Gaussians Processes and Deep Neural Networks: A tale of two non-parametric regressors

Eric Nalisnick

University of California, Irvine

Outline

1 **Background: Gaussian Processes**

2 **Deep Neural Networks and Deep Learning**

3 **Connecting GPs and Neural Networks**

4 ***Dis*connecting GPs and Neural Networks**

Gaussian Processes

Gaussian Processes

Assume we observe data:

$$\{\mathbf{X}, \mathbf{y}\} = \{(\mathbf{x}_i, y_i)_{i=1}^N\}$$

Gaussian Processes

Assume we observe data:

$$\{\mathbf{X}, \mathbf{y}\} = \{(\mathbf{x}_i, y_i)_{i=1}^N\}$$

Then a Gaussian Process (GP) model is specified as:

$$y_i = \phi(\mathbf{x}_i) + \epsilon_i$$

$$\phi \sim \mathrm{GP}(\cdot|\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}})$$

$$\epsilon_i \sim \mathrm{N}(0, \sigma^2)$$

Gaussian Processes

$$y_i = \phi(\mathbf{x}_i) + \epsilon_i, \;\; \phi \sim \mathrm{GP}(\cdot|\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}}) \;\; , \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2)$$

Definition: The GP is named as such because the joint distribution over function evaluations is multivariate Normal, i.e.

$$p\left(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\right) = \mathrm{N}(\boldsymbol{\mu}, \mathbf{K}_{\boldsymbol{\theta}})$$

Gaussian Processes

$$y_i = \phi(\mathbf{x}_i) + \epsilon_i, \;\; \phi \sim \mathrm{GP}(\cdot|\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}}) \;, \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2)$$

Definition: The GP is named as such because the joint distribution over function evaluations is multivariate Normal, i.e.

$$p\left(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\right) = \mathrm{N}(\boldsymbol{\mu}, \mathbf{K}_{\boldsymbol{\theta}})$$

Covariance Matrix: The (zero mean) GP is fully specified by the covariance matrix, whose elements are the covariance function evaluated between all pairs of feature vectors.

$$k_{\boldsymbol{\theta}}^{i,j} = \kappa(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta})$$

