Deep learning

Optimization and Regularization in deep networks

Hamid Beigy

Sharif university of technology

October 9, 2019

Table of contents

Table of contents

Batch gradient descent

1. Batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

2. We need to calculate the gradients for the whole dataset to perform just one update.
3. Batch gradient descent can be very slow and is intractable for datasets that don't fit in memory.
4. Batch gradient descent also doesn't allow us to update our model online, i.e. with new examples on-the-fly.

Stochastic gradient descent

1. Stochastic gradient descent (SGD) performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

2. Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update.
3. SGD does away with this redundancy by performing one update at a time.
4. It is therefore usually much faster and can also be used to learn online.
5. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

Mini-batch gradient descent

1. Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

2. This method reduces the variance of the parameter updates, which can lead to more stable convergence
3. It can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient (mini-batch very efficient).
4. Common mini-batch sizes range between 50 and 256, but can vary for different applications.

Mini-batch gradient descent (Challenges)

Mini-batch gradient descent does not guarantee good convergence and offers a few challenges that need to be addressed.

1. Choosing a proper learning rate can be difficult.
2. Choosing the parameters (schedules and thresholds) of learning rate schedules is difficult.
3. Are we using the same learning rate for all parameters?
4. How to avoid from getting trapped in suboptimal local minima.

Momentum

1. Momentum[1] is a method that helps accelerate SGD in the relevant direction and dampens oscillations.
2. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

[1] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks, 12(1), 145–151.

Nesterov accelerated gradient

1. A ball that rolls down a hill, blindly following the slope, is highly unsatisfactory.
2. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
3. Nesterov accelerated gradient (NAG)[2] is a way to give our momentum term this kind of prescience.

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

The value of $\theta - \gamma v_{t-1}$ gives an approximation of the next position of the parameters.

[2] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o(1/k2). Doklady ANSSSR (translated as Soviet.Math.Docl.), vol. 269, pp. 543– 54

Adagrad

1. Adagrad[3] is an algorithm for gradient-based optimization that adapts the learning rate to the parameters.
2. Adagrad updates the parameters in the following manner.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot \nabla_\theta J(\theta_{t,i})$$

   where $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element (i, i) is the sum of the squares of the gradients w.r.t. θ_i. ϵ is a smoothing term that avoids division by zero.
3. Adadelta[4] is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

[3] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159.

[4] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. Arxiv.

Adam

1. Adaptive Moment Estimation (Adam)[5] computes adaptive learning rates for each parameter.
2. Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.
3. Adam computes the decaying averages of past and past squared gradients m_t and v_t.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta J(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta J(\theta_t))^2 \end{aligned} \tag{1}$$

where m_t and v_t are estimates of the first moment and the second moment of the gradients, respectively,

[5] Kingma, D. P., and Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13.

Adam (cont.)

1. Initially m_t and v_t are set to 0.
2. m_t and v_t are biased towards zero.
3. Bias-corrected m_t and v_t are

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{2}$$

Then Adam updates parameters as

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Other optimization algorithms)

1. AdaMax changed parameter v_t of Adam
2. Nadam (Nesterov-accelerated Adaptive Moment Estimation) combines Adam and NAG.
3. For more information please read the following paper. Sebastian Ruder (2017), "An overview of gradient descent optimization algorithms", Arxiv.
4. Which optimizer to use?

[5] Dozat, T. (2016). Incorporating Nesterov Momentum into Adam. ICLR Workshop, (1), 2013–2016.

Table of contents

Model selection

1. Considering regression problem, in which the training set is

$$S = \{(x_1, t_1), (x_2, t_2), \ldots, (x_N, t_N)\}, t_k \in \mathbb{R}.$$

where

$$t_k = f(x_k) + \epsilon \qquad \forall k = 1, 2, \ldots, N$$

$f(x_k) \in \mathbb{R}$ is the unknown function and ϵ is the random noise.

2. The goal is to approximate the $f(x)$ by a function $g(x)$.
3. The empirical error on the training set S is measured using cost function $E_E(g(x)|S) = \frac{1}{2}\sum_{i=1}^{N}(t_i - g(x_i))^2$
4. The aim is to find $g(.)$ that minimizes the empirical error.
5. We assume that a hypothesis class for $g(.)$ with a small set of parameters.
6. Assume that $g(x)$ is linear

$$g(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_D x_D$$

Model selection

1. Define the following vectors and Matrix
   - Data matrix

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1D} \\ 1 & x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & & & \vdots & \\ 1 & x_{N1} & x_{N2} & \dots & x_{DD} \end{bmatrix}$$

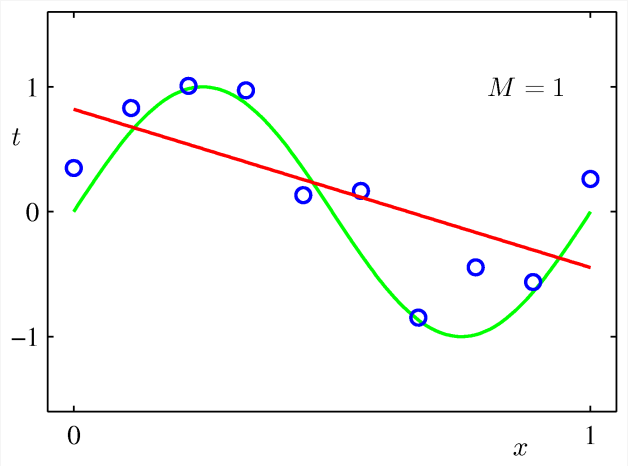
   - The k^{th} input vector: $X_k = (1, x_{k1}, x_{k2}, \dots, x_{kD})^T$
   - The weight vector: $W = (w_0, w_1, w_2, \dots, w_D)^T$
   - The target vector: $t = (t_1, t_2, t_3, \dots, t_N)^T$
2. The empirical error equals to: $E_E(g(x)|S) = \frac{1}{2}\sum_{k=1}^{N}\left(t_k - W^T X_k\right)^2$.
3. The gradient of $E_E(g(x)|S)$ equals to $\nabla_W E_E(g(x)|S) = \sum_{k=1}^{N} t_k X_k^T - W^T \sum_{k=1}^{N} X_k X_k^T = 0$
4. Solving for W, we obtain $W^* = \left(X^T X\right)^{-1} X^T t$

