Machine Learning (part II)

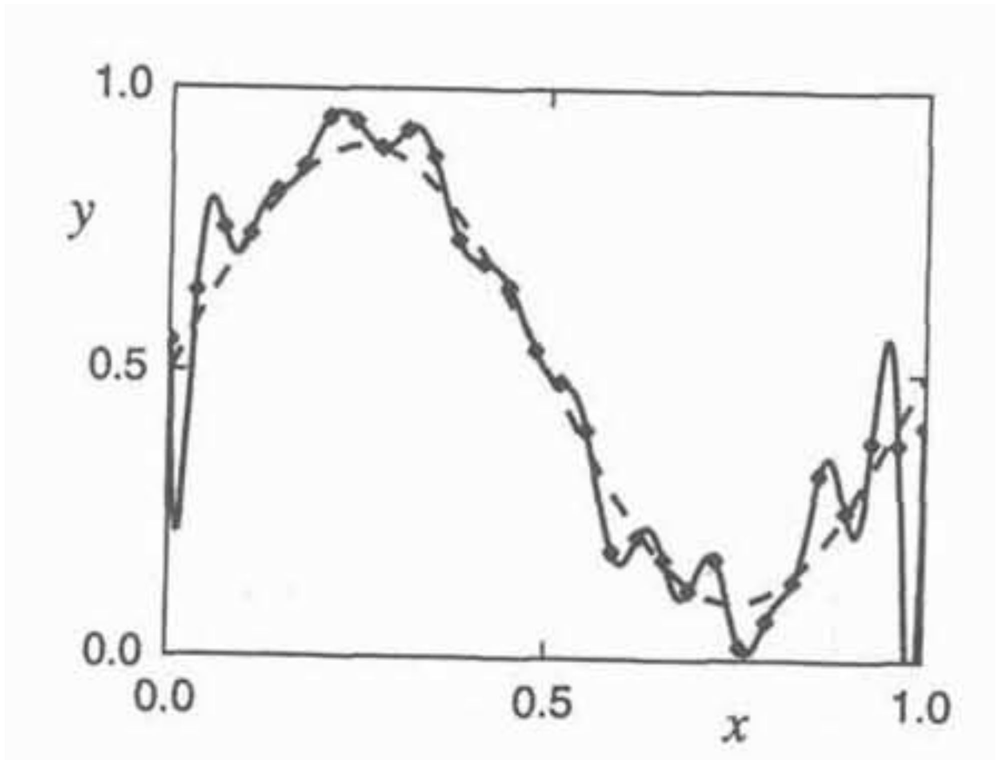
Regularization for NNs

Angelo Ciaramella

Introduction

- Problem
  - Generalization
  - How to make an algorithm that will perform well not just on the training data

overfitting



Introduction

- Generalization
  - Bias-variance trade-off
    - Model simple and inflexible – large bias
    - Model too much flexibility – large variance
  - Controlling the effective complexity of the model
    - NNs – number of adaptive parameters
- Regularization
  - Controlling the complexity of the model
  - Addition of a penality term
  - Cross-validation

Bias and variance

- MLP
  - Sum-of-squares error function
  - Single output

- In the limit of an infinite data set

$$E = \frac{1}{2}\int \{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2 p(\mathbf{x})\,d\mathbf{x}$$
$$+ \frac{1}{2}\int \{\langle t^2|\mathbf{x}\rangle - \langle t|\mathbf{x}\rangle^2\} p(\mathbf{x})\,d\mathbf{x}$$

$$\langle t|\mathbf{x}\rangle \equiv \int t p(t|\mathbf{x})\,dt$$ conditional average or regression

Bias and variance

- Practical situation
  - finite training set *D* of *N* patterns

- The error depends on the particular data set

$$\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2$$

- Eliminating this dependence by average pver the complete ensemble of data sets

$$\mathcal{E}_D[\{y(\mathbf{x}) - \langle t|\mathbf{x}\rangle\}^2]$$

Expectation or ensemble average

Bias and variance

- Bias and variance

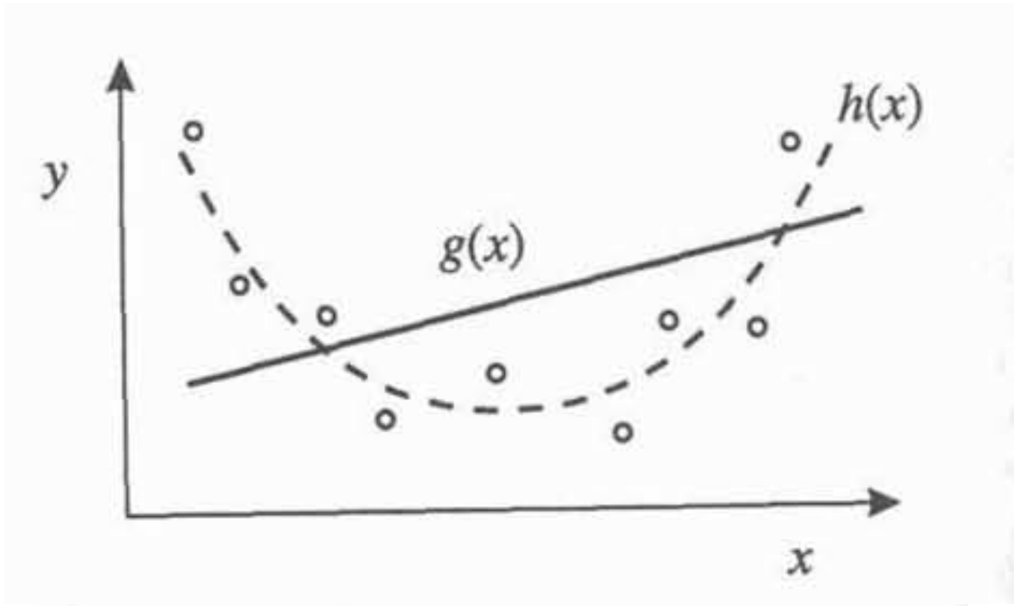
$$(\text{bias})^2 = \frac{1}{2}\int \{\mathcal{E}_D[y(\mathbf{x})] - \langle t|\mathbf{x}\rangle\}^2 p(\mathbf{x})\,d\mathbf{x}$$

$$\text{variance} = \frac{1}{2}\int \mathcal{E}_D[\{y(\mathbf{x}) - \mathcal{E}_D[y(\mathbf{x})]\}^2]p(\mathbf{x})\,d\mathbf{x}.$$