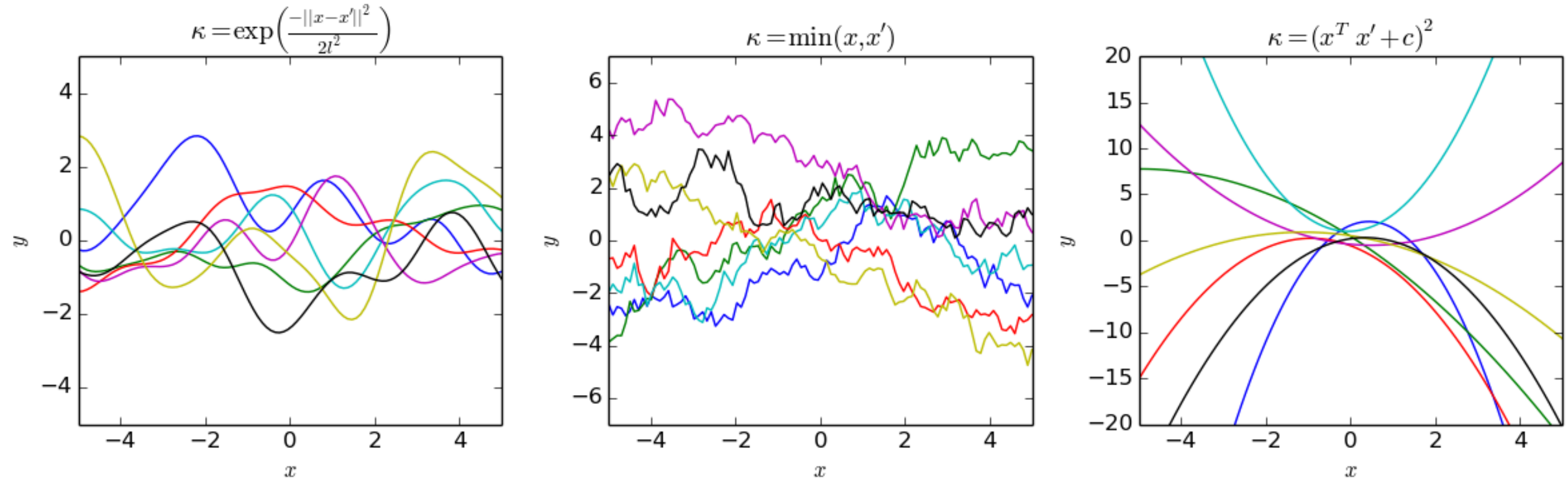
Example: Squared Exponential

$$\kappa(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta}) = \exp\left\{ \frac{||\mathbf{x}_i - \mathbf{x}_j||_2^2}{2\boldsymbol{\theta}} \right\}$$

Gaussian Processes

$$y_i = \phi(\mathbf{x}_i) + \epsilon_i, \;\; \phi \sim \mathrm{GP}(\cdot|\mathbf{0}, \mathbf{K}_{\boldsymbol{\theta}}) \;, \quad \epsilon_i \sim \mathrm{N}(0, \sigma^2)$$

Kernel Function: The choice of kernel function results in regression functions of varying smoothness.



https://en.wikipedia.org/wiki/Gaussian_process#/media/File:Gaussian_process_draws_from_prior_distribution.png

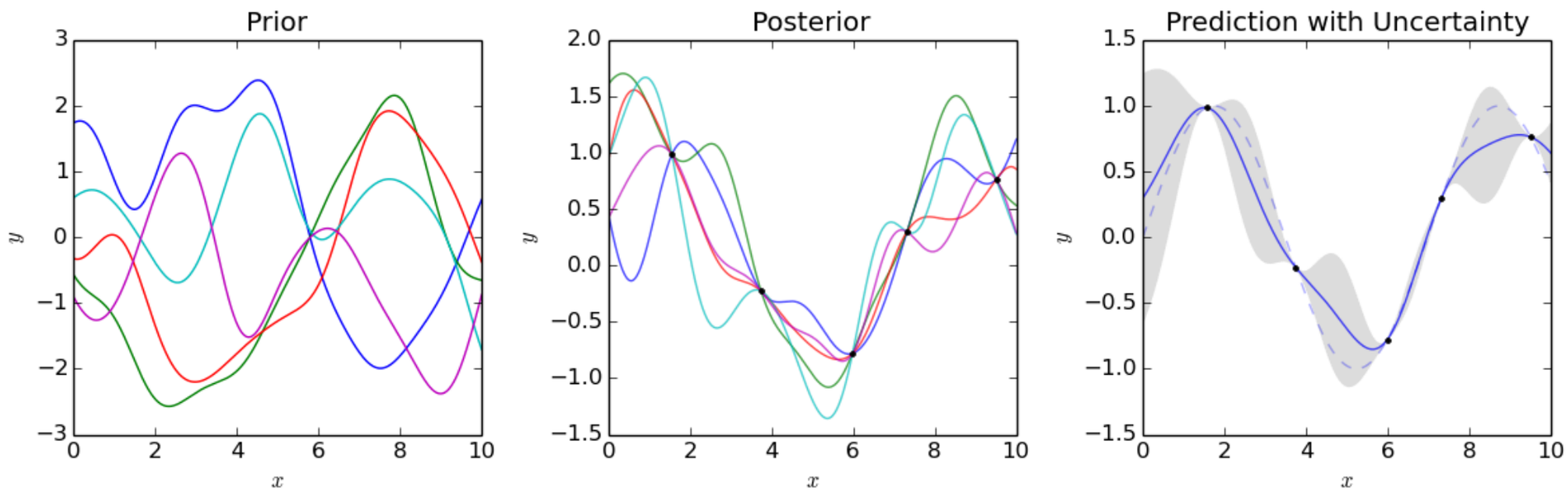
Gaussian Processes

Posterior Inference: Due to all variables being Gaussian, posterior quantities are available in closed-form.