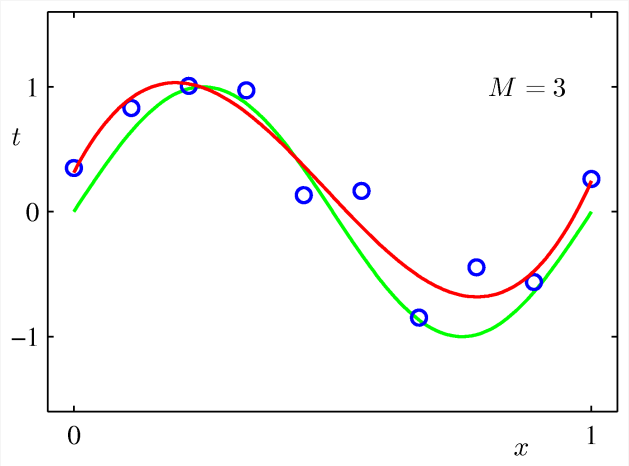
Model selection

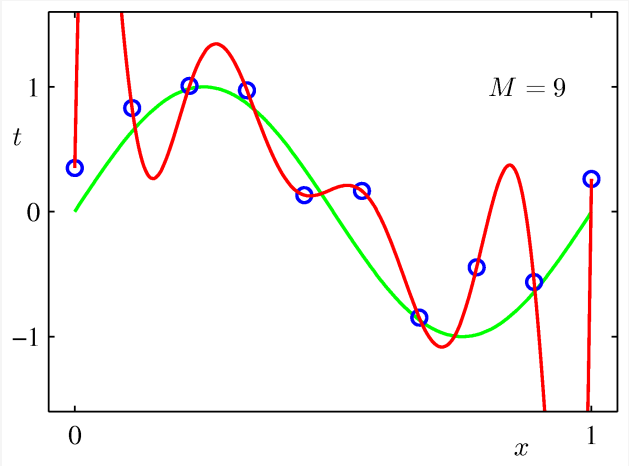
1. If the linear model is too simple, the model can be a polynomial (a more complex hypothesis set)

$$g(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M.$$

2. M is the order of the polynomial.
3. Choosing the right value of M is called model selection.







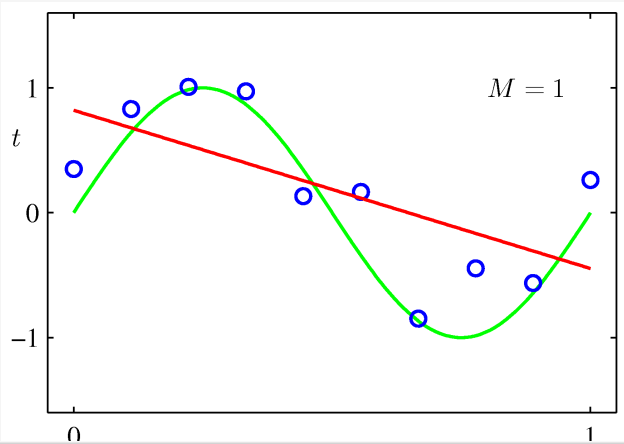
4. For $M = 1$, we have a too general model
5. For $M = 9$, we have a too specific model

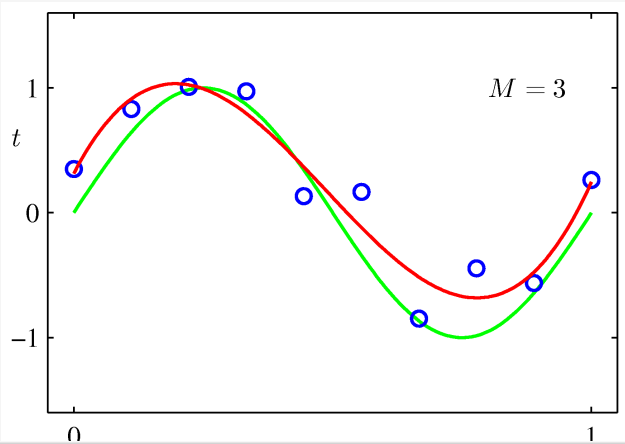
Training vs Testing error

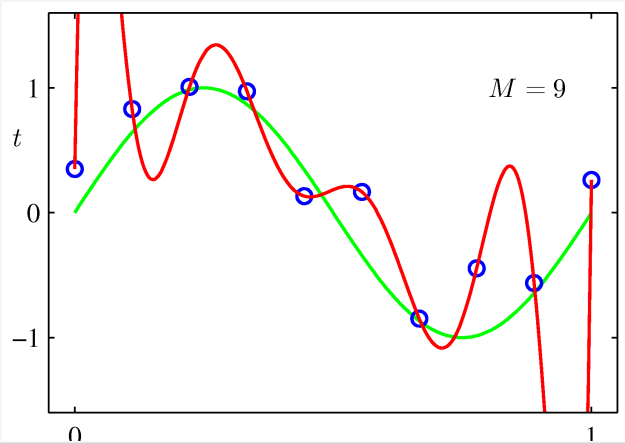
1. Given a new data point x, we would like to understand the expected prediction error
$$\mathbb{E}\left[(t - g(x))^2\right]$$
2. Assume that x generated by the same process as the training set. We decompose $\mathbb{E}\left[(t - g(x))^2\right]$ into bias, variance, and noise.
$$\mathbb{E}\left[(t - g(x))^2\right] = Bias^2(g(x)) + Var(g(x)) + \sigma^2$$
3. The first term describes the average error of $g(x)$.
4. The second term quantifies how much $g(x)$ deviates from one training set S to another one. This depends on both the estimator and the training set.This term is consequence of over-fitting.
5. The last term is the variance of the added noise. This error cannot be removed no matter what estimator we use. Note that the variance of the noise can not be minimized.

Training vs Testing error

1. In regression, as M increases, a small changes in the data sets causes a greater change in the fitted function; thus variance increases.
2. The goal is to minimize the expected loss. There is a trade-off between bias and variance.
   - Very flexible models have low bias and high variance.
   - Relative rigid models have high bias and low variance.
3. The model with optimal predictive capability is one that leads to the best balance between bias and variance.
4. If there is bias, there is under-fitting. why?
5. If there is variance, there is over-fitting. why?

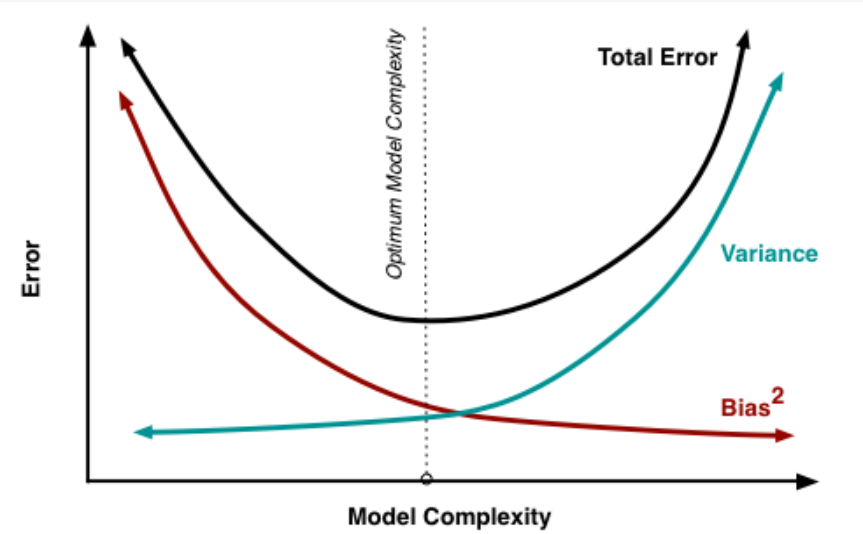






Bias-Variance trade-off

1. Consider bias-variance of a model.



2. The problem is how to balance between bias and variance.
3. Some solutions
   - Trial and error using validation dataset
   - Regularization
   - ⋮

Table of contents

Regularization

1. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.
2. In this manner, regularization is used to improve the generalization of the model. Other intuitions:
   - Regularization tends to increase the estimator bias while reducing the estimator variance.
   - Regularization can be seen as a way to prevent overfitting.
   - A common problem is in picking the model size and complexity. It may be appropriate to simply choose a large model that is regularized appropriately.
3. Regularization can also be studied from view point of statistical learning theory

Types of regularization

1. Regularizations may take on many different types of forms. The following list is not exhaustive, but includes regularizations one may consider.
2. It may be appropriate to add a soft constraint on the parameter values in the objective function.
   - To account for prior knowledge (e.g., that the parameters have a bias).
   - To prefer simpler model classes that promote generalization.
   - To make an under-determined problem determined. (e.g., least squares with indeterminate $X^T X$.)
3. Dataset augmentation
4. Ensemble methods (i.e., essentially combining the output of several models).
5. Some training algorithms (e.g., stopping training early, dropout) can be seen as a type of regularization.

