

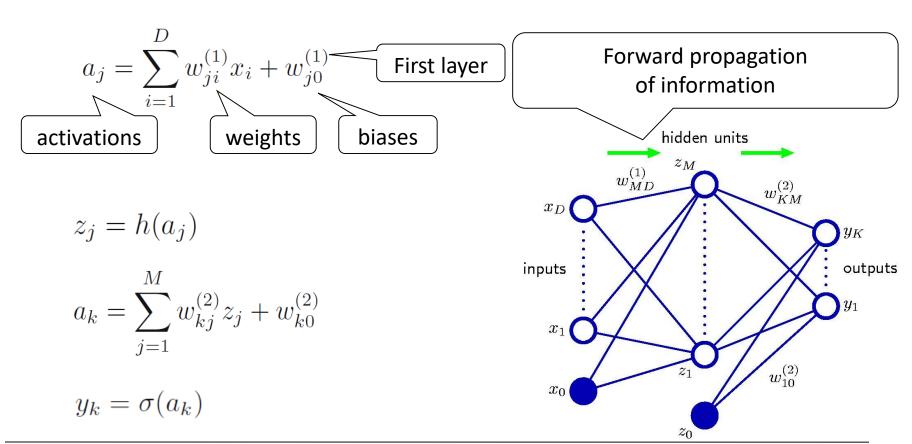
### Learning Objectives

- 1. What are feed-forward network functions?
- 2. What are cost functions for network training?
- 3. What are error propagation schemes for neural networks?
- 4. How to compute Jacobian and Hessian matrices?
- 5. How to achieve regularization for neural networks?
- 6. What are Bayesian MAP neural networks?
- 7. What are convolution neural networks?
- 8. What are generative adversarial networks?

### **Outlines**

- Feedforward Network Functions
- Network Training
- Error Backpropagation
- Jacobian and Hessian Matrices
- Network Regularization
- MAP Neural Networks
- CNN and GAN

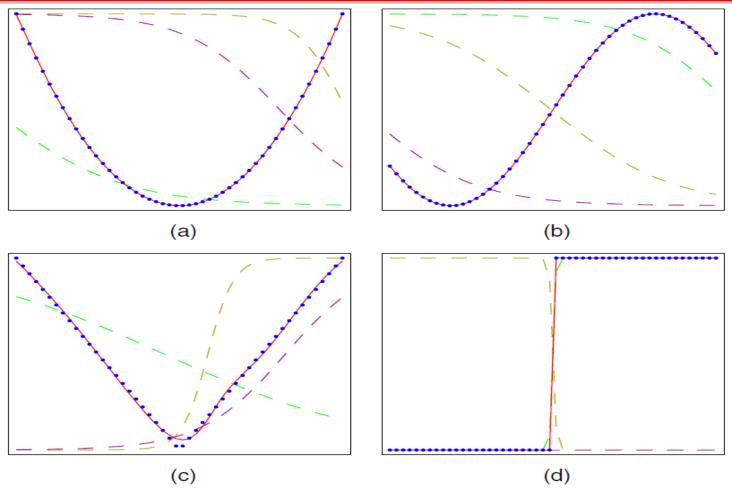
Goal: to extend linear model by making the basis functions depend on parameters, allow these parameters to be adjusted.



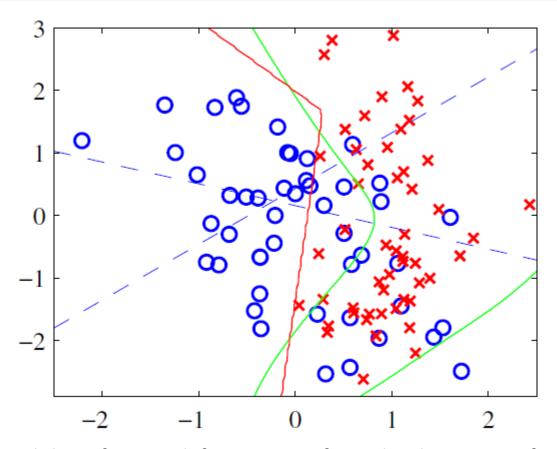
☐ The overall network function, comprising two stage processing, becomes a linear regression model with adaptive basis functions