MARGINAL LIKELIHOOD

$$\log p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta}) = \log \int p(\mathbf{y}|\mathbf{X},\phi)p(\phi;\boldsymbol{\theta})d\phi$$

$$= \frac{-1}{2}\log|\mathbf{K}_{\boldsymbol{\theta}} + \sigma^2\mathbb{I}| + \frac{-1}{2}\mathbf{y}^T(\mathbf{K}_{\boldsymbol{\theta}} + \sigma^2\mathbb{I})^{-1}\mathbf{y} + \text{const}$$

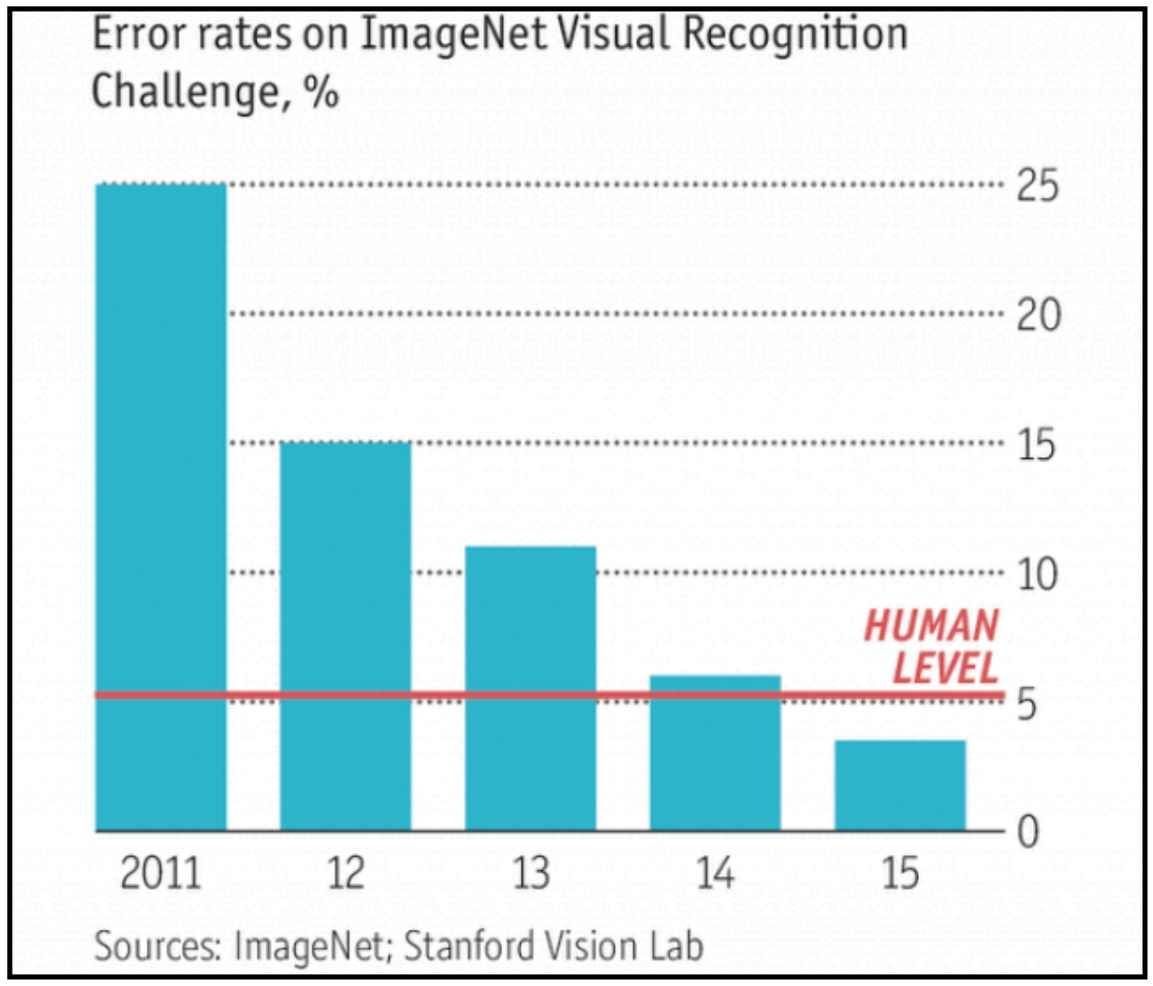


https://en.wikipedia.org/wiki/Gaussian_process#/media/File:Gaussian_Process_Regression.png