Stopping early

1. A straightforward (and popular) way to regularize is to constantly evaluate the training and validation loss on each training iteration, and return the model with the lowest validation error.
   - Requires caching the lowest validation error model.
   - Training will stop when after a pre-specified number of iterations, no model has decreased the validation error.
   - The number of training steps can be thought of as another hyper-parameter.
   - Validation set error can be evaluated in parallel to training.
   - It doesnt require changing the model or cost function.

Regularization via parameter norm penalties

1. A common (and simple to implement) type of regularization is to modify the cost function with a parameter norm penalty. This penalty is typically denoted as $\Omega(\theta)$ and results in a new cost function of the form:

$$\bar{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

with $\alpha \geq 0$

- α is a hyper-parameter that weights the contribution of the norm penalty.
- When $\alpha = 0$, there is no regularization.
- When$\alpha \to \infty$, the cost function is irrelevant and the model will set the parameters to minimize $\Omega(\theta)$.
- The choice of can strongly affect generalization performance.
- When regularizing parameters, we typically do not regularize biases, since they do not introduce substantial variance to the estimator.

L_2 regularization

1. A common form of parameter norm regularization is to penalize the size of the weights, which is also called ridge regression or Tikhonov regularization. Let θ is union of w and bias.

$$\Omega(\theta) = \frac{1}{2} w^T w$$

2. To prevent $\Omega(\theta)$ from getting large, L_2 regularization will cause the weights w to have small norm.
3. The new cost function is

$$\bar{J}(\theta) = J(\theta) + \frac{\alpha}{2} w^T w$$

4. Its gradient equals to

$$\nabla_w \bar{J}(\theta) = \nabla_w J(\theta) + \alpha w$$

5. Parameters are updated using

$$w^{(t+1)} = (1 - \eta\alpha) w^{(t)} - \eta \nabla_w J(\theta)$$

L_2 regularization

1. In linear regression, the least squares solution $w = (X^T X)^{-1} X^T y$ becomes:

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

2. In linear regression, as M increases, the magnitude of the coefficients typically gets larger.

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Extensions of L_2 regularization

1. Other related forms of L_2 regularization include:
   1. Instead of a soft constraint that w be small, one may have prior knowledge that w is close to some value b. Then, the regularizer may take the form:
   $$\Omega(\theta) = \|w - b\|_2.$$
   2. One may have prior knowledge that two parameters, $w^{(1)}$ and $w^{(2)}$, must be close to each other. Then, the regularizer may take the form:
   $$\Omega(\theta) = \|w^{(1)} - w^{(2)}\|_2.$$

L_1 regularization

1. L_1 regularization defines the parameter norm penalty as

$$\Omega(\theta) = \|w\|_1$$

2. This penalty also causes the weights to be small.
3. The new cost function is

$$\bar{J}(\theta) = J(\theta) + \alpha w$$

4. Its gradient equals to

$$\nabla_w \bar{J}(\theta) = \nabla_w J(\theta) + \alpha sign(w)$$

5. Empirically, this typically results in sparse solutions where $w_i = 0$ for several i.
6. This may be used for feature selection, where features corresponding to zero weights may be discarded.

Regularization

- L_2-Regularization has Closed form solution and can be solved in polynomial time

$$E_E(g(x)|S) = \frac{1}{2}\sum_{i=1}^{N}(t_i - g(x_i))^2 + \frac{\lambda}{2}||W||^2.$$

- L_1-Regularization can be approximated in polynomial time

$$E_E(g(x)|S) = \frac{1}{2}\sum_{i=1}^{N}(t_i - g(x_i))^2 + \lambda||W||_1.$$

- L_0-Regularization is NP-complete optimization problem

$$E_E(g(x)|S) = \frac{1}{2}\sum_{i=1}^{N}(t_i - g(x_i))^2 + \lambda\sum_{j=1}^{M-1}\delta(w_j \neq 0).$$

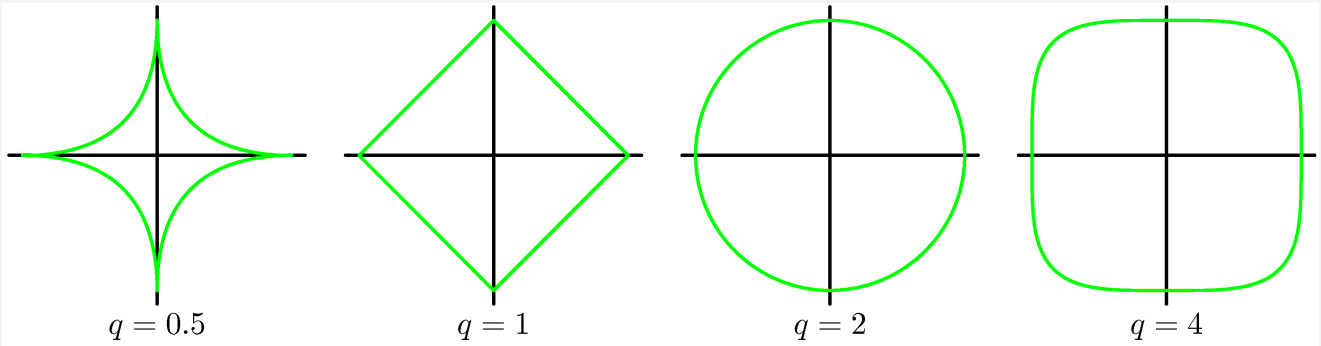
The L_0-norm represents the optimal subset of features needed by a Regression model.

Regularization

- A more general regularizer is sometimes used, for which the regularized error takes the form

$$E_E(g(x)|S) = \frac{1}{2}\sum_{i=1}^{N}(t_i - g(x_i))^2 + \frac{\lambda}{2}\sum_{j=1}^{M-1}|w_j|^q.$$

- When $q = 2$, it is called Ridge regularizer
- When $q = 1$, it is called Lasso regularizer



Regularization and prior knowledge

- Assume that w has distribution of

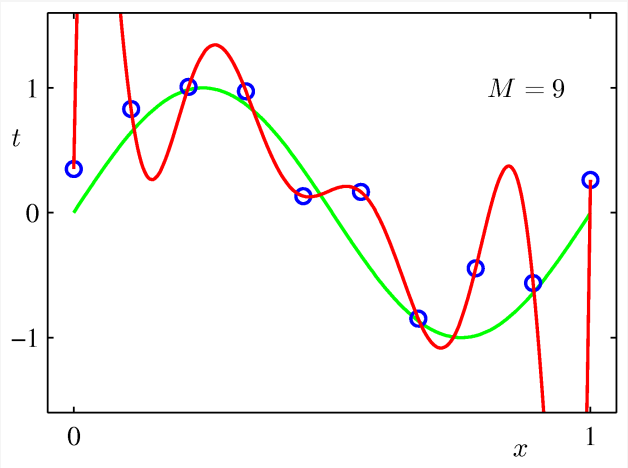
$$\mathbb{P}(w) = \mathcal{N}(0, \sigma^2 I_D).$$