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left( \sum_{j=1}^M w_{kj}^{(2)} h \left( \sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

Adaptive basis functions



The capability of network functions to approximate different functions



The capability of network functions to form the decision surface: **red line**; the optimal decision surface: **green line** 

#### **Outlines**

- Feedforward Network Functions
- Network Training
- Error Backpropagation
- Jacobian and Hessian Matrices
- Network Regularization
- MAP Neural Networks
- CNN and GAN

# **Network Training (I)**

 $\Box$  t has a Gaussian distribution with an x-dependent mean

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}\left(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}\right)$$

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta)$$
 (likelihood function)

$$\frac{\beta}{2} \sum_{1}^{N} \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2} \ln \beta + \frac{N}{2} \ln(2\pi) \quad \text{(Negative log)}$$

### **Network Training (II)**

■ Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

 $\blacksquare$  Having found  $\mathbf{w}_{\mathsf{ML}}$ , the value of  $\beta$  can be found by minimizing negative log likelihood

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}_{\mathrm{ML}}) - t_n\}^2$$

### **Network Training (III)**

- The choice of output unit activation function and matching error function
  - ✓ Standard regression problems:

Error function: Negative log-likelihood function

Output unit activation function: identity

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

$$y = a$$

### Network Training (IV)

- The choice of output unit activation function and matching error function
  - ✓ For binary classification problems:

**Error function**: cross-entropy error function

Output unit activation function: logistic sigmoid

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

$$y = \sigma(a) \equiv \frac{1}{1 + \exp(-a)}$$

### Network Training (V)

- The choice of output unit activation function and matching error function
  - ✓ For multiclass problems:

Error function: cross-entropy error function

Output unit activation function: softmax function

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

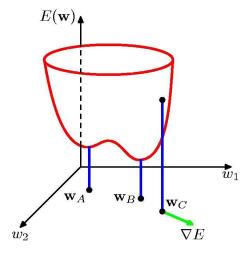
### Parameter Optimization

☐ Error E(w) is a smooth continuous function of w and smallest value will occur at a point in weight space

$$\nabla E(\mathbf{w}) = 0$$

Global minimum:  $\mathbf{w}_{\mathrm{B}}$ 

Local minima:  $\mathbf{w}_{\mathrm{A}}$ 



■ Most techniques involve choosing some initial value for weight vector and then moving through weight space in a succession of steps of the form

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta \mathbf{w}^{(\tau)}$$

### Local Quadratic Approximation

☐ Taylor expansion of E(w) around some point

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}} \mathbf{b} + \frac{1}{2} (\mathbf{w} - \widehat{\mathbf{w}})^{\mathrm{T}} \mathbf{H} (\mathbf{w} - \widehat{\mathbf{w}})$$

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w} = \widehat{\mathbf{w}}} \quad \text{(gradient)} \qquad (\mathbf{H})_{ij} \equiv \left. \frac{\partial E}{\partial w_i \partial w_j} \right|_{\mathbf{w} = \widehat{\mathbf{w}}} \text{(Hessian Matrix)}$$

☐ Local approximation of the gradient

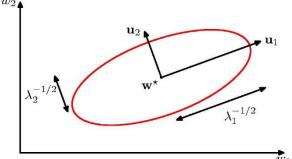
$$\nabla E \simeq \mathbf{b} + \mathbf{H}(\mathbf{w} - \widehat{\mathbf{w}})$$
 If  $b = 0$ ,  $\widehat{\mathbf{w}} = \mathbf{w} - \mathbf{H}^{-1} \nabla E$ 

 $\square$  Local quadratic approximation when b = 0 at w\*

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^{\mathrm{T}} \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_{i} \lambda_i \alpha_i^2$$

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2$$



### Use of Gradient Information

■ In the quadratic approximation, computational cost to find minimum is O(W³)

W is the dimensionality of w

- ✓ perform O(W²) evaluations, each of which would require O(W) steps.
- In an algorithm that makes use of the gradient information, computational cost is O(W<sup>2</sup>)
  - ✓ By using error backpropagation, O(W) gradient evaluations and each such evaluation takes only O(W) steps.

