on the objective function:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) + \frac{1}{2}\alpha\boldsymbol{w}^T\boldsymbol{w} \tag{21}$$

$$\tilde{\boldsymbol{w}}^* = \arg\min_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y} \tag{22}$$

L^2 Regularization on Linear Regression

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

$$\tilde{\boldsymbol{w}}^* = \arg\min_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = (\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

- Using L^2 Regularization replaces $\boldsymbol{X}^T\boldsymbol{X}$ with $(\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I})$.
- The original matrix is proportional to the covariance matrix with the diagonal entries corresponding to the variance of each input feature.
- The new matrix is the same as the origin one with the addition of α to the diagonal.
- L^2 Regularization causes the learning algorithm to "perceive" the input $\boldsymbol{X}$ as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^1 Regularization

- L^1 regularization term is formulated as $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_1$.
- Again, we assume no bias parameter ($\boldsymbol{\theta} = \boldsymbol{w}$).
- The regularized objective function and its gradient:

$$\tilde{J}(\boldsymbol{w}) = \alpha\|\boldsymbol{w}\|_1 + J(\boldsymbol{w}) \tag{23}$$

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}) = \alpha\text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}) \tag{24}$$

- The regularization contribution to the gradient is a constant factor with the sign of $\boldsymbol{w}$.

L^1 Regularization

- Make a quadratic approximation to the objective function via its Taylor series:

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{25}$$

- The L^1 penalty does not admit clean algebraic expressions in the case of a fully general Hessian, we will further assume that the Hessian is diagonal $\boldsymbol{H} = \text{diag}(H_{1,1}, ..., H_{n,n})$, where each $H_{i,i} > 0$.[5]
- The quadratic approximation of the L^1 regularized objective function decomposes into a sum over the parameters:

$$\tilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \sum_i \frac{1}{2} H_{i,i}(w_i - w_i^*)^2 + \alpha|w_i| \tag{26}$$

[5] This assumption holds if the data has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- The problem of minimizing the approximate objective function has an analytical solution for each i:

$$w_i = \text{sign}(w_i^*) \max\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\} \tag{27}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization just shifts the optimal value by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.
- L^1 regularization results in a solution that is more sparse. It has been used extensively as a feature selection mechanism.
- Bayesian prior interpretation: L^1 regularization can be explained as a Bayesian inference with an Laplace distribution over $\mathbf{w}$:

$$\log p(\mathbf{w}) = \sum_i \log \text{Laplace}(w_i; 0, \frac{1}{\alpha}) = -\alpha \|\mathbf{w}\|_1 + n \log \alpha - n \log 2$$

Norm Penalties as Constrained Optimization

- Consider a constrained optimization problem:

$$\min J(\boldsymbol{\theta})$$
$$\text{s.t. } \Omega(\boldsymbol{\theta}) - k \leq 0$$

  we can solve the constrained problem by constructing a generalized Lagrange function:

$$\mathcal{L}(\boldsymbol{\theta}, \alpha) = J(\boldsymbol{\theta}) + \alpha(\Omega(\boldsymbol{\theta}) - k) \tag{28}$$
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \alpha) \tag{29}$$

- If the value of α is fixed, we can get the solution:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) + \alpha^* \Omega(\boldsymbol{\theta}) \tag{30}$$

- This is exactly the same as the regularized objective function.

Norm Penalties as Constrained Optimization

- We can thus think of a parameter norm penalty as imposing a constraint on the weights.
  - The parameter α encourages $\Omega(\boldsymbol{\theta})$ to shrink: larger α will result in a smaller constraint region; smaller α will result in a larger one.
- Sometimes we may prefer explicit constraints rather than penalties.
  - If we have an idea of what value of k is appropriate and do not want to spend time searching for the value of α that corresponds to this k.
  - Penalties can cause non-convex optimization procedures to get stuck in local minimal.
  - Explicit constraints can be useful because they impose some stability on the optimization procedure.
- Hinton et al. (2012c) recommend using constraints combined with a high learning rate to allow rapid exploration of parameter space while maintaining some stability.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary for machine learning problems to be properly defined. Many linear models in machine learning depend on inverting the matrix $\boldsymbol{X}^T\boldsymbol{X}$. This is not possible whenever $\boldsymbol{X}^T\boldsymbol{X}$ is singular. In this case, many forms of regularization correspond to inverting $\boldsymbol{X}^T\boldsymbol{X} + \alpha\boldsymbol{I}$ instead.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems. For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Outline

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- For some machine learning tasks, the algorithm is expected to be robust to noise or be invariant to some transformations.
- It is reasonably straightforward to generate new fake data by transforming the inputs.
- Dataset argumentation has been a particularly effective technique for tasks like object recognition, speech recognition.

