Regularization in Neural Networks

Sargur Srihari

Topics in Neural Network Regularization

- What is regularization?
- Methods
  1. Determining optimal number of hidden units
  2. Use of regularizer in error function
     - Linear Transformations and Consistent Gaussian priors
  3. Early stopping
- Invariances
  - Tangent propagation
- Training with transformed data

What is Regularization?

- Central problem of ML is Generalization
  - to design algorithms that will perform well not just on training data but on new inputs as well
- Regularization is:
  - "any modification we make to a learning algorithm to reduce its generalization error but not its training error"
  - Reduce test error even at the expense of increasing training error
- Some goals of regularization
  1. Encode prior knowledge
  2. Express preference for simpler model
  3. Need to make underdetermined problem determined

Need for Regularization

- Generalization
  - Prevent over-fitting
- Occam's razor
- Bayesian point of view
  - Regularization corresponds to prior distributions on model parameters

What is the Best Model?

- Best fitting model obtained not by finding the right number of parameters
- Instead, best fitting model is a large model that has been regularized appropriately
- We review several strategies for how to create such a large, deep regularized model

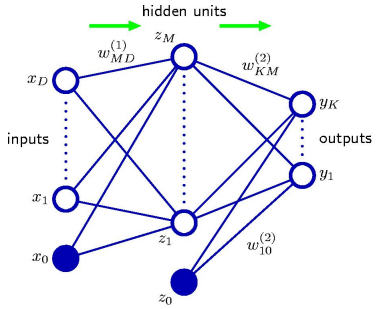
Common Regularization Strategies

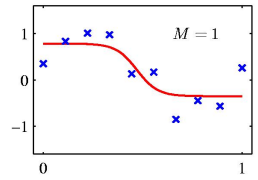
1. Parameter Norm Penalties
2. Early Stopping
3. Parameter tying and parameter sharing
4. Bagging and other ensemble methods
5. Dropout
6. Data Set Augmentation
7. Adversarial training
8. Tangent methods

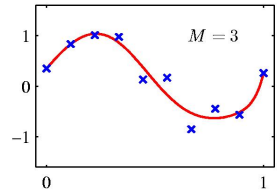
Optimizing no. of hidden units

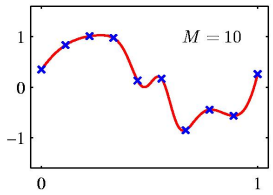
- Number of input and output units is determined by dimensionality of data set
- Number of hidden units M is a free parameter
  - Adjusted to get best predictive performance
- Possible approach is to get maximum likelihood estimate of M for balance between under-fitting and over-fitting

Effect of Varying Number of Hidden Units









Sinusoidal Regression Problem

Two layer network trained on 10 data points

$M = 1, 3$ and 10 hidden units

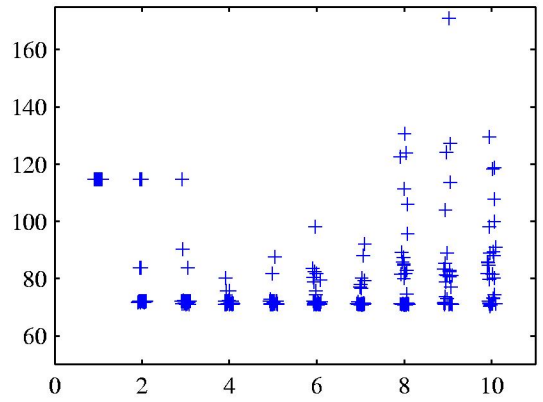
Minimizing sum-of-squared error function
Using conjugate gradient descent

Generalization error is not a simple function of M due to presence of local minima in error function

Using *Validation Set* to determine no of hidden units

- Plot a graph choosing random starts and different numbers of hidden units M and choose the specific solution having smallest generalization error

Sum of squares
Test error

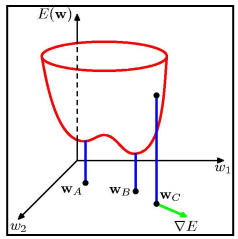


No. of hidden units, M

- 30 random starts for each M
  30 points in each column of graph
- Overall best *validation* set performance happened at $M=8$

- There are other ways to control the complexity of a neural network in order to avoid overfitting
- Alternative approach is to choose a relatively large value of M and then control complexity by adding a regularization term

