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
- Convolutional networks
- Soft weight sharing

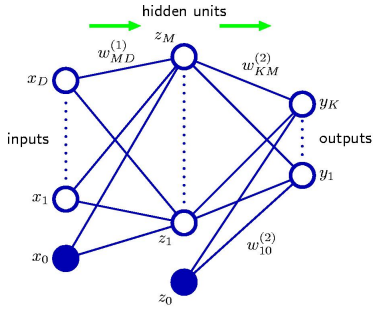
What is Regularization?

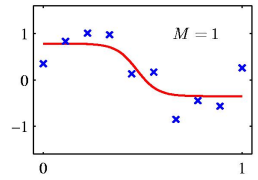
- In machine learning (also, statistics and inverse problems):
  - introducing additional information to prevent over-fitting ←
    (or solve ill-posed problem)
- This information is usually a penalty for complexity, e.g.,
  - restrictions for smoothness
  - bounds on the vector space norm
- Theoretical justification for regularization:
  - attempts to impose Occam's razor on the solution
- From a Bayesian point of view
  - Regularization corresponds to imposition of prior distributions on model parameters

1. Regularization by determining no. of hidden units

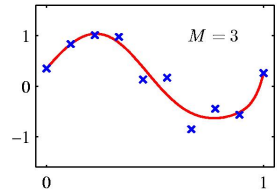
- Number of input and output units is determined by dimensionality of data set
- Number of hidden units M is a free parameter
  - Adjusted to get best predictive performance
- Possible approach is to get maximum likelihood estimate of M for balance between under-fitting and over-fitting

Effect of Varying Number of Hidden Units



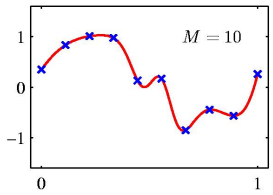


Sinusoidal Regression Problem



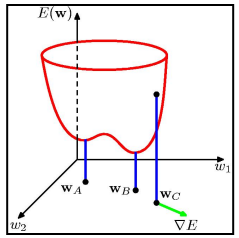
Two layer network trained on *10* data points

$M = 1, 3$ and *10* hidden units



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

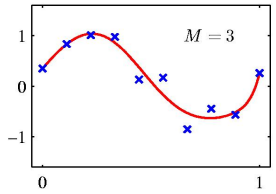
Generalization error is not a simple function of M due to presence of local minima in error function



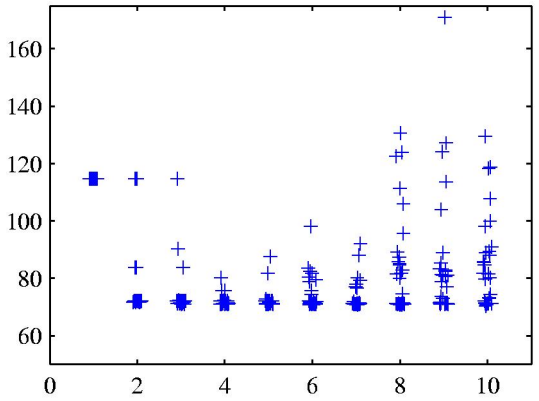
Using *Validation Set* to determine no of hidden units

Plot a graph choosing random starts and different numbers of hidden units M

30 random starts for each M

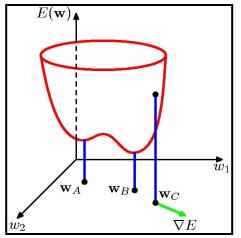


Sum of squares
Test error
for polynomial data



Overall best *validation*
Set performance happened at
M=8

Number of hidden units, M



2. 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

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

  - Effective model complexity determined by choice of regularization coefficient λ
  - Regularizer is equivalent to a zero mean Gaussian prior over weight vector $\mathbf{w}$
- Simple weight decay has certain shortcomings

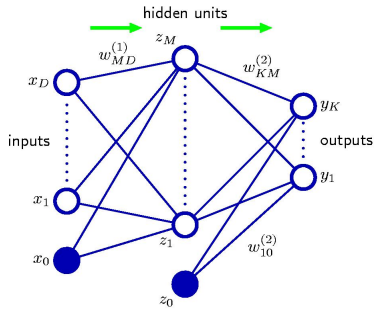
Consistent Gaussian priors

- 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 $\{x_i\}$ and output variables $\{y_i\}$
- 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

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

- We can arrange for mapping performed by network to be unchanged
  - if we transform weights and biases from inputs to 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 is