Neural Networks and Deep Learning

Deep Learning Results

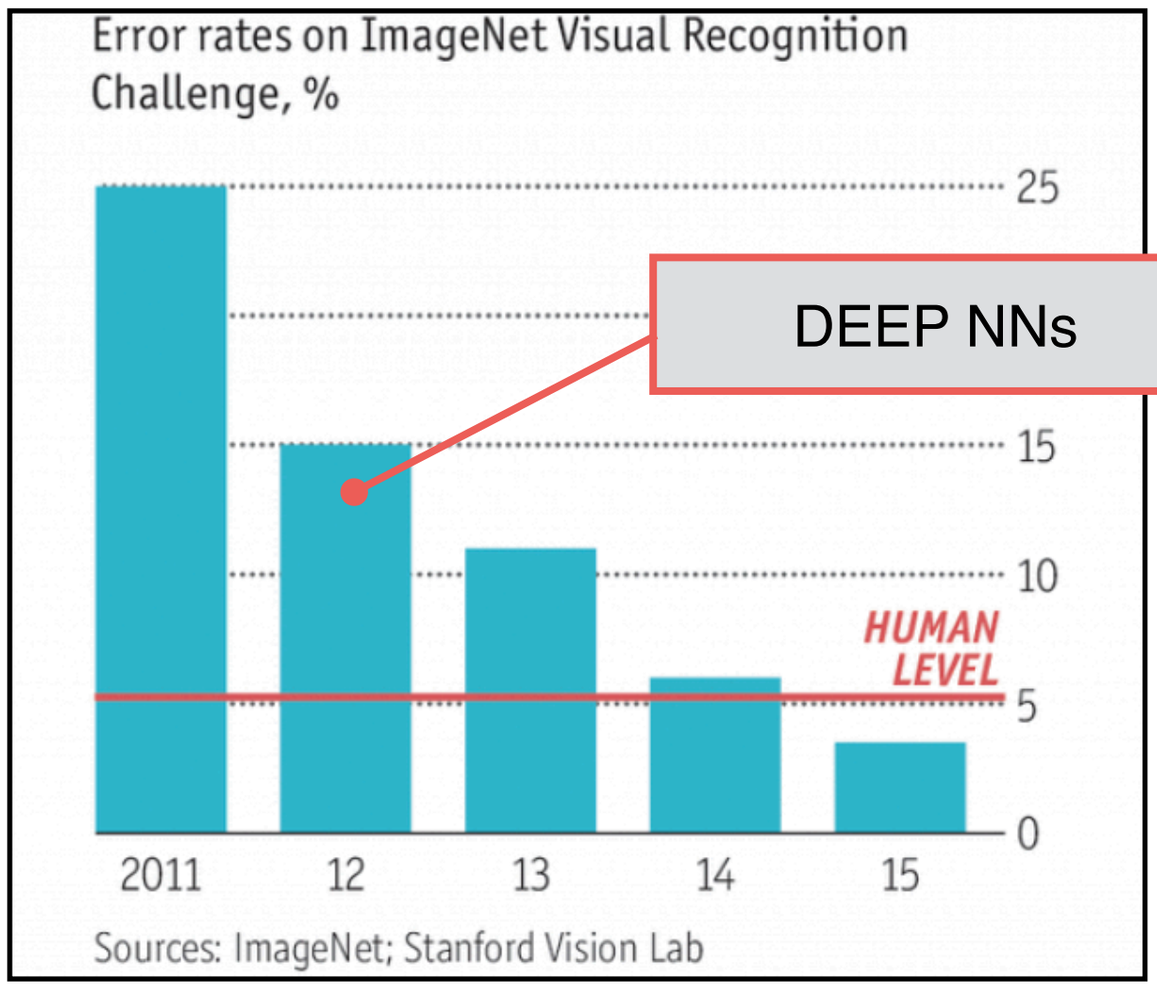
- **Computer Vision**: Results on ImageNet object classification dataset.