### **Gradient Descent Optimization**

■ Weight update to comprise a small step in the direction of the negative gradient

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

- Batch method
  - ✓ Techniques that use the whole data set at once.
  - ✓ The error function always decreases at each iteration unless the weight vector has arrived at a local or global minimum.
- On-line version of gradient descent
  - ✓ Sequential gradient descent or stochastic gradient descent
  - ✓ Update to the weight vector based on one data point at a time.
  - ✓ Can handle redundancy in the data much more efficiently.
  - ✓ The possibility of escaping from local minima.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)})$$

#### **Outlines**

- Feedforward Network Functions
- Network Training
- Error Backpropagation
- Jacobian and Hessian Matrices
- Network Regularization
- MAP Neural Networks
- CNN and GAN

# Error Backpropagation (I)

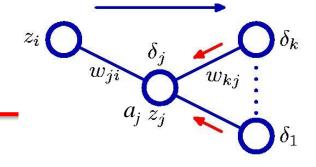
■ Least Square Error Function

$$E = \sum_{n} E_{n}$$
  $E_{n} = \frac{1}{2} \sum_{k} (y_{k} - t_{k})^{2}$ 

$$y_k = a_k \qquad \frac{\partial y_k}{\partial a_k} = 1$$

$$\frac{\partial E_n}{\partial y_k} = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

# Error Backpropagation (II)



 $a_k y_k$ 

$$a_j = \sum_i w_{ji} z_i$$
  $z_j = h(a_j)$   $\frac{\partial a_j}{\partial w_{ji}} = z_i$ 

$$a_k = \sum_{j} w_{kj} z_j$$
  $y_k = a_k$   $\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_{k} \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$ 

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} = \delta_k z_j \left| \frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \right|$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

 $\delta$  rules

### **Error Backpropagation (III)**

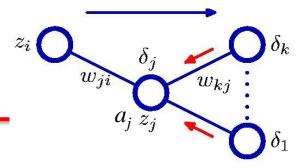
□ Cross-Entropy Error Function

$$E = \sum_{n} E_{n}$$
  $E_{n} = -\sum_{k} t_{k} \ln y_{k} + (1 - t_{k}) \ln(1 - y_{k})$ 

$$\frac{\partial E_n}{\partial y_k} = -\frac{t_k}{y_k} + \frac{1 - t_k}{1 - y_k} = \frac{y_k - t_k}{y_k (1 - y_k)}$$

$$\frac{\partial y_k}{\partial a_k} = y_k (1 - y_k) \qquad \qquad y_k = \sigma(a_k)$$

# Error Backpropagation (IV)



 $a_k y_k$ 

$$a_j = \sum_i w_{ji} z_i$$
  $z_j = h(a_j)$   $\frac{\partial a_j}{\partial w_{ji}} = z_i$ 

$$a_k = \sum_j w_{kj} z_j$$
  $y_k = \sigma(a_k)$   $\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$ 

$$\delta_k \equiv \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

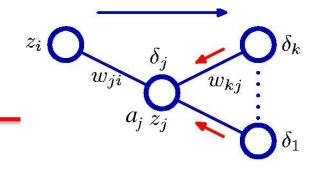
$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

$$\frac{\partial E_n}{\partial w_{kj}} = \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial w_{kj}} = \delta_k z_j \left[ \frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i \right]$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

 $\delta$  rules

# Error Backpropagation (V)



 $a_k y_k$ 

#### ■ Error Backpropagation

① Apply an input vector  $x_n$  to the network and forward propagate through the network using equations below to find the activations of all the hidden and output units.

$$a_j = \sum_i w_{ji} z_i \quad z_j = h(a_j)$$

- ② Evaluate the  $\delta_k$  for all the output units using  $\,\delta_k = y_k t_k\,$
- 3 Backpropagate the  $\delta$  using  $\delta_j = h'(a_j) \sum_k w_{jk} \delta_k$

to obtair  $\delta_j$  for each hidden unit in the network

4 Use  $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$  to evaluate the required derivatives

### A Simple Example

$$h(a) \equiv \tanh(a) \qquad \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

$$h'(a) = 1 - h(a)^2$$

$$E_n = \frac{1}{2} \sum_{k=1}^K (y_k - t_k)^2 \quad (y_k: \text{ output unit k, } t_k: \text{ the corresponding target})$$

$$a_j = \sum_{i=0}^D w_{ji}^{(1)} x_i$$

$$z_j = \tanh(a_j) \qquad \delta_j = (1 - z_j^2) \sum_{k=1}^K w_{kj} \delta_k$$

$$y_k = \sum_{j=0}^M w_{kj}^{(2)} z_j \qquad \frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i \quad \frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j \quad \text{for rules}$$

