

Stats 225: Bayesian Analysis

Neural Networks, Deep Learning, and Gaussian Processes

Babak Shahbaba
UC Irvine

Generalized Linear Models (GLMs)

$$\mathbf{x}_i \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

Link Function


$$g^{-1}(\mathbf{x}_i^T \mathbf{w} + b) = \mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

Generalized Linear Models (GLMs)

$$\mathbf{x}_i \in \mathbb{R}^d, \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

Link Function


$$g^{-1}(\mathbf{x}_i^T \mathbf{w} + b) = \mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

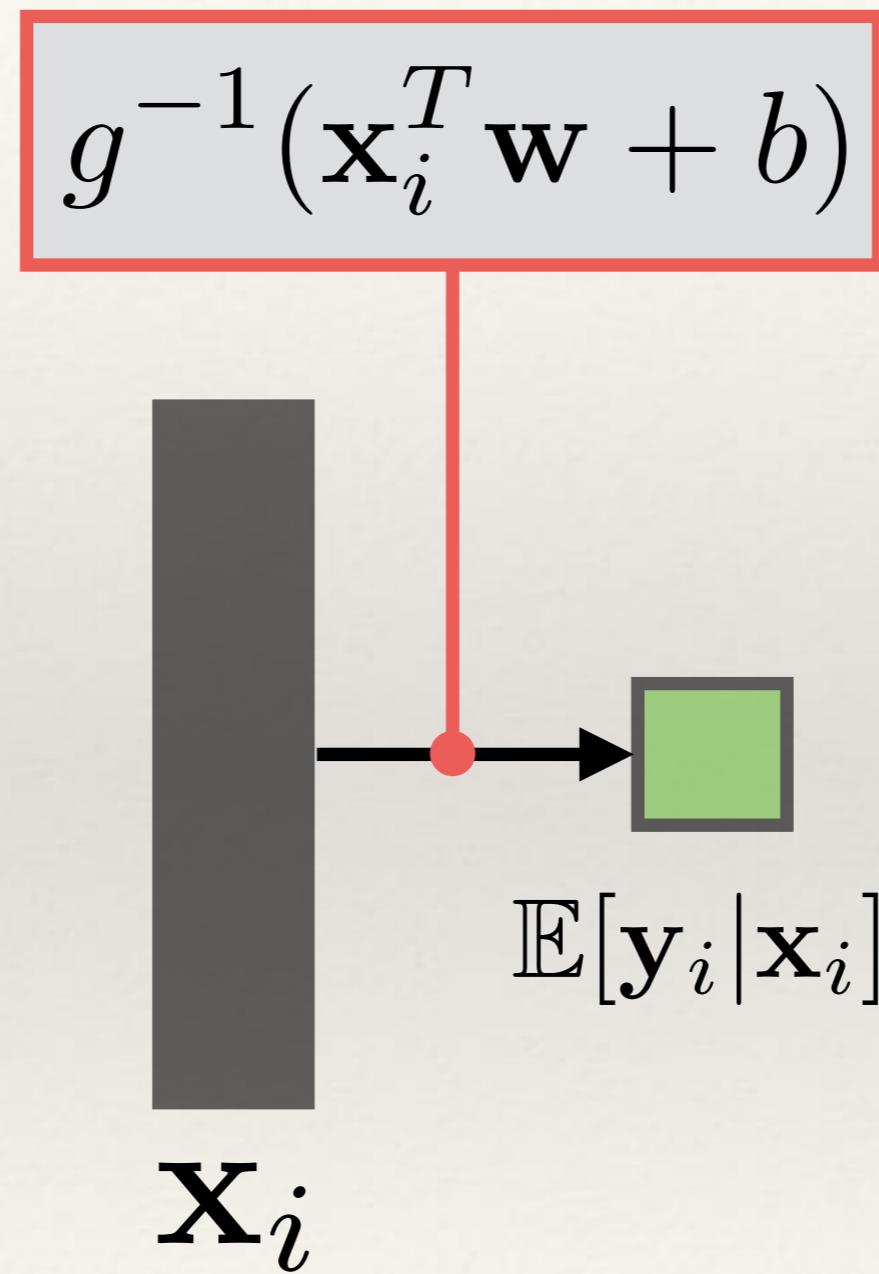
Example: Logistic Regression

$$\mathbf{y}_i \in \{0, 1\}$$

$$g^{-1}(z) = \frac{1}{1 + e^{-z}}$$

THE LOGISTIC FUNCTION

Generalized Linear Models (GLMs)



Recursive GLMs

Define a latent feature via a GLM:

$$g_1^{-1}(\mathbf{x}_i^T \mathbf{w}_1 + b_1) = h_i$$

Recursive GLMs

Define a latent feature via a GLM:

$$g_1^{-1}(\mathbf{x}_i^T \mathbf{w}_1 + b_1) = h_i$$

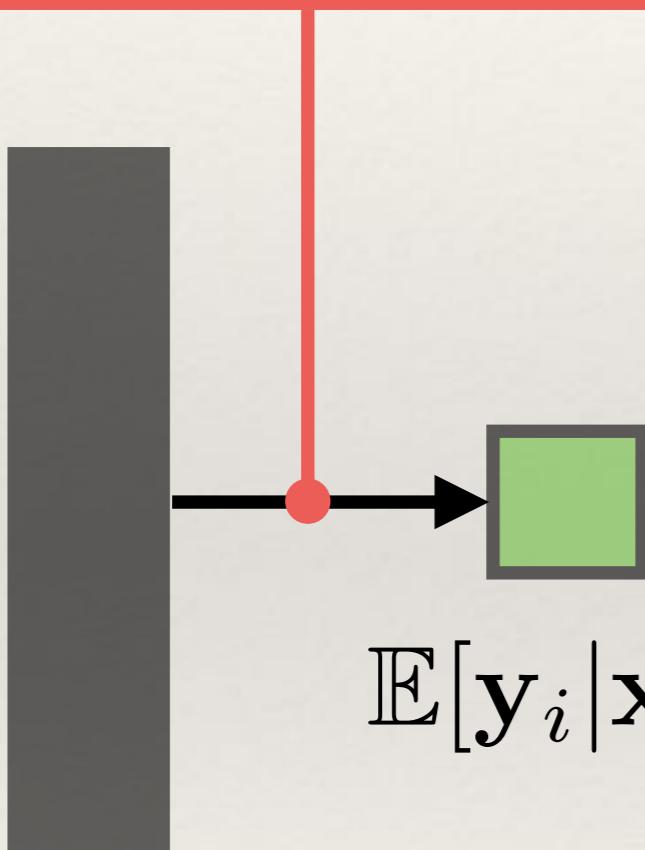
Define GLM on latent feature:

$$g_2^{-1}(h_i w_2 + b_2) = \mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