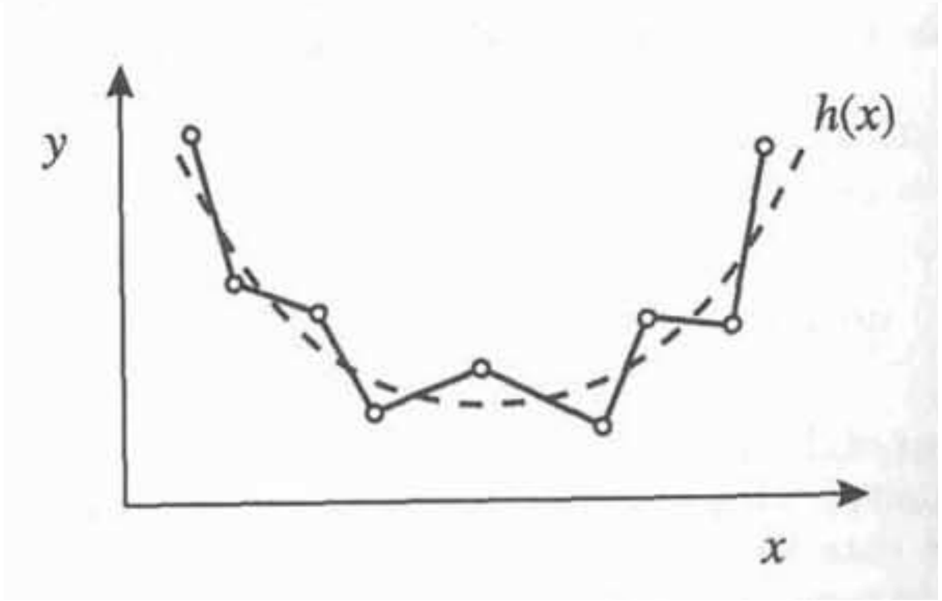
- Consider

$$t^n = h(\mathbf{x}^n) + \epsilon^n.$$

Bias and variance



High bias and low variance



Low bias and high variance

Regularization

- Regularized objective function

Penality term

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

$$\alpha \in [0, \infty)$$

Hyperparameter

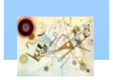
L_2 regularization

- Ridge regression (or Tikhonov regularization)
  - Drives weights closer to the origin

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

- Objective function

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}),$$



L_2 regularization

- By mean squared error, the approximation is

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- Minimum

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

L_2 regularization

- Gradient

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})$$

- Learning

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon\left(\alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y})\right)$$

L_2 regularization

- Consider the regularized error

$$\widetilde{E} = E + \nu\Omega. \qquad \nabla\widetilde{E} = \nabla E + \nu\mathbf{w}.$$

- Weight decay

$$\Omega = \frac{1}{2}\sum_i w_i^2$$

- Weight evolves

$$\frac{d\mathbf{w}}{d\tau} = -\eta\nabla E = -\eta\nu\mathbf{w} \qquad \mathbf{w}(\tau) = \mathbf{w}(0)\exp(-\eta\nu\tau)$$

L_2 regularization

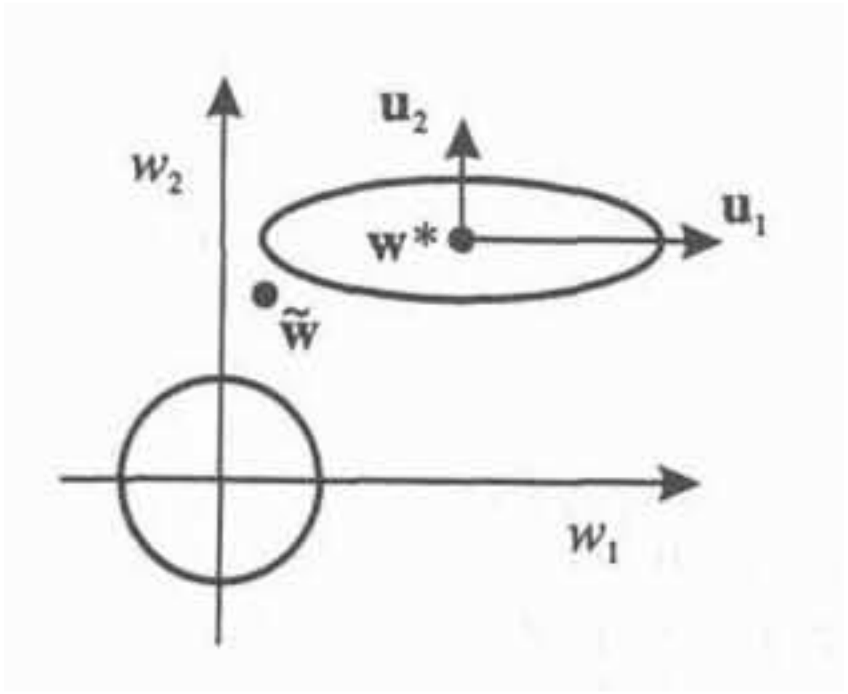
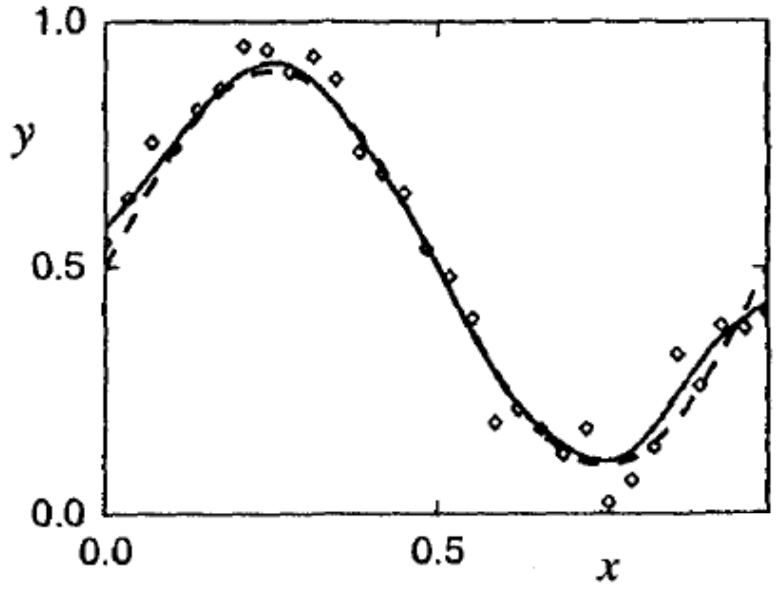
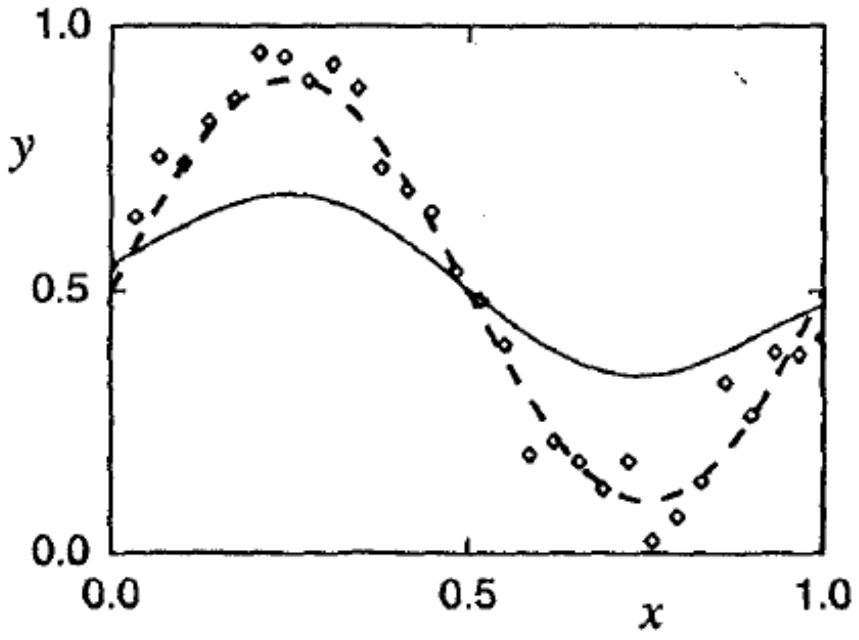


Illustration of the effect of a simple weight-decay regularizer on a quadratic error function

