ANLY-601
Advanced Pattern Recognition

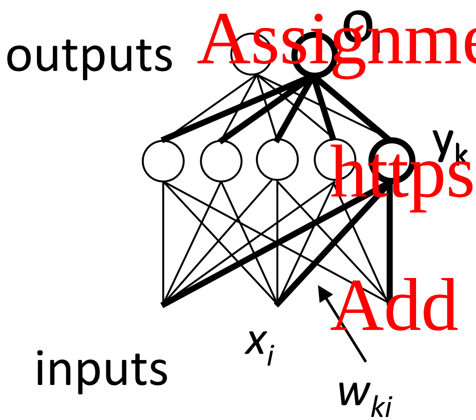
Spring 2018

L20 --- Neural Nets II

GEORGETOWN UNIVERSITY

MLP Output

Signal propagation (forward pass, bottom-up)



$$y_k = \sigma\left(\sum_i w_{ki}\, x_i\right) \equiv \sigma(net_k)$$

$$O_l = g\left(\sum_k w_{lk}\, y_k\right) \equiv g(net_l)$$

$$with \quad g(net_l) = \begin{cases} \sigma(net_l) & for\ classification \\ net_l & for\ regression \ldots \end{cases}$$

Gradient Descent in MLP

Cost function as before:

number of outputs

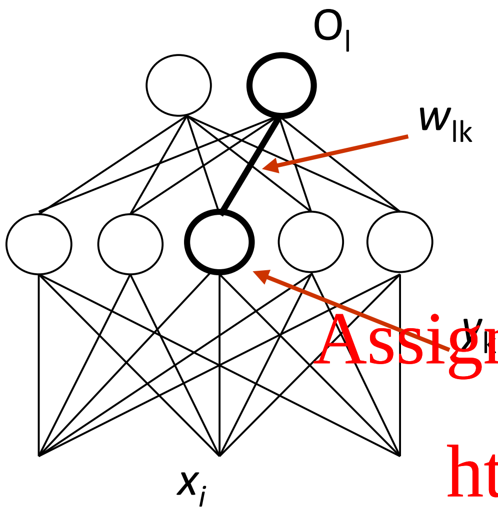
$$\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} \left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2$$

Learning by gradient descent

$$\Delta w_{ij} = -\eta \, \frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ij}}$$

Let's calculate the components of the gradient

MLP Error Gradient



$$\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} \left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2$$

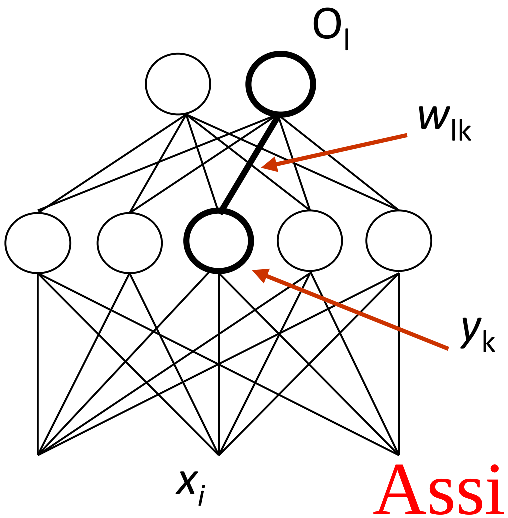
$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ij}}$$

1. Derivative with respect to a weight to the output.

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{lk}} = \frac{\partial}{\partial w_{lk}} \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} \left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2 = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} \frac{\partial}{\partial w_{lk}} \left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2$$

$$= \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} 2\left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right) \frac{\partial}{\partial w_{lk}} \left(-O_m(\vec{x}_\alpha)\right)$$

$$= \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} 2\left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)\left(-\delta_{ml}\right) \frac{\partial O_l(x_\alpha)}{\partial w_{lk}} = -\frac{1}{D} \sum_{\alpha=1}^{D} \left(t_{\alpha l} - O_l(\vec{x}_\alpha)\right) \frac{\partial O_l(x_\alpha)}{\partial w_{lk}}$$



MLP Error Gradient

Derivative with respect to a weight to the output.

We have so far:

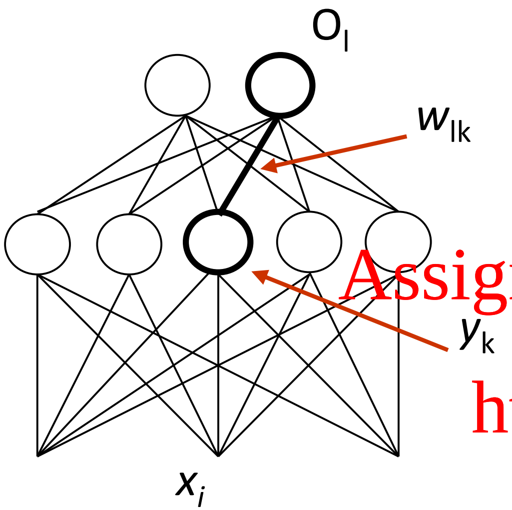
$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{lk}} = -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\frac{\partial O_l(\vec{x}_\alpha)}{\partial w_{lk}} = -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\frac{\partial g(net_{\alpha l})}{\partial w_{lk}}$$

$$= -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\frac{\partial net_{\alpha l}}{\partial w_{lk}} = -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\frac{\partial(\sum_i w_{li}\ y_{\alpha i})}{\partial w_{lk}}$$

$$= -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\sum_i \delta_{ik}\ y_{\alpha i} = -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\ y_{\alpha k}$$

GEORGETOWN UNIVERSITY

MLP Error Gradient



1. Derivative with respect to a weight to the output.

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{lk}} = -\frac{1}{D}\sum_{\alpha=1}^{D} \underbrace{(t_{\alpha l} - O_l(\vec{x}_{\alpha}))}\; g'(net_{\alpha l})\; y_{\alpha k}$$

Weight from node *k* to output node *l*.