$$w_2 \in \mathbb{R}, b_2 \in \mathbb{R}$$

Recursive GLMs

$$g^{-1}(\mathbf{x}_i^T \mathbf{w} + b)$$



\mathbf{x}_i

Recursive GLMs

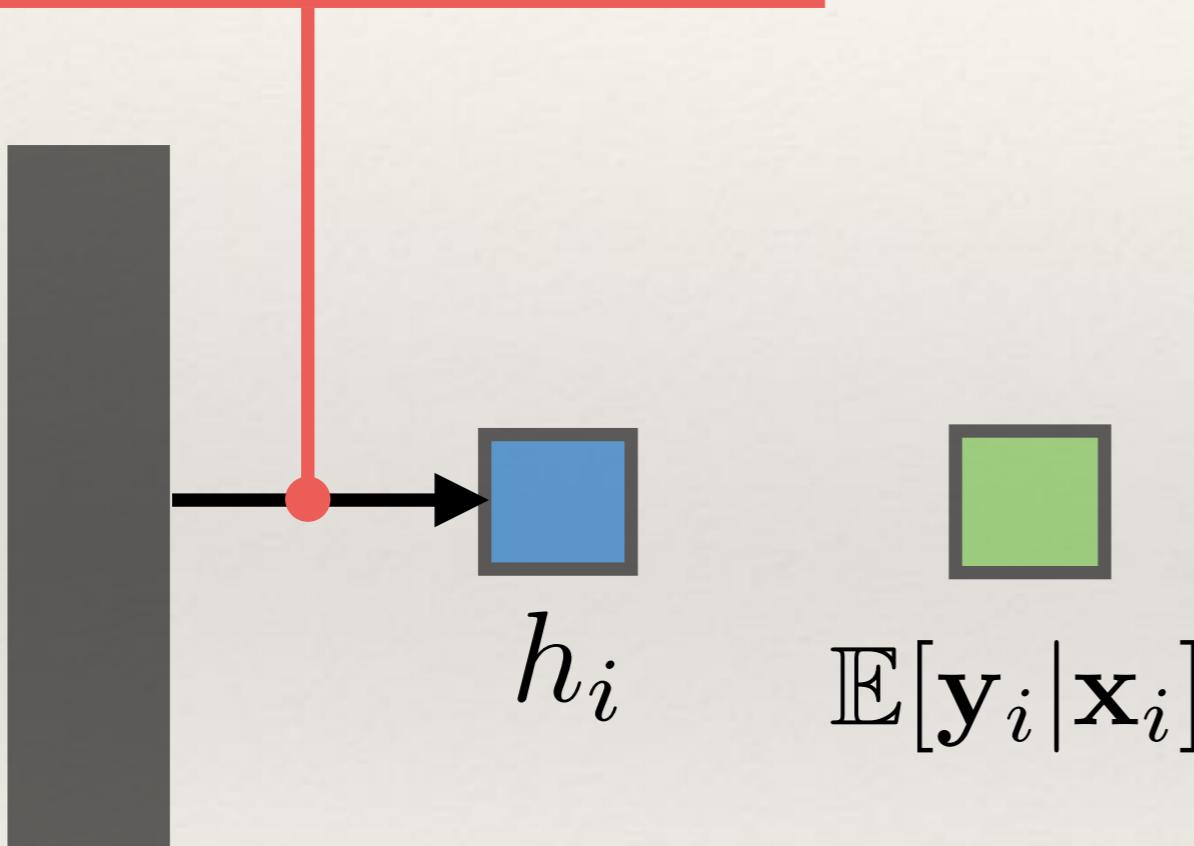


$$\mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

x_i

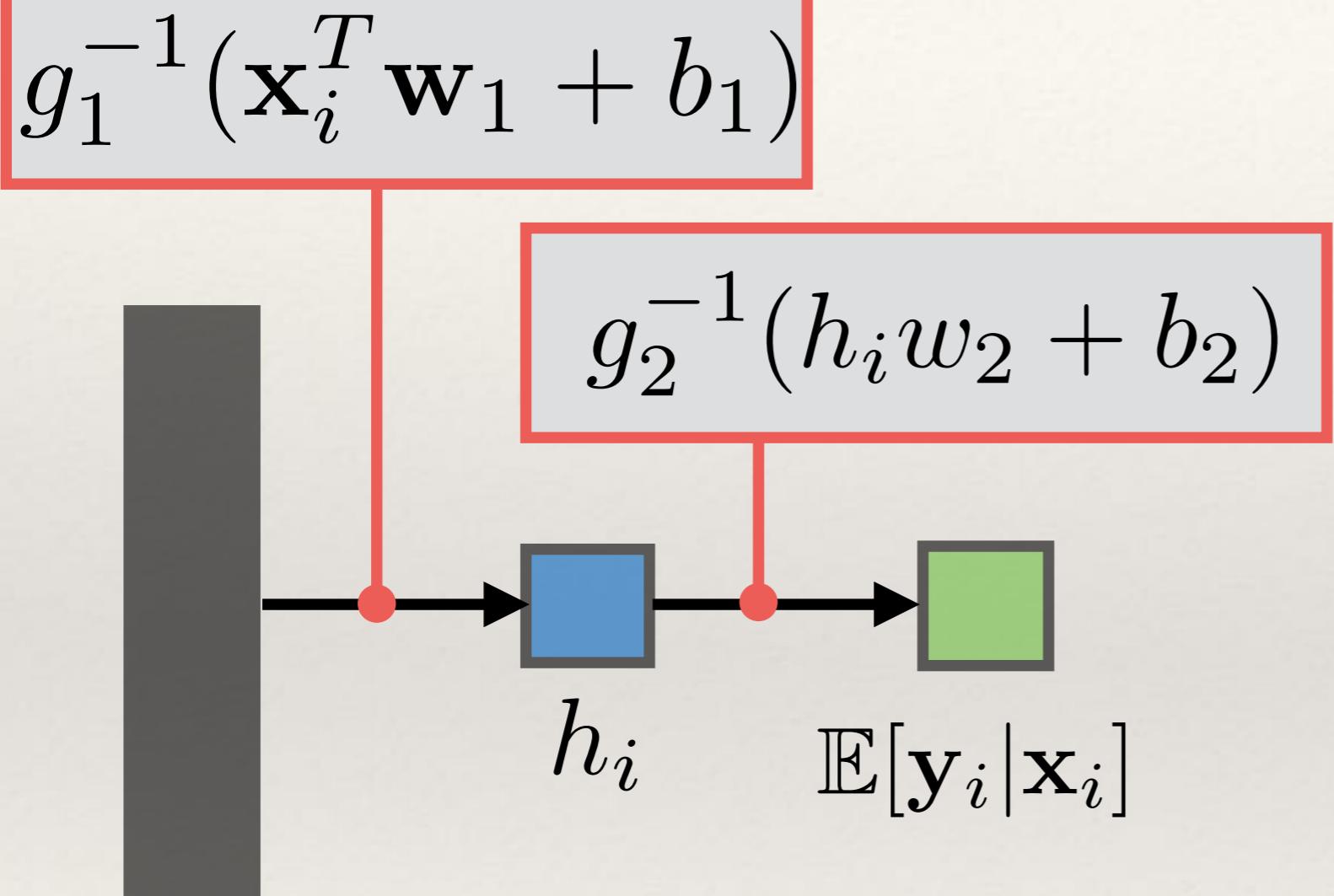
Recursive GLMs

$$g_1^{-1}(\mathbf{x}_i^T \mathbf{w}_1 + b_1)$$



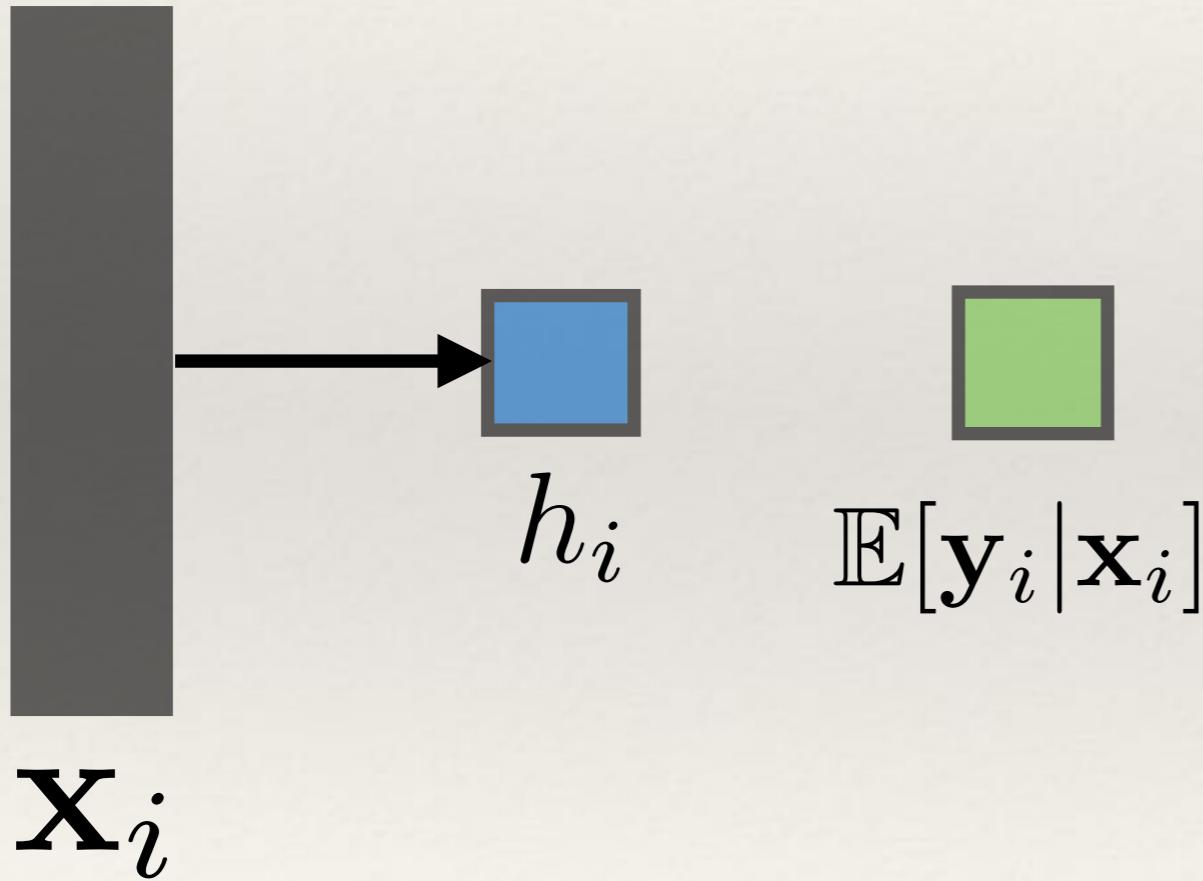
\mathbf{x}_i

Recursive GLMs

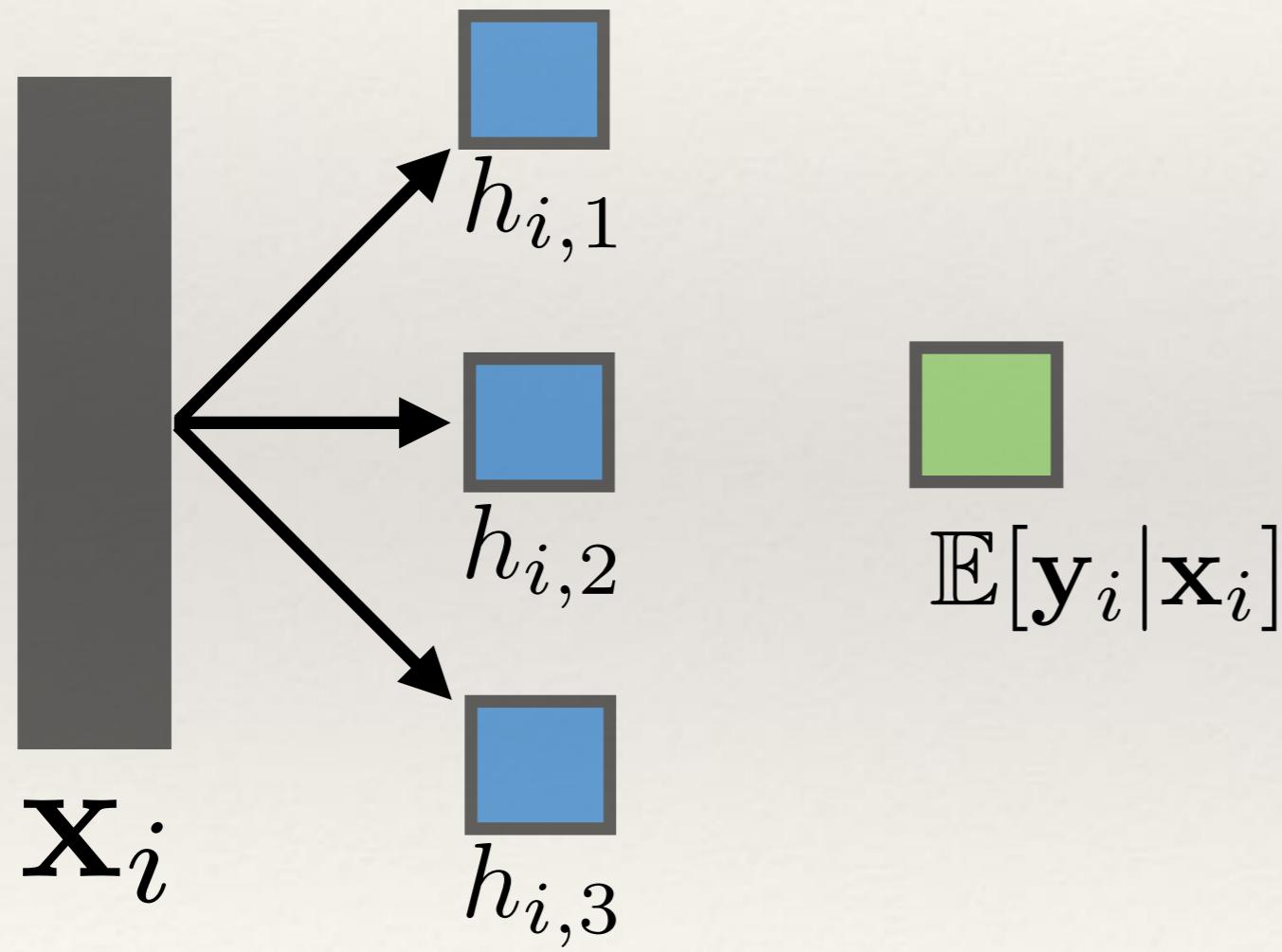


\mathbf{x}_i

Building Neural Nets from Recursive GLMs

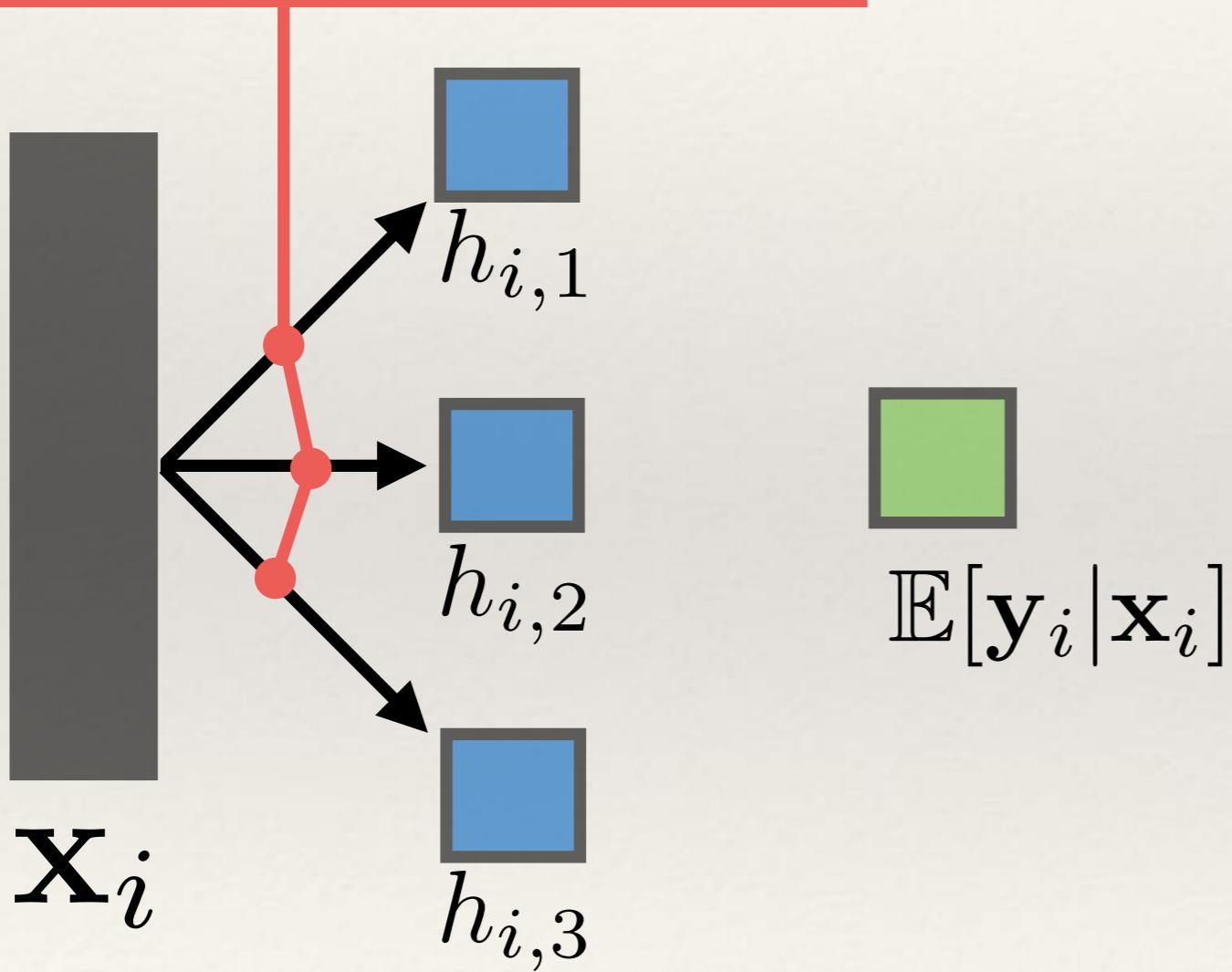


Building Neural Nets from Recursive GLMs



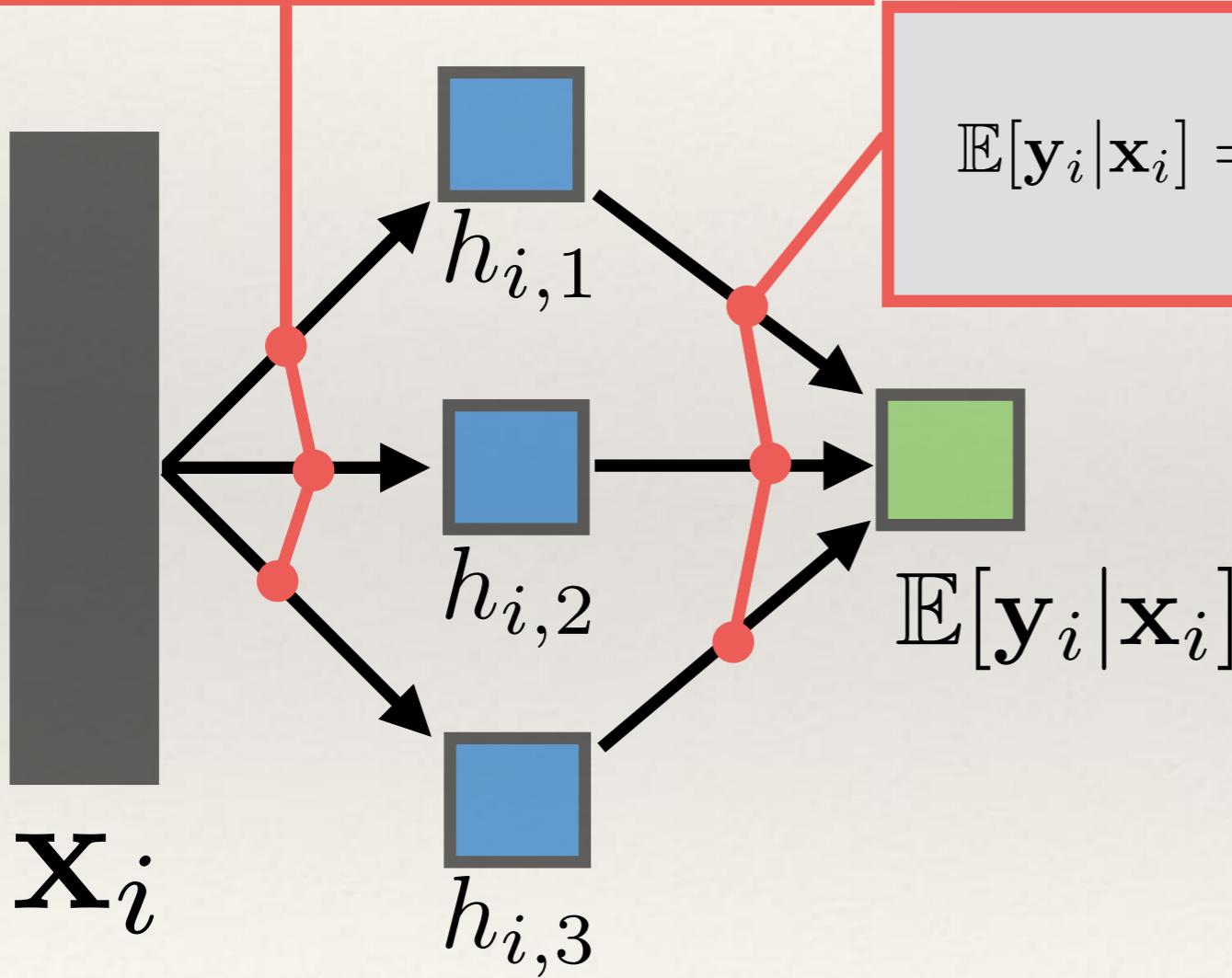