https://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

Deep Learning Results

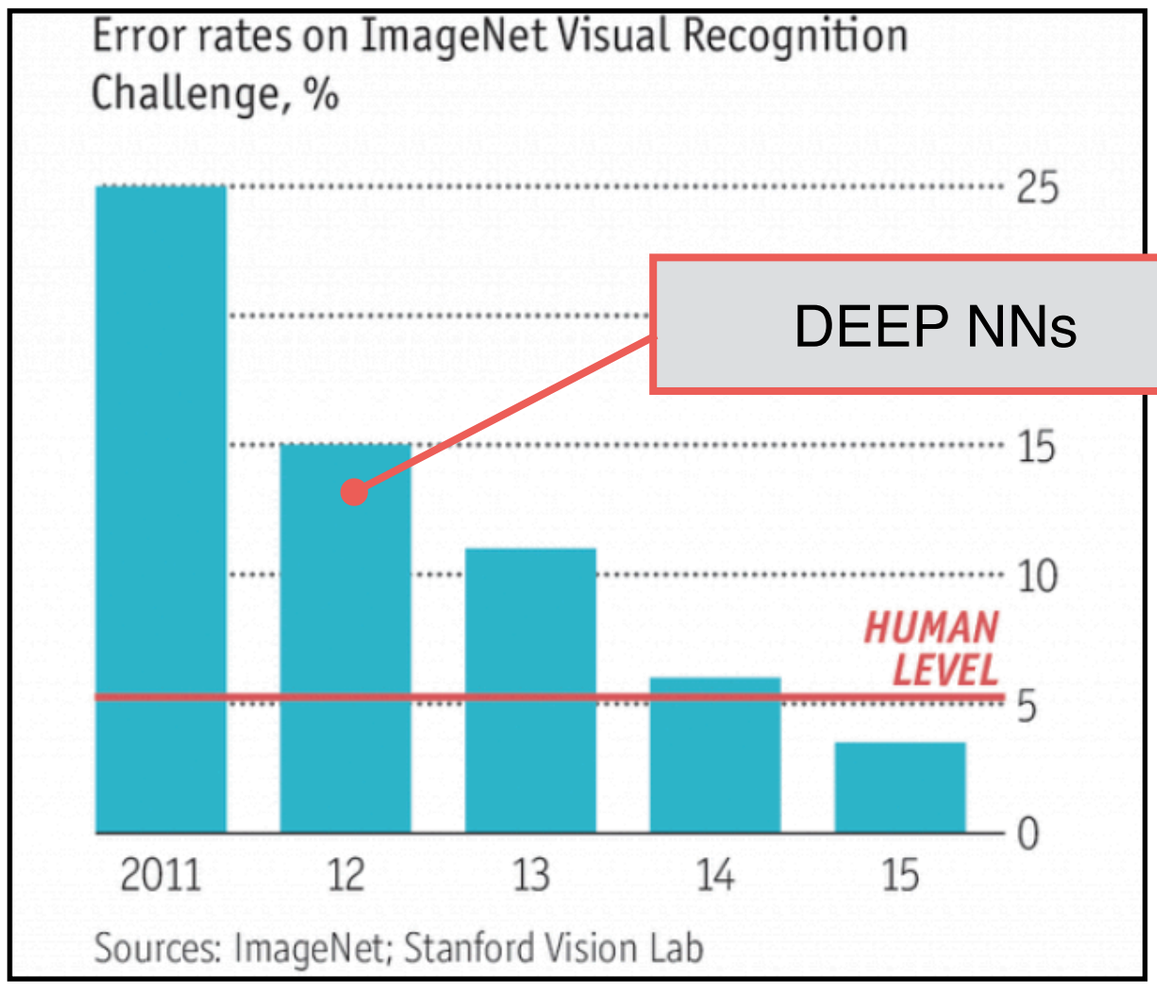
- **Computer Vision**: Results on ImageNet object classification dataset.