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

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

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

Desirable invariance property of regularizer

- If we train one network using original data
- Another for which input and/or target variables are transformed by one of the linear transformations
- Then they should only differ by the weights as given
- Regularizer should have this property
- Otherwise it arbitrarily favors one solution over another
- Simple weight decay does not have this property
  - Treats all weights and biases on an equal footing

Regularizer invariant under linear transformation

- A regularizer invariant to re-scaling of weights and shifts of 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
- This regularizer remains unchanged under weight transformations provided

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

- However, it corresponds to prior of the form

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

  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

Example: Effect of hyperparameters

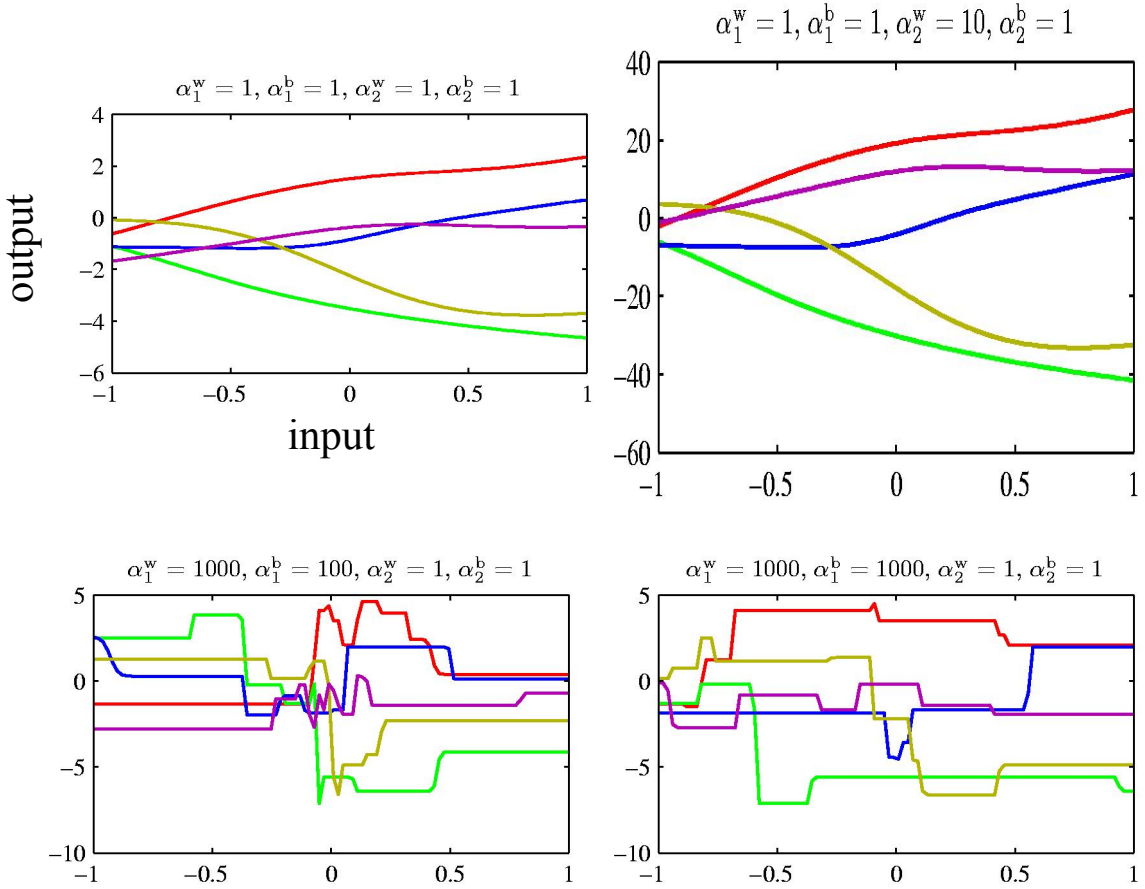
Network with single input (x value ranging from -1 to +1), single linear output (y value ranging from -60 to +40)