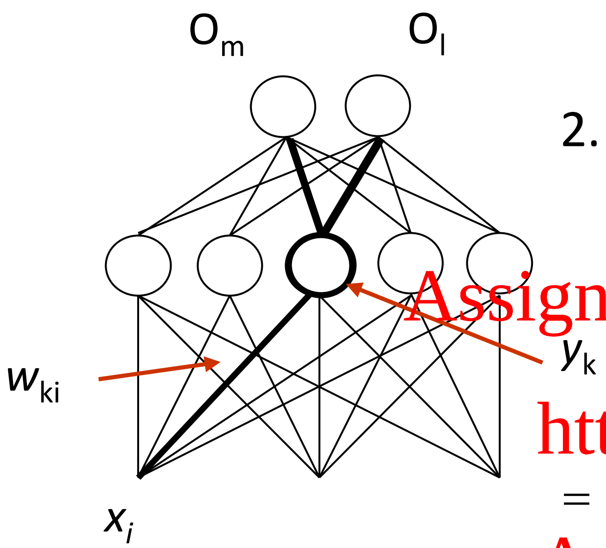
Error at node *l*

Slope of activation function for node *l*

Signal from node *k*

GEORGETOWN UNIVERSITY

MLP Gradient



2. Derivative with respect to weights to hidden units

$$\frac{\partial E(\vec{w})}{\partial w_{ki}} = \frac{\partial}{\partial w_{ki}} \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} (t_{\alpha n} - O_n(\vec{x}_\alpha))^2$$

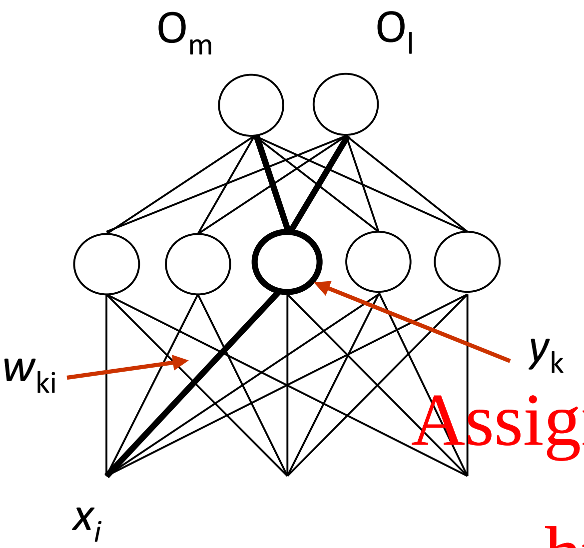
$$= \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} \frac{\partial}{\partial w_{ki}} (t_{\alpha n} - O_n(\vec{x}_\alpha))^2$$

$$= -\frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} 2(t_{\alpha n} - O_n(\vec{x}_\alpha)) \frac{\partial\, O_n(\vec{x}_\alpha)}{\partial w_{ki}} = -\frac{1}{D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} (t_{\alpha n} - O_n(\vec{x}_\alpha)) \frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}} \frac{\partial\, y_{\alpha k}}{\partial\, w_{ki}}$$

$$= -\frac{1}{D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} (t_{\alpha n} - O_n(\vec{x}_\alpha)) \frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}} \frac{\partial\, \sigma(net_{\alpha k})}{\partial\, w_{ki}}$$

GEORGETOWN UNIVERSITY

MLP Gradient



2. Derivative with respect to weights to hidden units

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ki}} = \frac{\partial}{\partial w_{ki}} \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} (t_{\alpha n} - O_n(\vec{x}_\alpha))^2$$

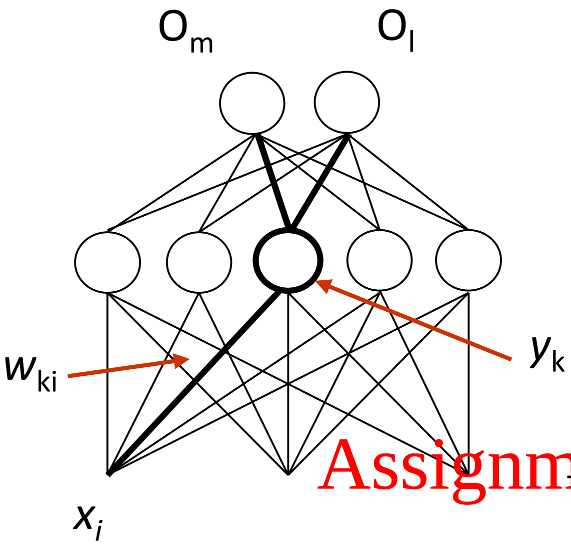
$$= -\frac{1}{D} \sum_{\alpha=1}^{D} \sum_{n=1}^{N_O} (t_{\alpha n} - O_n(\vec{x}_\alpha)) \frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}} \frac{\partial\, \sigma(net_{\alpha k})}{\partial\, w_{ki}} \quad (1)$$