Regularization using Simple Weight Decay



- Generalization error is not a simple function of M
  - Due to presence of local minima
  - Need to control network complexity to avoid over-fitting
  - Choose a relatively large M and control complexity by addition of regularization term
- Simplest regularizer is weight decay

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

  - Effective model complexity determined by choice of regularization coefficient λ
  - Regularizer is equivalent to a Gaussian prior over weight vector $\boldsymbol{w}$

- Simple weight decay has certain shortcomings
  - invariance to scaling

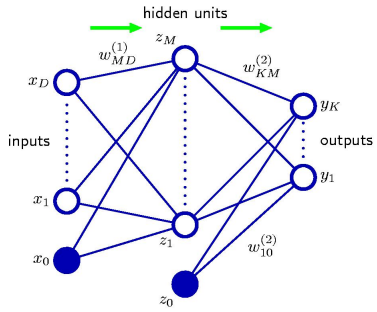
Consistent Gaussian priors

- Simple weight decay is inconsistent with certain scaling properties of network mappings
- To show this, consider a multi-layer perceptron network with two layers of weights and linear output units
- Set of input variables $\{\boldsymbol{x}_i\}$ and output variables $\{\boldsymbol{y}_k\}$
- Activations of hidden units in first layer have the form

$$z_j = h\left(\sum_i w_{ji} x_i + w_{j0}\right)$$

- Activations of output units are

$$y_k = \sum_j w_{kj} z_j + w_{k0}$$



Linear Transformations of input/output Variables

- Suppose we perform a linear transformation of input data

$$x_i \rightarrow \tilde{x}_i = a x_i + b$$

- Then we can arrange for mapping performed by network to be unchanged by making a corresponding linear transformation
  - Of the weights and biases from the inputs to the hidden units as

$$w_{ji} \rightarrow \tilde{w}_{ji} = \frac{1}{a} w_{ji} \quad \text{and} \quad w_{j0} = w_{j0} - \frac{b}{a} \sum_i w_{ji}$$

- Similar linear transformation of output variables of network of the form

$$y_k \rightarrow \tilde{y}_k = c y_k + d$$

  - Can be achieved by transformation of second layer weights and biases

$$w_{kj} \rightarrow \tilde{w}_{kj} = c w_{kj} \quad \text{and} \quad w_{k0} = c w_{k0} + d$$

Desirable invariance property of regularizer

- Suppose we train two different networks
  - First network trained using original data: $\{x_i\}, \{y_k\}$
  - Second network for which input and/or target variables are transformed by one of the linear transformations $x_i \rightarrow \tilde{x}_i = ax_i + b$ and/or $y_k \rightarrow \tilde{y}_k = cy_k + d$
- Then consistency requires that we should obtain equivalent networks that differ only by linear transformation of the weights

  For first layer: $w_{ji} \rightarrow \frac{1}{a} w_{ji}$ And/or or second layer: $w_{kj} \rightarrow c w_{kj}$
- Regularizer should have this property
  - Otherwise it arbitrarily favors one solution over another
- Simple weight decay $\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$ does not have this property
  - Treats all weights and biases on an equal footing
    - While resulting w_{ji} and w_{kj} should be treated differently
    - Consequently networks will have different weights and violate invariance
- We therefore look for a regularizer invariant under the linear transformations

Regularizer invariant under linear transformation

- The regularizer should be invariant to re-scaling of weights and shifts of biases
- Such a regularizer is

$$\frac{\lambda_1}{2}\sum_{w\in W_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in W_2} w^2$$

  - where W_1 are weights of first layer and
  - W_2 are the set of weights in the second layer
- This regularizer remains unchanged under the weight transformations provided the parameters are rescaled using

$$\lambda_1 \rightarrow a^{1/2}\lambda_1 \quad \text{and} \quad \lambda_2 \rightarrow c^{-1/2}\lambda_2$$

- We have seen before that weight decay is equivalent to a Gaussian prior. So what is the equivalent prior to this?