Building Neural Nets from Recursive GLMs

$$h_{i,j} = g_{1,j}^{-1}(\mathbf{x}_i^T \mathbf{w}_{1,j} + b_j)$$



Building Neural Nets from Recursive GLMs

$$h_{i,j} = g_{1,j}^{-1}(\mathbf{x}_i^T \mathbf{w}_{1,j} + b_j)$$

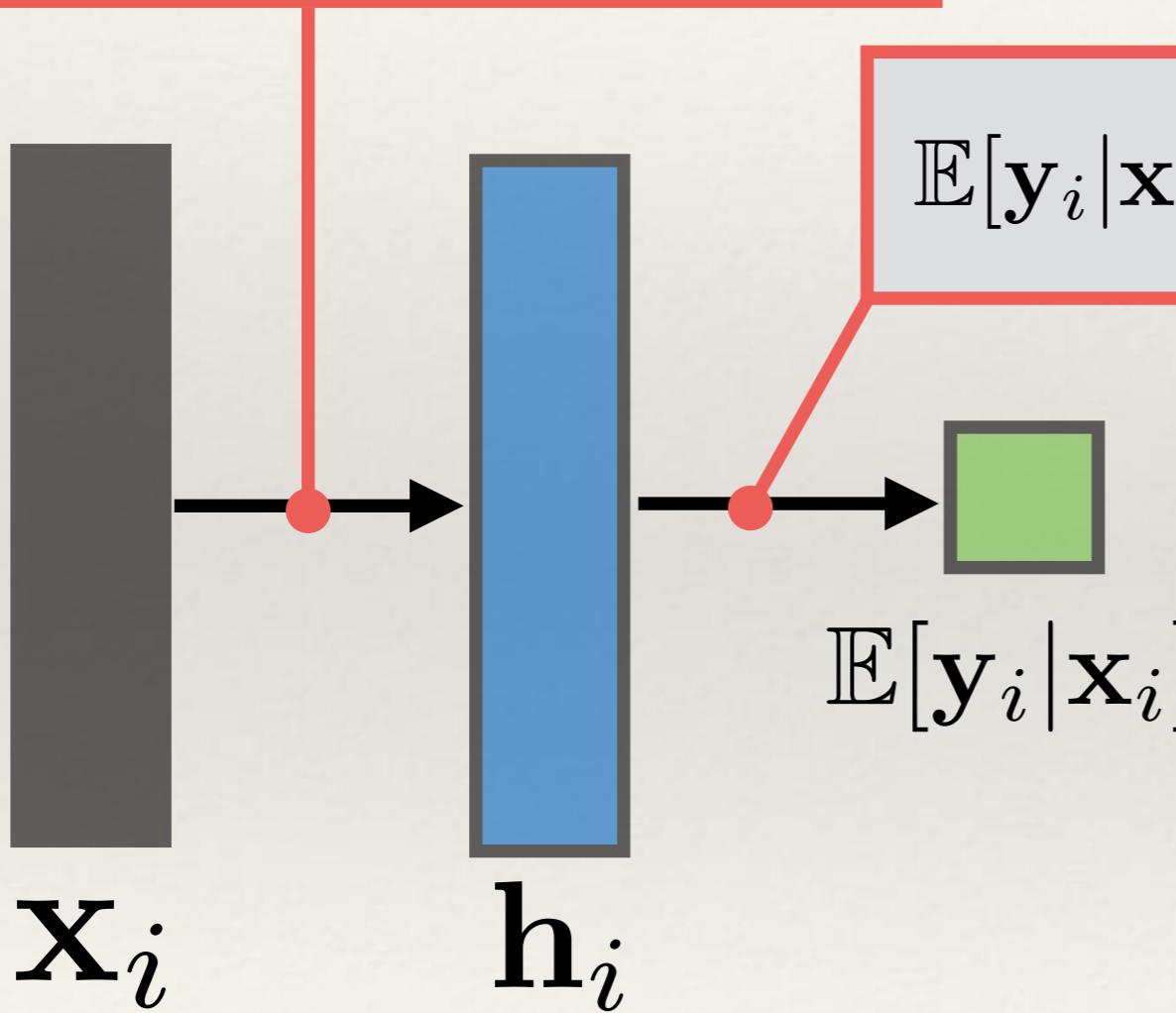


$$\mathbb{E}[\mathbf{y}_i | \mathbf{x}_i] = g_2^{-1} \left(\sum_{j=1}^H h_{i,j} w_{2,j} + b_2 \right)$$

Building Neural Nets from Recursive GLMs

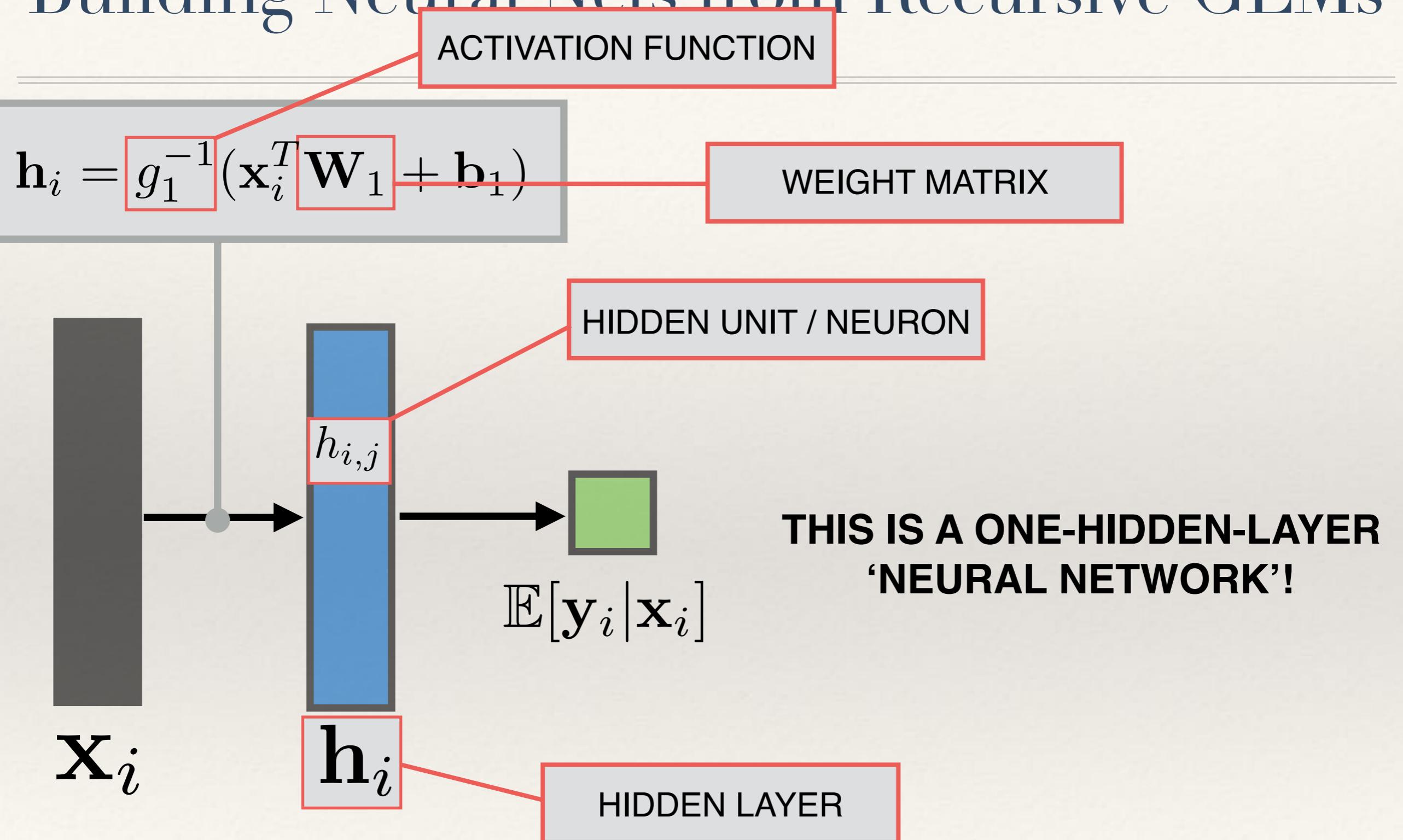
$$\mathbf{h}_i = g_1^{-1}(\mathbf{x}_i^T \mathbf{W}_1 + \mathbf{b}_1)$$

$$\mathbf{h}_i \in \mathbb{R}^H, \mathbf{W}_1 \in \mathbb{R}^{d \times H}, \mathbf{b}_1 \in \mathbb{R}^H$$