Now look at the two pieces:

$$\frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}} = \frac{\partial\, g(net_{\alpha n})}{\partial y_{\alpha k}} = g'(net_{\alpha n}) \frac{\partial\, net_{\alpha n}}{\partial y_{\alpha k}} = g'(net_{\alpha n}) \frac{\partial \sum_i w_{ni}\, y_{\alpha i}}{\partial y_{\alpha k}} = g'(net_{\alpha n})\, w_{nk}$$

$$\frac{\partial\, \sigma(net_{\alpha k})}{\partial\, w_{ki}} = \sigma'(net_{\alpha k}) \frac{\partial\, net_{\alpha k}}{\partial\, w_{ki}} = \sigma'(net_{\alpha k}) \frac{\partial \sum_j w_{kj}\, x_{\alpha j}}{\partial\, w_{ki}} = \sigma'(net_{\alpha k})\, x_{\alpha i}$$

Substitute into (1)



MLP Gradient

Derivative with respect
to weights to hidden units

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ki}} = -\frac{1}{D}\sum_{\alpha=1}^{D}\sum_{n=1}^{N_O}(t_{\alpha n} - O_n(\vec{x}_\alpha))\frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}}\frac{\partial\, \sigma(net_{\alpha k})}{\partial\, w_{ki}}$$

$$\frac{\partial\, O_n(\vec{x}_\alpha)}{\partial y_{\alpha k}} = g'(net_{\alpha n})\, w_{nk} \qquad \frac{\partial\, \sigma(net_{\alpha k})}{\partial\, w_{ki}} = \sigma'(net_{\alpha k})\; x_{\alpha i}$$

So $$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ki}} = -\frac{1}{D}\sum_{\alpha=1}^{D}\left[\sum_{n=1}^{N_0}\underbrace{(t_{\alpha n} - O_n(\vec{x}_\alpha))\, g'(net_{\alpha n})\; w_{nk}}\right]\sigma'(net_{\alpha k})\;\; x_{\alpha i}$$

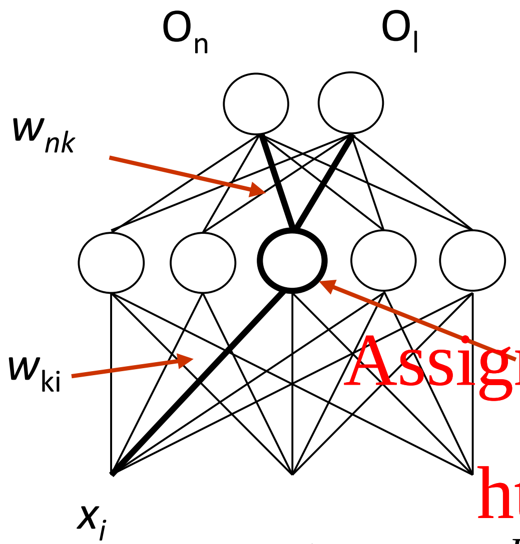
Pseudo-error at hidden node *k*

Activation function slope at node *k*

Signal at node *i*

GEORGETOWN UNIVERSITY

MLP Error Gradient



Derivative with respect to weights to hidden units

signal at node *i.*

$$\frac{\partial E(\vec{w})}{\partial w_{ki}} = -\frac{1}{D}\sum_{\alpha=1}^{D}\left[\overbrace{\sum_{n=1}^{N_0}(t_{\alpha n}-O_n(\vec{x}_\alpha))\,g'(net_{\alpha n})\,w_{nk}}^{\text{Pseudo error at hidden node } k}\right]\sigma'(net_{\alpha k})\;x_{\alpha i}$$

Error at output node *n*

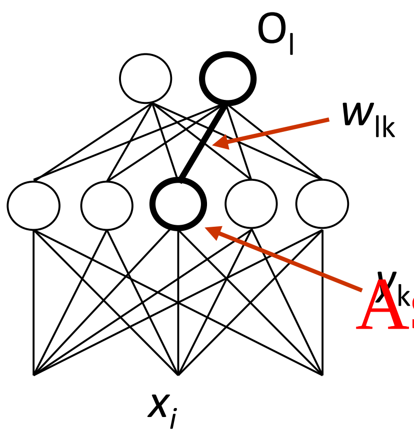
multiplied by slope at output node *n*

passed <u>backwards</u> through the weight
from node *n* to node *k,* hence “error back-propagation”

slope of activation function at node *k.*

GEORGETOWN UNIVERSITY

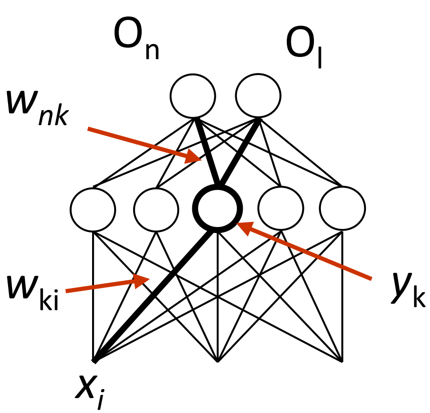
Summary MLP Error Gradients



1. Derivative with respect to a weight to an output.

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{lk}} = -\frac{1}{D}\sum_{\alpha=1}^{D}(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\ y_{\alpha k}$$

stochastic ver. $\frac{\partial \mathcal{E}_\alpha(\vec{w})}{\partial w_{lk}} = -(t_{\alpha l} - O_l(\vec{x}_\alpha))\ g'(net_{\alpha l})\ y_{dk}$