L$_2$ regularization



Regularization parameter = 40



Regularization parameter = 1000

L_1 regularization

- Absolute value

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

- Objective function

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

- Gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w})$$

L_1 regularization

- Approximation

$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha |w_i|\right]$$

- Analytical solution

$$w_i = \operatorname{sign}(w_i^*) \max\left\{|w_i^*| - \frac{\alpha}{H_{i,i}}, 0\right\}$$

- Sparse solution

Constrained optimization

- Minimize a function subject to constraints by constructing a generalized Lagrange function

- Each penalty is a product between a coefficient
  - called a Karush–Kuhn–Tucker (KKT) multiplier

- Contrain the penalty to be less than some constant k

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k).$$

- Solution

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \alpha).$$

Generalized Lagrangian

- Constraints

$$\mathbb{S} = \{\boldsymbol{x} \mid \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ and } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0\}$$

equality inequality

- Generalized Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x})$$

- Minnimization

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) \qquad \min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x})$$

Generalized Lagrangian

- Any time the constraints are satisfied

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = f(\boldsymbol{x})$$

- Any time the constraint is violated

$$\max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha}) = \infty$$

Training with noise

- Addition of noise to the input vectors during the learning process
  - It has demonstrated that can indeed lead to improvements in network generalization
  - Closely related to the technique of regulraization
  - Reduce over-fitting

- Considering random error

$$\widetilde{E} = \frac{1}{2}\sum_k \iiint \{y_k(\mathbf{x}+\xi) - t_k\}^2 p(t_k|\mathbf{x})p(\mathbf{x})\widetilde{p}(\xi)\, d\mathbf{x}\, dt_k\, d\xi.$$

Training with noise

- Taylor series

$$y_k(\mathbf{x}+\boldsymbol{\xi}) = y_k(\mathbf{x}) + \sum_i \xi_i \left.\frac{\partial y_k}{\partial x_i}\right|_{\boldsymbol{\xi}=0} + \frac{1}{2}\sum_i \sum_j \xi_i \xi_j \left.\frac{\partial^2 y_k}{\partial x_i \partial x_j}\right|_{\boldsymbol{\xi}=0} + \mathcal{O}(\boldsymbol{\xi}^3).$$

$$\int \xi_i \widetilde{p}(\boldsymbol{\xi})\, d\boldsymbol{\xi} = 0 \qquad \int \xi_i \xi_j \widetilde{p}(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \nu \delta_{ij}$$

- Integrating

$$\widetilde{E} = E + \nu\Omega$$

Training with noise

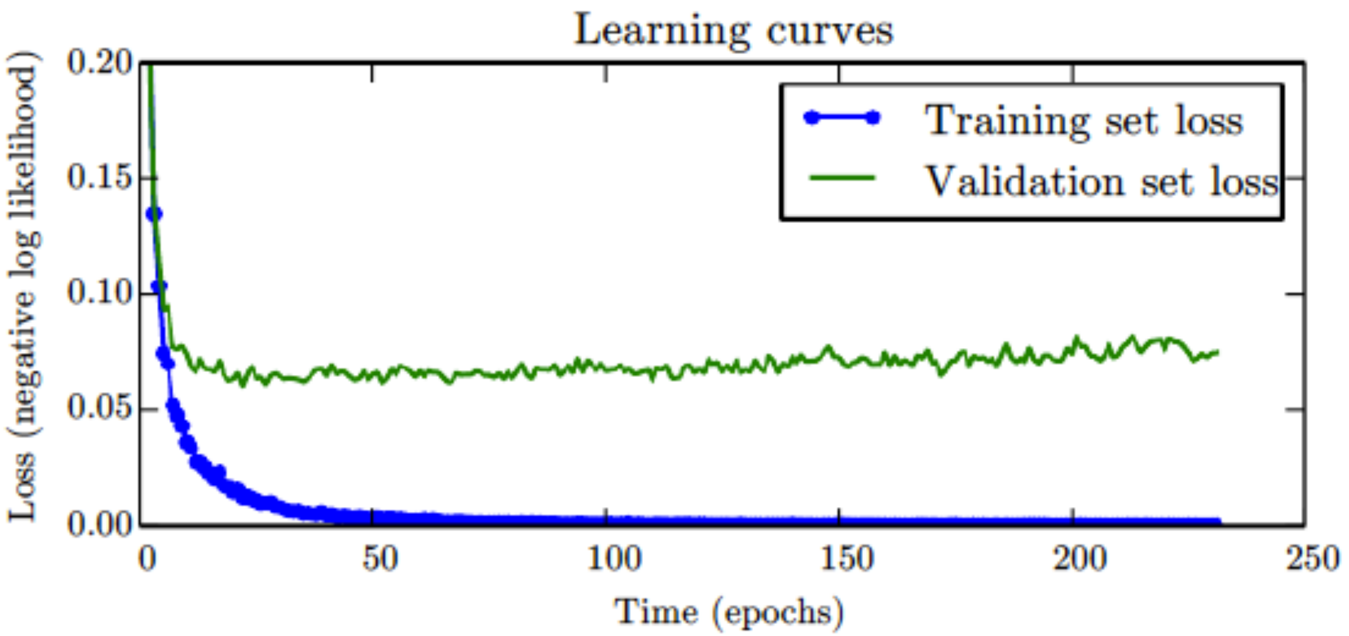
- Goal
  - addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights
  - Noise applied to the hidden units is an important topic
  - Adding the noise to the weights
    - Recurrent NNs
  - Injecting Noise at the Output Targets
    - explicitly model the noise on the labels

Semi-Supervised Learning

- Goal
  - learning a representation so that examples from the same class have similar representations
  - application of principal components analysis as a pre-processing step before applying a classifier

Early Stopping

- Training large models
  - training error decreases
  - validation set error rising



Early Stopping

- Goal
  - running our optimization algorithm until the error on the validation set has not improved for some amount of time
  - number of training steps is an hyperparameter

Early Stopping

- Strategies
  - initialize the model again and retrain on all of the data
    - we train for the same number of steps as the early stopping procedure determined in the first pass
  - keep the parameters obtained from the first round of training
    - continue training but now using all of the data

Parameter sharing

- Strategies
  - the parameters of one model trained as a classifier in a supervised paradigm to be close to the parameters of another model, trained in an unsupervised paradigm
  - to force sets of parameters to be equal
    - we interpret the various models or model components as sharing a unique set of parameters

Sparse representations

- Strategy
  - adding to the loss function J a norm penalty on the representation

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

$$\Omega(\boldsymbol{h}) = ||\boldsymbol{h}||_1 = \sum_i |h_i|$$

  - Orthogonal matching pursuit

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

Bagging

- Bagging (Bootstrap Aggregating)
  - reducing generalization error by combining several models
  - ensemble methods
  - bagging involves constructing k different datasets
    - same number of examples as the original dataset
    - constructed by sampling with replacement from the original dataset
    - Model i is then trained on dataset i

- Boosting
  - constructs an ensemble with higher capacity than the individual models
  - incrementally adding neural networks to the ensemble

Bagging

- Expected squared error (k regression models)

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{j\neq i} \epsilon_i \epsilon_j\right)\right]$$
$$= \frac{1}{k}v + \frac{k-1}{k}c.$$