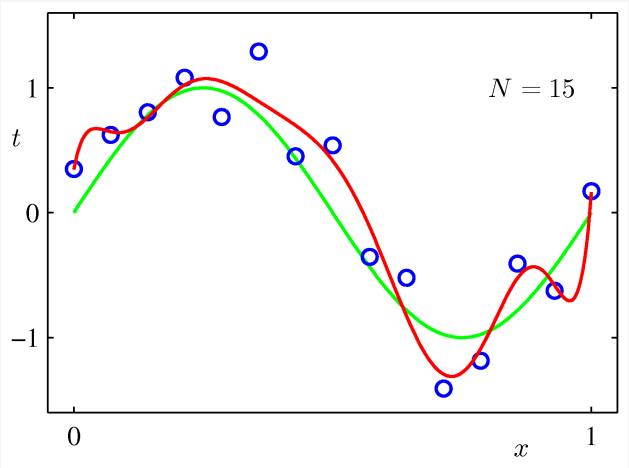
- Posterior density of w given set S results in L_2-regularization.
- Assume that w has isotropic Laplace distribution Posterior density of w given set S results in L_1-regularization.
- What about other types of regularizes?

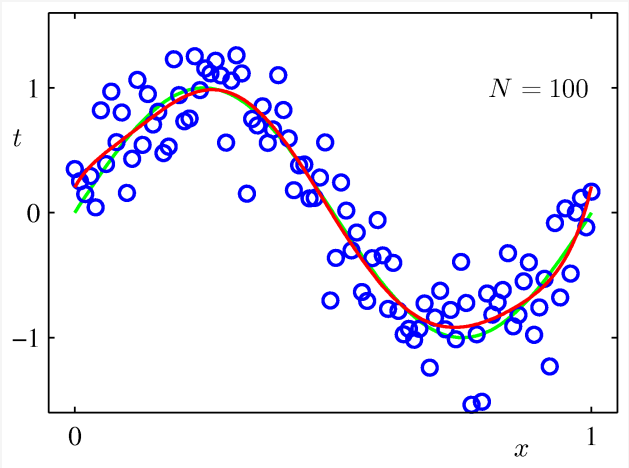
Table of contents

Dataset augmentation

1. Can the more training data prevent the model from over-fitting.
2. For a given model complexity, the over-fitting problem become less severe as the size of the data set increases.







Dataset augmentation

1. Best way to make a ML model to generalize better is to train it on more data
2. In practice amount of data is limited
3. Get around the problem by creating synthesized data
4. For some ML tasks it is straightforward to synthesize data

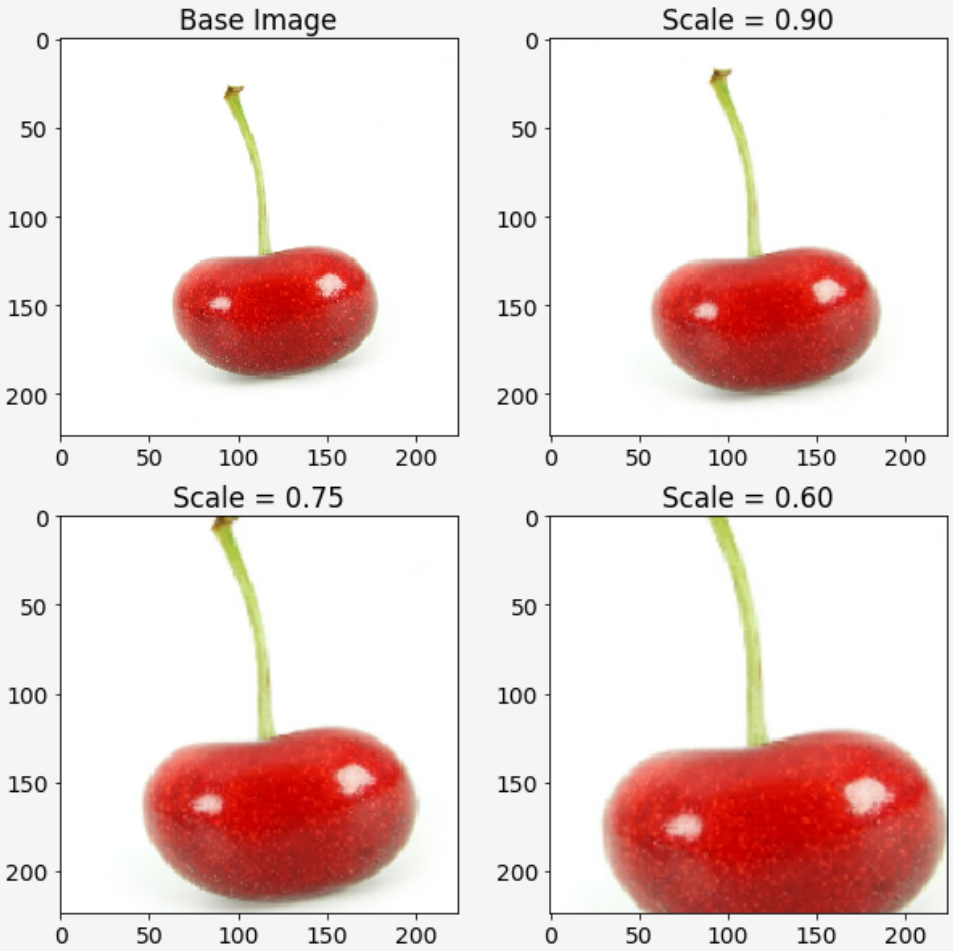
Dataset augmentation for classification

1. Data augmentation is easiest for classification
   - Classifier takes high-dimensional input x and summarizes it with a single category identity y
   - Main task of classifier is to be invariant to a wide variety of transformations
2. Generate new samples (x, y) just by transforming inputs
3. Approach not easily generalized to other problems

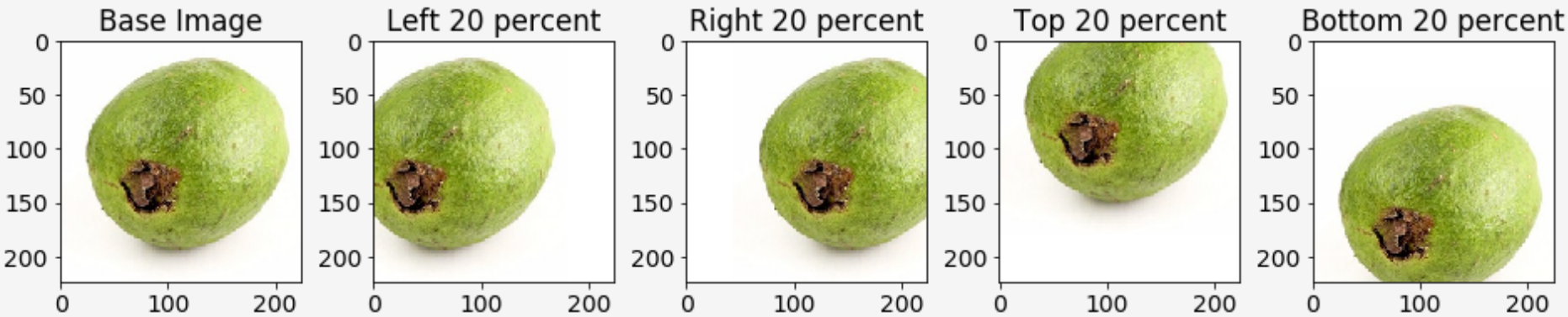
Dataset augmentation for object recognition

1. Data set augmentation very effective for the classification problem of object recognition
2. Images are high-dimensional and include a variety of variations, may easily simulated
3. Translating the images a few pixels can greatly improve performance
4. Rotating and scaling are also effective
   - Considering flipping between b and d.
   - Considering rotation between 6 and 9.