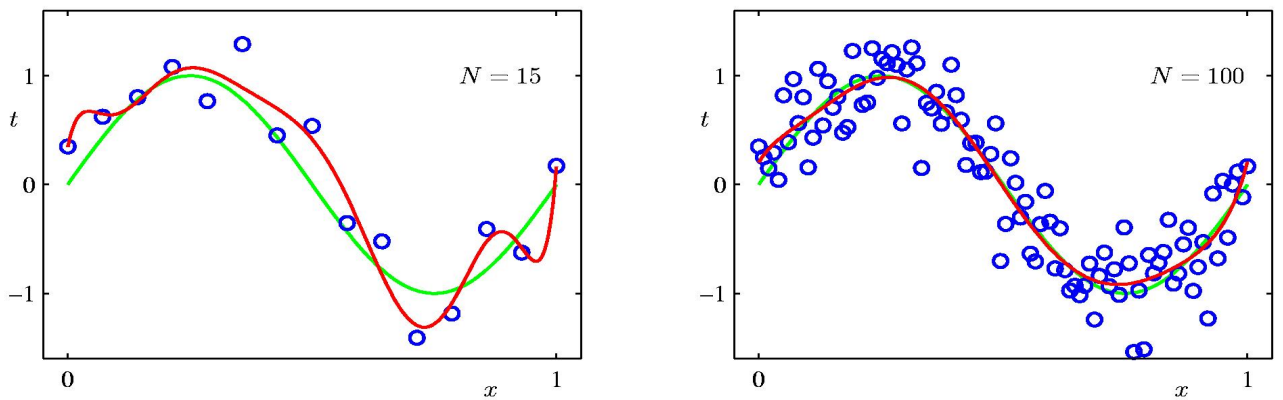
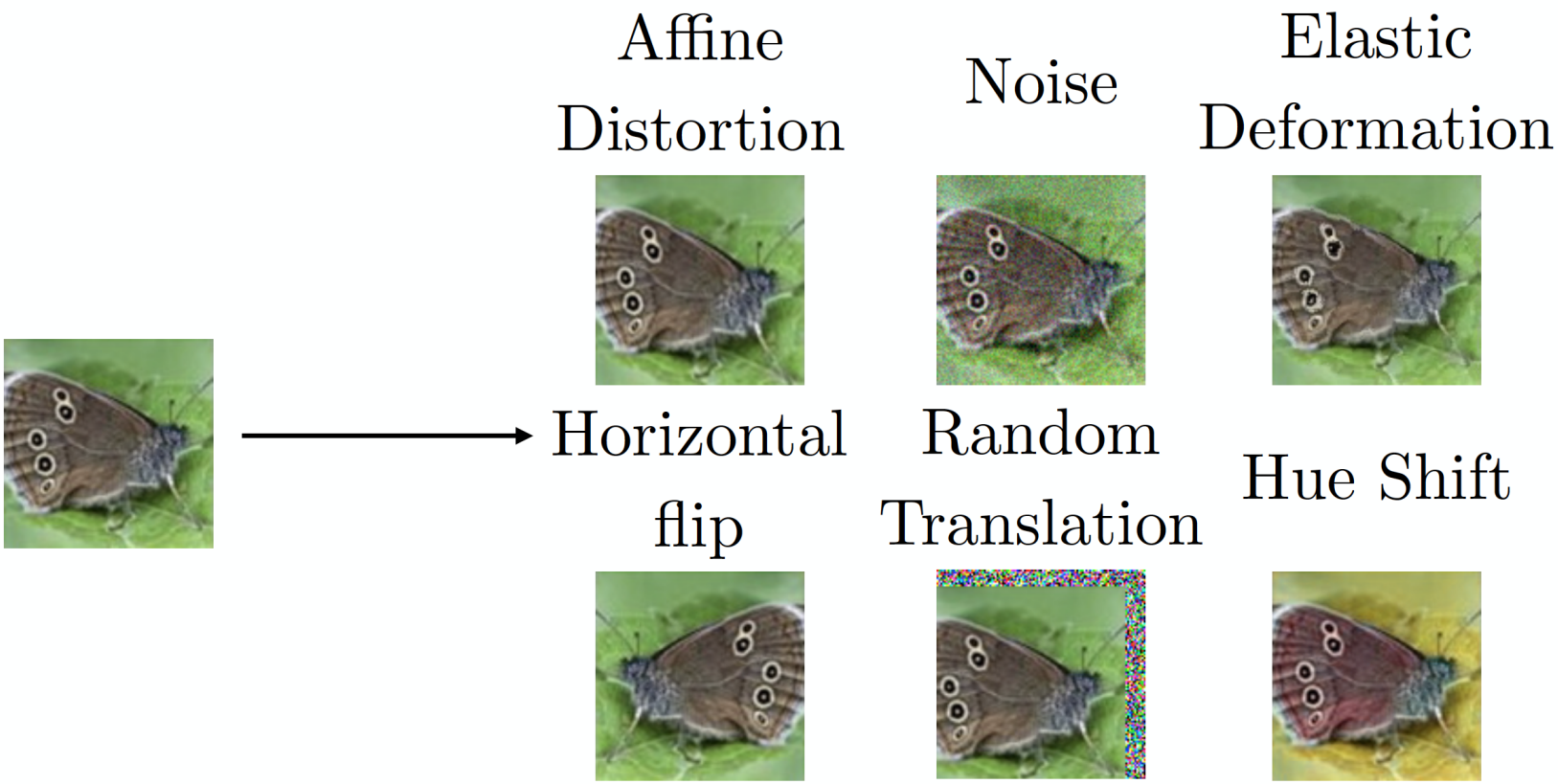