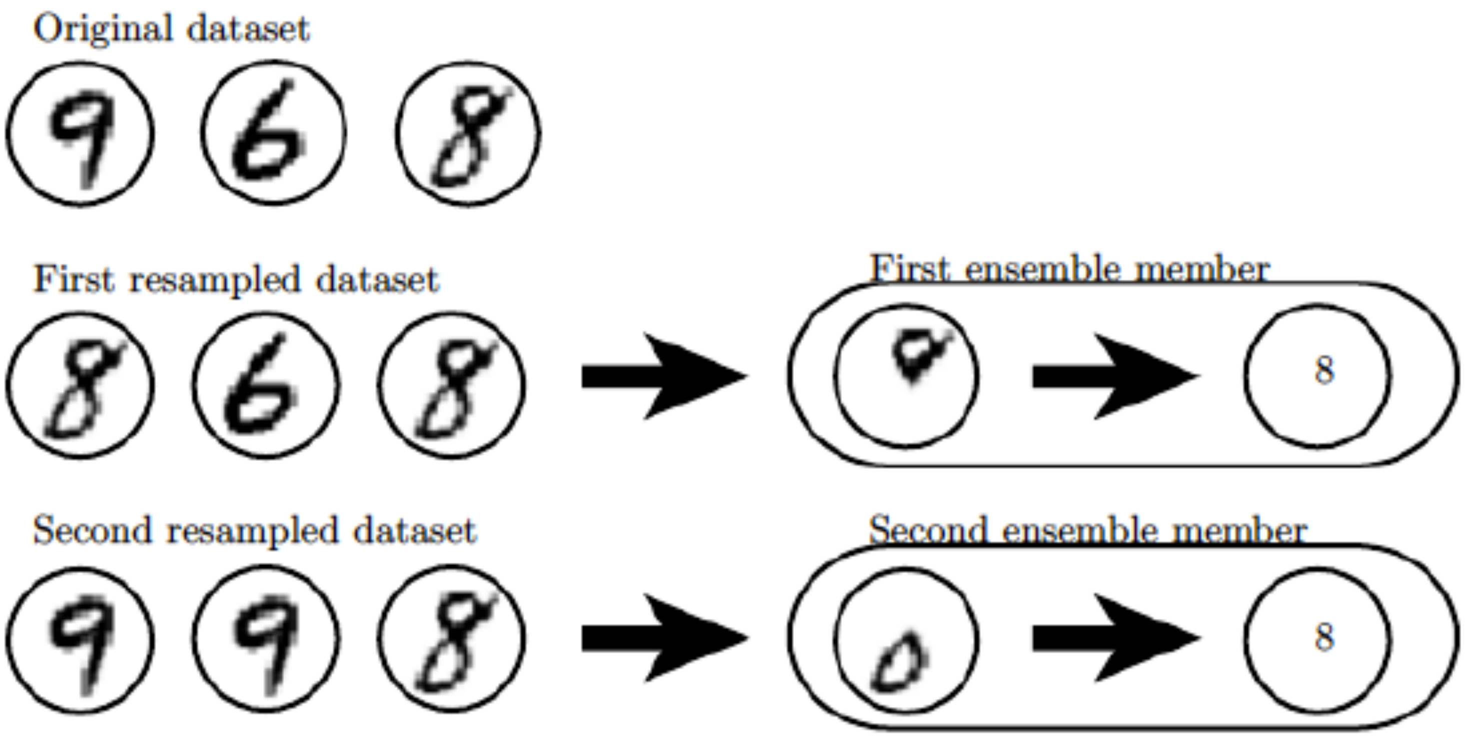
ϵ_i error on each example

$\frac{1}{k}\sum_i \epsilon_i.$ average prediction

Error with zero-mean multivariate normal distributions

$\mathbb{E}[\epsilon_i^2] = v$ $\mathbb{E}[\epsilon_i \epsilon_j] =$ c

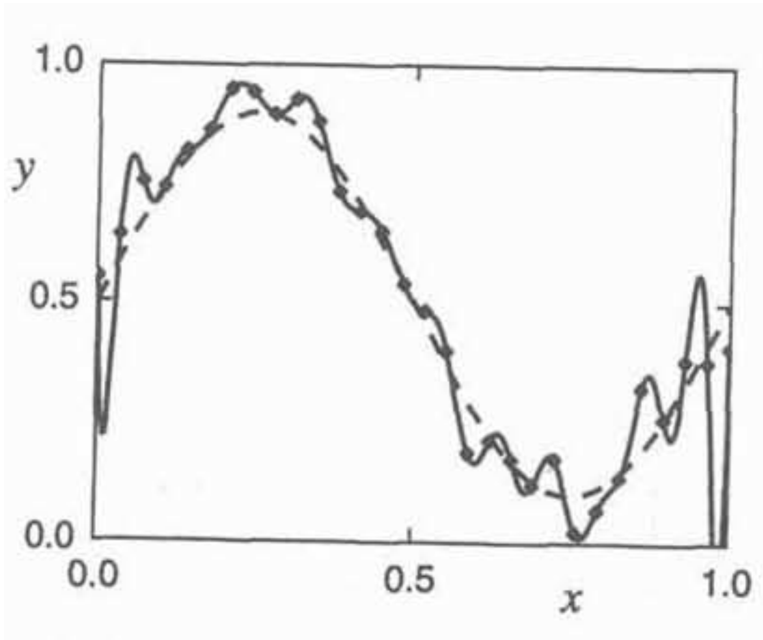
Bagging

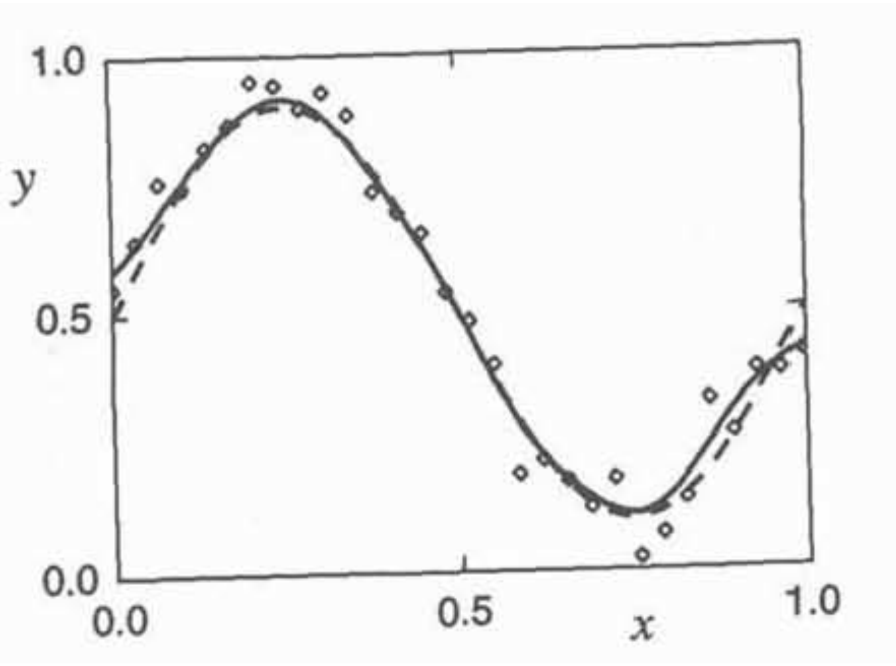

Original dataset
First resampled dataset
First ensemble member
8
Second resampled dataset
Second ensemble member
8