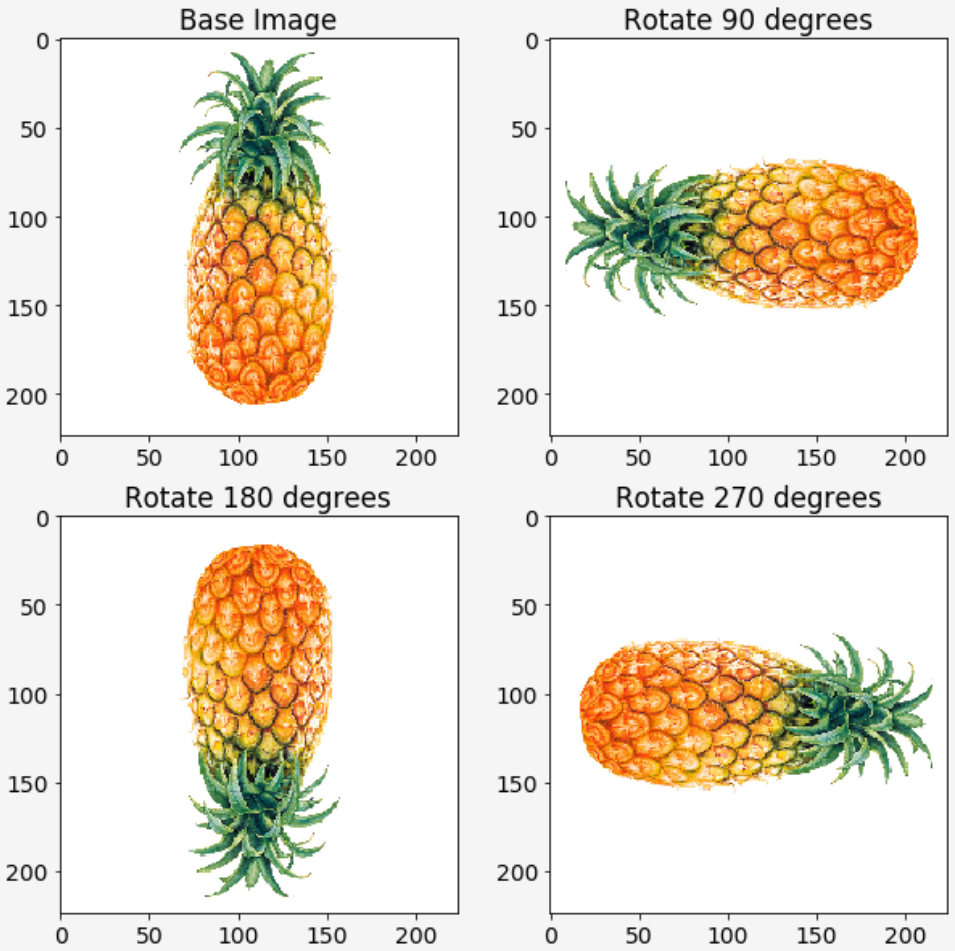
Dataset augmentation for Image processing (Scaling)



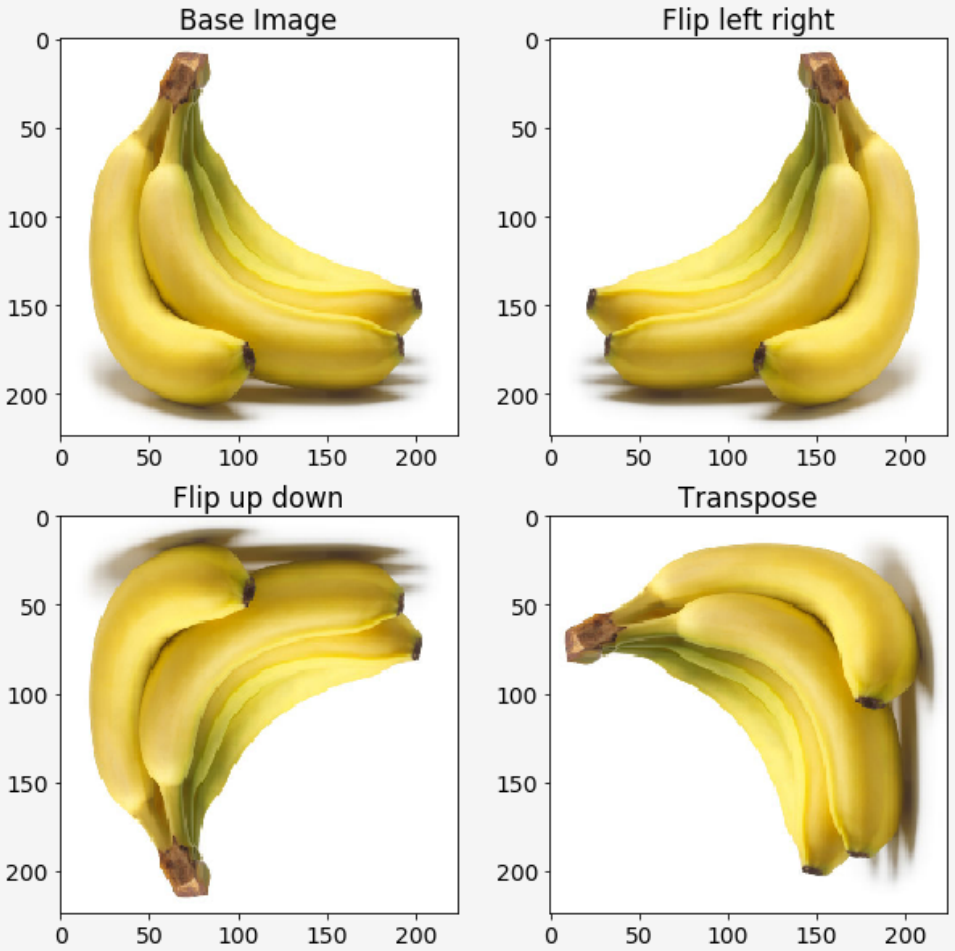
Dataset augmentation for Image processing (Translation)