2. Derivative with respect to weights to hidden units

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ki}} = -\frac{1}{D}\sum_{\alpha=1}^{D}\left[\sum_{n=1}^{N_0}(t_{\alpha n} - O_n(\vec{x}_\alpha))\, g'(net_{\alpha n})\ w_{nk}\right]\sigma'(net_{\alpha k})\ x_{\alpha i}$$

stochastic version

$$\frac{\partial \mathcal{E}_\alpha(\vec{w})}{\partial w_{ki}} = -\left[\sum_{n=1}^{N_0}(t_{\alpha n} - O_n(\vec{x}_\alpha))\, g'(net_{\alpha n})\ w_{nk}\right]\sigma'(net_{\alpha k})\ x_{\alpha i}$$

Backpropagation Learning Algorithm

Batch Mode (uses ALL data at each step)

choose learning rate η

initialize w_{ij} % Usually to "small" random numbers

while ($\Delta\mathcal{E} / \mathcal{E} > \varepsilon$) % Fractional change $\varepsilon \sim 10^{-4}$–10^{-6}

calculate mean square error $\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} (t_{\alpha m} - O_m(\vec{x}_\alpha))^2$

calculate <u>all</u> error derivatives and step downhill $\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ij}}$

endwhile

Backpropagation Learning Algorithm

Stochastic or On-Line Mode

(uses ONE input/target pair for each step)

choose learning rate η

initialize w_{ij} usually to "small" random numbers

while ($\Delta\mathcal{E} / \mathcal{E} > \varepsilon$)   % Fractional change $\varepsilon \sim 10^{-4}$ - 10^{-6}

  calculate mean square error $\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} (t_{\alpha m} - O_m(\vec{x}_\alpha))^2$

  for α = 1 … D   % Step through data, or do D random draws with replacement

    change all weights w_{ij} as $\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}_\alpha(\vec{w})}{\partial w_{ij}}$

  end for

endwhile

Comments

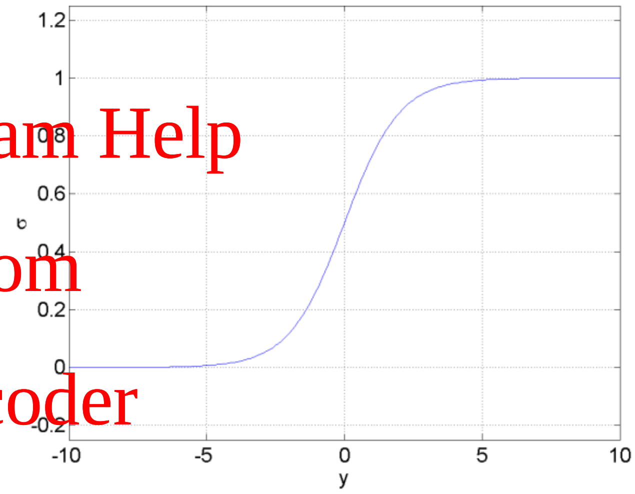
Cost function may not be convex, can have local optima, some may be quite poor. In practice, this is not a show-stopper.

Usually initialize with random weights close to zero. Then

$$net_k = \sum_i w_{ki} x_i$$

will be small,

and $\sigma(net_k) \cong net_k$



So early on, the network output will be nearly linear in the input. Non-linearities are added as training continues.

Comments

Learning algorithms are simply *optimization* methods. Trying to find w that minimizes $\mathcal{E}(w)$. Several other optimization methods, both classical and novel, are brought to bear on the problem.

Deep networks (several layers of sigmoidal hidden nodes) can be very slow to train; gradient with respect to weights near inputs will have multiple factors of σ' which decreases gradient signal. (And the condition number of the Hessian of $\mathcal{E}$ become small.)

Power

Universal approximation theorem

– Any continuous function on a compact subset of the input space (closed and bounded) can be approximated arbitrarily closely by a feedforward network with one layer of sigmoidal hidden units and linear output units.

$$O(\vec{x}) = \sum_{i=1}^{n_h} w_i \, \sigma\left(\sum_j w_{ij} \, x_j\right)$$

That is, weighted sums of sigmoidal functions of the inputs are <u>universal approximators</u>.

Power

- Approximation Accuracy
  - The magnitude of the approximation error decreases with increasing number n_h of hidden units as

$$\mathcal{E} = Order\,(1/n_h)$$

  - Techniques linear in the parameters (fixed basis functions with only their weighting fit)

$$O(\vec{x}) = \sum_{i=1}^{N} w_i \; \phi_i(\vec{x})$$

only achieve error bounded by

$$Order \; (1/n)^{2/d}$$

where d is the dimension of the input space.

CURSE OF DIMESIONALITY

Inductive Bias

The hypothesis space is the continuous weight space! Hard to characterize inductive bias.

Bias can be imposed by adding a *regularizer* to the cost function

$$\mathcal{E}(\vec{w},\lambda) = \frac{1}{2D}\sum_{\alpha=1}^{D}\sum_{m=1}^{N_O}\left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2 + \lambda\, F(\vec{w})$$

where *F(w)* carries the desired bias, and λ characterizes the *strength* with which the bias is imposed.

Inductive Bias

Bias can be imposed by adding a *regularizer* to the cost function

$$\mathcal{E}(\vec{w},\lambda) = \frac{1}{2D} \sum_{\alpha=1}^{D} \sum_{m=1}^{N_O} \left(t_{\alpha m} - O_m(\vec{x}_\alpha)\right)^2 + \lambda\, F(\vec{w})$$

where *F(w)* carries the desired bias, and λ characterizes the *strength* with which the bias is imposed.