Cross-validation

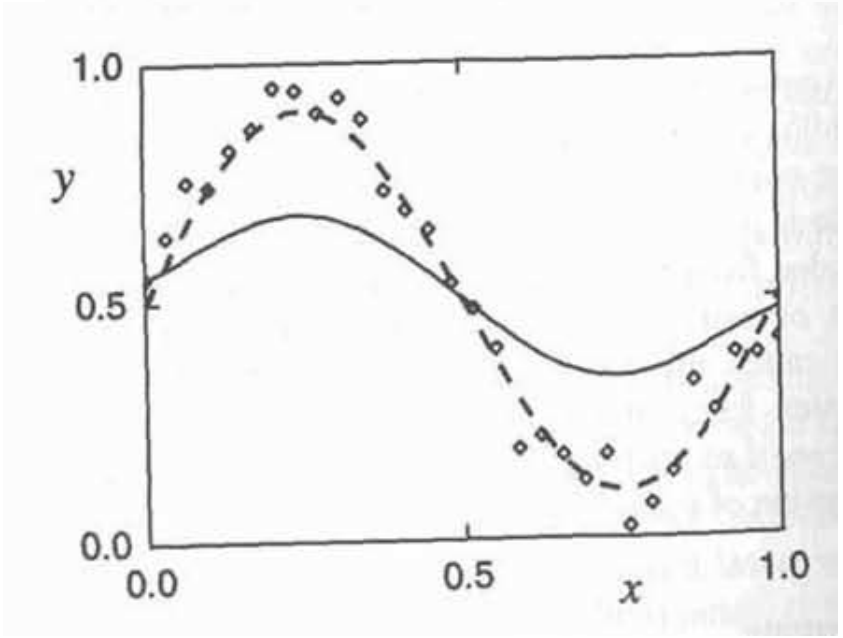
- Goal
  - Hold out method
    - Training, validation and test sets