12 hidden units with tanh activation functions

Priors are governed by four hyper-parameters

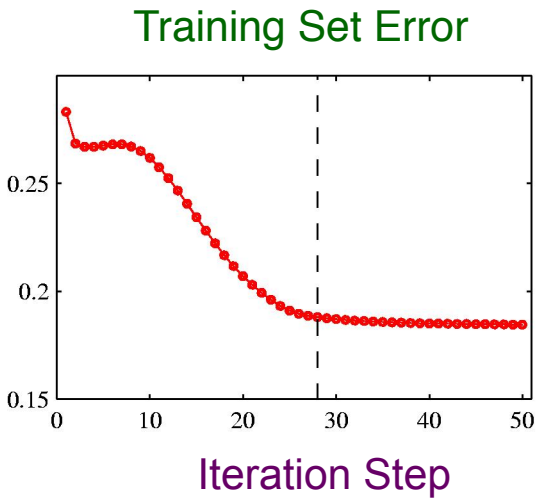
α_1^b, precision of Gaussian of *first layer bias*
α_1^w, …..of *first layer weights*
α_2^b,…. ..of *second layer bias*
α_2^w, …. of *second layer weights*

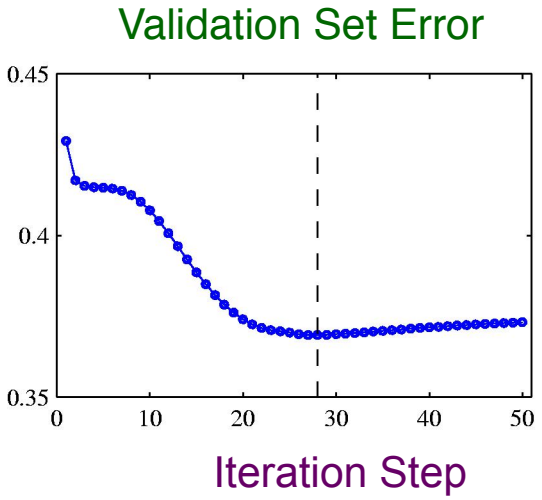
Draw samples from prior, plot network functions
Five samples correspond to five colors
For each setting function is learnt and plotted



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



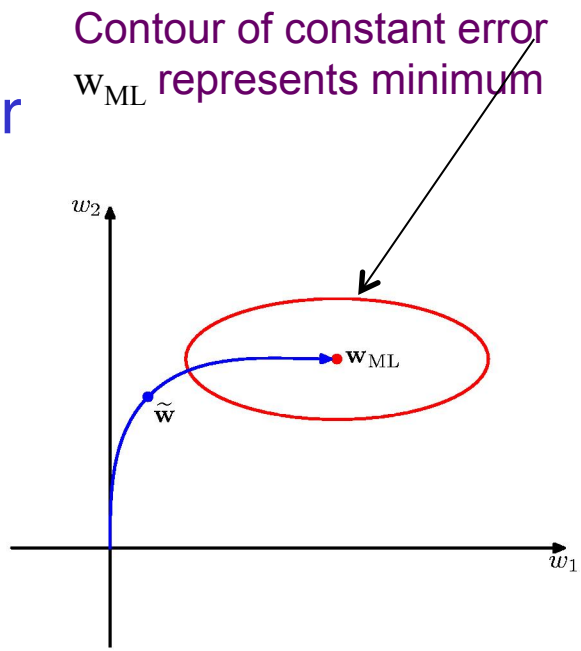


Interpreting the effect of Early Stopping

- 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 $\mathbf{w}_{\mathrm{ML}}$
- Stopping at $\tilde{w}$ is similar to weight decay