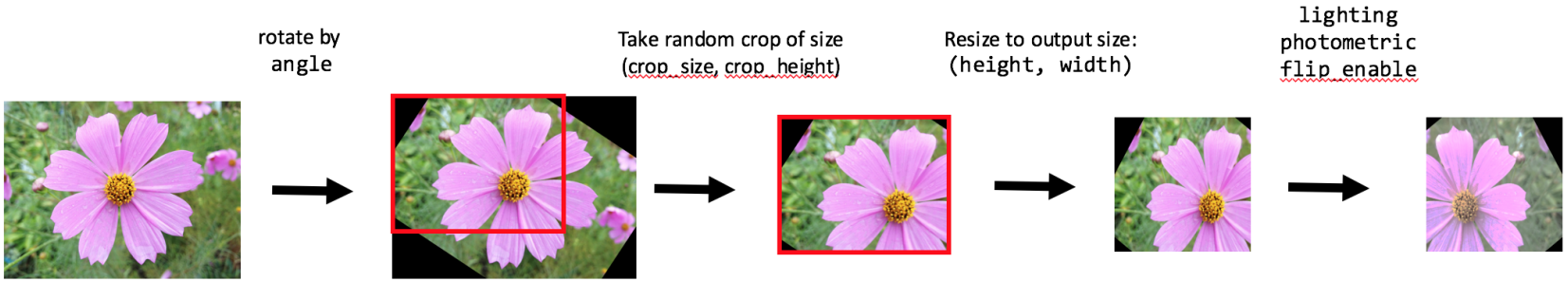
Figure 4: Recall the data-augment examples for fitting the polynomial functions.

Dataset Augmentation: Example

- In objective recognition, we would like our classifiers to be invariant to translations such as translating, rotating, scaling and etc.

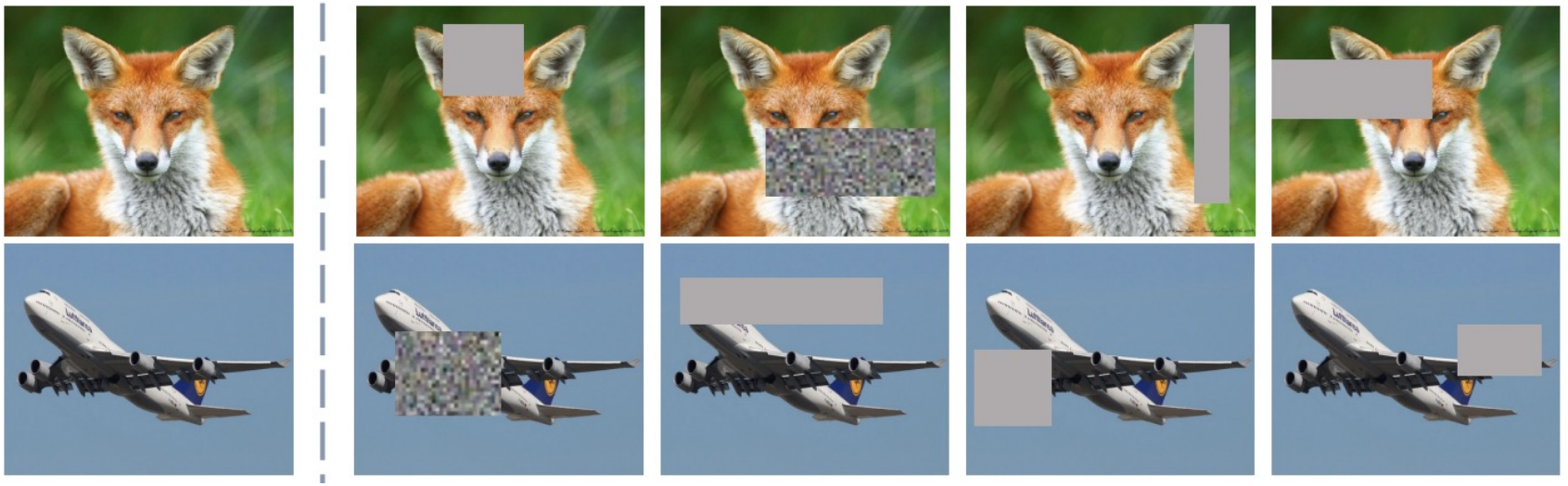


Random Crop



- Randomly crops a fix-sized rectangle of the image, and resize it to the input scale.
- Random Cropping is a very useful data augmentation technique that forces the deep learning model to learn robustly and ignore positional bias in the training set.

DropBlock



- Randomly selects a random-sized rectangle of the image, fill it with grey or random color.
- DropBlock(Ghiasi et al. 2018) forces the model not to rely on subtle local patterns of the image and guides it to learn the general concept.

Outline

Noise Injection at the Inputs

- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs. It can be motivated as a data augmentation strategy.
- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop, 1995a,b).
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder.
- Noise injection also works when the noise is applied to the hidden units, which can be seen as doing dataset augmentation at multiple levels of abstraction.

Noise Injection at the Weights

- Another way that noise has been used in the service of regularizing models is by adding it to the weights. This technique has been used primarily in the context of recurrent neural networks.
- This can be interpreted as a stochastic implementation of a Bayesian inference over the weights.
  - The Bayesian treatment of learning would consider the model weights to be uncertain and representable via a probability distribution.
  - Adding noise to the weights is a practical, stochastic way to reflect this uncertainty.