Cross-validation



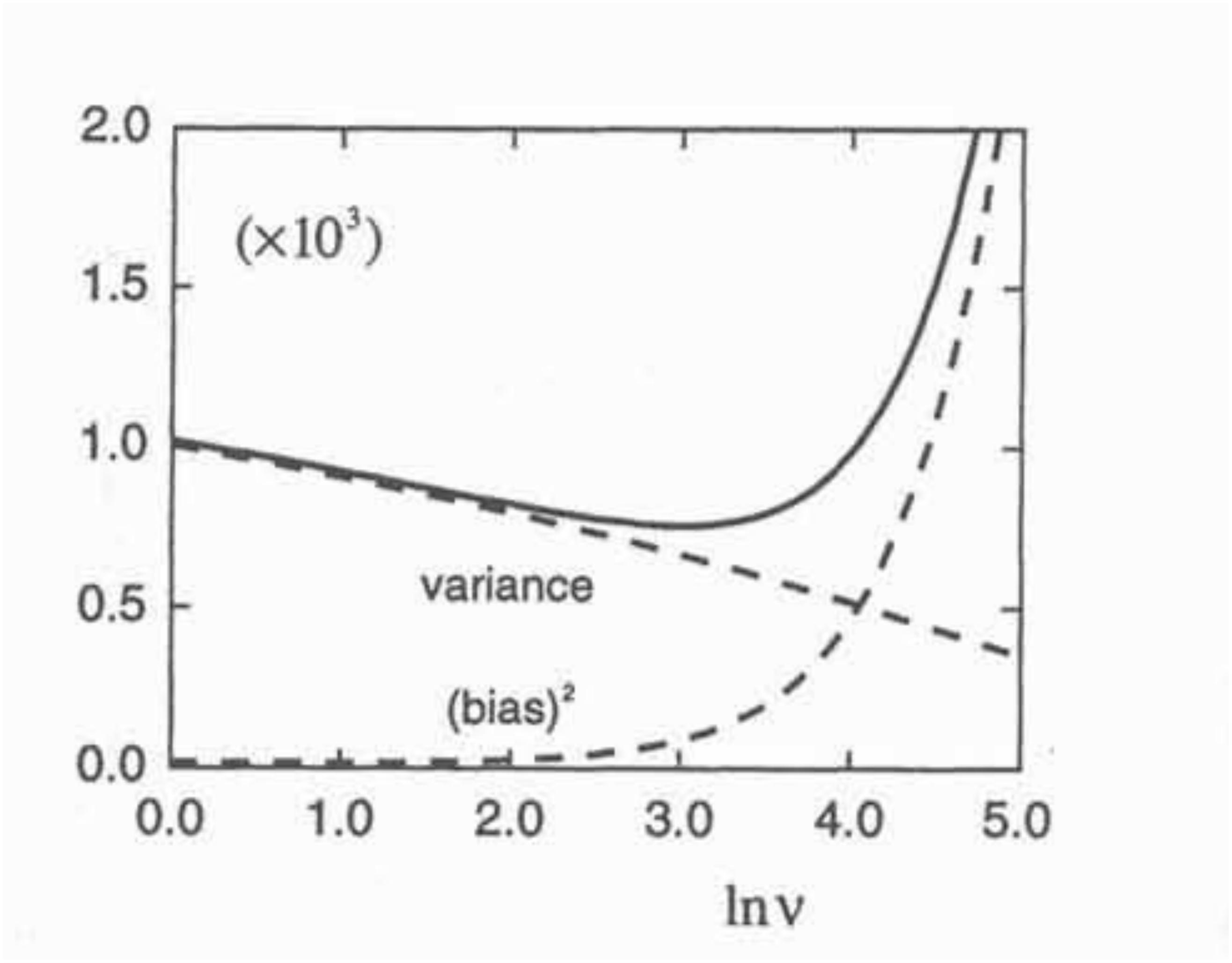


Regularization coefficient v = 40



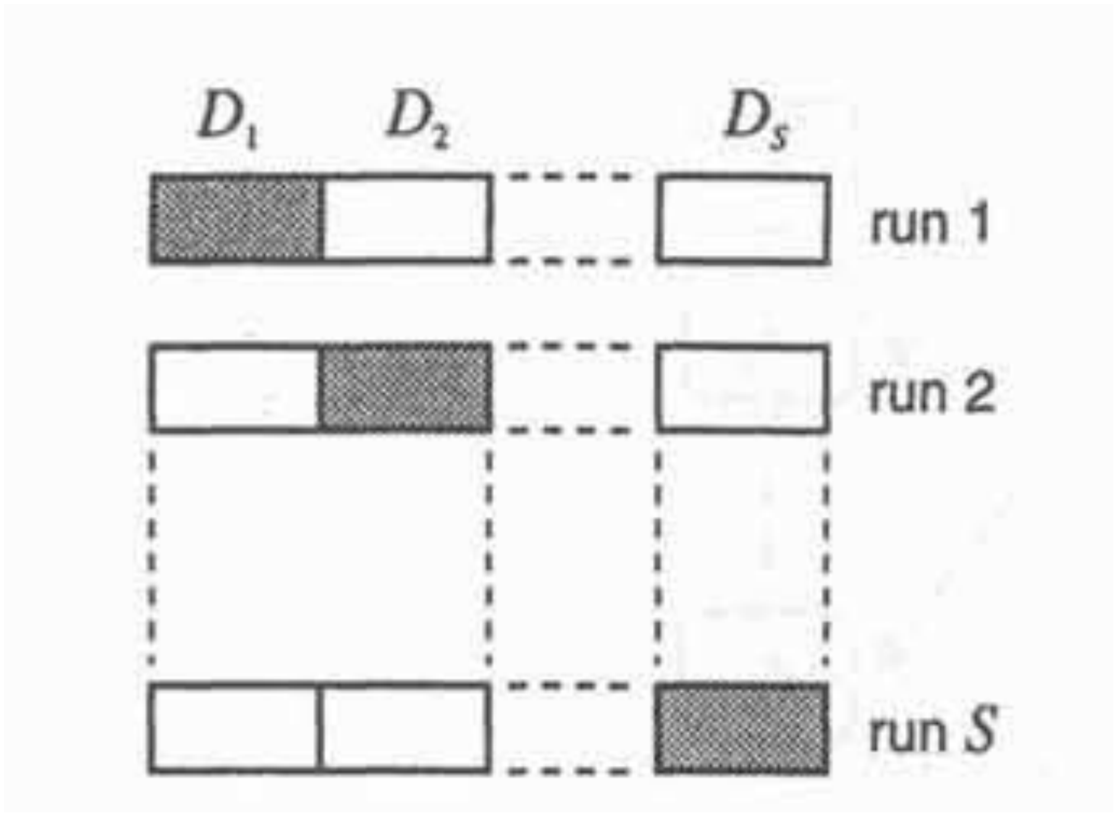
v = 1000

Cross-validation



Log of the regularization coefficient

Cross-validation



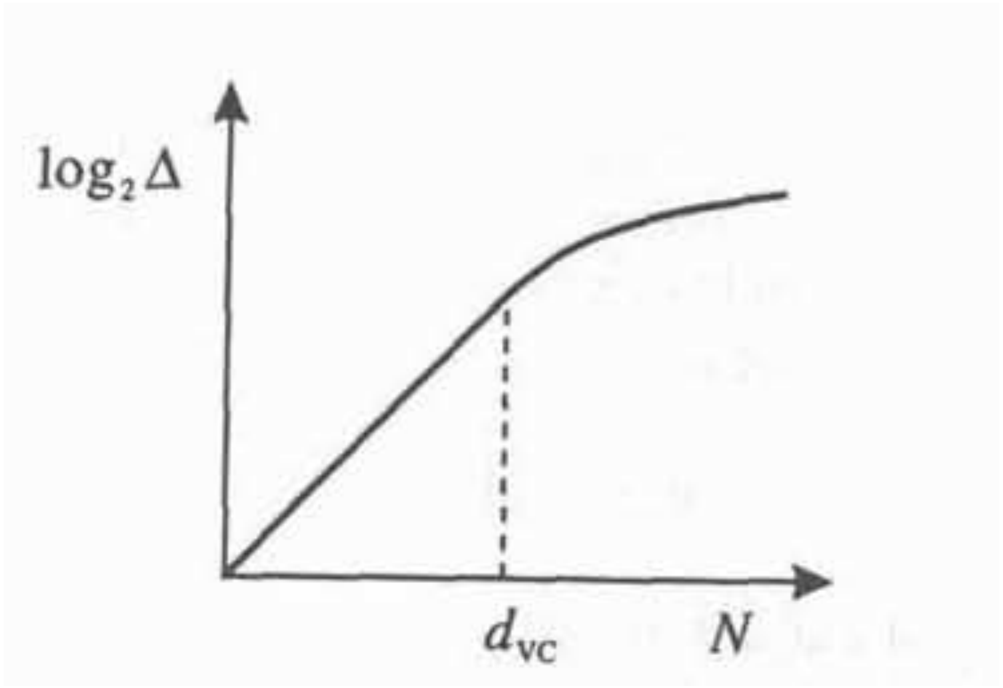
Partitionating of a data set into S segments for use cross-validation

Vapnik-Chervonenkis dimension

- Goal
  - Worst-case performance for a particular trained network
- Theorem

$$\Pr\left(\max_{\{y\}} |g_N(y) - g(y)| > \epsilon\right) \le 4\Delta(2N)\exp(-\epsilon^2 N/8)$$

Vapnik-Chervonenkis dimension



$$\Delta(N) \leq N^{d_{vc}} + 1$$

Vapnik-Chervonenkis dimension

- NN
  - M units, W weights

$$d_{\mathrm{VC}} \leq 2W \log_2(eM)$$

$$N \geq \frac{W}{\epsilon} \log_2\left(\frac{M}{\epsilon}\right)$$

  - Two layers and threshold units

$$d_{\mathrm{VC}} \geq 2\lfloor M/2 \rfloor d$$ d inputs

$$Md \simeq W$$

$$N_{\min} \simeq W/\epsilon.$$

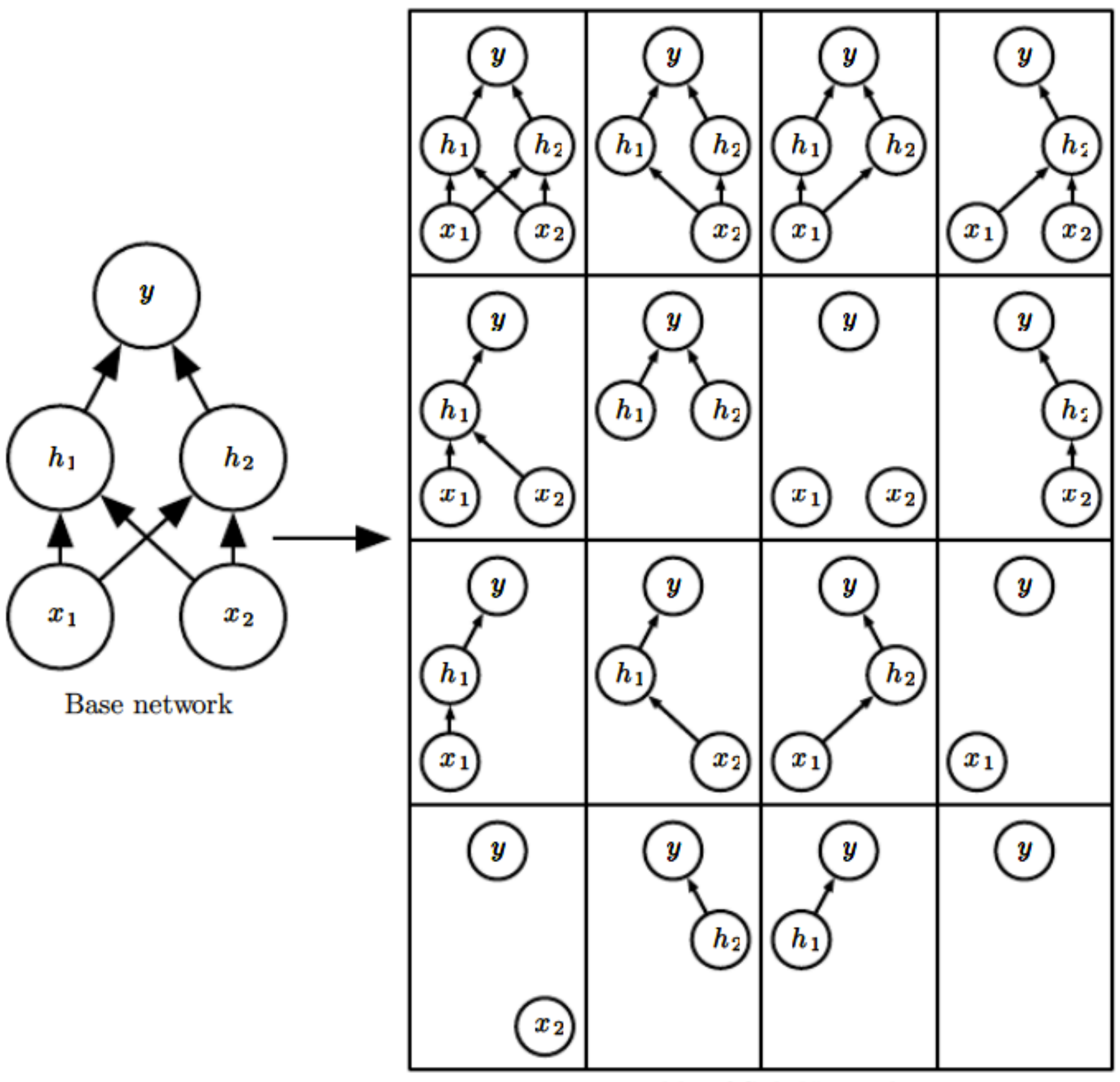
For large networks

Dropout

- Goal
  - making bagging practical for ensembles of very many large neural networks
  - trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network
  - dropout algorithm multiplying by zero