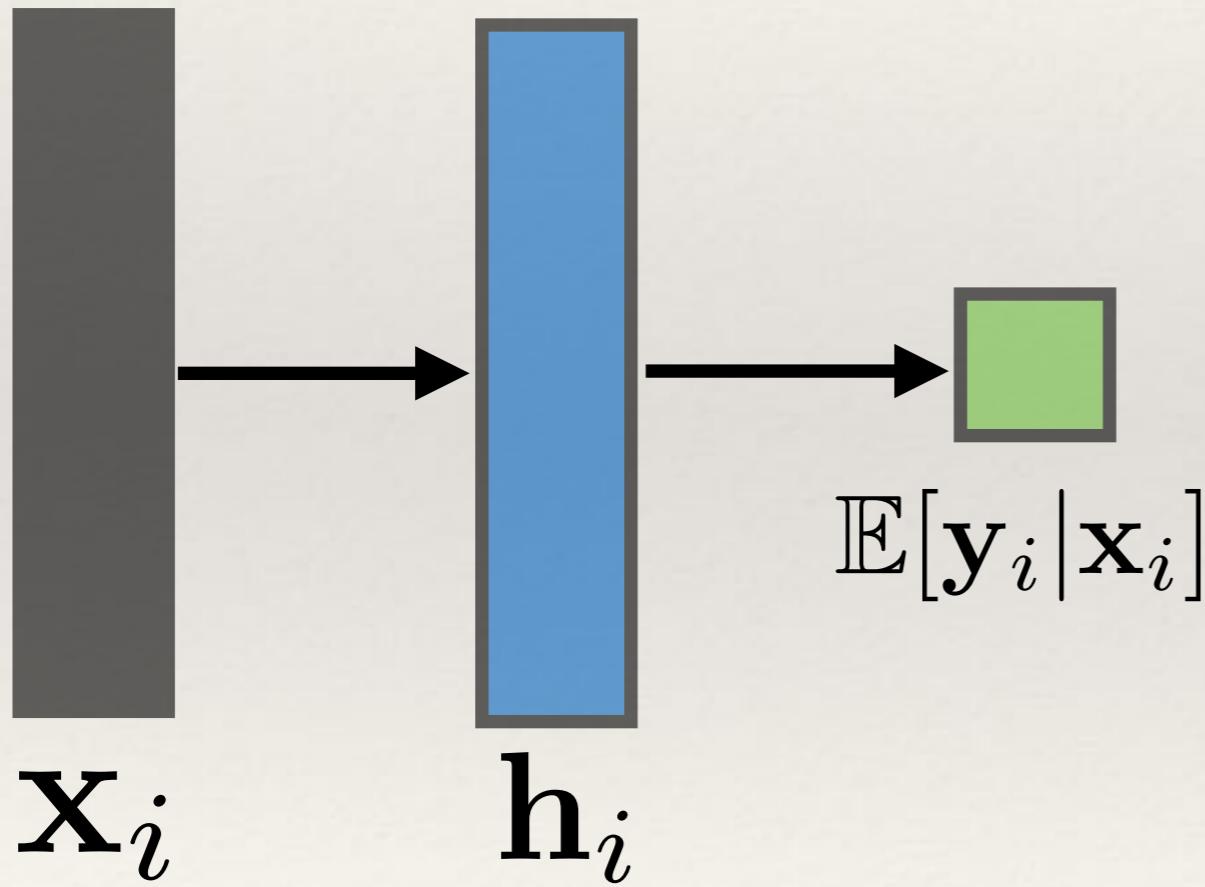


**THIS IS A ONE-HIDDEN-LAYER
'NEURAL NETWORK'!**

Building Neural Nets from Recursive GLMs



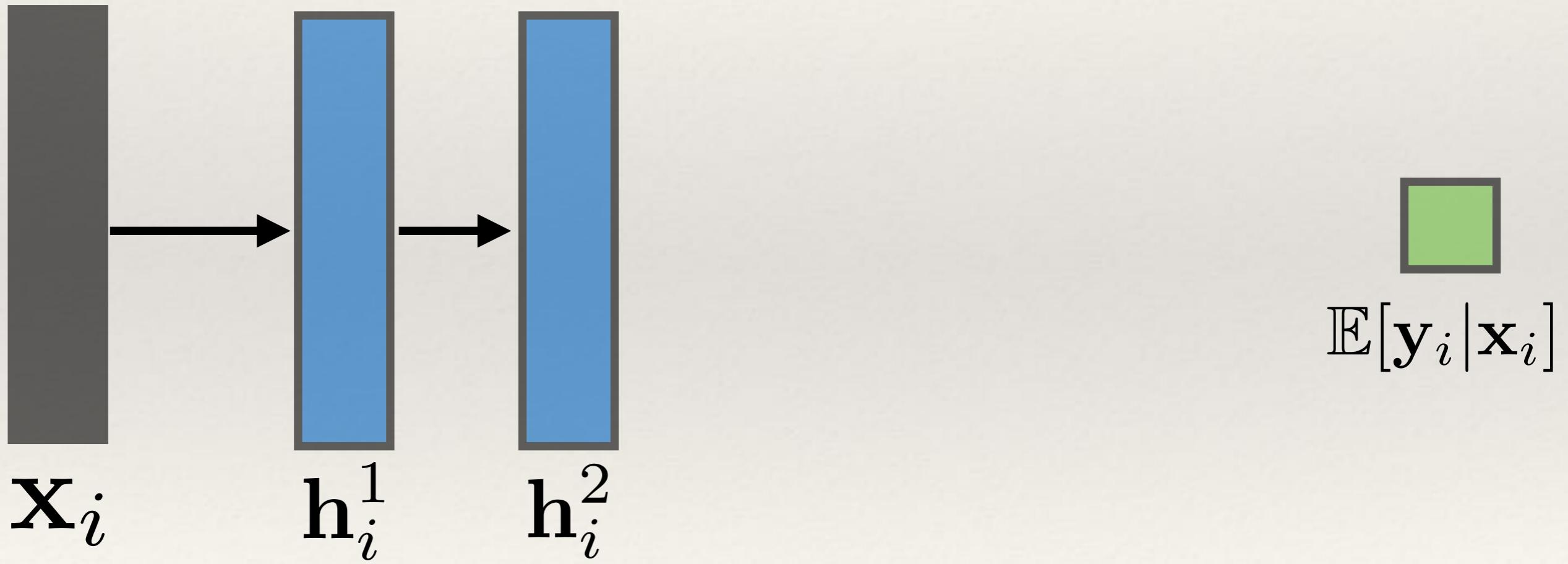
Deep Neural Networks



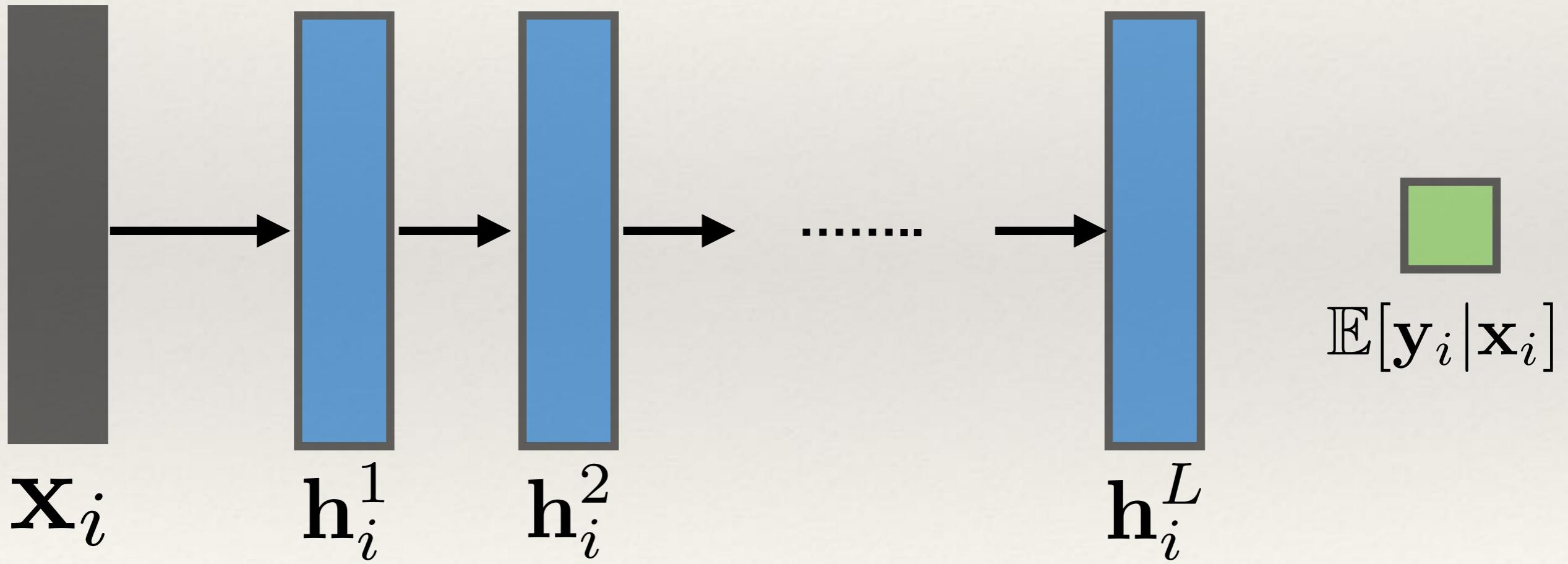
Deep Neural Networks



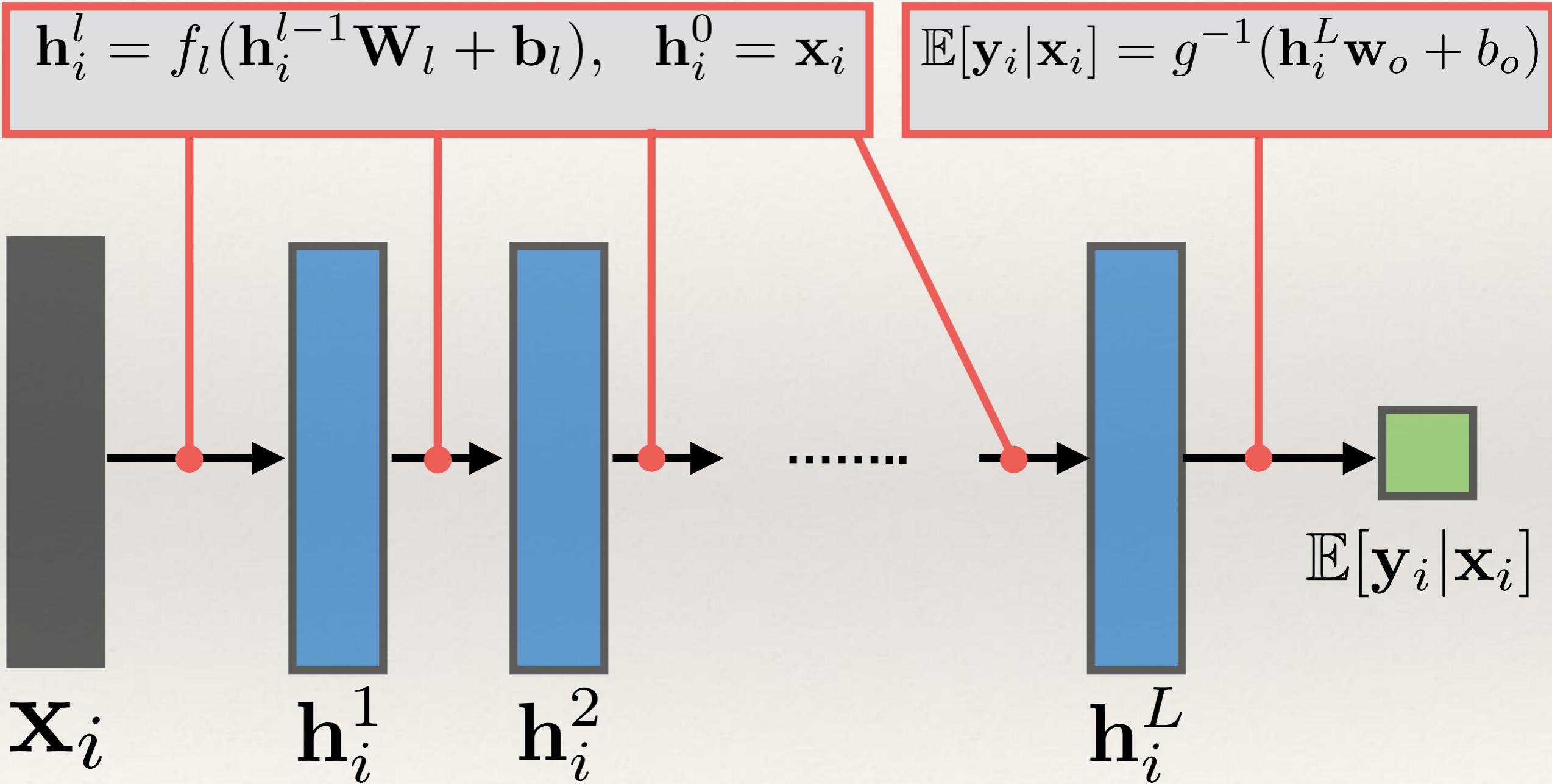
Deep Neural Networks



Deep Neural Networks



Deep Neural Networks



Deep Neural Networks

Parameter Estimation: All parameters are estimated via gradient ascent on the log likelihood.

LOG LIKELIHOOD

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1}) = \sum_{i=1}^N \log p(\mathbf{y}_i | \mathbf{x}_i, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

Deep Neural Networks

Parameter Estimation: All parameters are estimated via gradient ascent on the log likelihood.

LOG LIKELIHOOD

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1}) = \sum_{i=1}^N \log p(\mathbf{y}_i | \mathbf{x}_i, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

PARAMETER UPDATE

$$\mathbf{W}_l^{t+1} = \mathbf{W}_l^t + \alpha \nabla_{\mathbf{W}_l} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

Deep Neural Networks

Parameter Estimation: All parameters are estimated via gradient ascent on the log likelihood.

LOG LIKELIHOOD

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1}) = \sum_{i=1}^N \log p(\mathbf{y}_i | \mathbf{x}_i, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

PARAMETER UPDATE

$$\mathbf{W}_l^{t+1} = \mathbf{W}_l^t + \boxed{\alpha} \nabla_{\mathbf{W}_l} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

learning rate / step size.

Deep Neural Networks

Parameter Estimation: All parameters are estimated via gradient ascent on the log likelihood.

LOG LIKELIHOOD

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1}) = \sum_{i=1}^N \log p(\mathbf{y}_i | \mathbf{x}_i, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

PARAMETER UPDATE

$$\mathbf{W}_l^{t+1} = \mathbf{W}_l^t + \alpha \boxed{\nabla_{\mathbf{W}_l} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})}$$

learning rate / step size.