  Example:

$$\mu_N = \frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2} \cdot \left(\frac{1}{N}\sum_{n=1}^{N} x_n\right) + \frac{\sigma^2}{N\sigma_0^2 + \sigma^2}\mu_0 \tag{31}$$

- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Injection at the Weights

- Effect of weight noise in the regression setting:
  - The object is to minimize the least-square cost between the model predictions $\hat{y}(x)$ and the true values y:

$$J = \mathbb{E}_{p(\boldsymbol{x},y)}[(\hat{y}(\boldsymbol{x}) - y)^2] \tag{32}$$

  - Assume that with each input presentation we also include a random perturbation $\epsilon_{\boldsymbol{W}} \sim \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \eta \boldsymbol{I})$. We denote the perturbed model as $\hat{y}_{\epsilon_{\boldsymbol{W}}}(x)$. The objective function thus become:

$$\tilde{J}_{\boldsymbol{W}} = \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[(\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - y)^2] \tag{33}$$

$$= \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_{\boldsymbol{W}})}[\hat{y}^2_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) - 2y\hat{y}_{\epsilon_{\boldsymbol{W}}}(\boldsymbol{x}) + y^2] \tag{34}$$

  - For small η, the minimization of J with added weight noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimization of J with an additional regularization term: $\eta \mathbb{E}_{p(\boldsymbol{x},y)}[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2]$
- This form of regularization encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output.

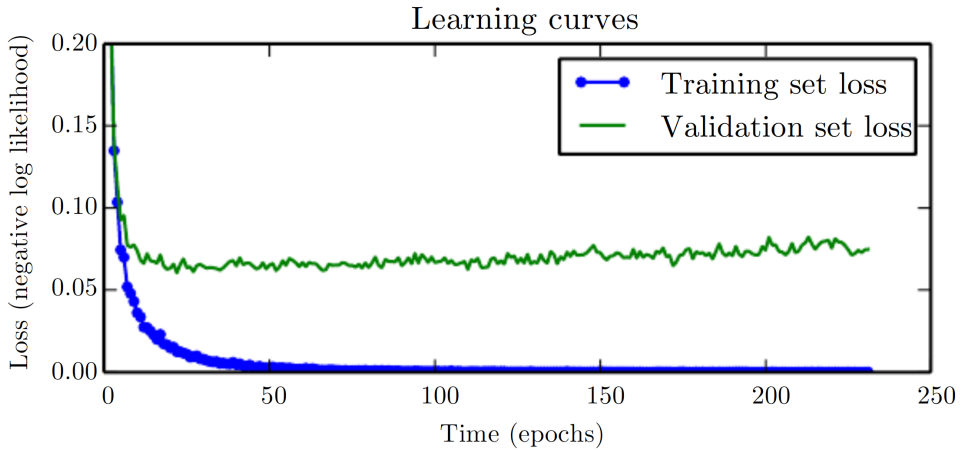
Noise Injection at the Outputs

- Most datasets have some amount of mistakes in the y labels. It can be harmful to maximize $\log p(y|x)$ when y is a mistake.
  - For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.
  - This assumption is easy to incorporate into the cost function analytically, rather than by explicitly drawing noise samples.
- Label smoothing regularizes a model based on a softmax with k output values by replacing the hard 0 and 1 classification targets with targets of $\frac{\epsilon}{k}$ and $\frac{1-\epsilon}{k}$, respectively.

Outline

Early Stopping

- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.



- We can obtain a model with better validation set error (and thus, hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.
- This strategy is known as early stopping. It is probably the most commonly used form of regularization in deep learning.

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^*

Early Stopping

- Early stopping requires a validation set, which means some training data is not fed to the model.
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed.
- There are two basic strategies one can use for this extra training procedure.

Early Stopping

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.

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

Early Stopping

- How does early stopping act as a regularizer?
- Bishop (1995a) and Sjoberg and Ljung (1995) argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value $\boldsymbol{\theta}_o$.
- We can show, in the case of a simple linear model with a quadratic error function and simple gradient descent, early stopping is equivalent to L^2 regularization.

Early Stopping

- We examine a simple setting where the only parameters are linear weights ($\boldsymbol{\theta} = \boldsymbol{w}$).
- Model the cost function J with a quadratic approximation in the neighborhood of the empirically optimal value of the weights $\boldsymbol{w}^*$:

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{35}$$

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \tag{36}$$

- For simplicity, initialize the parameter vector to the origin ($\boldsymbol{w}^{(0)} = \boldsymbol{0}$).
- The trajectory followed by the parameter vector during training:

$$\boldsymbol{w}^{(\tau)} = \boldsymbol{w}^{(\tau-1)} - \epsilon \nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}^{(\tau-1)}) \tag{37}$$

$$= \boldsymbol{w}^{(\tau-1)} - \epsilon \boldsymbol{H}(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{38}$$

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon \boldsymbol{H})(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{39}$$

Early Stopping

- Exploiting the eigen-decomposition of $\boldsymbol{H}$: $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix and $\boldsymbol{Q}$ is an orthonormal basis of eigenvectors.

$$\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^* = (\boldsymbol{I} - \epsilon\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T)(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{40}$$

$$\boldsymbol{Q}^T(\boldsymbol{w}^{(\tau)} - \boldsymbol{w}^*) = (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})\boldsymbol{Q}^T(\boldsymbol{w}^{(\tau-1)} - \boldsymbol{w}^*) \tag{41}$$

$$\boldsymbol{Q}^T\boldsymbol{w}^{(\tau)} = (\boldsymbol{I} - (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^r)\boldsymbol{Q}^T\boldsymbol{w}^* \tag{42}$$

- Recall Eq. 18 in L^2 regularization:

$$\tilde{\boldsymbol{w}} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{43}$$

$$\boldsymbol{Q}^T\tilde{\boldsymbol{w}} = (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^T\boldsymbol{w}^* \tag{44}$$

$$\boldsymbol{Q}^T\tilde{\boldsymbol{w}} = (\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha)\boldsymbol{Q}^T\boldsymbol{w}^* \tag{45}$$