Examples :

- small weights (L_2 norm -- decay) $F(\vec{w}) = |\vec{w}|^2 = \sum_i w_i^2$

- small curvature $F(\vec{w}) = \int \left| \frac{\partial^2 O(x)}{\partial x^2} \right|^2 dx$
- Sparse models (L_1 norm) $F(\vec{w}) = |\vec{w}| = \sum_i |w_i|$

Generalization
Overfitting / Underfitting

- We've been talking about fitting the network function to the training data $\{x_\alpha, t_\alpha\}$ $\alpha=1\ldots D$.

- But we really care about the performance of the network on unseen data.

Generalization Overfitting / Underfitting

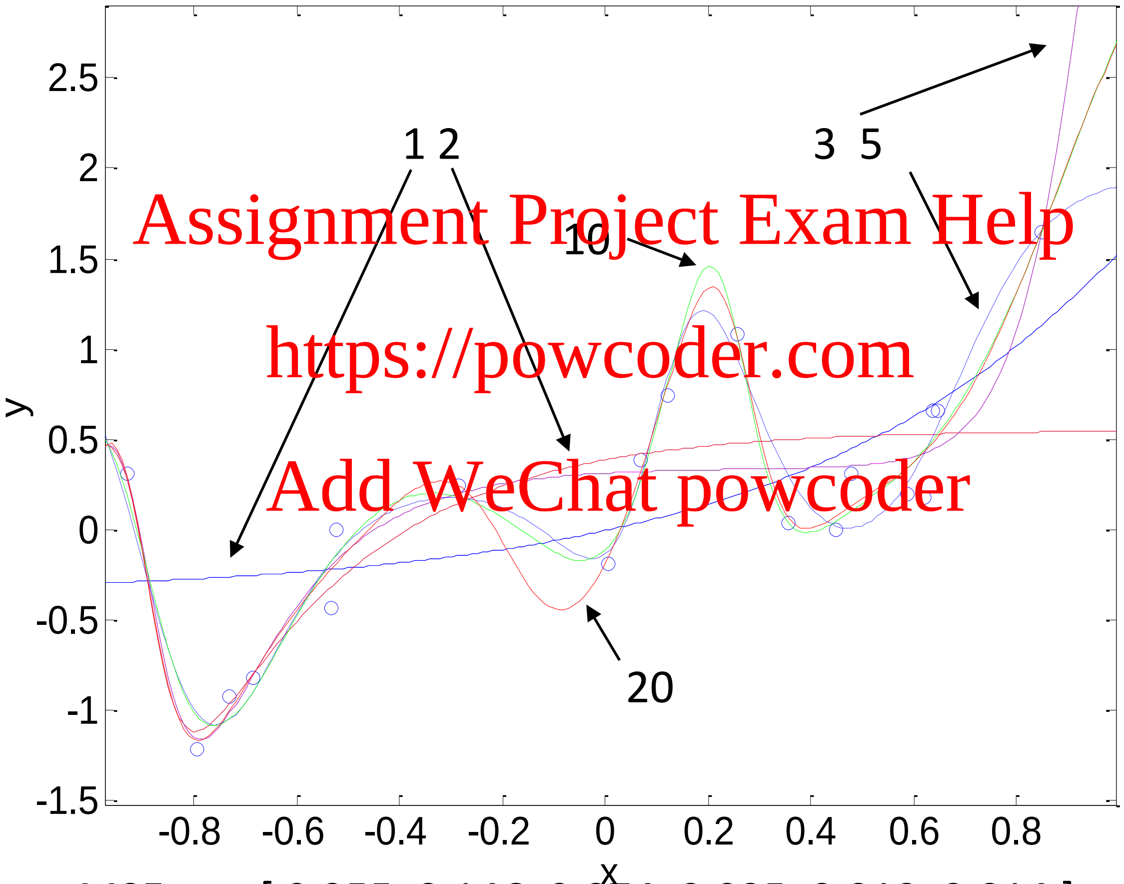
- We can build a network with a very large hidden layer, train it to the bottom of a deep local optimum and very closely approximate the training data. This may be precisely the wrong thing to do.

- Question is "how well does model generalize?" What's the average error on infinitely large test sets? (That is, "what's the statistical expectation of the error on the population?)

  This is called the generalization error.

Overfitting

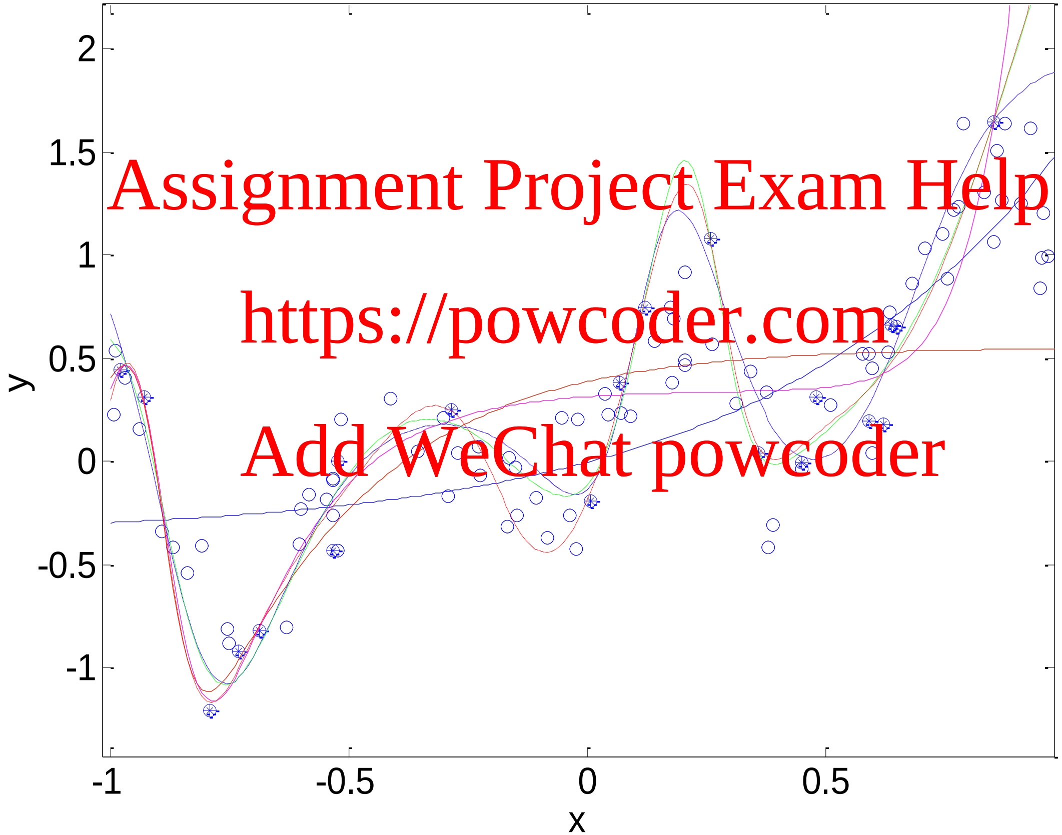
Regression problem, 20 data training points, six different neural network fits with 1, 2, 3, 5, 10, and 20 hidden units.