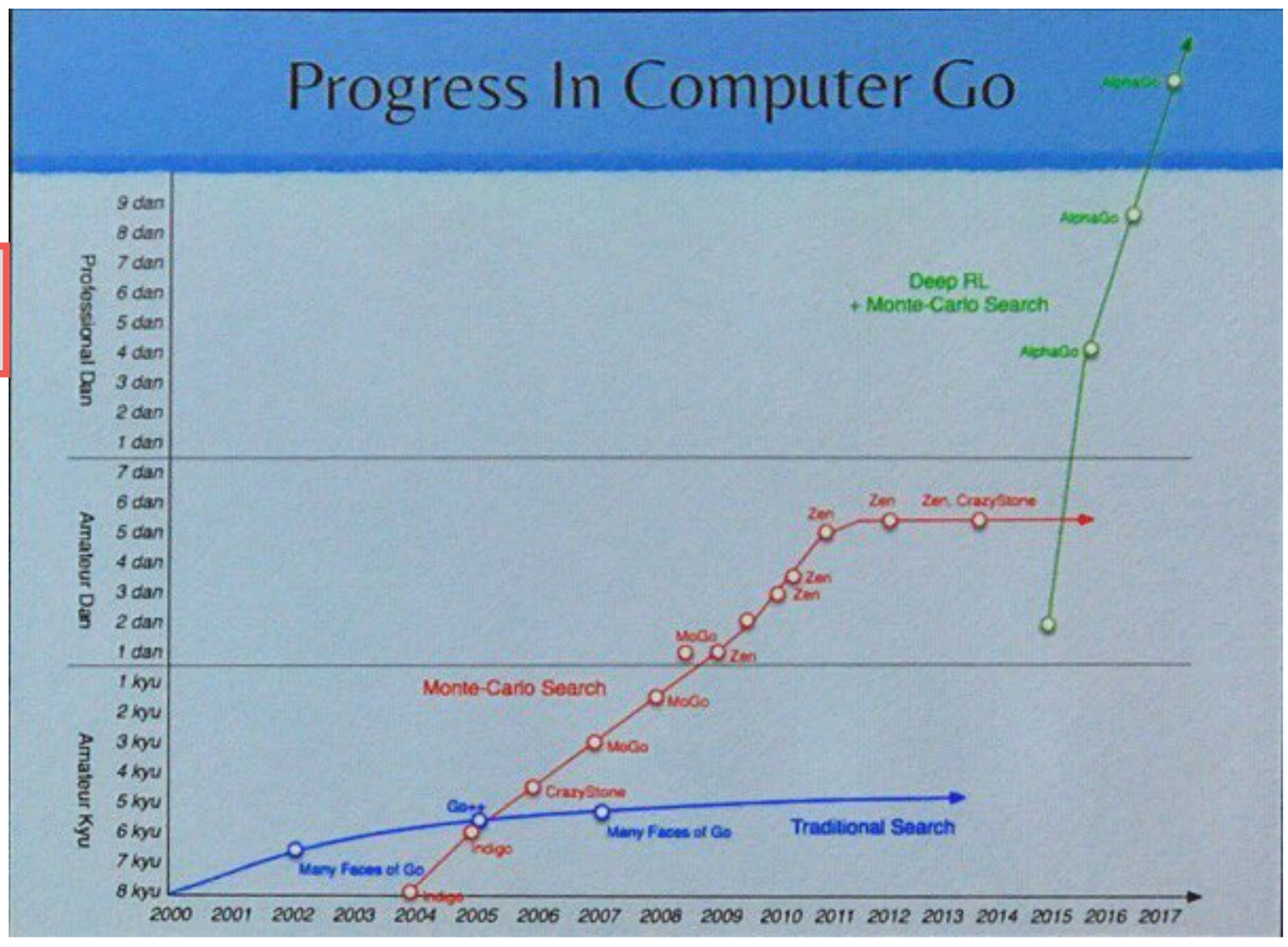


https://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

Deep Learning Results

- **Computer Vision**: Results on ImageNet object classification dataset.
- **Reinforcement Learning**: Results in playing Go.