Early Stopping

- Compare Eq. 42 and Eq. 45, we see that L^2 regularization and early stopping can be seen to be equivalent if the hyperparameters ϵ, α and τ are chosen such that:

$$\boldsymbol{I} - (\boldsymbol{I} - \epsilon\boldsymbol{\Lambda})^r = \boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha \tag{46}$$

- Going even further, by taking logarithms and using the series expansion for $\log(1+x)$, we can conclude that if all λ_i are small (that is, $\epsilon\lambda_i \ll 1$ and $\lambda_i/\alpha \ll 1$) then:

$$\tau \approx \frac{1}{\epsilon\alpha} \tag{47}$$

$$\alpha \approx \frac{1}{\tau\epsilon} \tag{48}$$

- Parameter values corresponding to directions of significant curvature are regularized less than directions of less curvature. In the context of early stopping, this means that parameters that correspond to directions of significant curvature tend to learn early relative to parameters corresponding to directions of less curvature.

Early Stopping

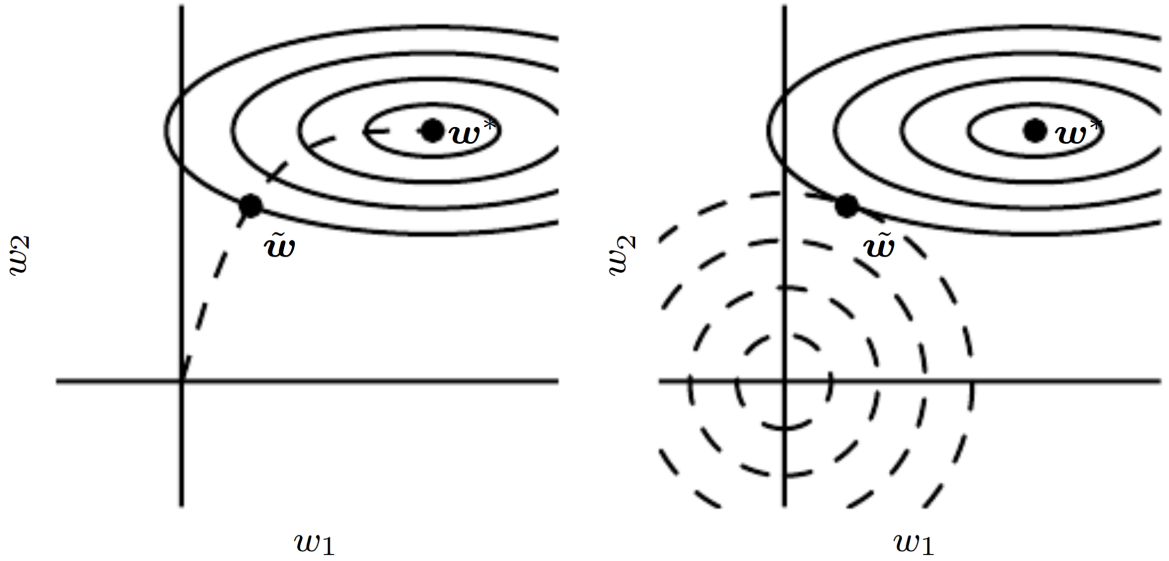


Figure 5: An illustration of the effect of early stopping. (Left) The solid contour lines indicate the contours of the negative log-likelihood. The dashed line indicates the trajectory taken by SGD beginning from the origin. Rather than stopping at the point $\boldsymbol{w}$ that minimizes the cost, early stopping results in the trajectory stopping at an earlier point $\tilde{\boldsymbol{w}}$. (Right) An illustration of the effect of L^2 regularization for comparison. The dashed circles indicate the contours of the L^2 penalty, which causes the minimum of the total cost to lie nearer the origin than the minimum of the unregularized cost.

Outline

Parameter Sharing

- Sometimes we might not know precisely what values the parameters should take, but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
  - Consider that there are two models performing the same classification task but with different input distributions. Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
  - Imagine that the tasks are similar enough that we believe the model parameters should be close to each other: $w_i^{(A)}$ should be close to $w_i^{(B)}$ for all i.
  - While we can use a parameter norm penalty $\|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\|_2^2$ to regularize parameters to be close to one another, the more popular way is to use constraints: to force sets of parameters to be equal.
- This method of regularization is often referred to as parameter sharing.
- By far the most popular and extensive use of parameter sharing occurs in convolutional neural networks (CNNs) applied to computer vision.

Semi-Supervised Learning

- In the context of deep learning, semi-supervised learning usually refers to learning a representation $\boldsymbol{h} = f(\boldsymbol{x})$.
  - The goal is to learn a representation so that examples from the same class have similar representations.
  - A linear classifier in the new space may achieve better generalization in many cases (Belkin and Niyogi, 2002; Chapelle et al., 2003).
- Instead of having separate unsupervised and supervised components in the model, one can construct models in which a generative model of either $P(\boldsymbol{x})$ or $P(\boldsymbol{x}, \boldsymbol{y})$ shares parameters with a discriminative model of $P(\boldsymbol{y}|\boldsymbol{x})$.
  - One can then trade-off the supervised criterion $-\log P(\boldsymbol{y}|\boldsymbol{x})$ with the unsupervised or generative one (such as $-\log P(\boldsymbol{x})$ or $-\log P(\boldsymbol{x}, \boldsymbol{y})$).
  - The generative criterion then expresses a particular form of prior belief that the structure of $P(\boldsymbol{x})$ is connected to the structure of $P(\boldsymbol{y}|\boldsymbol{x})$ in a way that is captured by the shared parametrization.