$\text{MSE}_{\text{train}}$= [0.255 0.146 0.074 0.025 0.019 0.014]

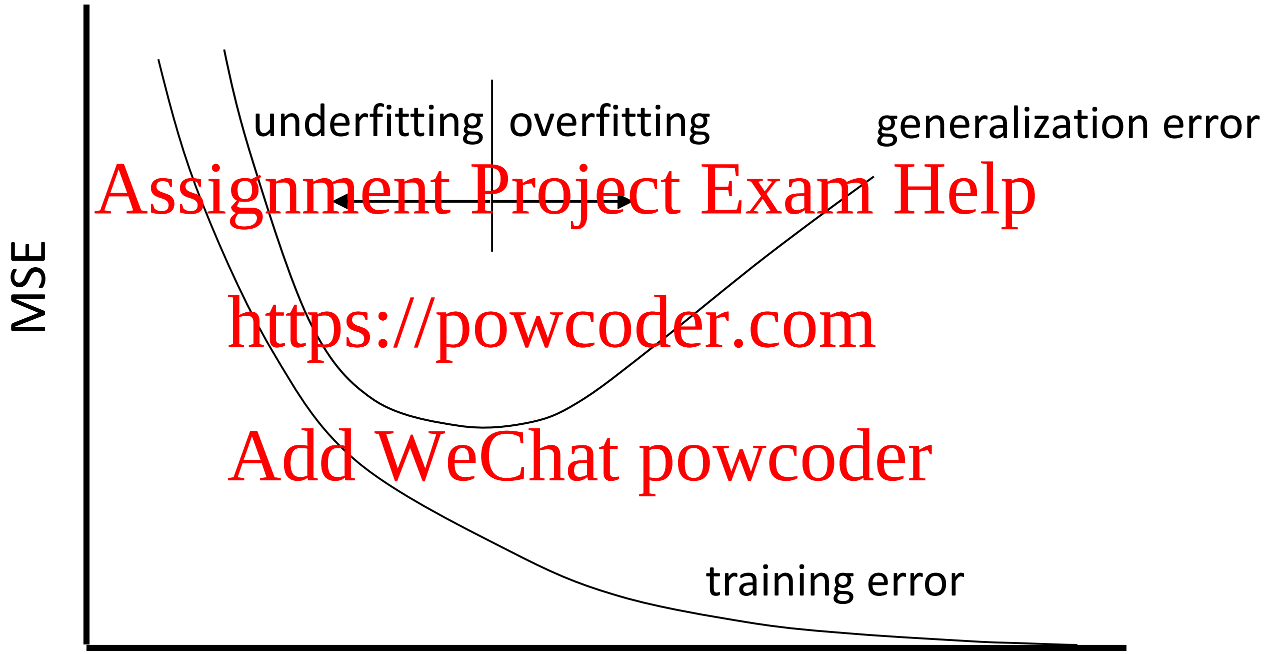
Overfitting

Here's fits to training data overlaid on *test* or *out-of-sample* data (i.e. data not used during the fitting).



MSE_{test}= [0.121 0.203 0.505 0.126 0.216 0.213]