Equivalent prior

- The regularizer invariant to re-scaling of weights/ biases is

$$\frac{\lambda_1}{2}\sum_{w\in W_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in W_2} w^2$$

  - where W_1 are weights of first layer and W_2 of second layer
- It corresponds to a prior of the form

$$p(\boldsymbol{w} \mid \alpha_1, \alpha_2) \quad \alpha \quad \exp\left(-\frac{\alpha_1}{2}\sum_{w\in W_1} w^2 - \frac{\alpha_2}{2}\sum_{w\in W_2} w^2\right)$$

  α_1 and α_2 are hyper-parameters

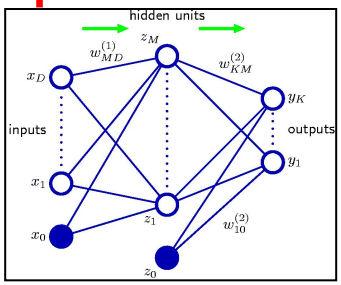
  - This is an *improper prior* which cannot be normalized
    - Leads to difficulties in selecting regularization coefficients and in model comparison within Bayesian framework
    - Instead include separate priors for biases with their own hyper-parameters
- We can illustrate effect of the resulting four hyperparameters
    - α_1^{b}, precision of Gaussian distribution of *first layer bias*
    - α_1^{w}, precision of Gaussian of *first layer weights*
    - α_2^{b}, precision of Gaussian distribution of *second layer bias*
    - α_2^{w}, precision of Gaussian distribution of *second layer weights*

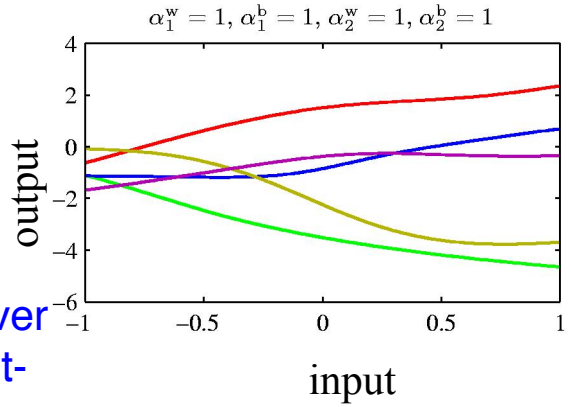
Effect of hyperparameters on input-output

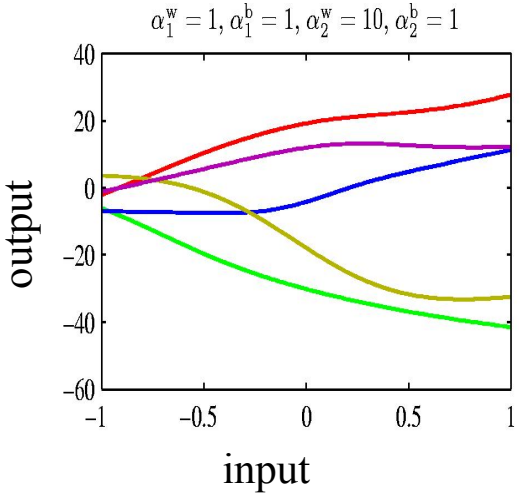
Network with single input (x value range: -1 to +1), single linear output (y value range: -60 to +40)

12 hidden units with tanh activation functions

Priors are governed by four hyper-parameters
α_1^b, precision of Gaussian distribution of *first layer bias*
α_1^w, precision of Gaussian of *first layer weights*
α_2^b,…. ..of *second layer bias*
α_2^w, …. of *second layer weights*

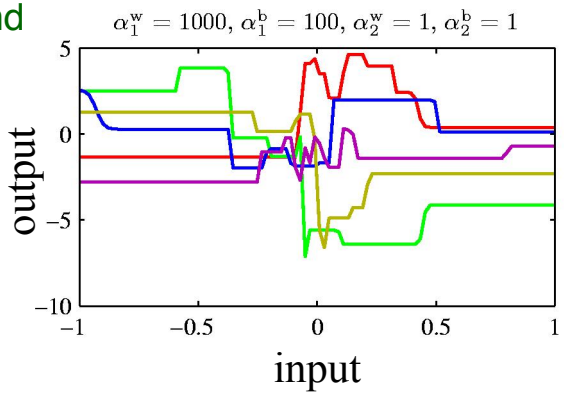


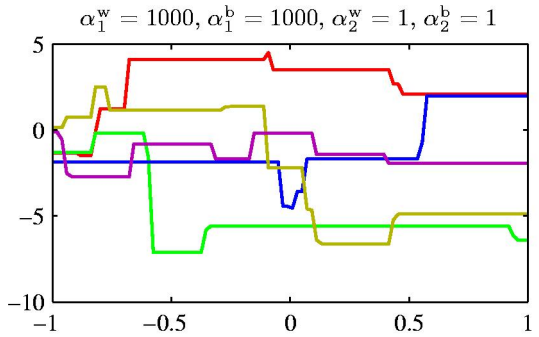




Observe: vertical scale controlled by α_2^w

Draw five samples from prior over hyperparameters, and plot input-output (network functions)
Five samples correspond to five colors
For each setting function is learnt and plotted