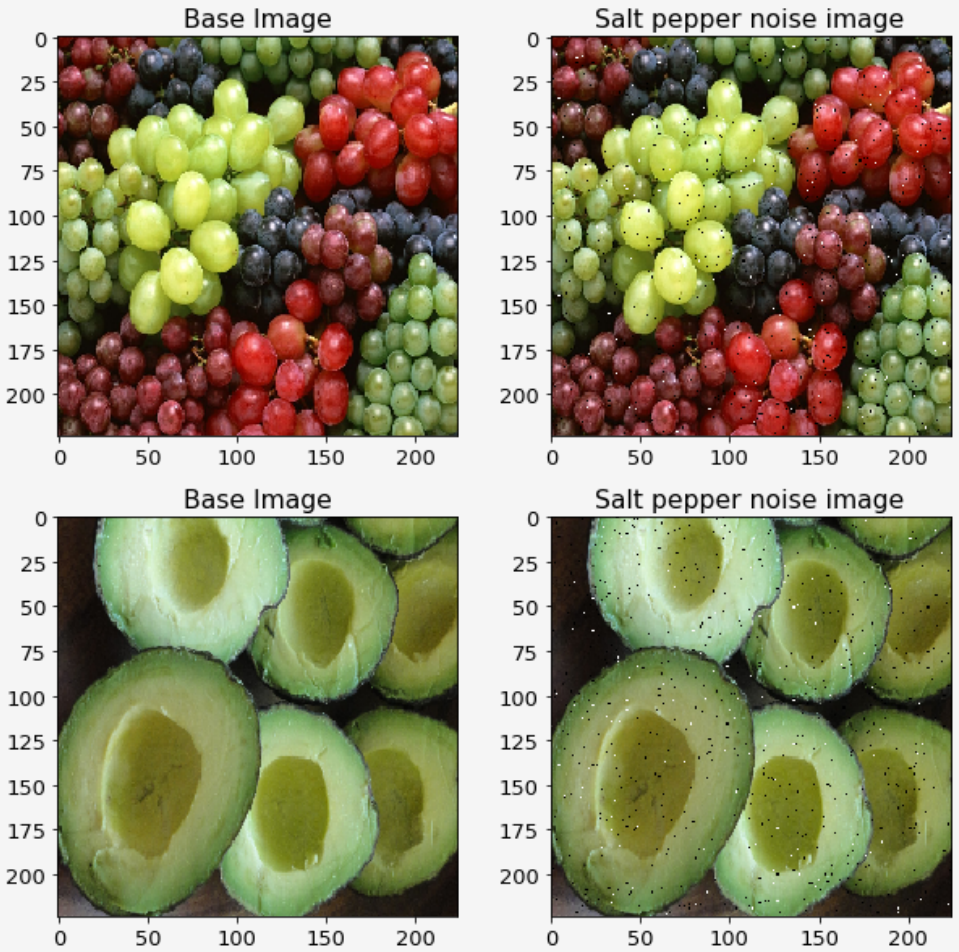
Dataset augmentation for Image processing (Rotation)



Dataset augmentation for Image processing (Flipping)

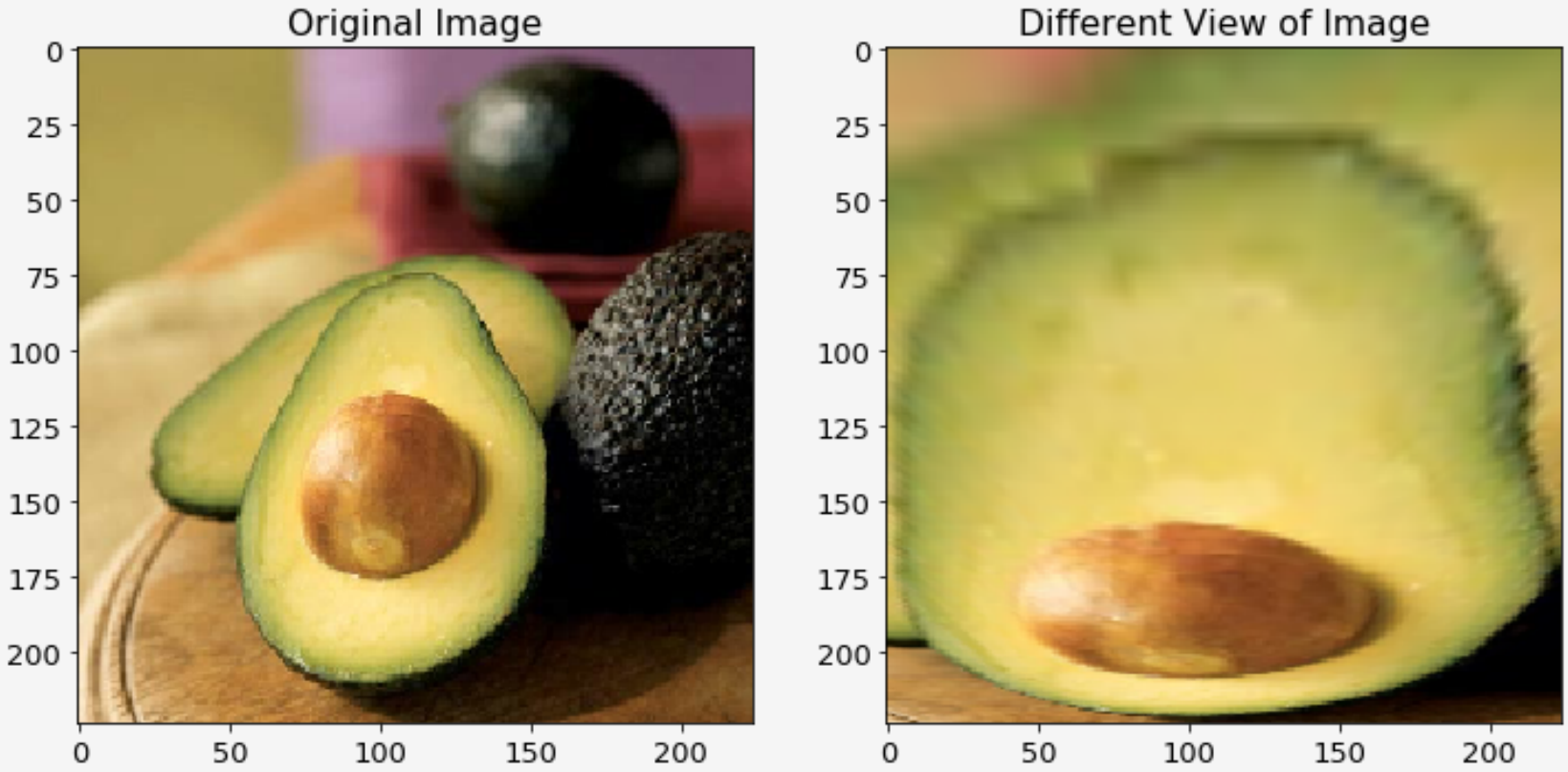


Dataset augmentation for Image processing (Adding Salt and Pepper noise)



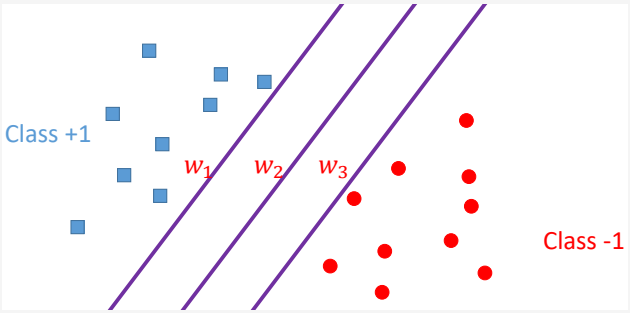
Dataset augmentation for Image processing (Lighting condition)