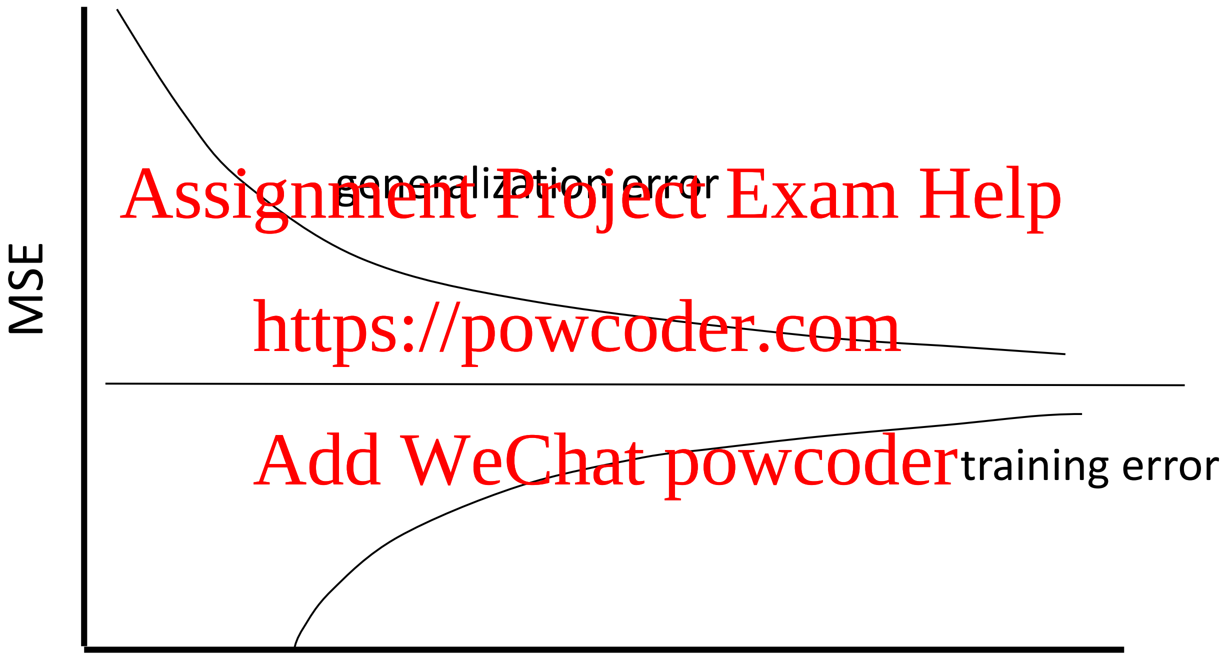
Fixed Training Set Size, Variable Model Size Expected Behavior



Model Size – Number of Hidden Units
(models trained to bottom of deep local optimum)

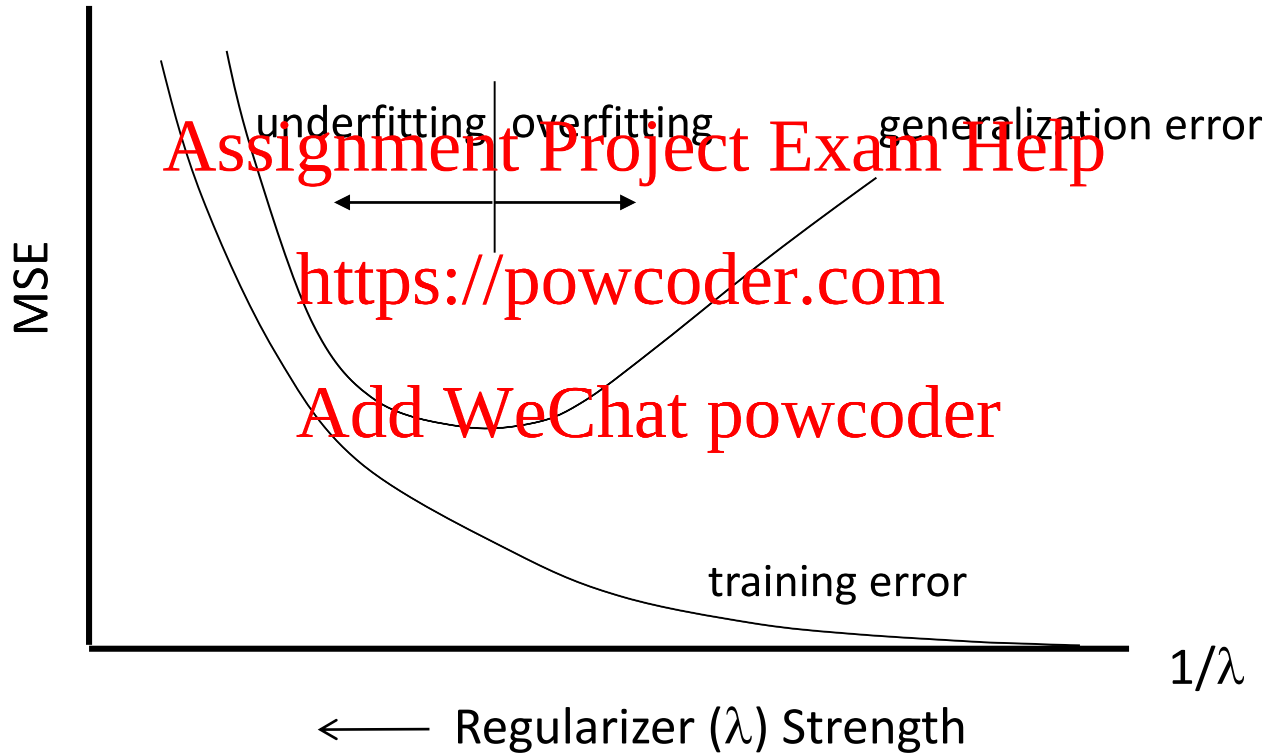
GEORGETOWN UNIVERSITY

Fixed Model Size, Variable Training Set Size Expected Behavior



Training Set Size
(model trained to bottom of deep local optimum)

Fixed Model and Training Data Regularized Cost Function Expected Behavior



Probabilistic Interpretations: Regression

Recall Lecture 8, page 9. The best possible function to minimize the MSE for a regression problem (curve fitting) is

$$O(x) = h(x) = E[t \mid x] \equiv \int t \; p_{t|x}(t \mid x)\, dy$$

the conditional mean of *y* at each point *x*.

Since NN are *universal approximators*, we expect that if we have

a large enough (hidden nodes) network

enough data

our network trained to minimize MSE will have outputs *O(x)* that approximate the regressor *E[t | x].*

Typically, regression networks have linear output nodes (but sigmoidal hidden nodes).