Usually estimated with a subsample (i.e. ‘stochastic gradient’)

$$\nabla_{\mathbf{W}_l} \mathcal{L}(\mathbf{X}, \mathbf{Y}, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1}) \approx \nabla_{\mathbf{W}_l} \sum_{k=1}^K \log p(\mathbf{y}_k | \mathbf{x}_k, \{\mathbf{W}_l\}_1^{L+1}, \{\mathbf{b}_l\}_1^{L+1})$$

for $K \ll N$

Deep Neural Networks

- **Adaptive Basis Function Regression:** Neural networks are just performing regression with adaptive basis functions.

$$\mathbb{E}[\mathbf{y}_i | \mathbf{x}_i] = g^{-1}(\mathbf{h}_i^L \mathbf{w}_o + b_o) = g^{-1}(\mathbf{h}(\mathbf{x}_i) \mathbf{w}_o + b_o)$$

Deep Neural Networks

- **Adaptive Basis Function Regression:** Neural networks are just performing regression with adaptive basis functions.

$$\mathbb{E}[\mathbf{y}_i | \mathbf{x}_i] = g^{-1}(\mathbf{h}_i^L \mathbf{w}_o + b_o) = g^{-1}(\boxed{\mathbf{h}(\mathbf{x}_i)} \mathbf{w}_o + b_o)$$

NEW REPRESENTATION
OF FEATURES

Deep Neural Networks

- **Adaptive Basis Function Regression:** Neural networks are just performing regression with adaptive basis functions.

$$\mathbb{E}[\mathbf{y}_i | \mathbf{x}_i] = g^{-1}(\mathbf{h}_i^L \mathbf{w}_o + b_o) = g^{-1}(\boxed{\mathbf{h}(\mathbf{x}_i)} \mathbf{w}_o + b_o)$$

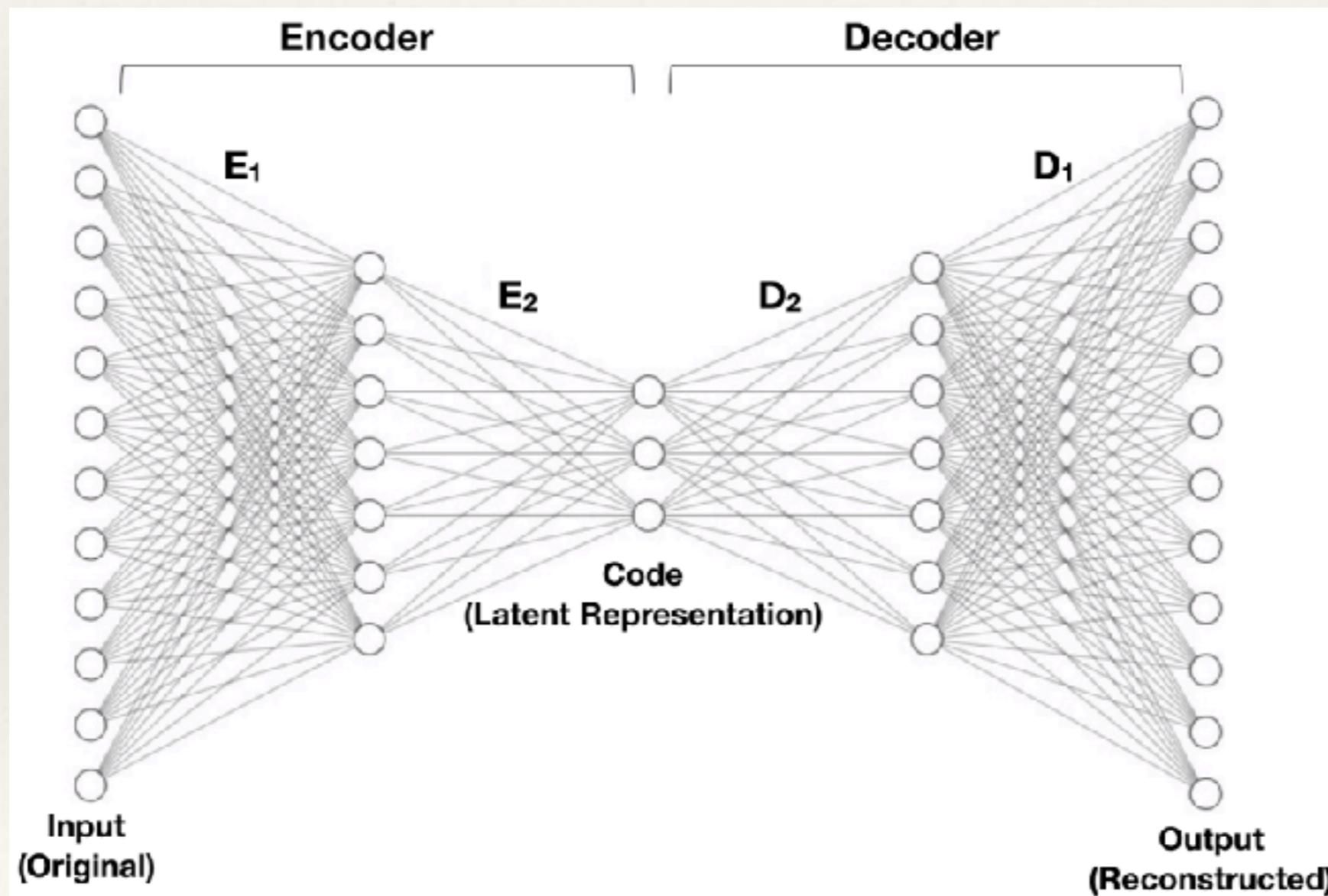
NEW REPRESENTATION
OF FEATURES

- **Backpropagation:** The ‘backpropagation’ algorithm [Rumelhart et al., 1986] is just an efficient way to compute the chain rule.

$$\frac{\partial}{\partial \mathbf{W}_l} \mathcal{L} = \frac{\partial}{\partial \mathbf{h}_i^L} \log p(\mathbf{y}_i | \mathbf{h}_i^L, \mathbf{w}_o, b_o) \frac{\partial \mathbf{h}_i^L}{\partial \mathbf{h}_i^{L-1}} \cdots \frac{\partial \mathbf{h}_i^{l+1}}{\partial \mathbf{h}_i^l} \frac{\partial}{\partial \mathbf{W}_l} f_l(\mathbf{h}_i^{l-1} \mathbf{W}_l + \mathbf{b}_l)$$

Autoencoder

Autoencoder is a special type of feed-forward neural network commonly used for learning latent representations of the data.



Autoencoder

The data are fed from the input layer and encoded into a low-dimensional latent representation; this is known as **coding**.

The code is then **decoded** into a reconstruction of the original data.

The goal of an autoencoder is to learn an identity map such that the output (reconstruction) is close to the input (original) data.

The model is trained to minimize the difference between the original and the reconstructed data.

Autoencoder

Theoretically, an auto-encoder with suitable activation functions could represent an identity map.

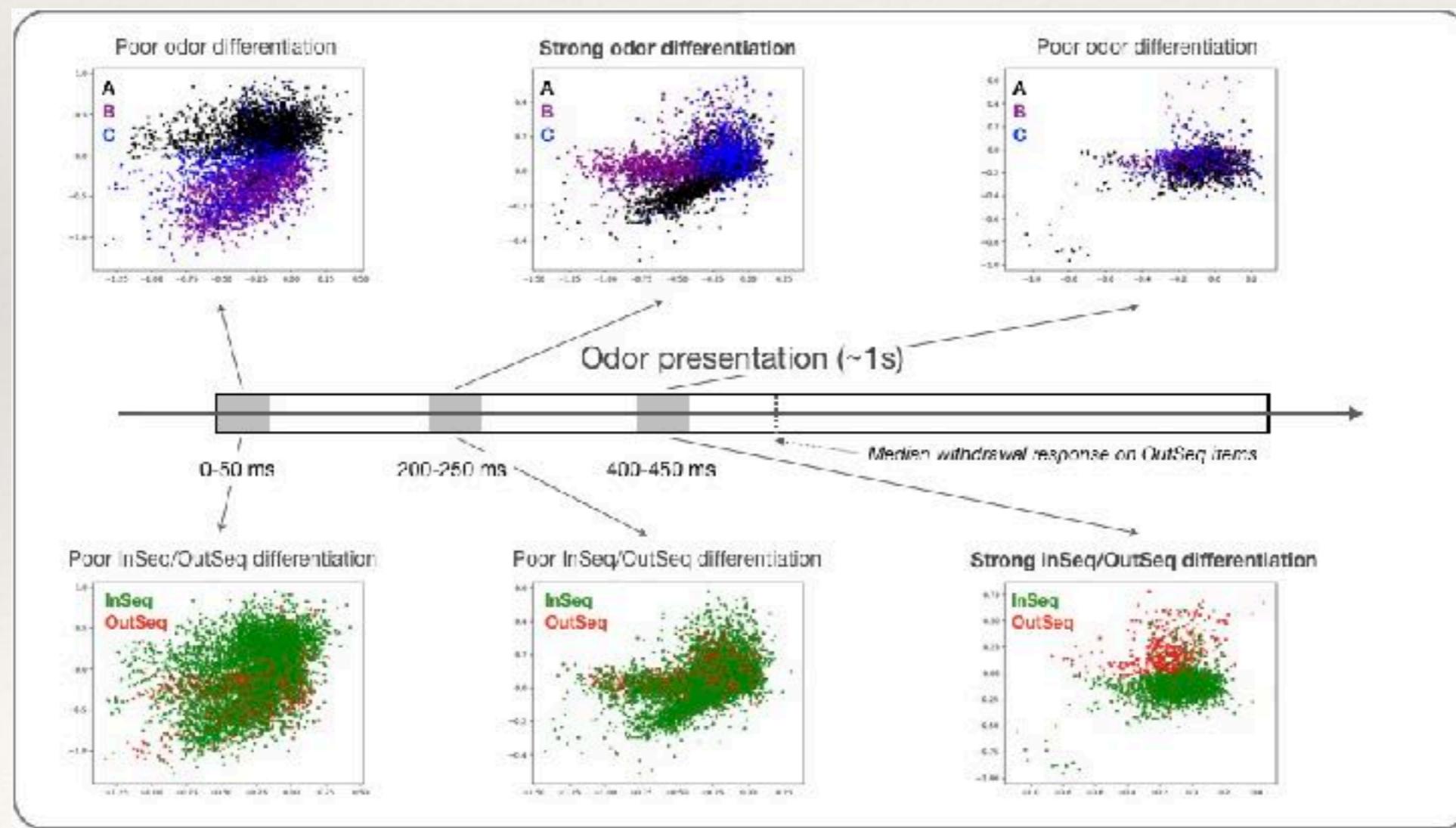
Denoting the encoder as ϕ and the decoder as ψ , ideally we want $\phi \circ \psi = I$.

This way, an accurate reconstruction of the data implies a good low-dimensional representation at the bottleneck (middle layer).

Therefore, autoencoders are commonly used for complicated nonlinear dimensionality reduction.

Autoencoder

The following plots shows low-dimensional representations of neural data obtained from rats learning a sequence of odors (A, B, C), and whether the odors are presented in the right sequence (InSeq) or out of sequence (OutSeq). The data are collected at the Fortin Lab at UCI.



Variational Autoencoder

Variational autoencoder (VAE, Kingma and Welling, 2013) is an extension of standard autoencoder, which can be used for learning deep generative models to generate realistic data such as images and texts.

Compared to a standard autoencoder, variational auto-encoder imposes a prior on the latent variable instead of treating it as a deterministic term. The prior can also be regarded as a regularizer on the latent variables.

More formally, given the observed data, X , we are interested in its latent representation Z . Using the framework of graphical models, we assume that the latent data z are generated from a prior $p_\theta(z)$, and the observed data x are generated from the model $p_\theta(x | z)$.

VAE aims at efficient marginal inference for x as well as finding an approximation for the posterior distribution of z using a simplified model $q_\phi(z | x)$.

Variational Autoencoder

Unlike mean-field variational Bayes, VAE simultaneously maximizes marginal likelihood of x (evidence) and the KL-divergence (KLD) between $q_\phi(z|x)$ and $p_\theta(z|x)$.

This is achieved by maximizing the evidence lower bound (ELBO) with respect to θ and ϕ :

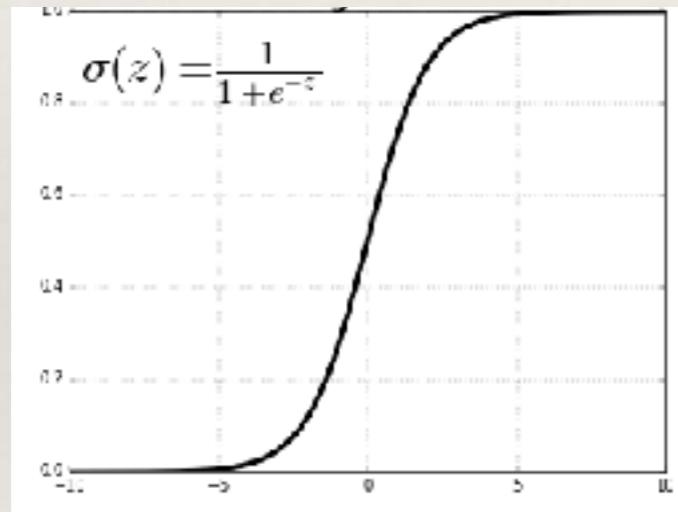
$$\begin{aligned} L(\theta, \phi; x) &= \log p_\theta(x) - KL(q_\phi(z|x) \| p_\theta(z|x)) \\ &= E_{q_\phi(z|x)}(\log p_\theta(x|z)) - KL(q_\phi(z|x) \| p_\theta(z)) \end{aligned}$$

This is equivalent to minimizing $-E_{q_\phi(z|x)}(\log p_\theta(x|z))$, which is called the reconstruction loss, while minimizing KLD between $q_\phi(z|x)$ and $p_\theta(z)$ at the same time.

Some Deep Learning Breakthroughs

Several small but crucial model changes allowed deep networks to start working well.

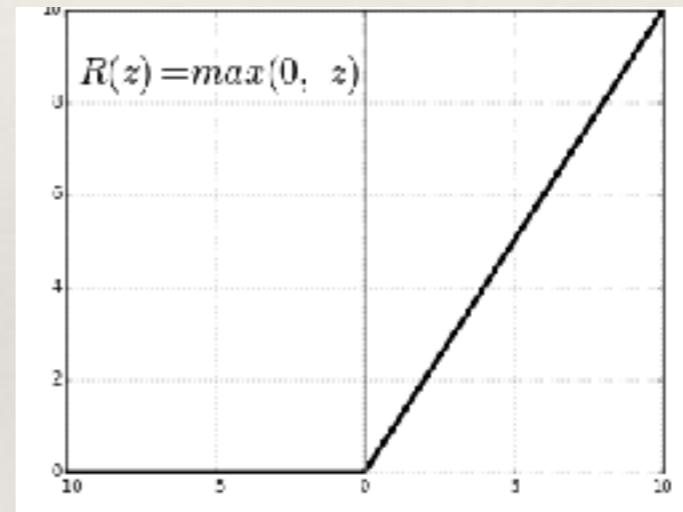
1. ReLU Activations: Scale-free non-linearities [Nair and Hinton, 2010].



LOGISTIC FUNCTION

OLD

VS



RECTIFIED LINEAR UNIT (ReLU)

NEW

Some Deep Learning Breakthroughs

Several small but crucial model changes allowed deep networks to start working well.

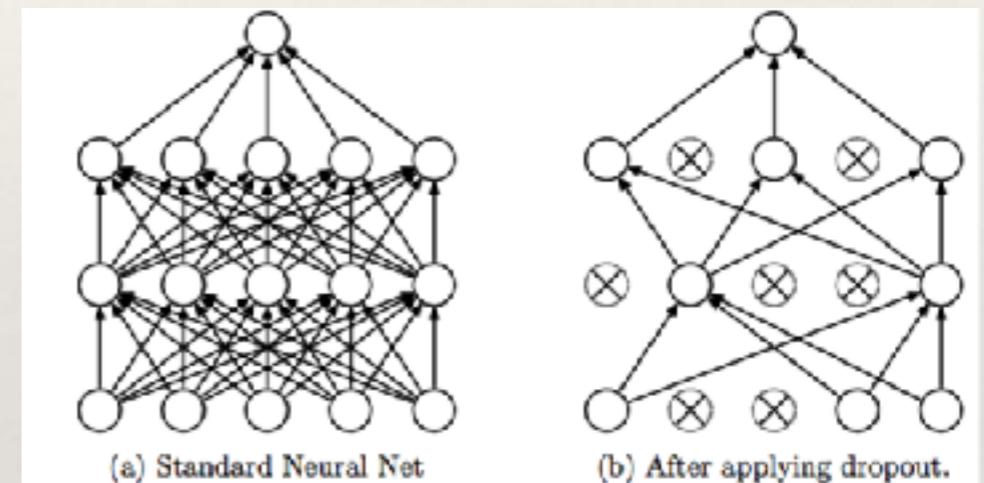
2. Better Regularization: Stochastic regularization strategies
[Srivastava et al., 2014].

$$\sum_{l=1}^{L+1} \|\mathbf{W}_l\|_2^2$$

RIDGE PENALTY

OLD

VS



DROPOUT

NEW

Some Deep Learning Breakthroughs

Several small but crucial model changes allowed deep networks to start working well.

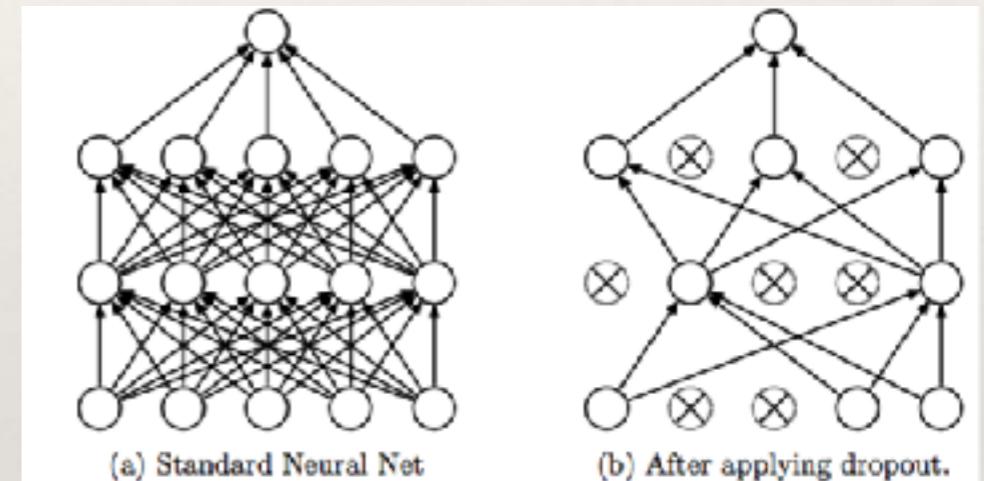
2. Better Regularization: Stochastic regularization strategies [Srivastava et al., 2014].

$$\sum_{l=1}^{L+1} \|\mathbf{W}_l\|_2^2$$

RIDGE PENALTY

OLD

VS



DROPOUT

NEW

Can think of dropout as spike and slab variable selection [Kuo and Mallick, 1998]:

$$\mathbf{h}_i^l = f_l \left((\mathbf{h}_i^{l-1} \odot \boldsymbol{\xi}) \mathbf{W}_l + \mathbf{b}_l \right), \boldsymbol{\xi} \sim \text{Bernoulli}(p)$$

Connecting GPs and Neural Nets

Infinitely Wide Neural Networks as GPs

Recall our formulation of NNs as regression with adaptive basis functions:

$$\psi(\mathbf{x}_i) = \mathbf{h}(\mathbf{x}_i)\mathbf{w}_o = \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i)$$

Infinitely Wide Neural Networks as GPs

Recall our formulation of NNs as regression with adaptive basis functions:

$$\psi(\mathbf{x}_i) = \mathbf{h}(\mathbf{x}_i)\mathbf{w}_o = \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i)$$

Next, assume a Gaussian prior on the weights:

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

Infinitely Wide Neural Networks as GPs

Recall our formulation of NNs as regression with adaptive basis functions:

$$\psi(\mathbf{x}_i) = \mathbf{h}(\mathbf{x}_i)\mathbf{w}_o = \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i)$$

Next, assume a Gaussian prior on the weights:

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

Assuming the hidden units are bounded, then the CLT applies as the number of weights approaches infinity:

$$p(\psi(\mathbf{x}_i)) \sim N\left(0, \frac{\sigma^2}{H} H V(\mathbf{x}_i)\right) = N\left(0, \sigma^2 V(\mathbf{x}_i)\right)$$

$$V(\mathbf{x}_i) = \mathbb{E}[h_j^2(\mathbf{x}_i)]$$

Infinitely Wide Neural Networks as GPs

To show the equivalence to GPs, recall we must show joint Normality.
Consider the vector:

$$\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \\ \sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \end{pmatrix}$$

Infinitely Wide Neural Networks as GPs

To show the equivalence to GPs, recall we must show joint Normality.
Consider the vector:

$$\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \\ \sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \end{pmatrix}$$

Sending H to infinity we find [Neal, 1995]...

$$\mathbb{E} \left[\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} \right] = \begin{pmatrix} \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_i) \\ \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Infinitely Wide Neural Networks as GPs

To show the equivalence to GPs, recall we must show joint Normality.
Consider the vector:

$$\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \\ \sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \end{pmatrix}$$

Sending H to infinity we find [Neal, 1995]...

$$\mathbb{E} \left[\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} \right] = \begin{pmatrix} \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_i) \\ \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\mathbb{E} [\psi(\mathbf{x}_i)\psi(\mathbf{x}_k)] = \mathbb{E} \left[\left(\sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \right) \left(\sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \right) \right] = \sum_{j=1}^H \mathbb{E} [(w_j^o)^2] \mathbb{E} [h_j(\mathbf{x}_i)h_j(\mathbf{x}_k)] = \sigma^2 V(\mathbf{x}_i, \mathbf{x}_k)$$

Infinitely Wide Neural Networks as GPs

To show the equivalence to GPs, recall we must show joint Normality.
Consider the vector:

$$\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \\ \sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \end{pmatrix}$$

Sending H to infinity we find [Neal, 1995]...

$$\mathbb{E} \left[\begin{pmatrix} \psi(\mathbf{x}_i) \\ \psi(\mathbf{x}_k) \end{pmatrix} \right] = \begin{pmatrix} \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_i) \\ \sum_{j=1}^H \mathbb{E}[w_j^o] h_j(\mathbf{x}_k) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\mathbb{E} [\psi(\mathbf{x}_i)\psi(\mathbf{x}_k)] = \mathbb{E} \left[\left(\sum_{j=1}^H w_j^o h_j(\mathbf{x}_i) \right) \left(\sum_{j=1}^H w_j^o h_j(\mathbf{x}_k) \right) \right] = \sum_{j=1}^H \mathbb{E} [(w_j^o)^2] \mathbb{E} [h_j(\mathbf{x}_i)h_j(\mathbf{x}_k)] = \sigma^2 V(\mathbf{x}_i, \mathbf{x}_k)$$

And therefore...

$$(\psi(\mathbf{x}_i), \psi(\mathbf{x}_k)) \sim N(0, \sigma^2 \mathbf{V})$$

Infinitely Wide Neural Networks as GPs

What if we place Gaussian priors on weights at the previous layers?:

$$w_{m,n}^l \sim N(0, \sigma^2/H)$$

Infinitely Wide Neural Networks as GPs

What if we place Gaussian priors on weights at the previous layers?:

$$w_{m,n}^l \sim N(0, \sigma^2/H)$$

The GP equivalence can be found recursively [Lee et al., 2018]:

$$\mathbb{E} [\psi_l(\mathbf{x}_k) \psi_l(\mathbf{x}_i)] = \sigma_l^2 \mathbb{E}_{\psi_{l-1} \sim \text{GP}(\mathbf{0}, \mathbf{K}^{l-1})} [f_l(\psi_{l-1}(\mathbf{x}_k)) f_l(\psi_{l-1}(\mathbf{x}_i))]$$

Infinitely Wide Neural Networks as GPs

What if we place Gaussian priors on weights at the previous layers?:

$$w_{m,n}^l \sim N(0, \sigma^2/H)$$

The GP equivalence can be found recursively [Lee et al., 2018]:

$$\mathbb{E} [\psi_l(\mathbf{x}_k) \psi_l(\mathbf{x}_i)] = \sigma_l^2 \mathbb{E}_{\psi_{l-1} \sim \text{GP}(\mathbf{0}, \mathbf{K}^{l-1})} [f_l(\psi_{l-1}(\mathbf{x}_k)) f_l(\psi_{l-1}(\mathbf{x}_i))]$$

ACTIVATION FUNCTION

Infinitely Wide Neural Networks as GPs

What if we place Gaussian priors on weights at the previous layers?:

$$w_{m,n}^l \sim N(0, \sigma^2/H)$$

The GP equivalence can be found recursively [Lee et al., 2018]:

ACTIVATION FUNCTION

$$\mathbb{E} [\psi_l(\mathbf{x}_k) \psi_l(\mathbf{x}_i)] = \sigma_l^2 \mathbb{E}_{\psi_{l-1} \sim \text{GP}(\mathbf{0}, \mathbf{K}^{l-1})} [f_l(\psi_{l-1}(\mathbf{x}_k)) f_l(\psi_{l-1}(\mathbf{x}_i))]$$

For ReLU activations, the covariance is [Lee et al., 2018]:

$$\mathbb{E} [\psi_l(\mathbf{x}_k) \psi_l(\mathbf{x}_i)] = \frac{\sigma_l^2}{2\pi} \sqrt{\kappa^{l-1}(\mathbf{x}_i, \mathbf{x}_i) \kappa^{l-1}(\mathbf{x}_k, \mathbf{x}_k)} [\sin \theta(\mathbf{x}_i, \mathbf{x}_k) + (\pi - \theta(\mathbf{x}_i, \mathbf{x}_k)) \cos \theta(\mathbf{x}_i, \mathbf{x}_k)]$$

$$\theta(\mathbf{x}_i, \mathbf{x}_k) = \cos^{-1} \frac{\kappa^{l-1}(\mathbf{x}_i, \mathbf{x}_k)}{\sqrt{\kappa^{l-1}(\mathbf{x}_i, \mathbf{x}_i) \kappa^{l-1}(\mathbf{x}_k, \mathbf{x}_k)}}$$

GPs Don't Learn Latent Representations

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

“With Gaussian priors the contributions of individual units are all negligible, and consequently, these units do not represent ‘hidden features’ that capture important aspects of the data” [Neal, 1995]

GPs Don't Learn Latent Representations

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

“With Gaussian priors the contributions of individual units are all negligible, and consequently, these units do not represent ‘hidden features’ that capture important aspects of the data” [Neal, 1995]

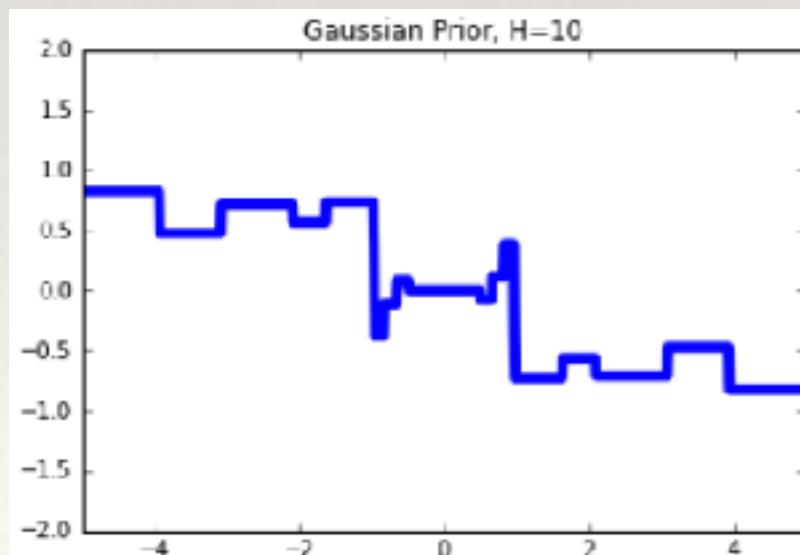
Function draws from a NN with H hidden units and a Gaussian prior:

GPs Don't Learn Latent Representations

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

“With Gaussian priors the contributions of individual units are all negligible, and consequently, these units do not represent ‘hidden features’ that capture important aspects of the data” [Neal, 1995]

Function draws from a NN with H hidden units and a Gaussian prior:

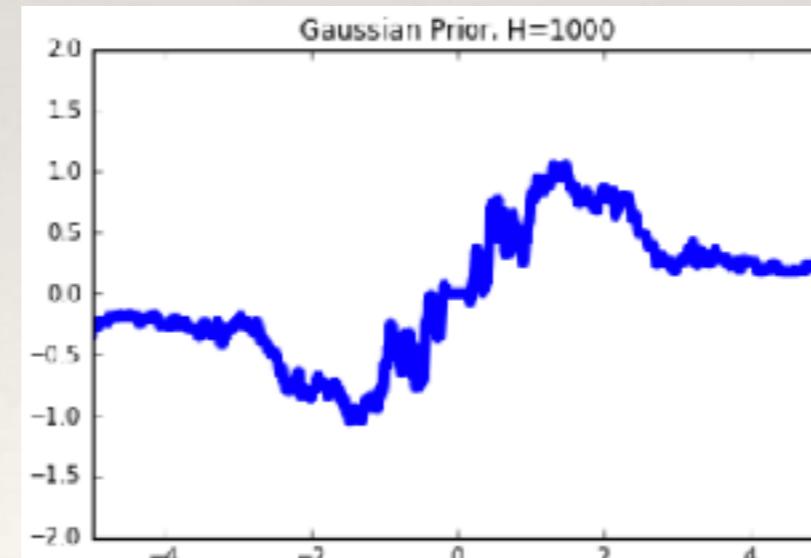
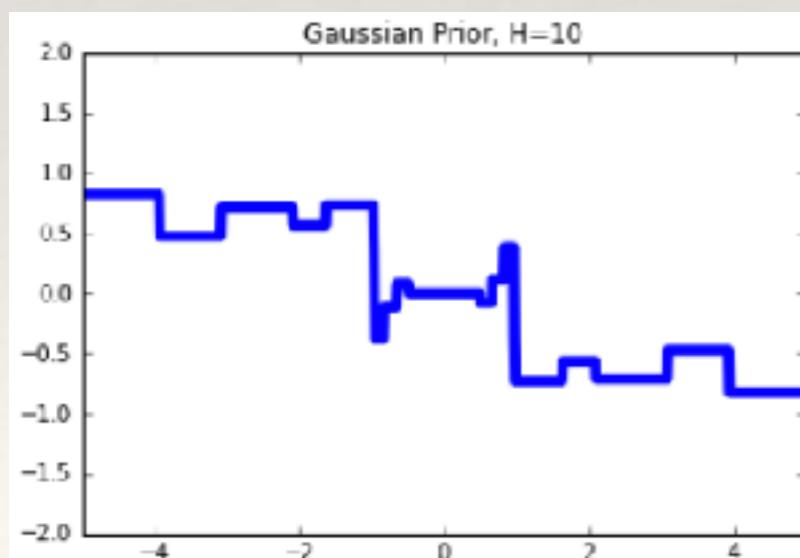


GPs Don't Learn Latent Representations

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

“With Gaussian priors the contributions of individual units are all negligible, and consequently, these units do not represent ‘hidden features’ that capture important aspects of the data” [Neal, 1995]

Function draws from a NN with H hidden units and a Gaussian prior:

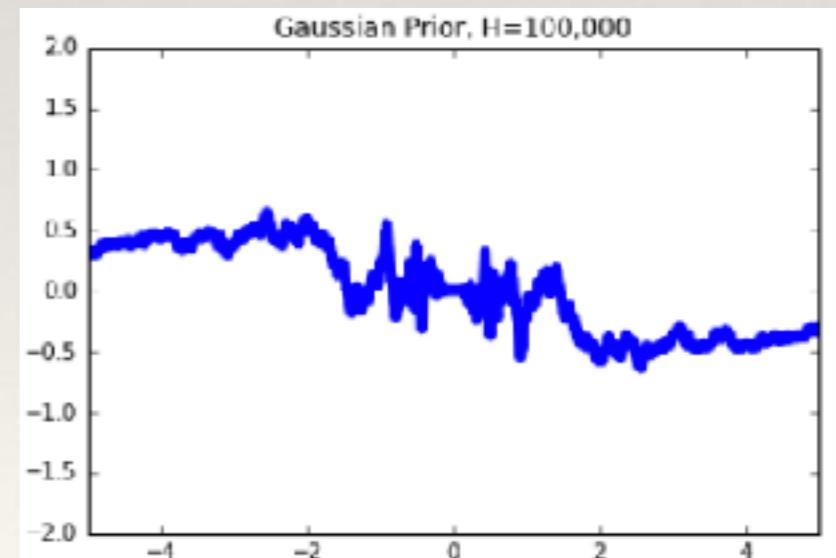
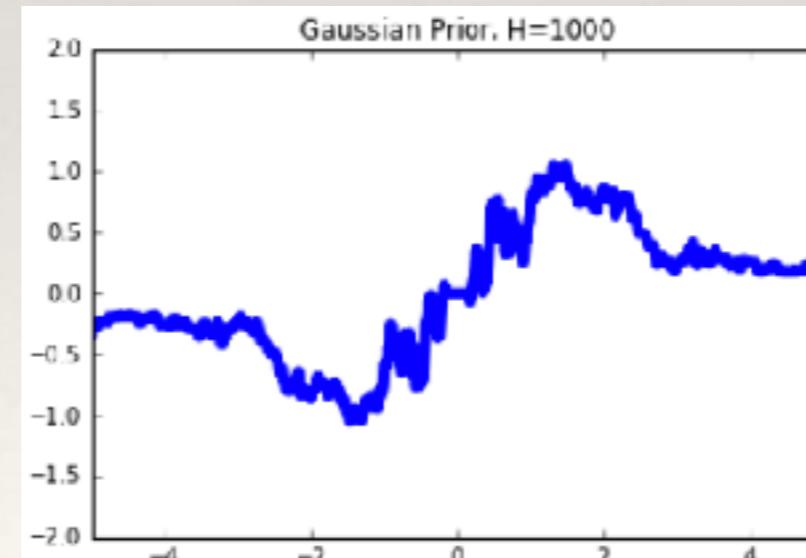
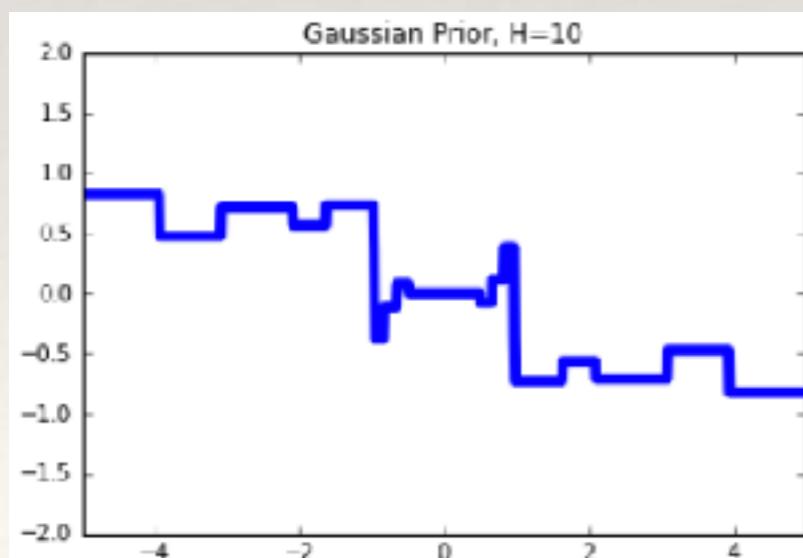


GPs Don't Learn Latent Representations

$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

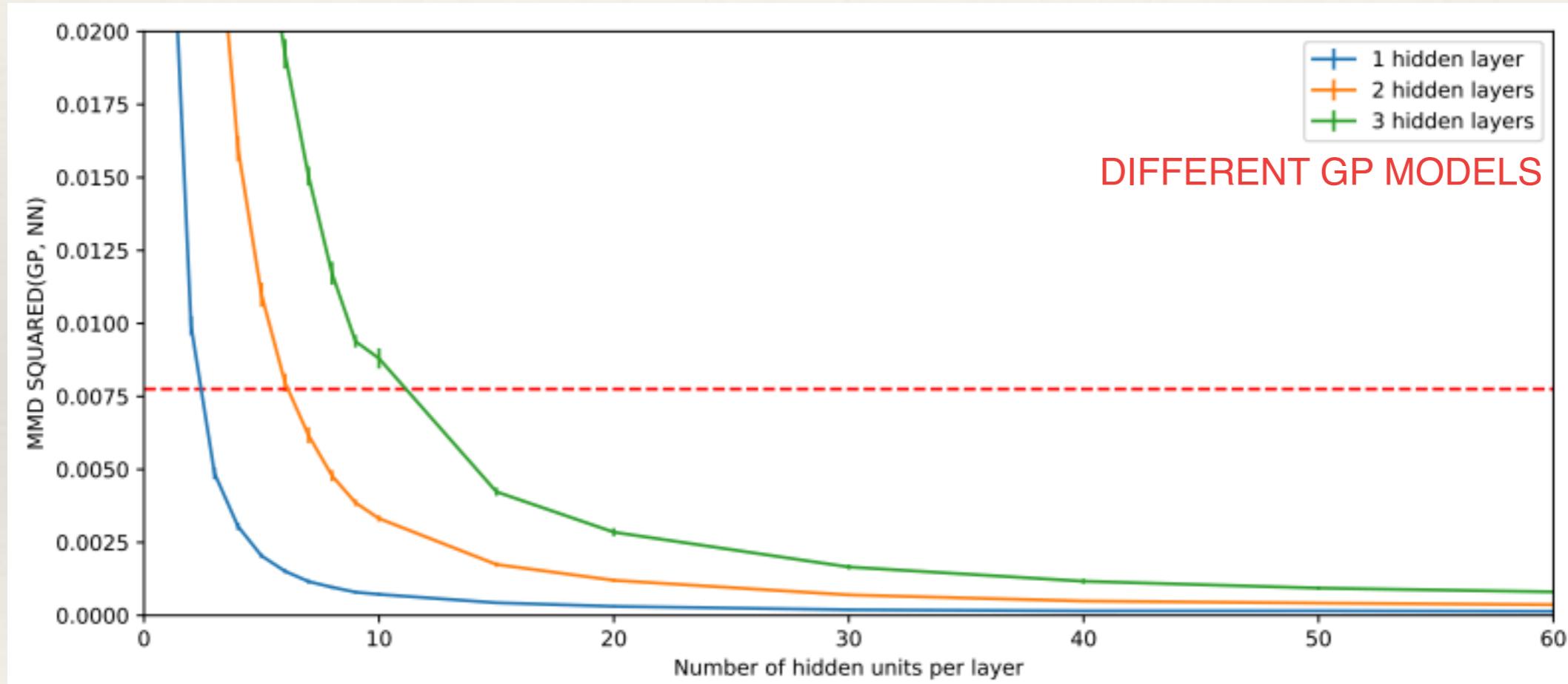
“With Gaussian priors the contributions of individual units are all negligible, and consequently, these units do not represent ‘hidden features’ that capture important aspects of the data” [Neal, 1995]

Function draws from a NN with H hidden units and a Gaussian prior:



Bayesian NNs Behave Like GPs

Empirical Investigation of GP behavior in Bayesian NNs:



Building Neural Nets from Recursive GLMs

What if we don't want GP behavior? How should we specify our Bayesian NN?