Multi-Task Learning

- Multi-task learning (Caruana, 1993) is a way to improve generalization by pooling the examples arising out several tasks.
- In the same way that additional training examples put more pressure on the parameters towards values that generalize well, when part of a model is shared across tasks, that part of the model is more constrained towards good values, often yielding better generalization.
- From the point of view of deep learning, the underlying prior belief is the following: *among the factors that explain the variations observed in the data associated with the different tasks, some are shared across two or more tasks*.

Multi-Task Learning

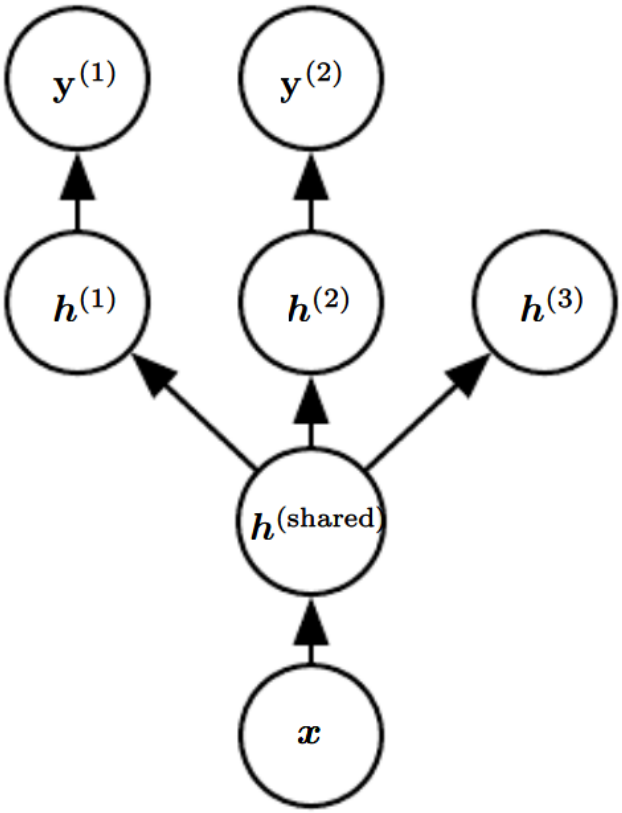


Figure 6: Multi-task learning

- Left figure illustrates a very common form of multi-task learning, in which different supervised tasks share the same input $\boldsymbol{x}$, as well as some intermediate-level representation $\boldsymbol{h}^{(shared)}$.
- Improved generalization can be achieved because of the shared parameters, for which statistical strength can greatly improved.
- This will happen only if some assumptions about the statistical relationship between the different tasks are valid.

Outline

Bagging and Ensemble Methods

- Bagging (short for bootstrap aggregating) is a technique for reducing generalization error by combining several models.
- The idea is to train several different models separately, then have all of the models vote on the output for test examples. This is an example of a general strategy in machine learning called model averaging. Techniques employing this strategy are known as ensemble methods.
- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
    - Neural networks can often benefit from model averaging even if all of the models are trained on the same dataset. Differences in random initialization, random selection of minibatches, hyperparameters, or different outcomes of non-deterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors.

Ensemble Methods

- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example. The errors are drawn from a multivariate normal distribution with $\mathbb{E}(\epsilon_i^2) = v$ and $\mathbb{E}(\epsilon_i \epsilon_j) = c$. Then the error made by the average prediction is $\frac{1}{k}\sum_i \epsilon_i$. The expected squared error of the ensemble predictor is

$$\mathbb{E}((\frac{1}{k}\sum_i \epsilon_i)^2) = \frac{1}{k^2}\mathbb{E}(\sum_i(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j)) = \frac{1}{k}v + \frac{k-1}{k}c \tag{49}$$

- If the errors are perfectly correlated and $c == v$, the mean error reduces to v, the model averaging does not help at all. If the errors are perfectly uncorrelated and $c = 0$, the expected squared error of the ensemble is only $\frac{1}{k}v$.
- 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.

Bagging

- Different ensemble methods construct the ensemble of models in different ways.
  - For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Bagging first construct k different datasets by sampling with replacement from the original dataset.
- Model i is then trained on dataset i. The differences between which examples are included in each dataset result in differences between the trained models.