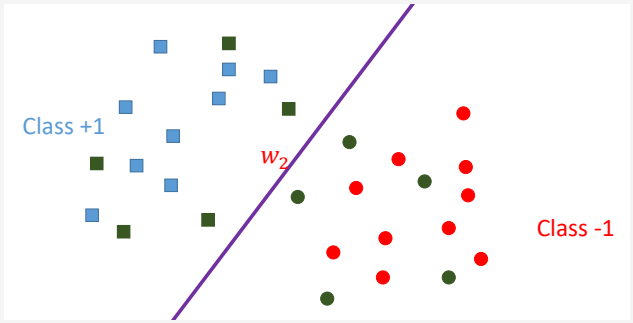
Dataset augmentation for Image processing (Perspective transform)

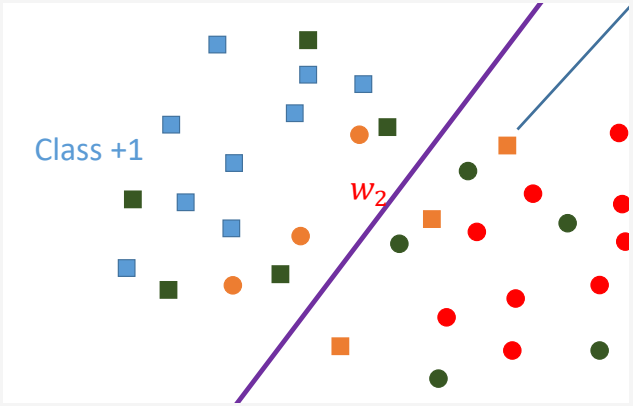


Noise injection

1. Noise injection to the input can be seen as a form of data augmentation.
2. Neural networks are robust to noise.
3. The robustness of neural networks can be improved by adding noise to inputs and outputs of hidden units.
4. Caution for using noise







Dataset augmentation for Image processing (results)

The result of dataset augmentation on Caltech101 dataset[6].

	Top-1 Accuracy	Top-5 Accuracy
Baseline	$48.13 \pm 0.42\%$	$64.50 \pm 0.65\%$
Flipping	$49.73 \pm 1.13\%$	$67.36 \pm 1.38\%$
Rotating	$50.80 \pm 0.63\%$	$69.41 \pm 0.48\%$
Cropping	$\mathbf{61.95 \pm 1.01\%}$	$\mathbf{79.10 \pm 0.80\%}$
Color Jittering	$\mathbf{49.57 \pm 0.53\%}$	$67.18 \pm 0.42\%$
Edge Enhancement	$49.29 \pm 1.16\%$	$66.49 \pm 0.84\%$
Fancy PCA	$49.41 \pm 0.84\%$	$\mathbf{67.54 \pm 1.01\%}$

Top-1 score is the number of times the highest probability is associated with the correct target over all testing images.
Top-5 score is the number of times the correct label is contained within the 5 highest probabilities.

[6] Luke Taylor and Geoff Nitschke, "Improving Deep Learning using Generic Data Augmentation", Arxiv, 2017.

Dataset augmentation for Image processing

1. For more data augmentation techniques used in deep learning for image data, please read the following paper.
   Connor Shorten and Taghi M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning", Journal of Big Data, 2019.
2. This paper can be downloaded from the following url.
   https://link.springer.com/article/10.1186/s40537-019-0197-0
3. and
   Alex Hernandez-Garcia and Peter Konig, "Data augmentation instead of explicit regularization", Arxiv, 2019.

Adding noise is equivalent to weight decay

1. Suppose we add noise to input: $x + \epsilon$
2. Suppose the hypothesis is $h(x) = w^T x$, noise is $\epsilon \sim \mathcal{N}(0, \lambda I)$
3. Before adding noise, the loss is

$$J(\theta) = \mathbb{E}_{x,y}(f(x) - y)^2$$

4. After adding noise, the loss is

$$J(\theta) = \mathbb{E}_{x,y,\epsilon}(f(x + \epsilon) - y)^2$$

5. Simplifying the above equation yields to

$$J(\theta) = \mathbb{E}_{x,y,\epsilon}(f(x) - y)^2 + \lambda \|w\|^2$$

6. This is equivalent to weight decay (L_2 regularization)

Adding noise to labels

1. Many datasets has mistakes in label y.
2. In this case maximizing $-\log p(y|x)$ is harmful.
3. We can assume that for small constant ϵ, the training label is correct or vise versa.
4. Extending the above case to k output values is label smoothing[7].

[7] Rafael Muller, Simon Kornblith, and Geoffrey Hinton, "When Does Label Smoothing Help?", Arxiv, 2019.

Adding noise to weights

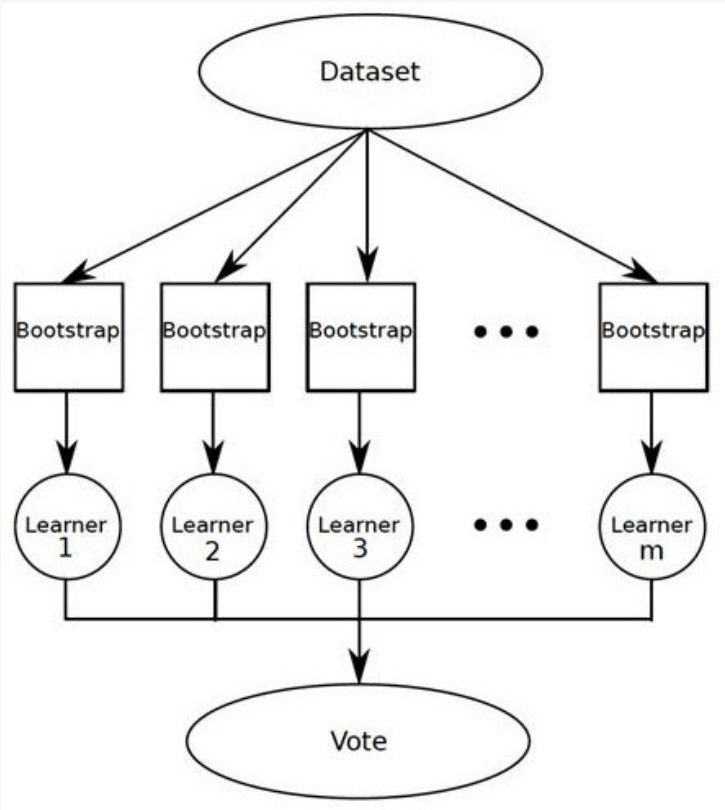
1. This technique primarily used with RNNs
2. This can be interpreted as a stochastic implementation of Bayesian inference over the weights
3. Adding noise to weights is a practical, stochastic way to reflect this uncertainty
4. Noise applied to weights is equivalent to traditional regularization, encouraging stability.

Table of contents

Bagging

1. It is short for Bootstrap Aggregating
2. It is a technique for reducing generalization error by combining several models
   Idea is to train several models separately, then have all the models vote on the output for test examples
3. This strategy is called model averaging
4. Techniques employing this strategy are known as ensemble methods.
5. Model averaging works because different models will not make the same mistake

Bagging


Dataset
Bootstrap
Bootstrap
Bootstrap
Bootstrap
Learner 1
Learner 2
Learner 3
Learner m
Vote

Bagging

Consider the set of k regression models

1. Each model i makes error ϵ_i on each example
2. Errors drawn from a zero-mean multivariate normal with variance $\mathbb{E}[\epsilon_i^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$
3. Error of average prediction of all ensemble models: $\frac{1}{k}\sum_i \epsilon_i$
4. Expected squared error of ensemble prediction is

$$\mathbb{E}\left[\frac{1}{k}\sum_i \epsilon_i\right]^2 = \frac{1}{k}v + \frac{k-1}{k}c$$

5. If errors are perfectly correlated, $c = v$, and mean squared error reduces to v, so model averaging does not help.
6. If errors are perfectly uncorrelated and $c = 0$, expected squared error of ensemble is only $\frac{v}{k}$ and Ensemble error decreases linearly with ensemble size

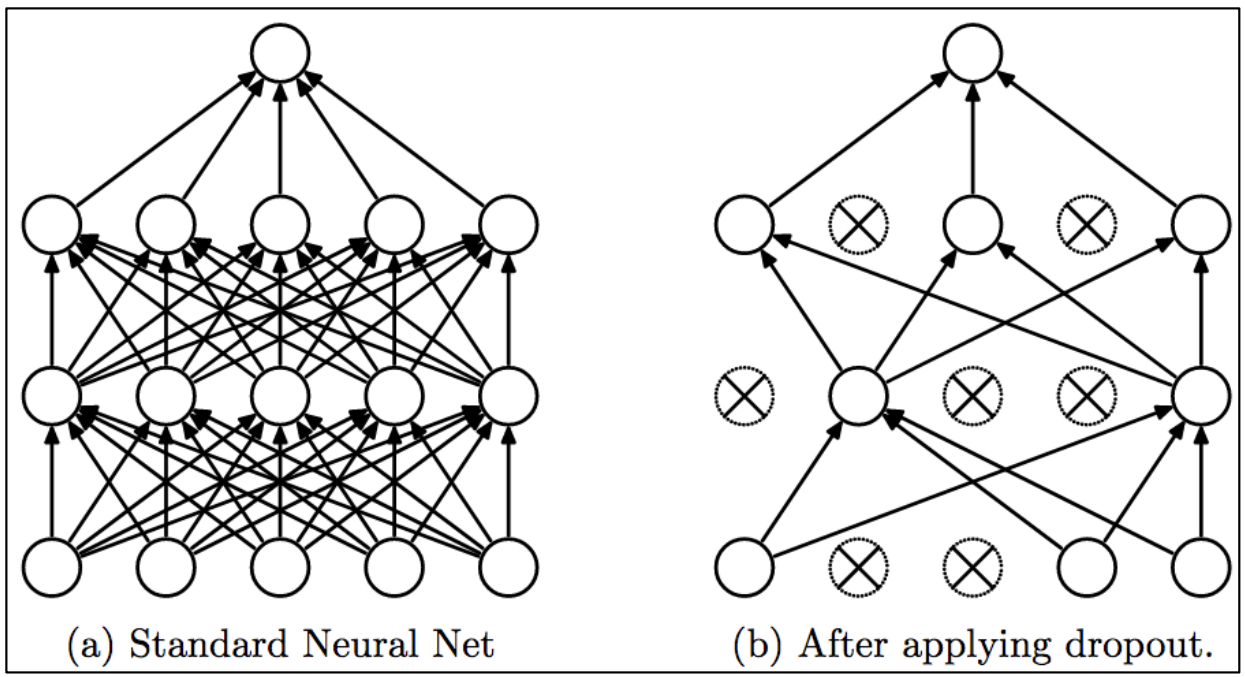
Table of contents

Dropout

1. Bagging is a method of averaging over several models to improve generalization
2. How do you apply the bagging to neural networks?
3. Impractical to train many neural networks since it is expensive in time and memory
4. Dropout makes it practical to apply bagging to very many large neural networks
5. It is a method of bagging applied to neural networks
6. Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout

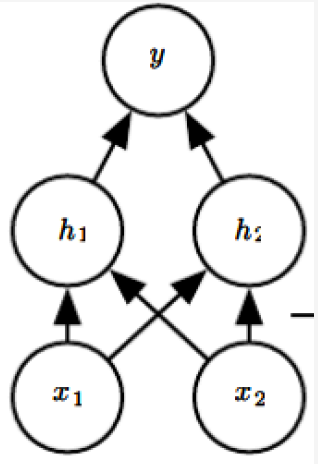
1. Dropout trains an ensemble of all subnetworks
2. Sub-networks formed by removing non-output units from an underlying base network
3. We can effectively remove units by multiplying its output value by zero



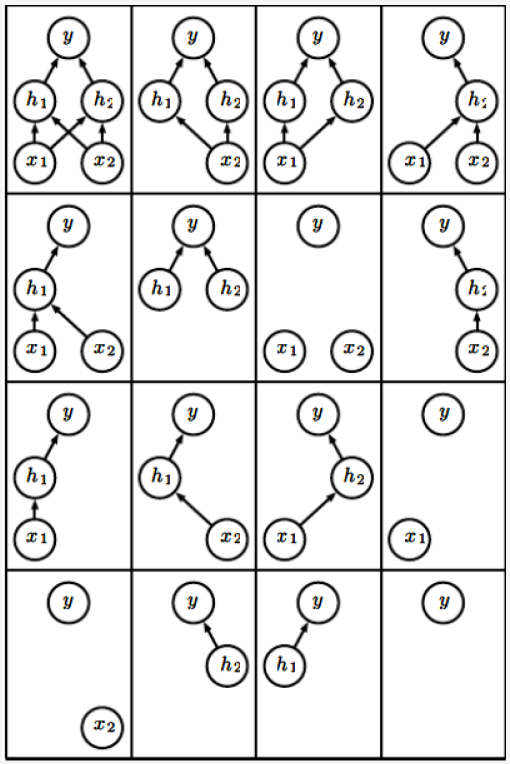
(a) Standard Neural Net (b) After applying dropout.

Dropout as bagging

1. In bagging we define k different models, construct k different data sets by sampling from the dataset with replacement, and train model i on dataset i
2. Dropout aims to approximate this process, but with an exponentially large number of neural networks



=



Mask for dropout training

1. To train with dropout we use mini-batch based learning algorithm that takes small steps such as SGD
2. At each step randomly sample a binary mask
3. Probability of including a unit is a hyper-parameter: 0.5 for hidden units and 0.8 for input units
4. We run forward and backward propagation as usual

Dropout prediction

1. Each sub-model defines mask vector μ defines a probability distribution $p(y|x,\mu)$
2. Dropout prediction is
$$\sum_{\mu} p(y|x,\mu)$$
3. It is intractable to evaluate due to an exponential number of terms
4. We can approximate inference using sampling
5. By averaging together the output from many masks
6. 10-20 masks are sufficient for good performance
7. Even better approach, at the cost of a single forward propagation

Other approaches for increasing generalization

1. Multi-task learning
2. Semi-supervised learning
3. other ensemble methods such as boosting?

What regularizations are frequently used?

1. L_2 regularization
2. Early stopping
3. Dropout
4. Data augmentation if the transformations known/easy to implement

Table of contents

Reading

Please read chapter 7 of Deep Learning Book.