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

- 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

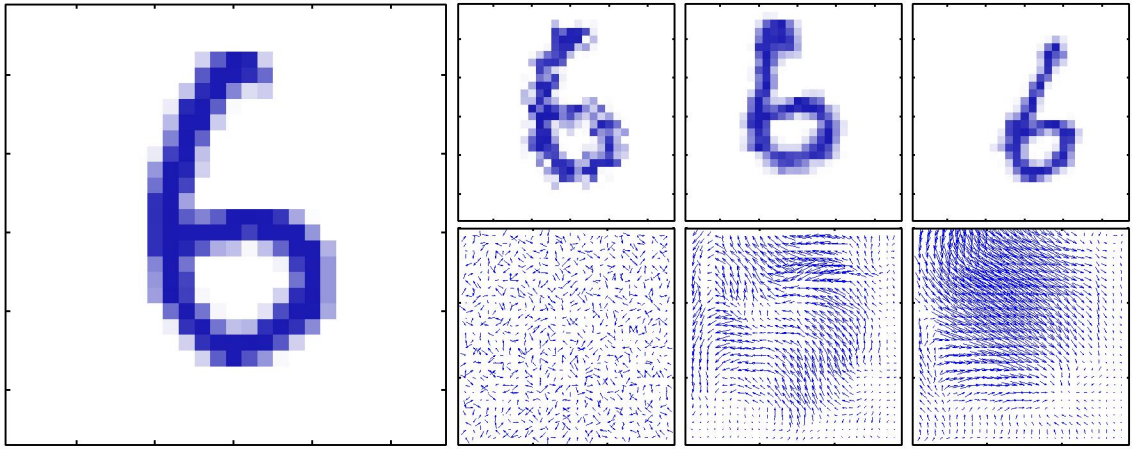
Approaches to Invariance

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

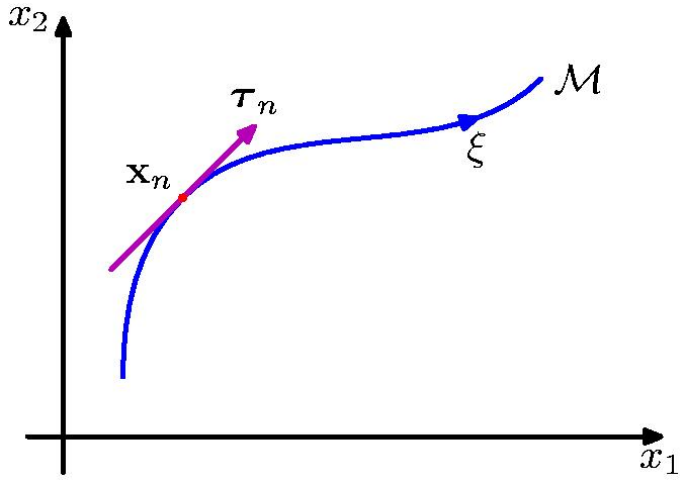
Original Image Warped Images



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
- input vector $\mathbf{x}_n$
- Continuous transformation sweeps a manifold $\mathcal{M}$ in D-dimensional input space



Two-dimensional input space showing effect of continuous transformation with single parameter ξ
Vector resulting from transformation is $s(x,\xi)$ so that $s(x,0)=x$
Tangent to curve $\mathcal{M}$ is given by

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

Tangent Propagation as Regularization

- Under transformation of input vector
  - 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
  - To encourage local invariance in neighborhood of data point

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

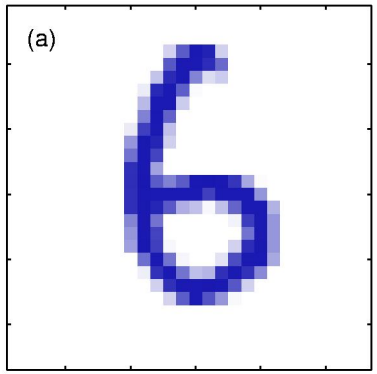
where λ is a regularization coefficient and

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

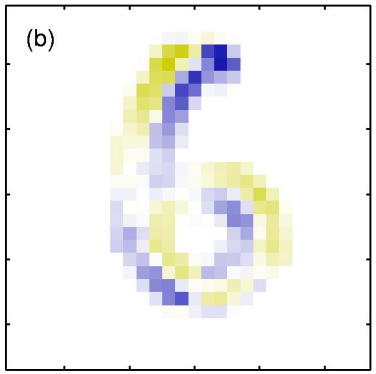
Tangent vector from finite differences

- 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
- *Tangent Distance* used to build invariance properties with *nearest-neighbor* classifiers is a related technique

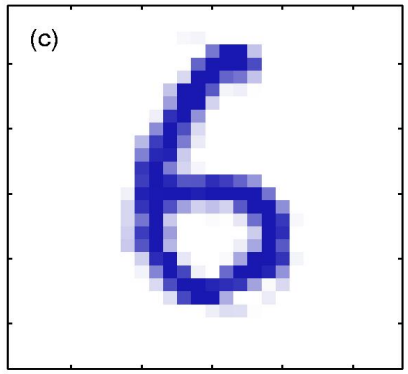
Original Image x



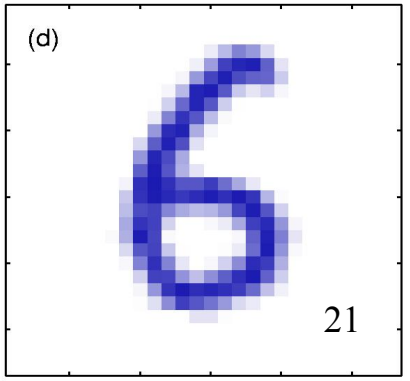
Tangent vector τ corresponding to small rotation



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