Probabilistic Interpretations: Classification

Consider a classification problem with *L* classes. (Each sample is a member of one and only one class.) The usual NN for such a problem has *L* output nodes with targets t_i , *i=1…L* :

$$t_i(x) = \begin{cases} 1 & if \quad x \in \omega_i \\ 0 & if \quad x \notin \omega_i \end{cases}$$

e.g. for a 4 class problem, for an input x from class 2, the target outputs are [0 1 0 0]. This is called a *one-of-many* representation.

Probabilistic Interpretations: Classification

A simple extension of the result of Lecture 8, page 13 says that the *best possible* function to minimize the MSE of the output for such a representation is to have each output equal to the class posterior

$$O_i(x) = E[t_i \,|\, x] \equiv \sum_{t_i=\{0,1\}} t_i \; p_{t_i|x}(t_i \,|\, x) = p(\omega_i \,|\, x)$$

Typically, networks trained for classification use a sigmoidal output function – usually the logistic

$$\sigma(u) = \frac{1}{1+\exp(-u)}$$

which is naturally bounded to the range [0,1].

Probabilistic Interpretations: Classification

A similar extension of the result in Lecture 8, page 14 says that the absolute minimum of the cross-entropy error measure

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{l=1}^{L} t_l(x_n)\,\ln\left(\frac{O_l(x_n)}{t_l(x_n)}\right) + (1-t_l(x_n))\,\ln\left(\frac{1-O_l(x_n)}{1-t_l(x_n)}\right)$$

is given when each network output equal to the class posterior.

Networks trained to minimize the cross-entropy typically use a logistic output

$$O_l(u_l) = \frac{1}{1+\exp(-u_l)}$$

Notice that this setup is also useful when an object can belong to several classes simultaneously (e.g. medical diagnosis).

Probabilistic Interpretations: Classification

Finally, for multiclass problems, a third cost function emerges. It is based on the multinomial distribution for target values and is exclusively for the case where each object is a member of <u>one and only one class</u>

$$p(\{t_1,\dots,t_L\}\,|\,x) = \prod_{l=1}^{L} O_l(x)^{t_l(x)}$$

Taking negative log-likelihood for a set of examples x_n $n=1\dots N$ results in the cost function

$$E = -\frac{1}{N}\sum_{n=1}^{N}\sum_{l=1}^{L} t_l(x_n)\,\ln\left(O_l(x_n)\right) = -\frac{1}{N}\sum_{n=1}^{N}\sum_{l=1}^{L} t_l(x_n)\,\ln\left(\frac{O_l(x_n)}{t_l(x_n)}\right) - E_0$$

Probabilistic Interpretations: Classification

Networks trained with this cost function ('Cross-entropy 2') typically use the soft-max activation

$$O_l(u_l) = \frac{\exp(u_l)}{\sum_{l'=1}^{L} \exp(u_{l'})}$$

Which is naturally bounded in the interval [0,1].

Note also that $\sum_{l=1}^{L} O_l(u_l) = 1$ as must be the case when each object belongs to one and only one class.

GEORGETOWN UNIVERSITY

Probabilistic Interpretations: Classification

Since NN are universal approximators, we expect that for

Large enough networks (hidden nodes)

Enough data

a classifier NN trained to minimize <u>either</u> the MSE or the cross-entropy error measure will have network outputs that approximate the <u>class posteriors</u>.

This is the usual interpretation of the NN classifier outputs – but care is essential, since the pre-requisites (large networks and enough data) may not be met.

Weight Estimation – Maximum Likelihood

Training a neural network is an estimation problem, where the parameters being estimated are the weights in the network.

Recalling the results in Lecture 10, pages 9 and 10

- Minimizing the MSE is equivalent to maximum likelihood estimation under the assumption of targets with a Gaussian distribution (as usually assumed for regression problems).

- Minimizing the CROSS-ENTROPY error is equivalent to maximum likelihood estimation under the assumption of targets with a Bernouli distribution (as usually assumed for classification problems).

Weight Estimation – Maximum Likelihood

– Minimizing CROSS-ENTROPY 2 is equivalent to maximum likelihood estimation under the assumption of targets with a multinomial distribution as given on p30.

Weight Estimation – MAP

Following earlier discussion of estimation methods, can introduce a prior over network weights p(w) and move from the ML estimate, to the MAP estimate.

This change is mirrored by the change from a cost function, to a regularized cost function

$$U = E - \ln(P(w))$$

Regularizers help improve generalization by reducing the variance of the estimates of the network weights. They do so at the price of introducing bias into the weight estimates.

Weight Estimation – MAP

An often-used regularizer is weight decay

which is equivalent to a Gaussian prior on the weights with mean zero, and covariance (spherically symmetric) $\Sigma = 1/(2\,\lambda)$.

$$U = E + \lambda \sum_{j=1}^{Q} w_j^2$$

Bayesian Inference with Neural Networks

A Bayesian would not pick a set of network weights to use in a regression or classification model, but would rather compute the posterior distribution on the network weights

$$p(w|D) = p(D\,|\,w)\ P(w)\,/\,P(D)$$

and perform inference by averaging models over this posterior distribution.

For example, the predictor for a regression problem takes the form

$$O(x|D) = \int O(x;w)\ \ p(w\,|\,D)\ dw$$

Needless to say, this is an ambitious program (multimodal posterior, intractable integrals) requiring Monte Carlo techniques, or extreme approximations.

1789

GEORGETOWN UNIVERSITY