#### **Outlines**

- Feedforward Network Functions
- Network Training
- Error Backpropagation
- Jacobian and Hessian Matrices
- Network Regularization
- Bayesian Neural Networks
- CNN and GAN

### The Jacobian Matrix

- ☐ The technique of backpropagation can also be applied to the calculation of other derivatives.
- ☐ The Jacobian matrix
  - ✓ Elements are given by the derivatives of the network outputs w.r.t. the inputs

$$J_{ki} \equiv \frac{\partial y_k}{\partial x_i}$$

 $\checkmark$  Minimizing an error function E w.r.t. the parameter w

$$\frac{\partial E}{\partial w} = \sum_{k,j} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial z_j} \frac{\partial z_j}{\partial w}$$

### The Jacobian Matrix

- A measure of the *local sensitivity* of the outputs to change in each of the input variables.
  - ✓ In general, the network mapping is nonlinear, and so the elements will not be constants. This is valid provided the  $|\Delta x_i|$  are small.

$$\Delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \Delta x_i$$

✓ The evaluation of the Jacobian Matrix

$$J_{ki} = \frac{\partial y_k}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} \frac{\partial a_j}{\partial x_i} \qquad \frac{\partial y_k}{\partial a_j} = \sum_l \frac{\partial y_k}{\partial a_l} \frac{\partial a_l}{\partial a_j}$$
$$= \sum_j w_{ji} \frac{\partial y_k}{\partial a_j} \qquad = h'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l}$$

### The Hessian Matrix

#### ☐ The Hessian matrix

- ✓ The second derivatives form the elements of H,
- $rac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}$

- ✓ Important roles:
  - Considering the second-order properties of the error surfaces.
  - Forms the basis of a fast procedure for re-training a feedforward network following a small change in the training data
  - The inverse of the Hessian is used to identify the least significant weights in a network (pruning algorithm)
  - In the Laplace approximation for a Bayesian NN
- ✓ Various approximation schemes are used to evaluate the Hessian matrix for a NN.

### Hessian: Diagonal Approximation

- ☐ Inverse of the Hessian is useful, and inverse of diagonal matrix is trivial to evaluate.
- Consider an error function
  - ✓ Replaces the off-diagonal elements with zeros
  - ✓ The diagonal elements of the Hessian:

$$\begin{split} \frac{\partial^2 E_n}{\partial w_{ji}^2} &= \frac{\partial^2 E_n}{\partial a_j^2} z_i^2 \\ \frac{\partial^2 E_n}{\partial a_j^2} &= h'(a_j)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''(a_j) \sum_k w_{kj} \frac{\partial E^n}{\partial a_k} \\ \frac{\partial^2 E_n}{\partial a_j^2} &= h'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j) \sum_k w_{kj} \frac{\partial E_n}{\partial a_k} \text{ (neglect off-diagonal)} \end{split}$$

### **Outer Product Approximation**

$$\mathbf{H} = \nabla \nabla E = \sum_{n=1}^{N} \nabla y_n \nabla y_n + \sum_{n=1}^{N} (y_n - t_n) \nabla \nabla y_n$$

- Output y happen to be very close to the target values t, the second term will be small and can be neglected.
  - Or the value of y-t is uncorrelated with the value of the second derivative term, then the whole term will average to zero.

$$\mathbf{H} \simeq \sum_{n=1}^{N} \mathbf{b}_{n} \mathbf{b}_{n}^{\mathrm{T}} \qquad \mathbf{b}_{n} = \nabla y_{n} = \nabla a_{n}$$