Bagging

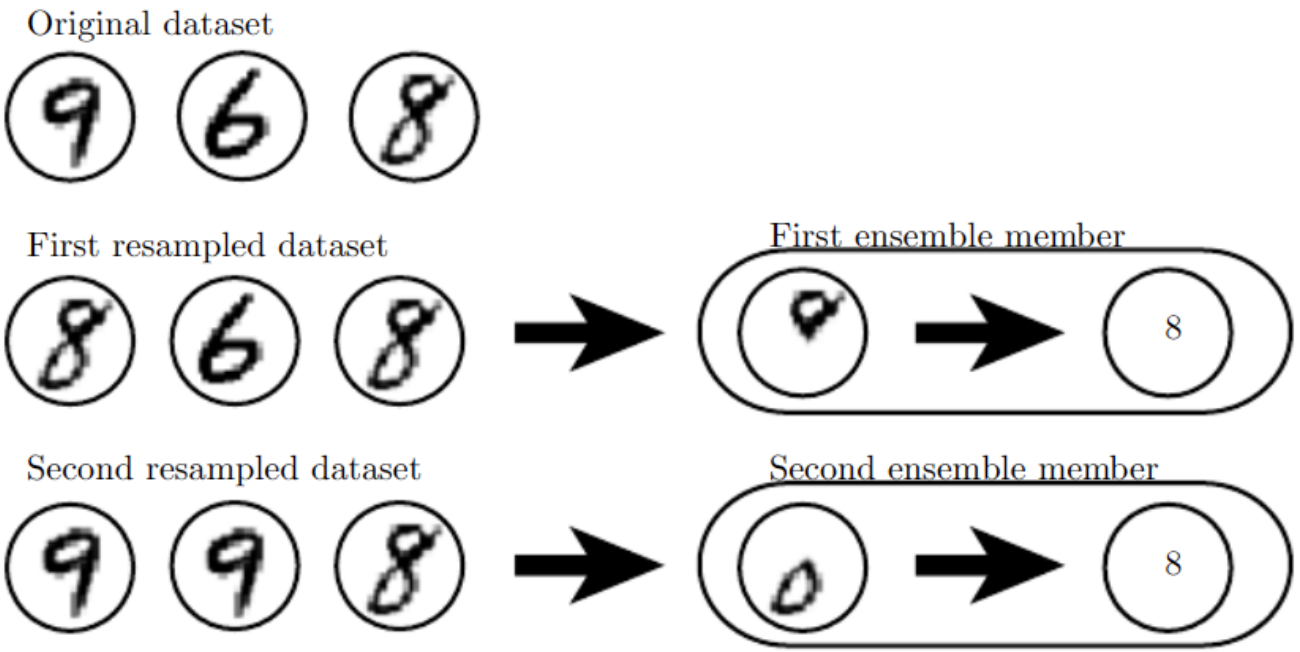


Figure 7: An example of training an 8 detector on the dataset containing an 8, a 6 and a 9. On the first dataset, the detector learns that a loop on top of the digit corresponds to an 8. On the second dataset, the detector learns that a loop on the bottom of the digit corresponds to an 8. Each of these individual classification rules is brittle, but if we average their output then the detector is robust.

Outline

Dropout

- Dropout (Srivastava et al., 2014) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
  - Bagging involves training and evaluating multiple models, which seems impractical when each model is a large neural network.
  - Dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output unit from an underlying base network.
  - In most modern neural networks, we can effectively remove a unit from a network by multiplying its output value by zero.

Dropout

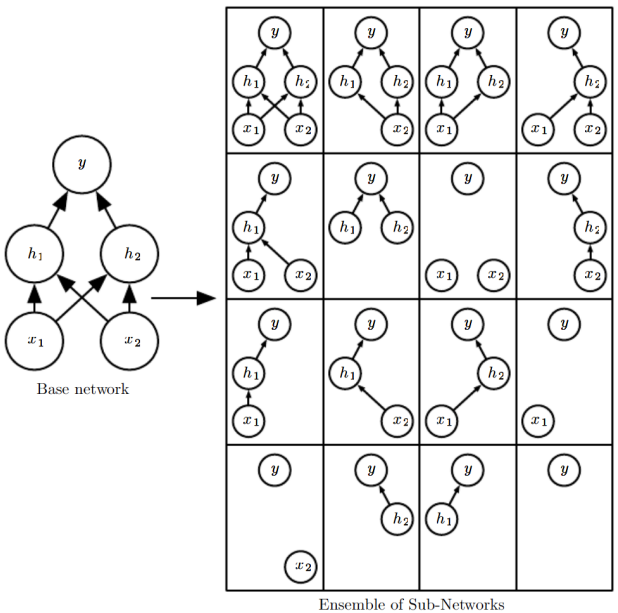


Figure 8: There are 16 possible subsets of a base network with 2 visible units and 2 hidden units.

- Dropout aims to approximate the bagging process, but with an *exponentially* large number of neural networks.
  - To train with dropout, we use a minibatch-based algorithm.
  - Each time we load an example into a minibatch, we randomly sample a binary mask for each of the input and hidden units.[a]
  - We then run forward propagation, back-propagation, and the learning update as usual.

[a] The mask probability is a set hyperparameter .

Dropout

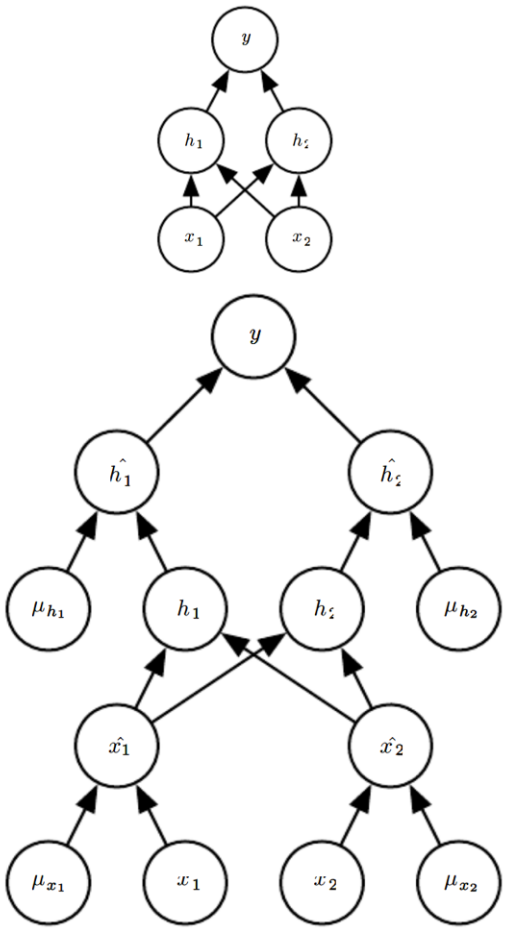


Figure 9: An example of forward propagation with dropout.