“...stop the onset of the central limit theorem”

~ [Matthews et al., 2018]

Priors with Infinite Variance

"For priors based on symmetric stable distributions of index < 2 (e.g. Cauchy), some of the hidden units in an infinite network have output weights of significant size, allowing them to represent 'hidden features'." [Neal, 1995]

Recall that stable distributions have the following characteristic function:

$$\phi(t; \alpha, \beta, c, \mu) = \exp(it\mu - |ct|^\alpha(1 - i\beta \operatorname{sgn}(t)\tan(\pi\alpha/2))) \quad \alpha \neq 1$$

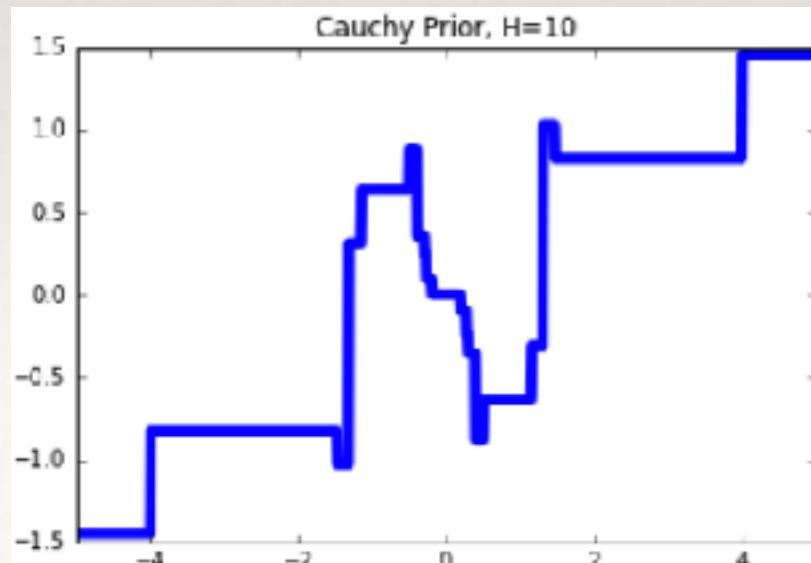
Priors with Infinite Variance

"For priors based on symmetric stable distributions of index < 2 (e.g. Cauchy), some of the hidden units in an infinite network have output weights of significant size, allowing them to represent 'hidden features'." [Neal, 1995]

Recall that stable distributions have the following characteristic function:

$$\phi(t; \alpha, \beta, c, \mu) = \exp(it\mu - |ct|^\alpha(1 - i\beta \operatorname{sgn}(t)\tan(\pi\alpha/2))) \quad \alpha \neq 1$$

Function draws from a NN with H hidden units and a Cauchy prior:



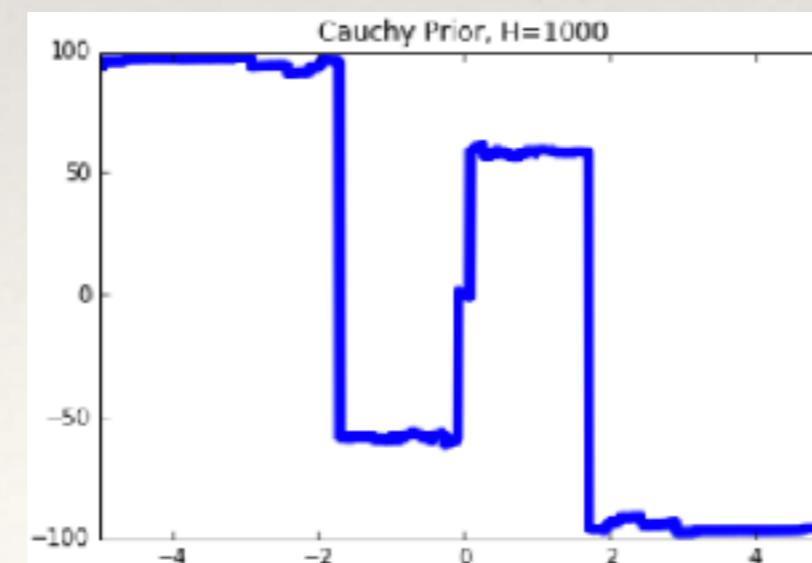
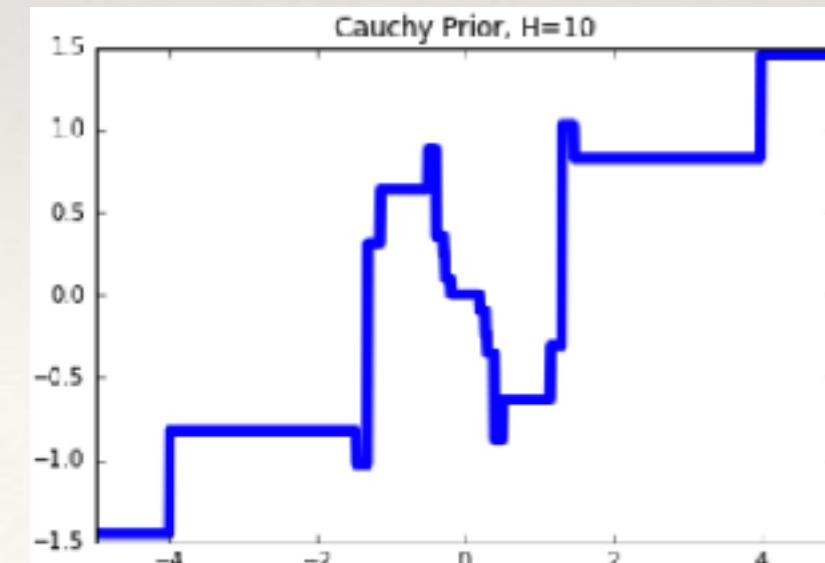
Priors with Infinite Variance

"For priors based on symmetric stable distributions of index < 2 (e.g. Cauchy), some of the hidden units in an infinite network have output weights of significant size, allowing them to represent 'hidden features'." [Neal, 1995]

Recall that stable distributions have the following characteristic function:

$$\phi(t; \alpha, \beta, c, \mu) = \exp(it\mu - |ct|^\alpha(1 - i\beta \operatorname{sgn}(t)\tan(\pi\alpha/2))) \quad \alpha \neq 1$$

Function draws from a NN with H hidden units and a Cauchy prior:



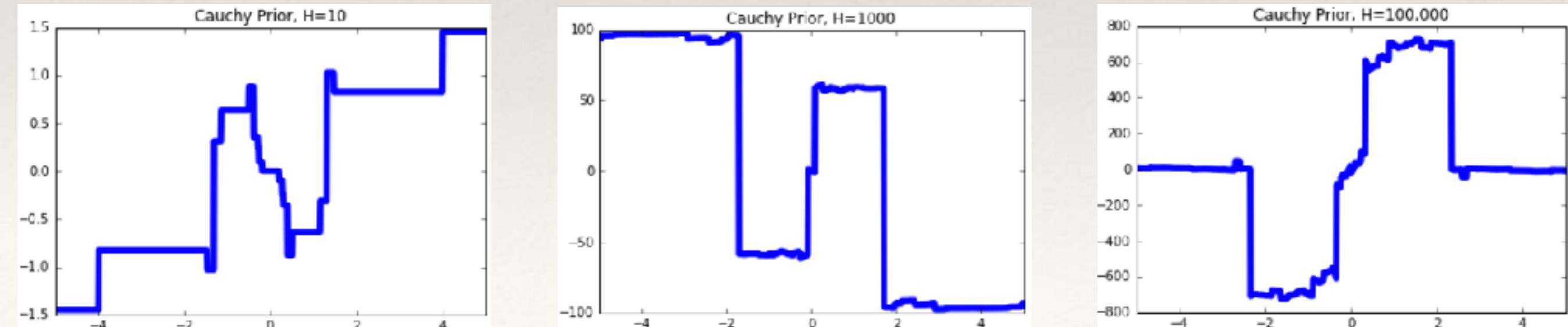
Priors with Infinite Variance

"For priors based on symmetric stable distributions of index < 2 (e.g. Cauchy), some of the hidden units in an infinite network have output weights of significant size, allowing them to represent 'hidden features'." [Neal, 1995]

Recall that stable distributions have the following characteristic function:

$$\phi(t; \alpha, \beta, c, \mu) = \exp(it\mu - |ct|^\alpha(1 - i\beta \operatorname{sgn}(t)\tan(\pi\alpha/2))) \quad \alpha \neq 1$$

Function draws from a NN with H hidden units and a Cauchy prior:



Neal's Recommendation: Student-t

$$w_j^o \sim N(0, \sigma^2)$$

$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

The marginal density is student-t with ν degrees of freedom. Admits Gibbs sampling strategies.

Neal's Recommendation: Student-t

$$w_j^o \sim N(0, \sigma^2)$$

$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

The marginal density is student-t with v degrees of freedom. Admits Gibbs sampling strategies.

Function draws from a NN with H hidden units and a Student-t prior:

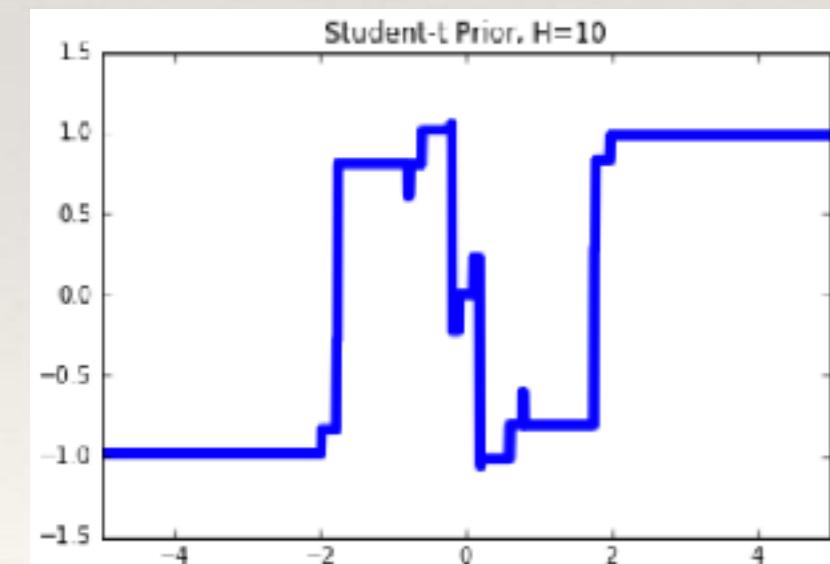
Neal's Recommendation: Student-t

$$w_j^o \sim N(0, \sigma^2)$$

$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

The marginal density is student-t with ν degrees of freedom. Admits Gibbs sampling strategies.

Function draws from a NN with H hidden units and a Student-t prior:



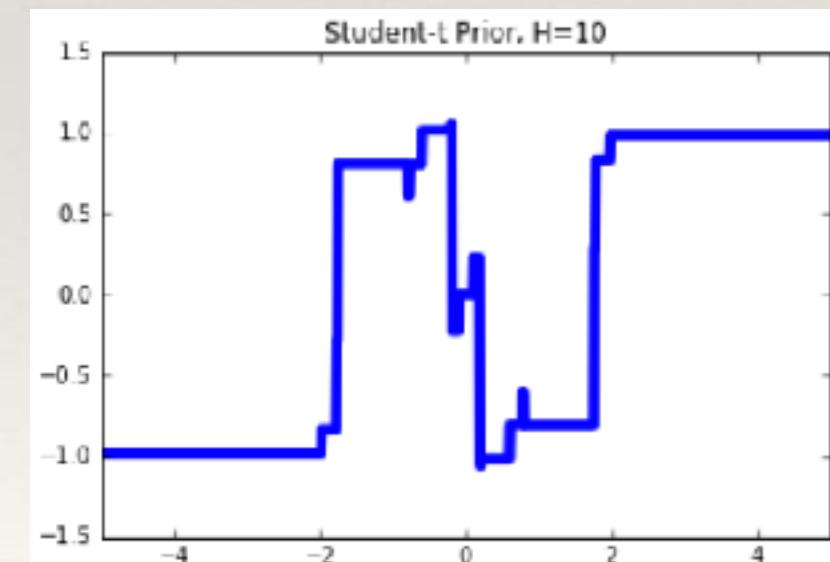
Neal's Recommendation: Student-t

$$w_j^o \sim N(0, \sigma^2)$$

$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

The marginal density is student-t with ν degrees of freedom. Admits Gibbs sampling strategies.

Function draws from a NN with H hidden units and a Student-t prior:



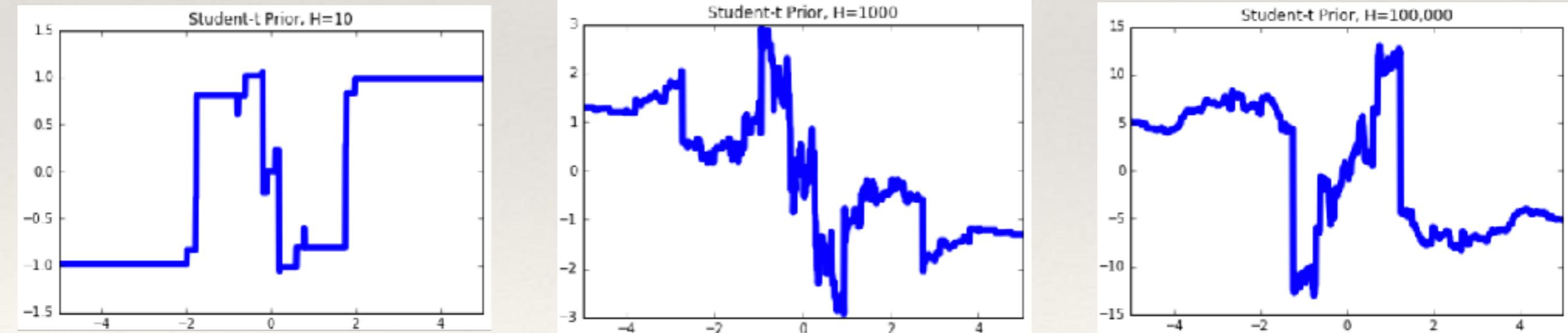
Neal's Recommendation: Student-t

$$w_j^o \sim N(0, \sigma^2)$$

$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

The marginal density is student-t with ν degrees of freedom. Admits Gibbs sampling strategies.

Function draws from a NN with H hidden units and a Student-t prior:



References

- Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.
- Alexander G. de G Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *International Conference on Learning Representations (ICLR)*, 2018.
- J. Deng, K. Li, M. Du, H. Su, and L. Fei-Fei. Construction and Analysis of a Large Scale Image Ontology. Vision Sciences Society, 2009.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- Lynn Kuo and Bani Mallick. Variable selection for regression models. *Sankhyā: The Indian Journal of Statistics, Series B*, pages 65–81, 1998.
- Jachoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *International Conference on Learning Representations (ICLR)*, 2018.
- David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.
- Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- Radford M Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 1995.
- David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.
- Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, pages 2171–2180, 2015.
- Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.