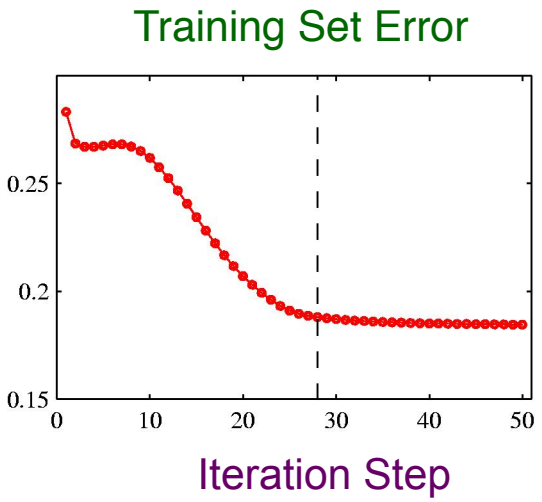


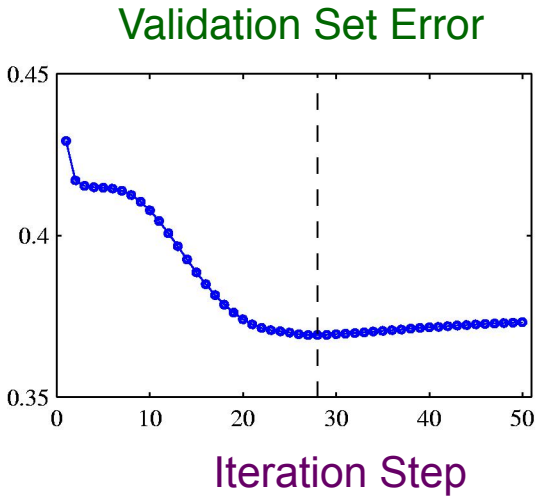


Observe: horizontal scale controlled by α_1^w

3. Early Stopping

- Alternative to regularization
  - In controlling complexity
- Error measured with an independent validation set
  - shows initial decrease in error and then an increase
- Training stopped at point of smallest error with validation data
  - Effectively limits network complexity



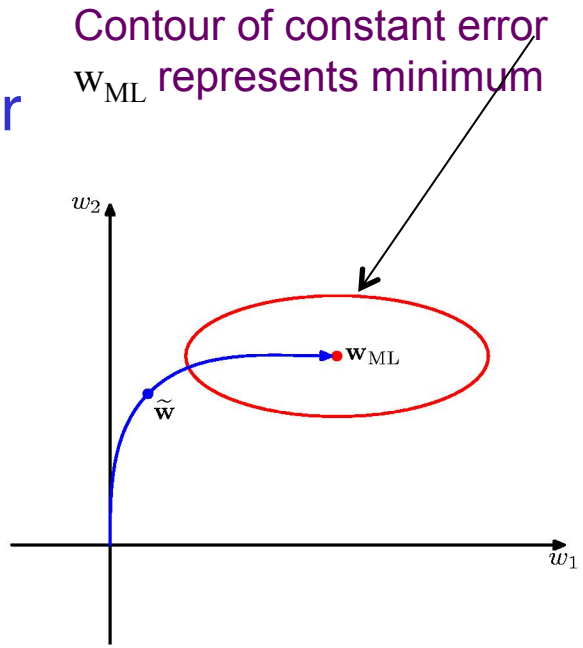


Interpreting the effect of Early Stopping

- Consider quadratic error function
- Axes in weight space are parallel to eigen vectors of Hessian
- In absence of weight decay, weight vector starts at origin and proceeds to $\boldsymbol{w}_{\mathrm{ML}}$
- Stopping at $\tilde{w}$ is similar to weight decay