- To perform forward propagation with dropout, we randomly sample a vector $\boldsymbol{\mu}$ with one entry for each input or hidden unit int the network.
- Each unit in the network is multiplied by the corresponding mask, and then forward propagation continues through the rest of the network as usual.
- This is equivalent to randomly selecting one of the sub-networks from the base network and running forward propagation through it.

Dropout

- Dropout training is not quite the same as bagging training.
- Bagging:
  - The models are all independent.
  - Each model is trained to convergence on its respective training set.
- Dropout:
  - The models share parameters, with each model inheriting a different subset of parameters from the base network.
  - Typically most models are not explicitly trained at all—the model is large enough that it would be infeasible to sample all possible sub-networks. Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.

Outline

Sparse Representations

- We have already discussed how L^1 penalization induces a sparse parametrization—meaning that many of the parameters become zero.
- Another sparsity regularization, representational sparsity, describes a presentation where many of the elements of the representation are 0 (or close to 0).

$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}} = \underset{\boldsymbol{A} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}} \underset{\boldsymbol{x} \in \mathbb{R}^n}{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}$$

(a) A linear regression model with parameter sparsity.

$$\underset{\boldsymbol{y} \in \mathbb{R}^m}{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}} = \underset{\boldsymbol{B} \in \mathbb{R}^{m \times n}}{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}} \underset{\boldsymbol{h} \in \mathbb{R}^n}{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}$$

(b) A linear regression model with representational sparsity

Figure 10: An illustration of the distinction between sparse parametrization and sparse representation. In the right expression, $\boldsymbol{h}$ is a sparse representation of the data $\boldsymbol{x}$. That is, $\boldsymbol{h}$ is a function of $\boldsymbol{x}$ that represents the information present in $\boldsymbol{x}$ with a sparse vector.

Sparse Representations

- Norm penalty regularization of representations is performed by adding to the loss function J a norm penalty on the *representation*.

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h}) \tag{50}$$

- Like on the parameters, an L^1 penalty on the representation induces representational sparsity as well.
- Other approaches can result in a sparse representation:
  - Penalty derived from a Student-t prior of the representation.
  - KL divergence penalties that constrain the representations to lie on the unit interval.
  - Regularize the average activation across several examples to be near some target value.
  - Orthogonal matching pursuit (OMP) encodes an input $\boldsymbol{x}$ with the representation $\boldsymbol{h}$ that solves the constrained optimization problem:

$$\underset{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k}{\arg\min} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

    where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.

Adversarial Training

- In many cases, neural networks have begun to reach human performance, but it is unclear whether these models have obtained a rue human-level understanding of these tasks.
- Szegedy el al. (2014b) found hat even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed.
- An optimization procedure can be used to search for an input $\boldsymbol{x}'$ near a data point $\boldsymbol{x}$ such that the model output is very different at $\boldsymbol{x}'$.
- In many cases, $\boldsymbol{x}'$ can be so similar to $\boldsymbol{x}$ that a human observer cannot tell the difference between the original example and the **adversarial example**, but the network can make highly different predictions.

Adversarial Training

$+ .007 \times$

$=$

$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda"	"nematode"	"gibbon"
w/ 57.7%	w/ 8.2%	w/ 99.3 %
confidence	confidence	confidence

Figure 11: A demonstration of adversarial example generation applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image.

Adversarial Training

- Adversarial training is to train on adversarially perturbed examples from the training set.
- Goodfellow et al. (2014b) showed that one of the primary causes of these adersarial examples in excessive linearity.
  - Neural networks are built out of primarily linear building blocks, which results that the overall function to be high linear.
  - The value of a linear function can change very rapidly if it has numerous inputs. If we change each input by ϵ, then a linear function with weights $\boldsymbol{w}$ can change by as much as $\epsilon \|\boldsymbol{w}\|_1$, which can be very large if $\boldsymbol{w}$ is high-dimensional.
- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data.

Adversarial Training

- Adversarial training helps to illustrate the power of using a large function family in combination with aggressive regularization.
  - Purely linear models are not able to resist adversarial examples.
  - Neural networks are able to represent all sorts of functions, thus can capture the linear trends in the training data while still learning to resist local perturbation.
- Adversarial examples also provide a meaning of accomplishing semi-supervised learning.
  - Given a point $\boldsymbol{x}$ where the model outputs $\hat{y}$ equals to the real label y, we can seek an adversarial example $\boldsymbol{x}'$ close to $\boldsymbol{x}$ that causes the classifier to output a label $\hat{y}' \neq \hat{y}$.
  - The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$. This encourages the classifier to learn a function robust to small changes anywhere along the manifold.
  - The assumption motivating this approach is that different classes usually lie on disconnected manifolds, and a small perturbation should not be able to jump from one class manifold to another class manifold.(So the real label $y' = y$ still holds for $\boldsymbol{x}$ and $\boldsymbol{x}'$.)

Overview

1. Background: Overfitting
2. Parameter Norm Penalties
3. Dataset Augmentation
4. Noise Robustness
5. Early Stopping
6. Parameter Sharing
7. Ensemble methods
8. Dropout
9. Other regularization methods

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn