CS 340: Machine Learning
Lecture 17: Neural Networks

AD

March 2010

Limitation of Linear Models

- Until now, we have worked primarily with linear models.
- In the models we have previously discussed, we select beforehand the basis functions.
- If we have too many basis functions (i.e. one for each training point), we tend to overfit.
- Solutions to reduce these problems consists of using priors or different loss functions.
- These methods dominate Machine Learning nowadays.

Neural Networks

- Very network, not very neural after all.
- In this case, we fix the number of basis functions but their parameters are adapted during training.
- These models are (too?) flexible.
- They can perform well but it is difficult to train them and their interpretability is difficult.
- Revival of these approaches over recent years.

Feed-forward network functions

- We have worked with models where for regression

$$y(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\Phi(\mathbf{x}) = \sum_{j=1}^{M} w_j \phi_j(\mathbf{x})$$

and for binary classification

$$\Pr(y = 1|\mathbf{x}) = g\left(\mathbf{w}^{\mathsf{T}}\Phi(\mathbf{x})\right).$$

- For the basic neural network (NN), we build first M linear combinations of $\mathbf{x} = (x_1, ..., x_D)$

$$a_j = \mathbf{w}_j^{(1)\,\mathsf{T}}\mathbf{x} = \sum_{l=1}^{D} \underbrace{w_{jl}^{(1)}}_{\text{weights}} x_l + \underbrace{w_{j0}^{(1)}}_{\text{bias}} \text{ for } j = 1, ..., M$$

- We then apply a nonlinear transformation - activation function $z_j = g(a_j)$. We can use the logistic sigmoid or the hyperbolic tangent. These are called *hidden units*.

- We obtain K output unit activations by setting for regression problems

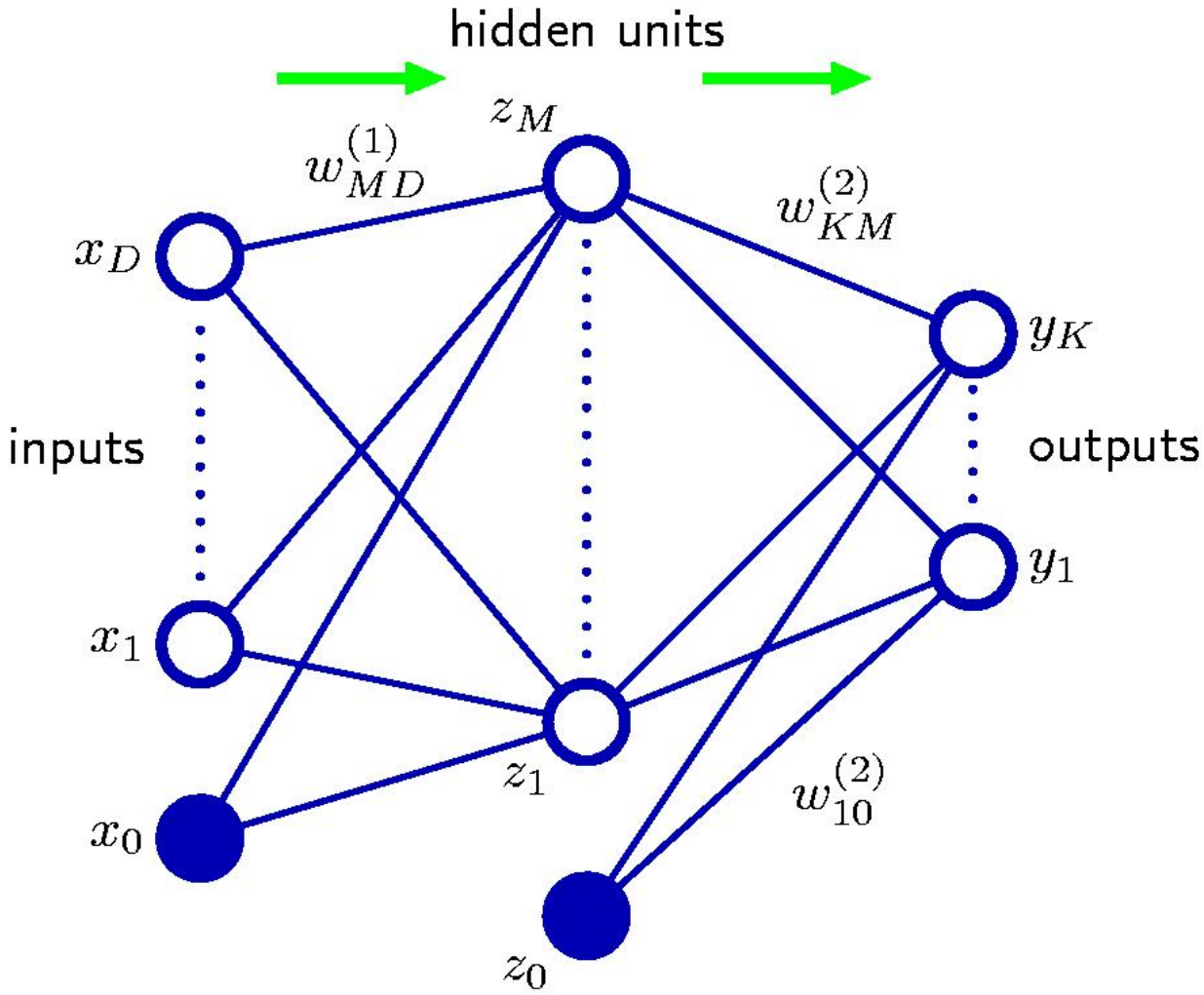
$$\begin{aligned} y_k(\mathbf{x}, \mathbf{w}) &= \mathbf{w}_k^{(2)\,\mathsf{T}} \mathbf{z} = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \text{ for } k = 1, ..., K \\ &= \sum_{j=1}^{M} w_{kj}^{(2)} g\left(\sum_{l=1}^{D} w_{jl}^{(1)} x_l + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \end{aligned}$$

- For classification problems, we have

$$y_k(\mathbf{x}, \mathbf{w}) = g(a_k) = g\left(\sum_{j=1}^{M} w_{kj}^{(2)} g\left(\sum_{l=1}^{D} w_{jl}^{(1)} x_l + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right)$$

with $K = 1$ and $g(\cdot)$ logistic function for binary classification and $K = C - 1$ and softmax link for C classes.

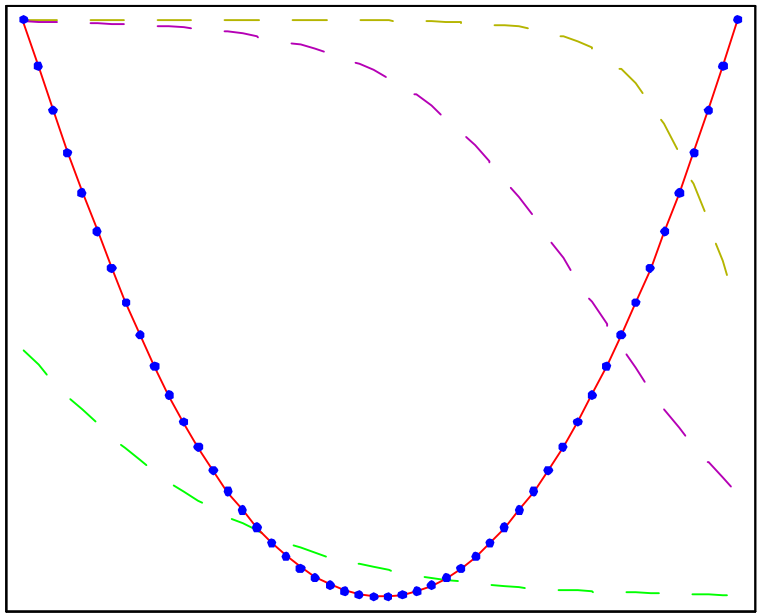
Two-layer NN

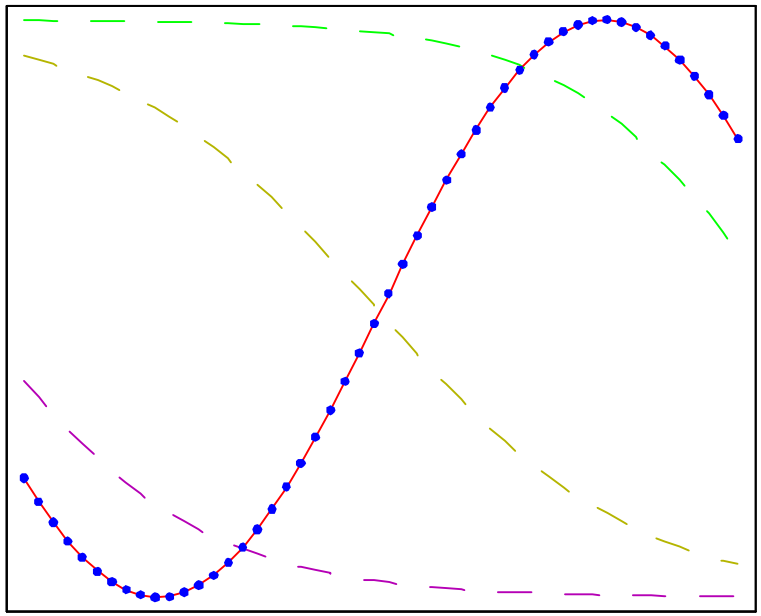

hidden units
z_M
$w^{(1)}_{MD}$
$w^{(2)}_{KM}$
x_D
y_K
inputs
outputs
y_1
x_1
z_1
$w^{(2)}_{10}$
x_0
z_0

- The NN architecture presented here is the most common one.
- We can add layers of hidden units.
- We can have sparse architectures where some of the connections are not included.
- *Theoretical justification*: Many results have established that a two-layer network with linear outputs can approximate any continuous function on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units.

- You should be a bit critical about these properties.
- Essentially it tells you then if your model can be as complex as you want then you can approximate anything.
- However, it is true that NN can perform well in some scenarios.

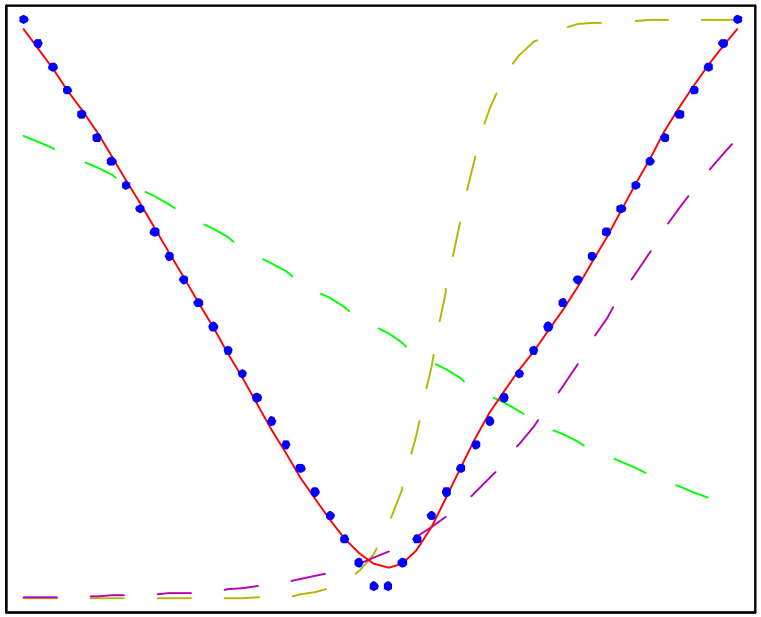
Regression examples

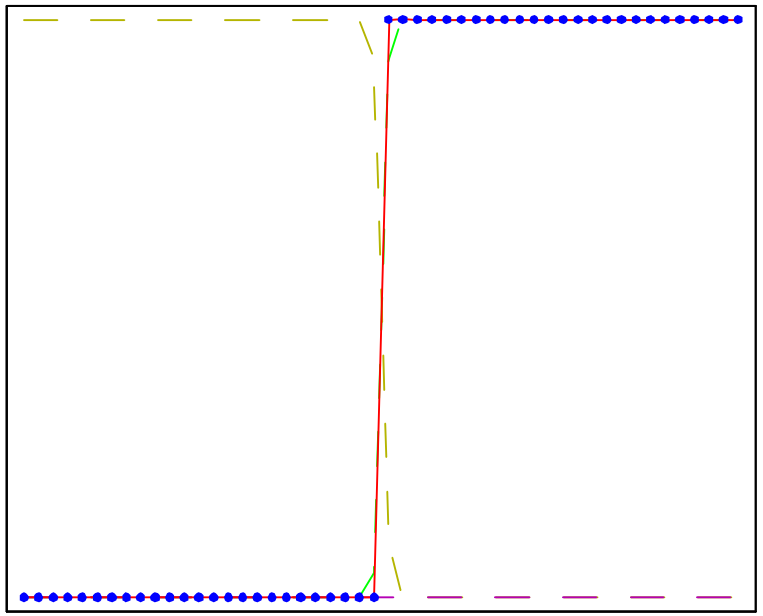




NN trained using 50 data on various functions using 3 hidden units with 'tanh' activation functions and a linear output. Ouput of the hidden units are in dashed lines.