#### Inverse Hessian

- ☐ A procedure for approximating the inverse of the Hessian
  - ✓ First, we write the outer-product approximation
  - ✓ Derive a sequential procedure for building up the Hessian by including data points one at a time.

$$\mathbf{H}_{L+1} = \mathbf{H}_L + \mathbf{b}_{L+1} \mathbf{b}_{L+1}^{\mathrm{T}}$$

$$\mathbf{H}_{L+1}^{-1} = \mathbf{H}_{L}^{-1} - \frac{\mathbf{H}_{L}^{-1}\mathbf{b}_{L+1}\mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_{L}^{-1}}{1 + \mathbf{b}_{L+1}^{\mathrm{T}}\mathbf{H}_{L}^{-1}\mathbf{b}_{L+1}}$$

✓ The initial matrix  $H_0 = H + \alpha I$ α is a small quantity, not sensitive to the precise value of α.

#### **Outlines**

- Feedforward Network Functions
- Network Training
- Error Backpropagation
- Jacobian and Hessian Matrices
- Network Regularization
- MAP Neural Networks
- CNN and GAN

### Neural Network Regularization (I)

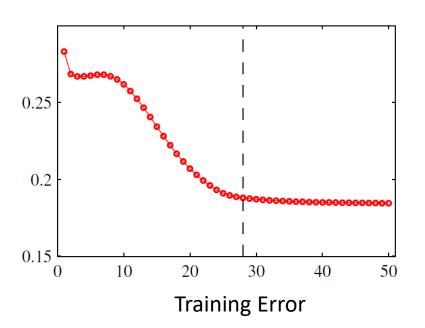
Consistent Gaussian prior

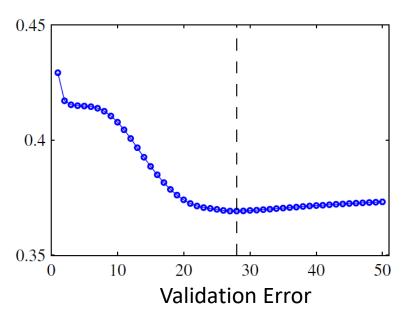
$$p(\mathbf{w}) \propto \exp\left(-\frac{1}{2}\sum_{k} \alpha_{k} \|\mathbf{w}\|_{k}^{2}\right)$$

$$\|\mathbf{w}\|_k^2 = \sum_{j \in \mathcal{W}_k} w_j^2.$$

### Neural Network Regularization (II)

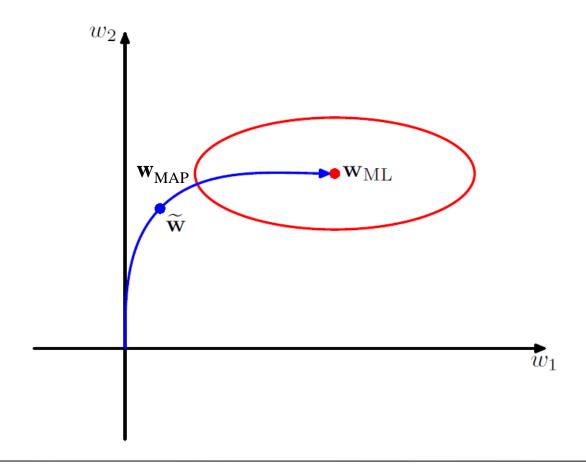
#### ■ Early stopping





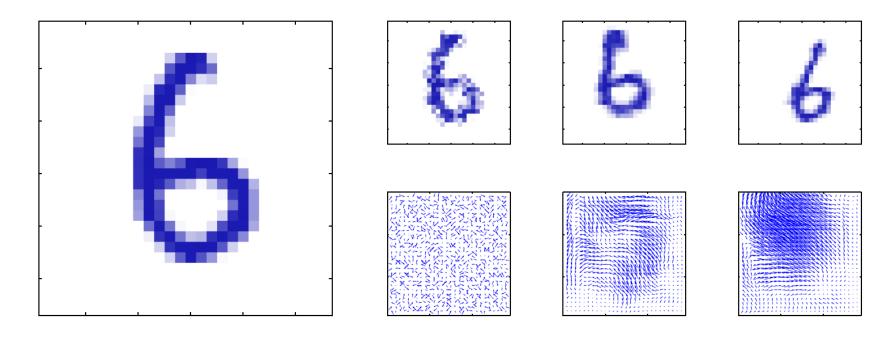
# Neural Network Regularization (II)