Dropout

Sixteen possible subsets



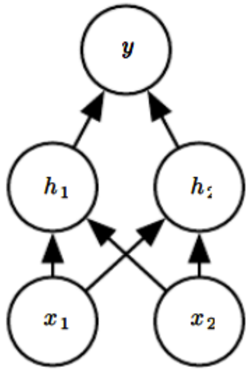
Base network

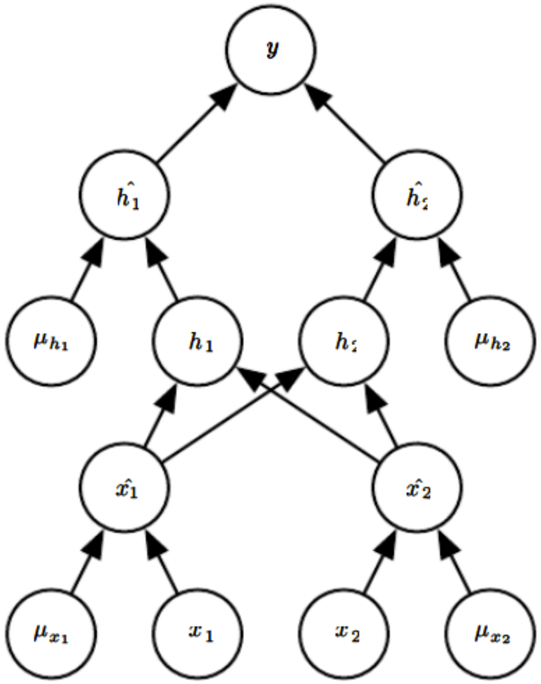
Ensemble of Sub-Networks

Dropout

- Train
  - we use a minibatch-based learning algorithm that makes small steps
    - such as stochastic gradient descent
  - Each time we load an example into a minibatch
    - randomly sample a different binary mask to apply to all of the input and hidden units in the network
    - The mask for each unit is sampled independently from all of the others
  - Probability of sampling a mask value
    - hyperparameter fixed before training begins
    - e.g., input unit is included with probability 0.8 and a hidden unit is included with probability 0.5

Dropout





randomly sample a vector µ with one entry for each input or hidden unit in the network

Dropout

- Error mimimization

$$J(\boldsymbol{\theta}, \boldsymbol{\mu}) \qquad\qquad \mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$$

- Weights
  - the models share parameters, with each model inheriting a different subset of parameters from the parent neural network
  - exponential number of models with a tractable amount of memory
    - tiny fraction of the possible sub-networks are each trained for a single step
    - the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters

Dropout

- Bagging
  - Model i produces a probability distribution

$$p^{(i)}(y \mid \boldsymbol{x})$$

  - Prediction of the ensemble

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y \mid \boldsymbol{x}).$$

- Dropout

arithmetic mean over all masks

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$$

$\boldsymbol{\mu}$ mask vector

Dropout

- Exopnential number of terms
  - geometric mean rather than the arithmetic mean of the ensemble members' predicted distributions
  - unnormalized probability distribution

$$\tilde{p}_{\text{ensemble}}(y \mid \boldsymbol{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y \mid \boldsymbol{x}, \boldsymbol{\mu})}$$

Uniform distribution over μ

d is the number of units that may be dropped

- Prediction

re-normalize the ensemble

$$p_{\text{ensemble}}(y \mid \boldsymbol{x}) = \frac{\tilde{p}_{\text{ensemble}}(y \mid \boldsymbol{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y' \mid \boldsymbol{x})}.$$

Dropout

- Weight scaling inference rule
  - Weights going out of unit i multiplied by the probability of including unit i
    - capture the right expected value of the output from that unit
  - Consider a softmax regression classifier with n input variables represented by the vector v

$$P(\mathrm{y} = y \mid \mathbf{v}) = \mathrm{softmax}\left(\boldsymbol{W}^{\top}\mathbf{v} + \boldsymbol{b}\right)_y$$

Dropout

- sub-models by element-wise multiplication of the input with a binary vector d

$$P(\mathrm{y}=y \mid \mathbf{v}; \boldsymbol{d}) = \mathrm{softmax}\left(\boldsymbol{W}^{\top}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)_y$$

- ensemble predictor is defined by re-normalizing the geometric mean

$$P_{\text{ensemble}}(\mathrm{y}=y \mid \mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y}=y \mid \mathbf{v})}{\sum_{y'} \tilde{P}_{\text{ensemble}}(\mathrm{y}=y' \mid \mathbf{v})}$$

$$\tilde{P}_{\text{ensemble}}(\mathrm{y}=y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y}=y \mid \mathbf{v}; \boldsymbol{d})}.$$

Dropout

- Simplify

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y \mid \mathbf{v}; \boldsymbol{d})}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \operatorname{softmax}\left(\boldsymbol{W}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)_y}$$

$$= \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \frac{\exp\left(\boldsymbol{W}_{y,:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)}{\sum_{y'} \exp\left(\boldsymbol{W}_{y',:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)}}$$

$$= \frac{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp\left(\boldsymbol{W}_{y,:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)}}{\sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \sum_{y'} \exp\left(\boldsymbol{W}_{y',:}^\top (\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)}}$$

Dropout

- Ignore multiplication by factors that are constant with respect to y

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y \mid \mathbf{v}) \propto \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} \exp\left(\boldsymbol{W}_{y,:}^{\top}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)}$$

$$= \exp\left(\frac{1}{2^n} \sum_{\boldsymbol{d} \in \{0,1\}^n} \boldsymbol{W}_{y,:}^{\top}(\boldsymbol{d} \odot \mathbf{v}) + \boldsymbol{b}\right)$$

$$= \exp\left(\frac{1}{2}\boldsymbol{W}_{y,:}^{\top}\mathbf{v} + \boldsymbol{b}\right)$$

Dropout

- One advantage of dropout is that it is very computationally cheap

- It works well for models that uses a distributed representation
  - Feedforward neural networks, probabilistic models such as restricted Boltzmann machines and recurrent neural networks
  - When extremely few labeled training examples are available, dropout is less effective
  - Applied to linear regression, dropout is equivalent to L^2 weight decay
  - Multiplying the weights by $\mu \sim N(1, I)$ can outperform dropout based on binary masks

Dropout

- Idea from biology
  - Hidden units must be prepared to be swapped and interchanged between models
  - sexual reproduction
    - involves swapping genes between two different organisms
    - creates evolutionary pressure for genes to become not just good
    - become readily swapped between different organisms
- Finally
  - Dropout regularizes each hidden unit to be not merely a good feature but a feature that is good in many contexts

Dropout

- Important
  - highly intelligent
    - adaptive destruction of the information content of the input rather than destruction of the raw values of the input
  - e.g.
    - if the model learns a hidden unit hi that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image
    - The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth
- noise is multiplicative
  - Multiplicative noise does not allow such a pathological solution to the noise robustness problem