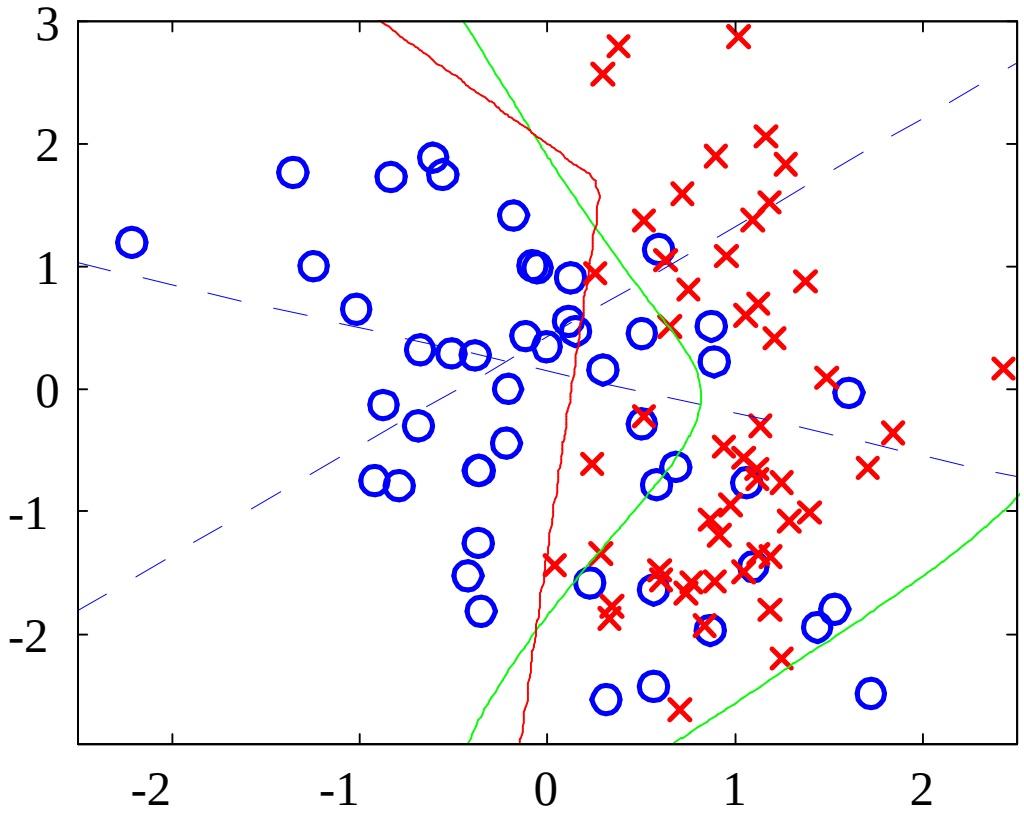
Regression examples





NN trained using 50 data on various functions using 3 hidden units with 'tanh' activation functions and a linear output. Ouput of the hidden units are in dashed lines.

Classification example



Two input, two hidden units with 'tanh' activation and a single output with logististic. Dashed blue lines show $z = 0.5$ for each hidden units, red line is output $y = 0.5$ and green line is the true Bayes classifier.

NN Training

- Assume we are considering a regression problem $\left\{\mathbf{x}^i, y^i\right\}_{i=1}^N$. To learn the parameters in a regression case, we seek to minimize

$$E\left(\mathbf{w}\right) = \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(y_k^i - y_k\left(\mathbf{x}^i, \mathbf{w}\right)\right)^2$$

  which corresponds to maximizing the likelihood for a Gaussian model.
- In the binary logistic regression case, we have

$$E\left(\mathbf{w}\right) = -\sum_{i=1}^{N}\left\{y^i \log\left(y\left(\mathbf{x}^i, \mathbf{w}\right)\right) + \left(1 - y^i\right)\log\left(1 - y\left(\mathbf{x}^i, \mathbf{w}\right)\right)\right\}$$

- In both cases, these functions are not convex and it is difficult to minimize $E(\mathbf{w})$.
- We can use a gradient descent method

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \delta \left. \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}^{(t)}}$$

- We can also use Newton-Raphson

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \left[\left. \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^{\mathsf{T}}} \right|_{\mathbf{w}^{(t)}} \right]^{-1} \left. \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}^{(t)}}$$

  which provides usually algorithms converging faster.
- We can also cycle over the observations using

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \delta \left. \frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}^{(t)}}$$

  where $E_i(\mathbf{w})$ corresponds to observation i.

A Regression Example

- Consider the case where

$$a_j = \sum_{l=0}^{D} w_{jl}^{(1)} x_l, \qquad z_j = \tanh\left(a_j\right) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}},$$

$$y_k\left(\mathbf{x}, \mathbf{w}\right) = \sum_{j=0}^{M} w_{kj}^{(2)} z_j.$$

- We have

$$E_i\left(\mathbf{w}\right) = \frac{1}{2} \sum_{k=1}^{K} \left(y_k^i - y_k\left(\mathbf{x}^i, \mathbf{w}\right)\right)^2.$$

- We want to compute

$$\frac{\partial E_i\left(\mathbf{w}\right)}{\partial w_{jl}^{(1)}} \text{ and } \frac{\partial E_i\left(\mathbf{w}\right)}{\partial w_{kj}^{(2)}}$$

Backpropagation algorithm

- We have

$$\begin{aligned}\frac{\partial E_i(\mathbf{w})}{\partial w_{kj}^{(2)}} &= \frac{\partial E_i(\mathbf{w})}{\partial y_k(\mathbf{x}^i, \mathbf{w})} \frac{\partial y_k(\mathbf{x}^i, \mathbf{w})}{\partial w_{kj}^{(2)}} \\ &= \left(y_k(\mathbf{x}^i, \mathbf{w}) - y_k^i\right) z_j\end{aligned}$$

- $E_i(\mathbf{w})$ only depends on $w_{jl}^{(1)}$ via the summed input z_j so

$$\frac{\partial E_i(\mathbf{w})}{\partial w_{jl}^{(1)}} = \frac{\partial E_i(\mathbf{w})}{\partial z_j} \frac{\partial z_j}{\partial w_{jl}^{(1)}}$$

where

$$\frac{\partial z_j}{\partial w_{jl}^{(1)}} = x_l^i \left(1 - z_j^2\right)$$

as $\left[\tanh(x)\right]' = 1 - \tanh(x)^2$.

Backpropagation algorithm

- We have

$$\begin{aligned}\frac{\partial E_i(\mathbf{w})}{\partial z_j} &= \sum_{k=1}^{K} \frac{\partial E_i(\mathbf{w})}{\partial y_k(\mathbf{x}^i, \mathbf{w})} \frac{\partial y_k(\mathbf{x}^i, \mathbf{w})}{\partial z_j} \\ &= \sum_{k=1}^{K} \left(y_k(\mathbf{x}^i, \mathbf{w}) - y_k^i\right) w_{kj}^{(2)}\end{aligned}$$

- So putting all the terms together we have

$$\frac{\partial E_i(\mathbf{w})}{\partial w_{jl}^{(1)}} = x_l^i \left(1 - z_j^2\right) \sum_{k=1}^{K} \left(y_k(\mathbf{x}^i, \mathbf{w}) - y_k^i\right) w_{kj}^{(2)}$$

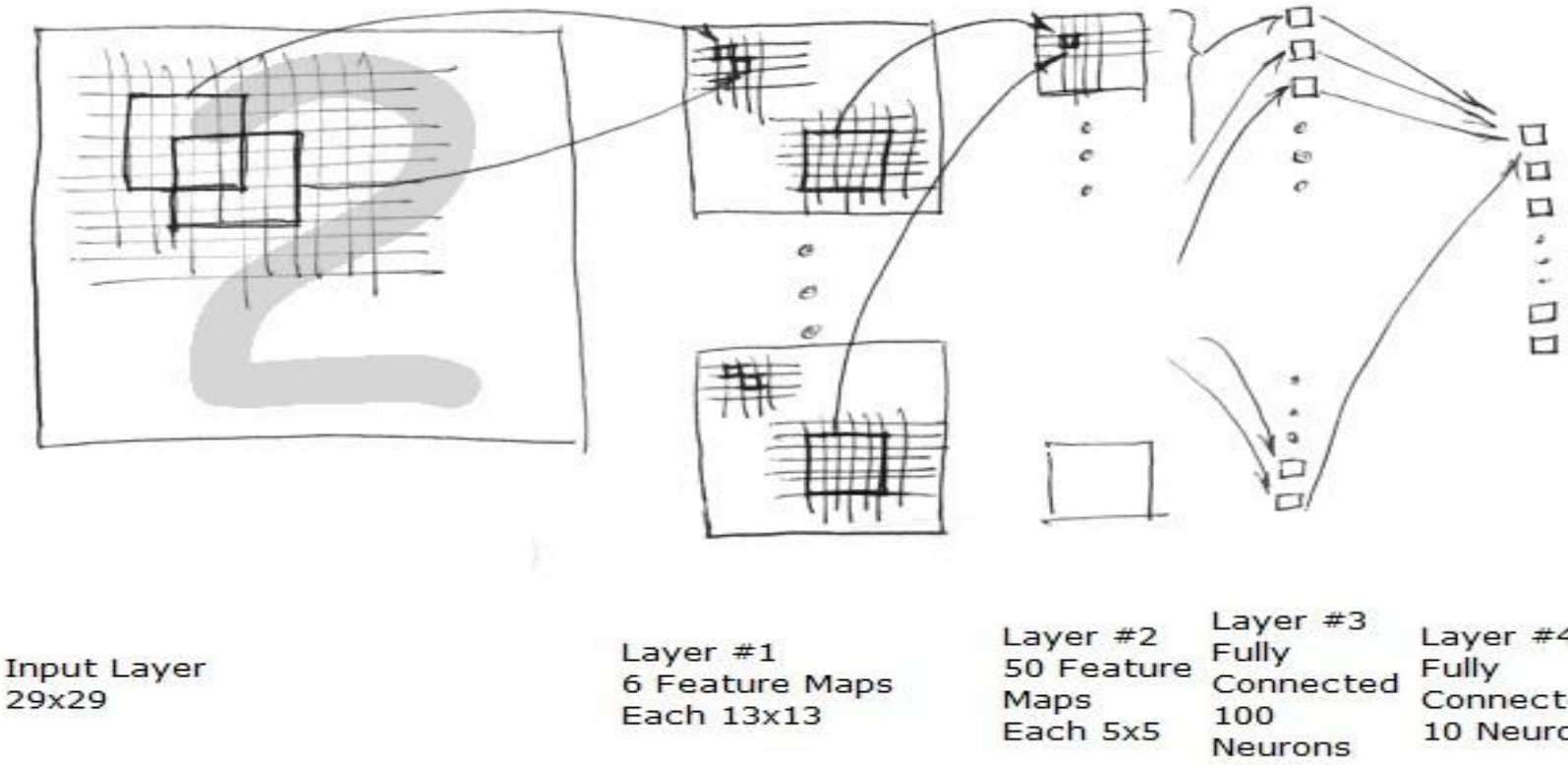
Summary

- Apply $\mathbf{x}^i$ to the network and propagate forward through $a_j = \sum_{l=0}^{D} w_{jl}^{(1)} x_l^i$, $z_j = h(a_j)$
- Evaluate $\varepsilon_k = y_k\left(\mathbf{x}^i, \mathbf{w}\right) - y_k^i$ for the ouput units and compute $\frac{\partial E_i(\mathbf{w})}{\partial w_{kj}^{(2)}} = \varepsilon_k z_j$.
- "Backpropagate" the ε's to compute

$$\frac{\partial E_i(\mathbf{w})}{\partial w_{jl}^{(1)}} = x_l^i \left(1 - z_j^2\right) \sum_{k=1}^{K} \varepsilon_k w_{kj}^{(2)}.$$

- Perform a gradient descent step $\mathbf{w} \leftarrow \mathbf{w} - \delta \left.\frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}}$.

Application to Digit Recognition



- Over 100,000 parameters trained using backpropagation, 1.40% test error on MNIST database.

Bayesian Neural Networks

- In practice, it can help to regularize the solution using a Gaussian prior

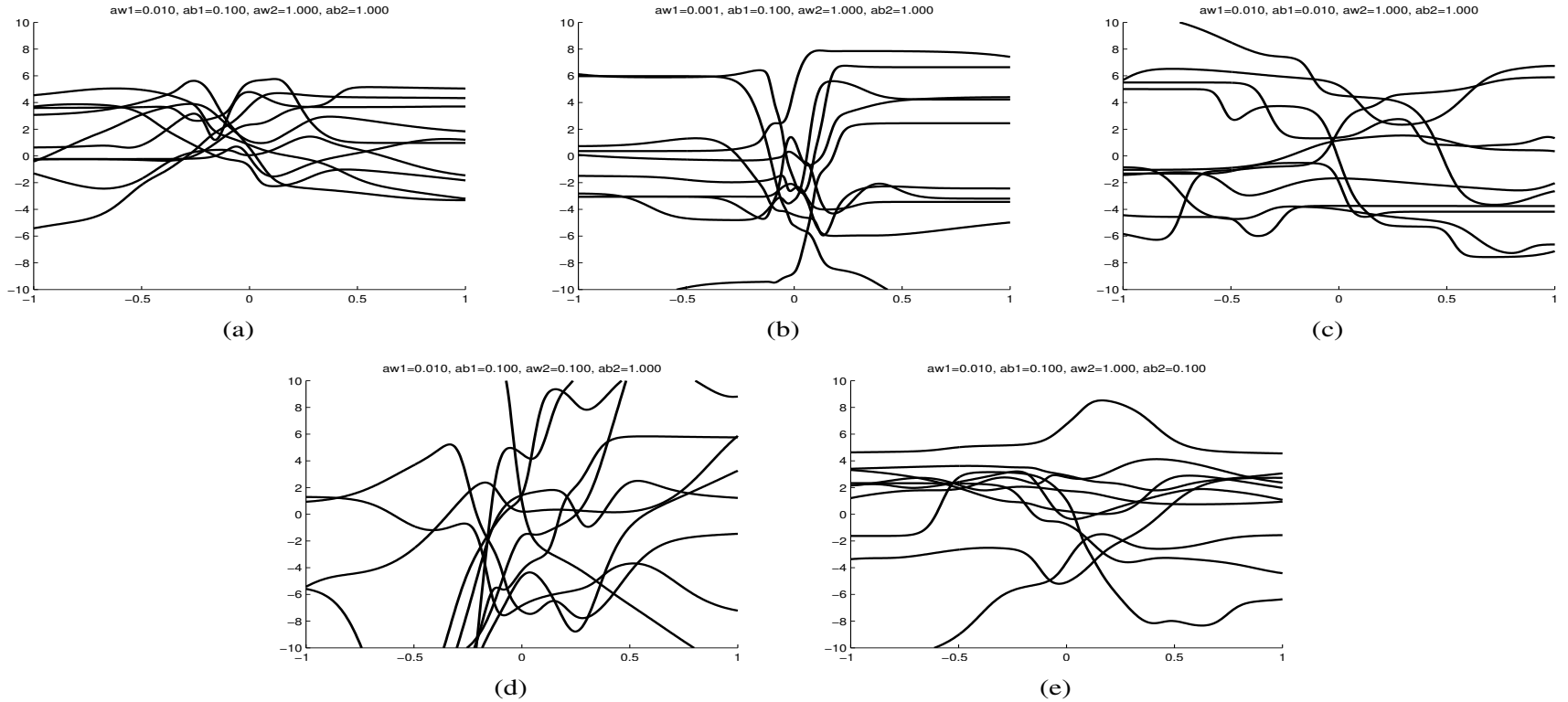
$$\begin{aligned} p\left(\mathbf{w}\middle|\,\alpha\right) &= \prod_{jl} p\left(w_{jl}^{(1)}\middle|\,\alpha\right) \prod_{kj} p\left(w_{kj}^{(2)}\middle|\,\alpha\right) \\ &= \prod_{jl} \mathcal{N}\left(w_{jl}^{(1)}; 0, \alpha^{-1}\right) \prod_{kj} \mathcal{N}\left(w_{kj}^{(2)}; 0, \alpha^{-1}\right) \end{aligned}$$

  but this is inefficient as $\left\{w_{jl}^{(1)}\right\}$ and $\left\{w_{kj}^{(2)}\right\}$ play different roles.
- It is much more efficient to have a layer specific regularization

$$\begin{aligned} p\left(\mathbf{w}\middle|\,\alpha_1, \alpha_2\right) &= \prod_{jl} p\left(w_{jl}^{(1)}\middle|\,\alpha_1\right) \prod_{kj} p\left(w_{kj}^{(2)}\middle|\,\alpha_2\right) \\ &= \prod_{jl} \mathcal{N}\left(w_{jl}^{(1)}; 0, \alpha_1^{-1}\right) \prod_{kj} \mathcal{N}\left(w_{kj}^{(2)}; 0, \alpha_2^{-1}\right). \end{aligned}$$

- In practice, we also use specific very vague priors for the bias (as in ridge regression).

Bayesian Neural Networks



Samples from the regression function $y(\mathbf{x}, \mathbf{w})$ for various values of the prior parameters.

Bayesian Neural Networks for Regression

- Assume that $K = 1$ and

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

Additionally, for sake of simplicity we set $\alpha_1 = \alpha_2$ so that

$$p\left(\mathbf{w}|\,\alpha\right) = \prod_{jl}\mathcal{N}\left(w_{jl}^{(1)}; 0, \alpha^{-1}\right)\prod_{kj}\mathcal{N}\left(w_{kj}^{(2)}; 0, \alpha^{-1}\right)$$

- For data $D = \left\{\mathbf{x}^i, y^i\right\}_{n=1}^N$, we are interested in the posterior

$$p\left(\mathbf{w}|\,D,\alpha,\beta\right) = \frac{p\left(\left\{y^i\right\}_{n=1}^N \middle|\, \left\{\mathbf{x}^i\right\}_{n=1}^N, \mathbf{w},\beta\right) p\left(\mathbf{w}|\,\alpha\right)}{p\left(\left\{y^i\right\}_{n=1}^N \middle|\, \left\{\mathbf{x}^i\right\}_{n=1}^N, \alpha, \beta\right)} \propto \exp\left(-E\left(\mathbf{w}\right)\right)$$

where

$$E\left(\mathbf{w}\right) = \frac{\beta}{2}\sum_{i=1}^N\left(y^i - y\left(\mathbf{x}^i,\mathbf{w}\right)\right)^2 + \frac{\alpha}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}$$

- Because the likelihood is highly non-linear in $\mathbf{w}$, there is no closed-form solution for the posterior.

Laplace approximation of the Posterior and Predictive

- Assuming we have found the MAP estimate $\mathbf{w}_{\text{MAP}}$ then the Laplace approximation approximates the posterior by a multivariate Gaussian distribution (more next week!) centered around the MAP

$$p\left(\mathbf{w}|\, D, \alpha, \beta\right) \approx q\left(\mathbf{w}|\, D, \alpha, \beta\right) = \mathcal{N}\left(\mathbf{w}; \mathbf{w}_{\text{MAP}}, A\right)$$

where

$$A = -\frac{\partial^2 \log p\left(\mathbf{w}|\, D, \alpha, \beta\right)}{\partial \mathbf{w} \partial \mathbf{w}^{\mathsf{T}}} = \alpha I + \beta H$$

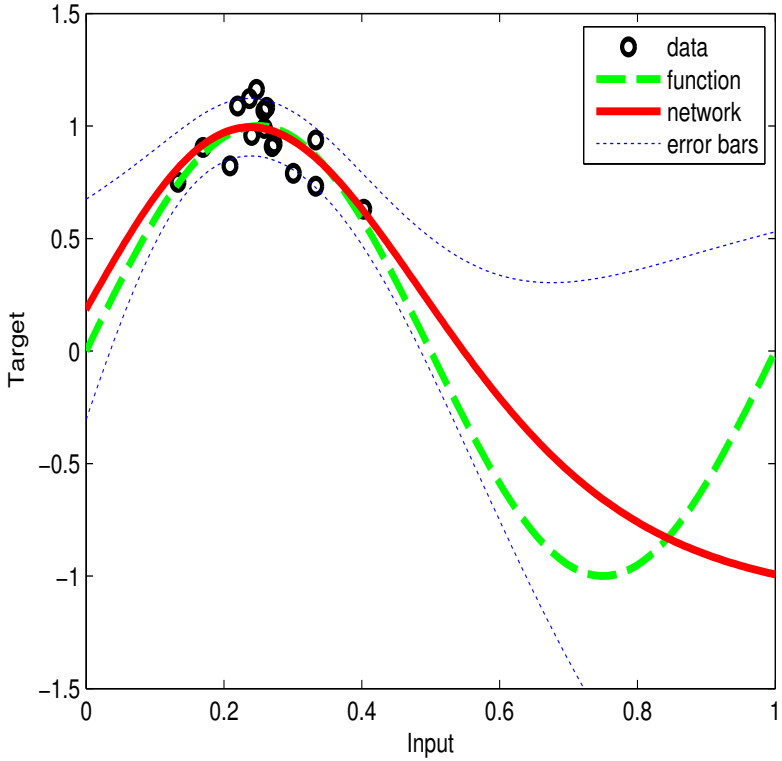
with H the Hessian of the sum of squared prediction errors.

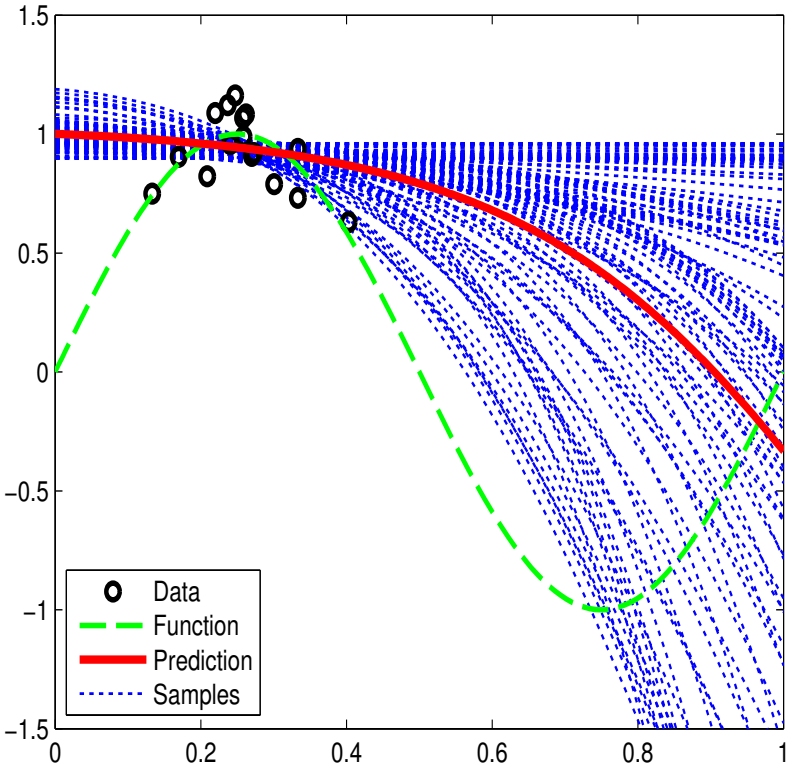
- We have

$$\begin{aligned} p\left(y|\, D, \alpha, \beta, \mathbf{x}\right) &= \int p\left(y|\, \mathbf{x}, \mathbf{w}\right) p\left(\mathbf{w}|\, D, \alpha, \beta\right) d\mathbf{w} \\ &\approx \int p\left(y|\, \mathbf{x}, \mathbf{w}\right) q\left(\mathbf{w}|\, D, \alpha, \beta\right) d\mathbf{w} \\ &\approx \mathcal{N}\left(y; y\left(\mathbf{x}, \mathbf{w}_{\text{MAP}}\right), \sigma^2\left(\mathbf{x}\right)\right) \end{aligned}$$

where $\sigma^2\left(\mathbf{x}\right) = \beta^{-1} + \left.\frac{\partial y(\mathbf{x},\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_{\text{MAP}}}^{-1} A \left.\frac{\partial y(\mathbf{x},\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}_{\text{MAP}}}$.

Example





True function (green), $y(\mathbf{x}, \mathbf{w}_{\text{MAP}})$ (red) and $y(\mathbf{x}, \mathbf{w}_{\text{MAP}}) \mp \sigma(\mathbf{x})$ (blue) for an MLP with 3 hidden nodes, trained on 16 data points. (a) Laplace approximation, after performing empirical Bayes to optimize (α, β). (b) Samples from $y(\mathbf{x}, \mathbf{w})$ where $\mathbf{w} \sim p(\mathbf{w}|D, \alpha, \beta)$ obtained using hybrid Monte Carlo, using the same (α, β) as in (a).