**□** Early stopping



# Neural Network Regularization (III)

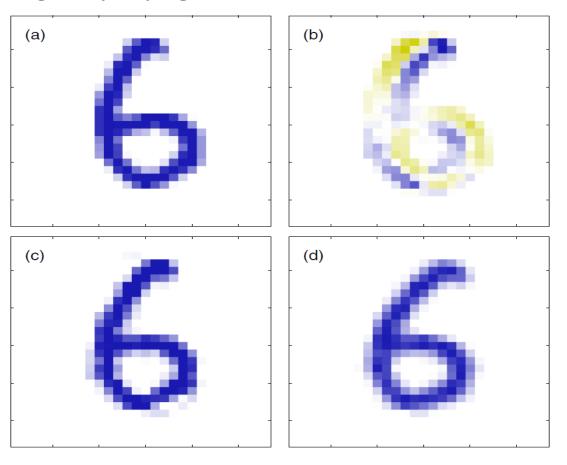
■ Invariances



Data Augmentation with random displacement fields and Gaussian smoothing

# Neural Network Regularization (IV)

#### ■ Tangent propagation



**Data Augmentation** 

### Neural Network Regularization (V)

#### □ Training with transformed data

$$E = \frac{1}{2} \iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \, dt$$

$$\widetilde{E} = \frac{1}{2} \iiint \{y(\mathbf{s}(\mathbf{x}, \xi)) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) p(\xi) \, d\mathbf{x} \, dt \, d\xi$$

$$\widetilde{E} = \frac{1}{2} \iint \{y(\mathbf{x}) - t\}^2 p(t|\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \, dt$$

$$+ \mathbb{E}[\xi] \iint \{y(\mathbf{x}) - t\} \boldsymbol{\tau}^{\mathrm{T}} \nabla y(\mathbf{x}) p(t|\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \, dt$$

$$+ \mathbb{E}[\xi^2] \iint \left[ \{y(\mathbf{x}) - t\} \frac{1}{2} \left\{ (\boldsymbol{\tau}')^{\mathrm{T}} \nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}} \nabla \nabla y(\mathbf{x}) \boldsymbol{\tau} \right\} + \left( \boldsymbol{\tau}^{\mathrm{T}} \nabla y(\mathbf{x}) \right)^2 \right] p(t|\mathbf{x}) p(\mathbf{x}) \, d\mathbf{x} \, dt + O(\xi^3).$$

### Neural Network Regularization (V)

#### □ Training with transformed data

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

$$\Omega = \int \left[ \{ y(\mathbf{x}) - \mathbb{E}[t|\mathbf{x}] \} \frac{1}{2} \left\{ (\boldsymbol{\tau}')^{\mathrm{T}} \nabla y(\mathbf{x}) + \boldsymbol{\tau}^{\mathrm{T}} \nabla \nabla y(\mathbf{x}) \boldsymbol{\tau} \right\} + \left( \boldsymbol{\tau}^{T} \nabla y(\mathbf{x}) \right)^{2} \right] p(\mathbf{x}) d\mathbf{x}$$

$$\Omega = \frac{1}{2} \int \left( \boldsymbol{\tau}^T \nabla y(\mathbf{x}) \right)^2 p(\mathbf{x}) \, d\mathbf{x}$$

Tangent propagation regularization

$$\Omega = \frac{1}{2} \int \|\nabla y(\mathbf{x})\|^2 p(\mathbf{x}) \, d\mathbf{x}$$

Tikhonov regularization additional noise

### Neural Network Regularization (VI)

■ Soft weight sharing

$$p(\mathbf{w}) = \prod_{i} p(w_i) \qquad p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(w_i | \mu_j, \sigma_j^2)$$

$$\Omega(\mathbf{w}) = -\sum_{i} \ln \left( \sum_{j=1}^{M} \pi_{j} \mathcal{N}(w_{i} | \mu_{j}, \sigma_{j}^{2}) \right)$$

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \Omega(\mathbf{w})$$