Adversial training

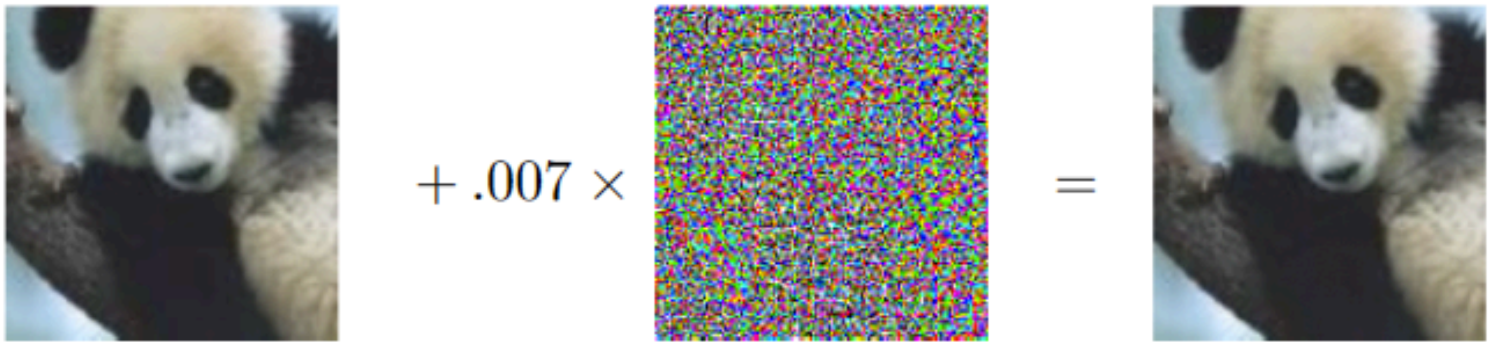
- Adversial training
  - probe the level of understanding a network has of the underlying task, we can search for examples that the model misclassifies

- Adversial example
  - Adversarial examples have many implications, for example, in computer security

- Adversarial pertubation
  - training on adversarially perturbed examples from the training set of the primary causes of these adversarial examples is excessive linearity

  - Adversarial examples also provide a means of accomplishing semi-supervised learning

Adversial training



$$\boldsymbol{x} \; + .007 \times \; \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)) \; = \; \boldsymbol{x} + \epsilon\, \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

$y =$"panda" w/ 57.7% confidence

"nematode" w/ 8.2% confidence

"gibbon" w/ 99.3 % confidence

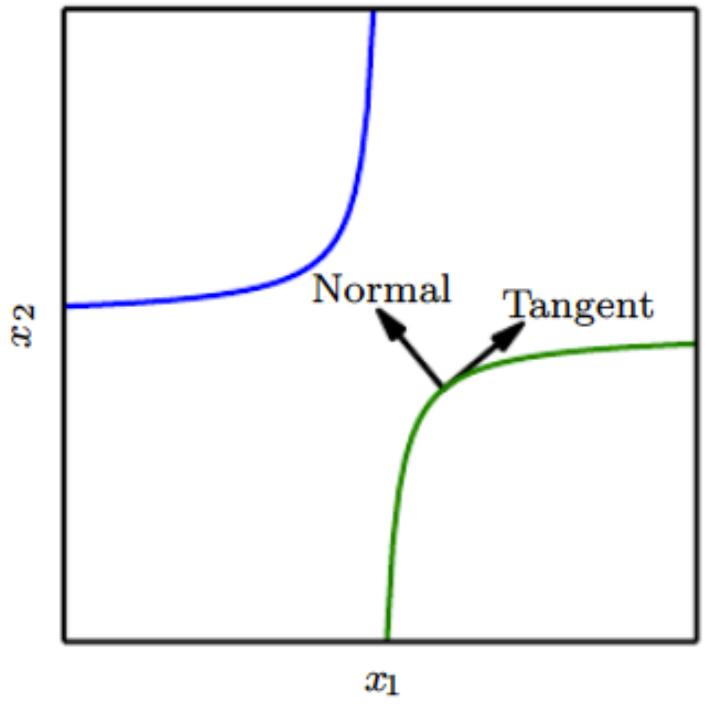
Tangent distance

- Tangent distance
  - It is a non-parametric nearest-neighbor algorithm
    - metric used is not the generic Euclidean distance but one that is derived from knowledge of the manifolds near which probability concentrates

- Tangent prop
  - These factors of variation correspond to movement along the manifold near which examples of the same class concentrate

$$\Omega(f) = \sum_i \left(\left(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \right)^\top \boldsymbol{v}^{(i)} \right)^2$$

Tangent prop

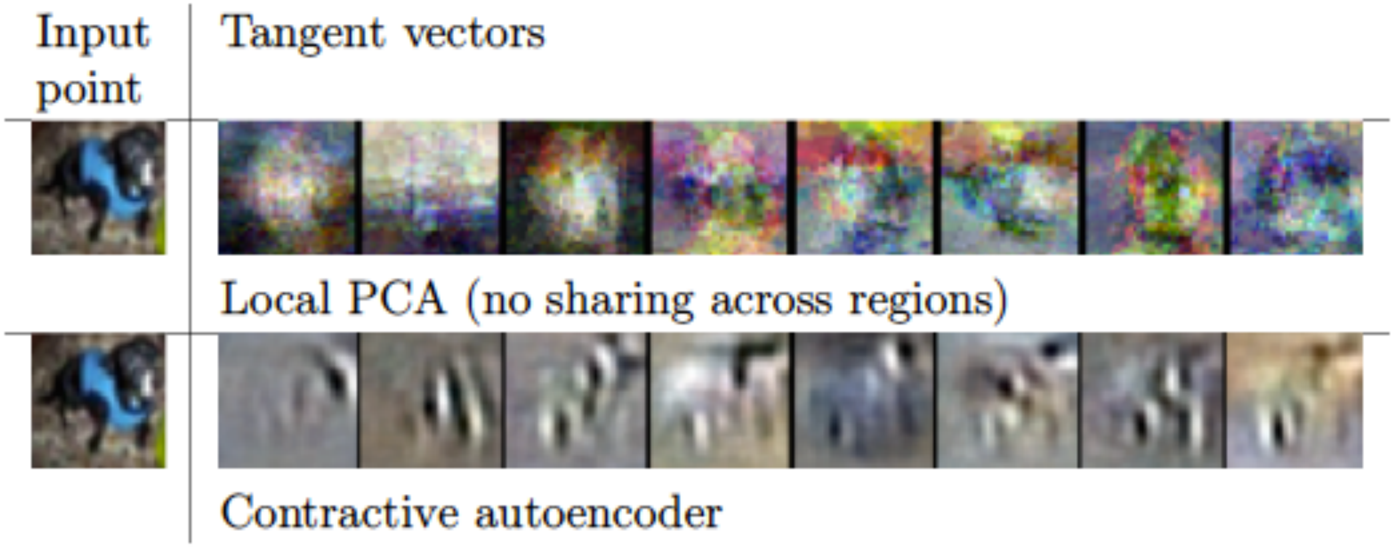


the classification function to change rapidly as it moves in the direction normal to the manifold, and not to change as it moves along the class manifold.

Tangent prop

- Features
  - related to dataset augmentation
  - related to double backprop
    - regularizes the Jacobian to be small
  - adversarial training
    - finds inputs near the original inputs and trains the model to produce the same output on these as on the original inputs
- The manifold tangent classifier
    - eliminates the need to know the tangent vectors a priori
    - Autoencoders estimate the manifold tangent vectors

Tangent prop


Input point
Tangent vectors
Local PCA (no sharing across regions)
Contractive autoencoder