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

- Effective number of parameters in the network grows during course of training



Invariances

- Quite often in classification problems there is a need
  - Predictions should be invariant under one or more transformations of input variable
- Example: handwritten digit should be assigned same classification irrespective of position in the image (translation) and size (scale)
- Such transformations produce significant changes in raw data, yet need to produce same output from classifier
  - Examples: pixel intensities,

  in speech recognition, nonlinear time warping along time axis

Simple Approach for Invariance

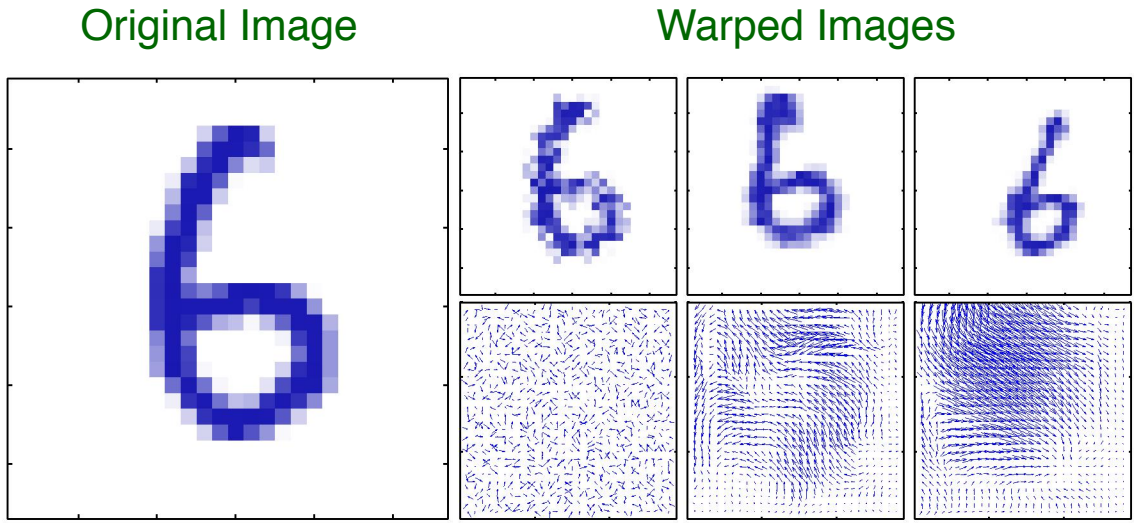
- Large sample set where all transformations are present
  - E.g., for translation invariance, examples of objects in may different positions
- Impractical
  - Number of different samples grows exponentially with number of transformations
- Seek alternative approaches for adaptive model to exhibit required invariances

Approaches to Invariance

1. Training set augmented by transforming training patterns according to desired invariances
   - E.g., shift each image into different positions
2. Add regularization term to error function that penalizes changes in model output when input is transformed.
   - Leads to tangent propagation.
3. Invariance built into pre-processing by extracting features invariant to required transformations
4. Build invariance property into structure of neural network (convolutional networks)
   - Local receptive fields and shared weights

Approach 1: Transform each input

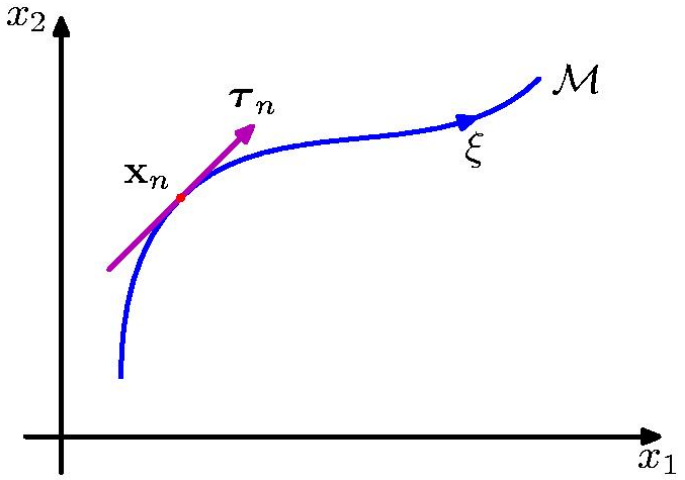
- Synthetically warp each handwritten digit image before presentation to model
- Easy to implement but computationally costly



Random displacements
$\Delta x, \Delta y \in (0,1)$ at each pixel
Then smooth by convolutions
of width 0.01, 30 and 60 resply

Approach 2: Tangent Propagation

- Regularization can be used to encourage models to be invariant to transformations
  - by techniques of tangent propagation
- Consider effect of a transformation on input vector $\mathbf{x}_n$
  - A one-dimensional continuous transformation parameterized by ξ applied to $\mathbf{x}_n$ sweeps a manifold $\mathcal{M}$ in D-dimensional input space



Two-dimensional input space showing effect of continuous transformation with single parameter ξ

Let the vector resulting from acting on $\mathbf{x}_n$ by this transformation be denoted by $s(\mathbf{x}_n,\xi)$ defined so that $s(\mathbf{x},0)=\mathbf{x}$.

Then the tangent to the curve $\mathcal{M}$ is given by the directional derivative $\tau = \delta s/\delta\xi$ and the tangent vector at point $\mathbf{x}_n$ is given by

$$\boldsymbol{\tau}_n = \left.\frac{\partial s(\mathbf{x}_n,\xi)}{\partial \xi}\right|_{\xi=0}$$

Tangent Propagation as Regularization

- Under a transformation of input vector
  - The network output vector will change
  - Derivative of output k wrt ξ is given by

$$\left.\frac{\partial y_k}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} \frac{\partial y_k}{\partial x_i} \left.\frac{\partial x_i}{\partial \xi}\right|_{\xi=0} = \sum_{i=1}^{D} J_{ki} \tau_i$$

  - where J_{ki} is the (k,i) element of the Jacobian Matrix J
- Result is used to modify the standard error function
  - So as to encourage local invariance in neighborhood of data point

$$\tilde{E} = E + \lambda \Omega$$

where λ is a regularization coefficient and

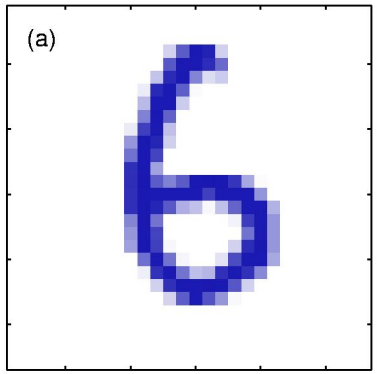
$$\Omega = \frac{1}{2} \sum_n \sum_k \left(\left.\frac{\partial y_{nk}}{\partial \xi}\right|_{\xi=0} \right)^2 = \frac{1}{2} \sum_n \sum_k \left(\sum_{i=1}^{D} J_{nki} \tau_{ni} \right)^2$$

Tangent vector from finite differences

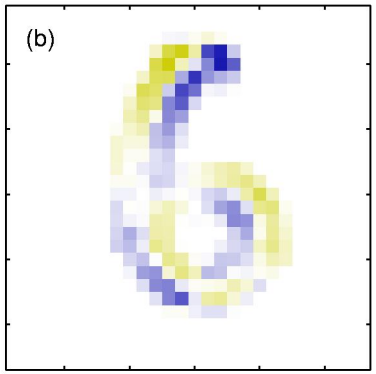
- In practical implementation tangent vector τ_n is approximated
  - using finite differences
  - by subtracting original vector x_n
  - from the corresponding vector
  - after transformations using
  - a small value of ξ and then
  - dividing by ξ

- A related technique *Tangent Distance is* used to build invariance properties into distance-based methods such as *nearest-neighbor* classifiers

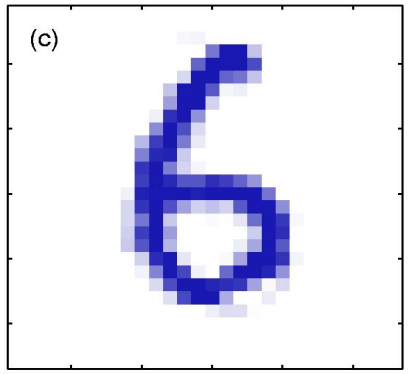
Original Image x



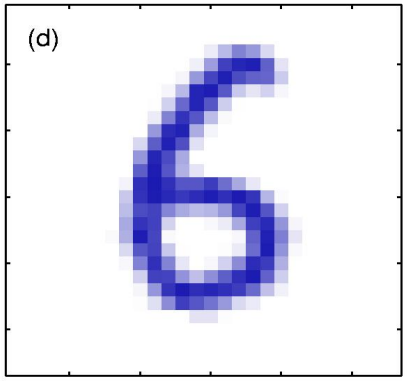
Tangent vector τ corresponding to small clockwise rotation



Adding small contribution from tangent vector to image $x+\varepsilon\tau$



True image rotated for comparison



Equivalence of Approaches 1 and 2

- Expanding the training set is closely related to tangent propagation
- Small transformations of the original input vectors together with *Sum-of-squared* error function can be shown to be equivalent to the *tangent propagation* regularizer