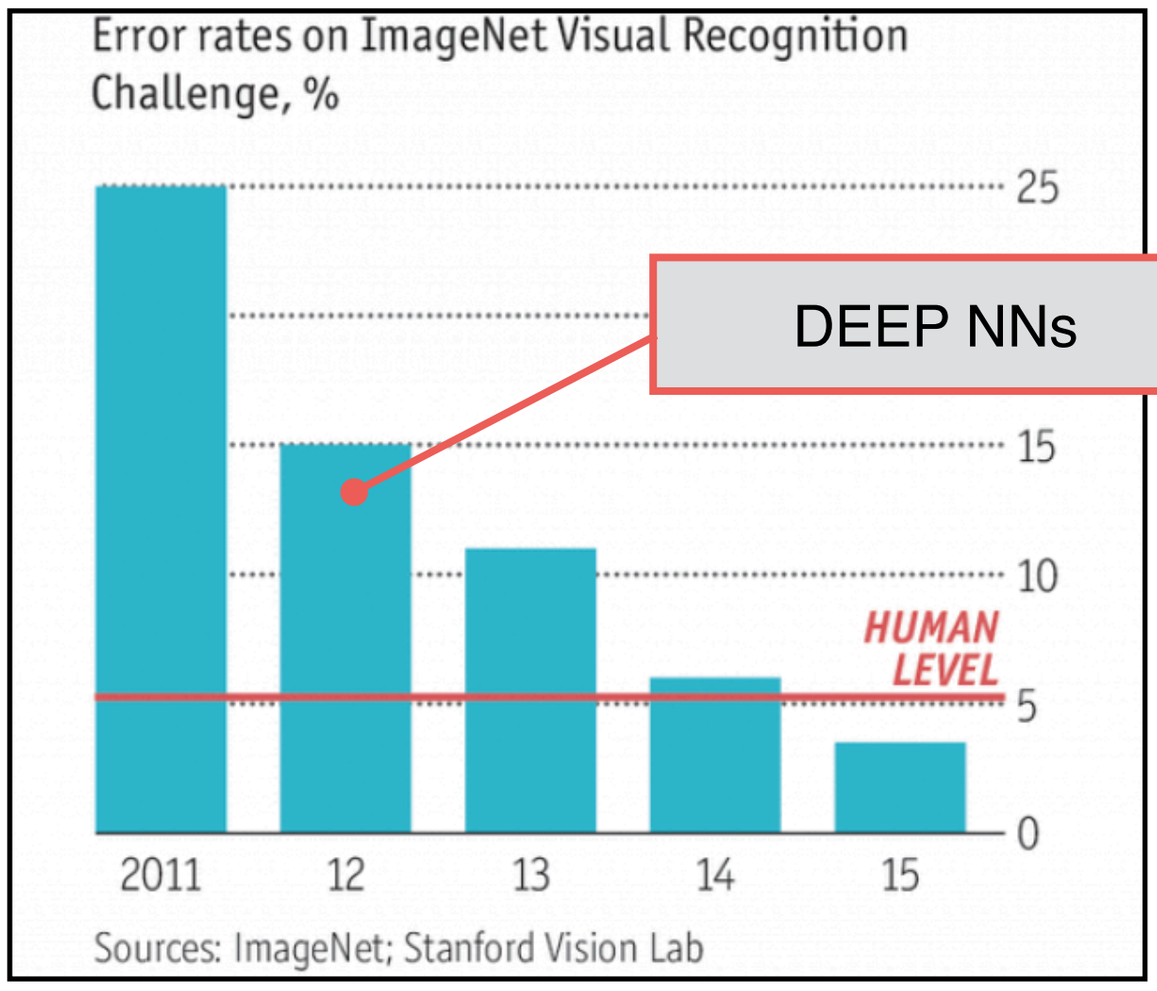




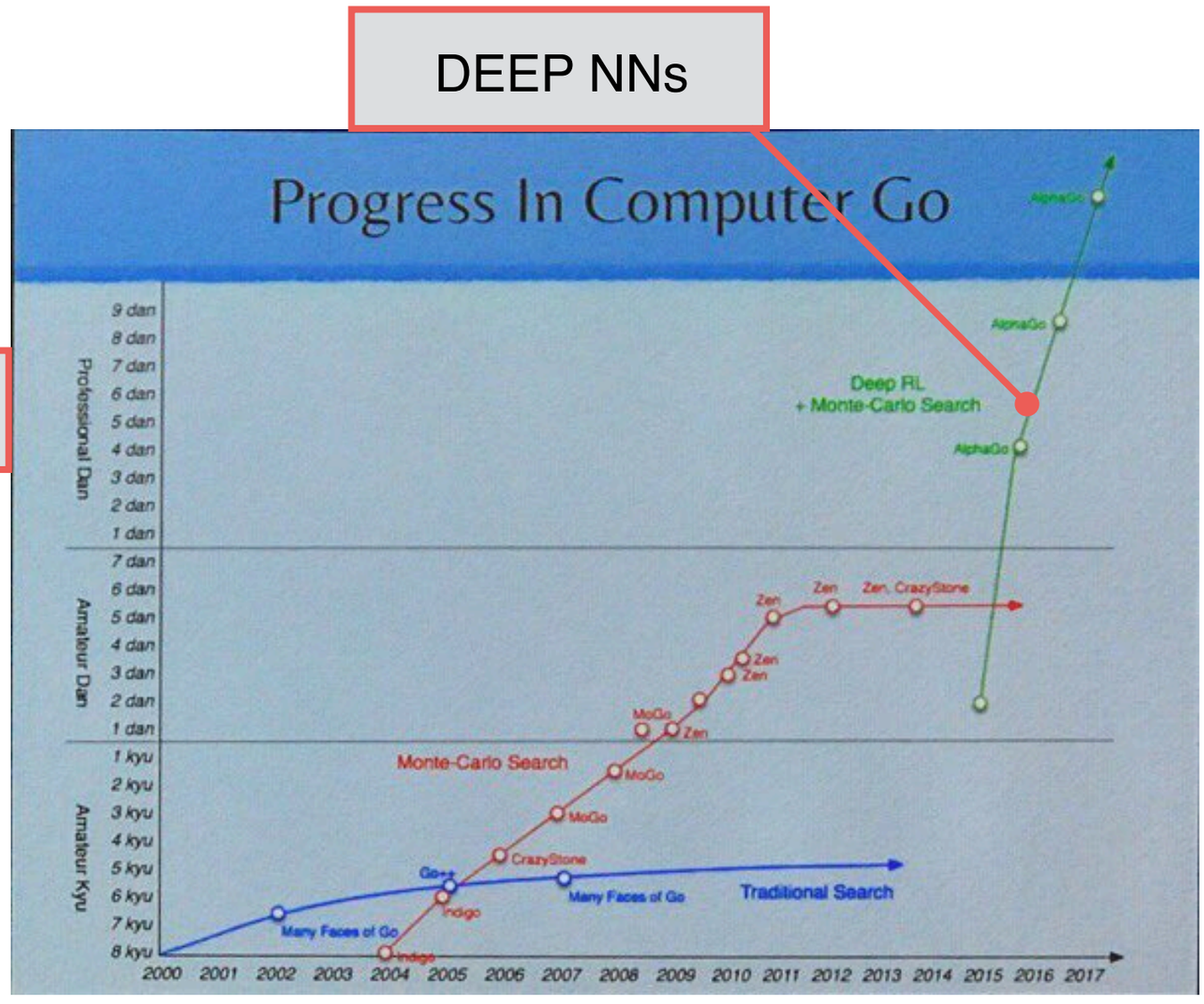
https://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

Deep Learning Results

- **Computer Vision**: Results on ImageNet object classification dataset.



- **Reinforcement Learning**: Results in playing Go.



https://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

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

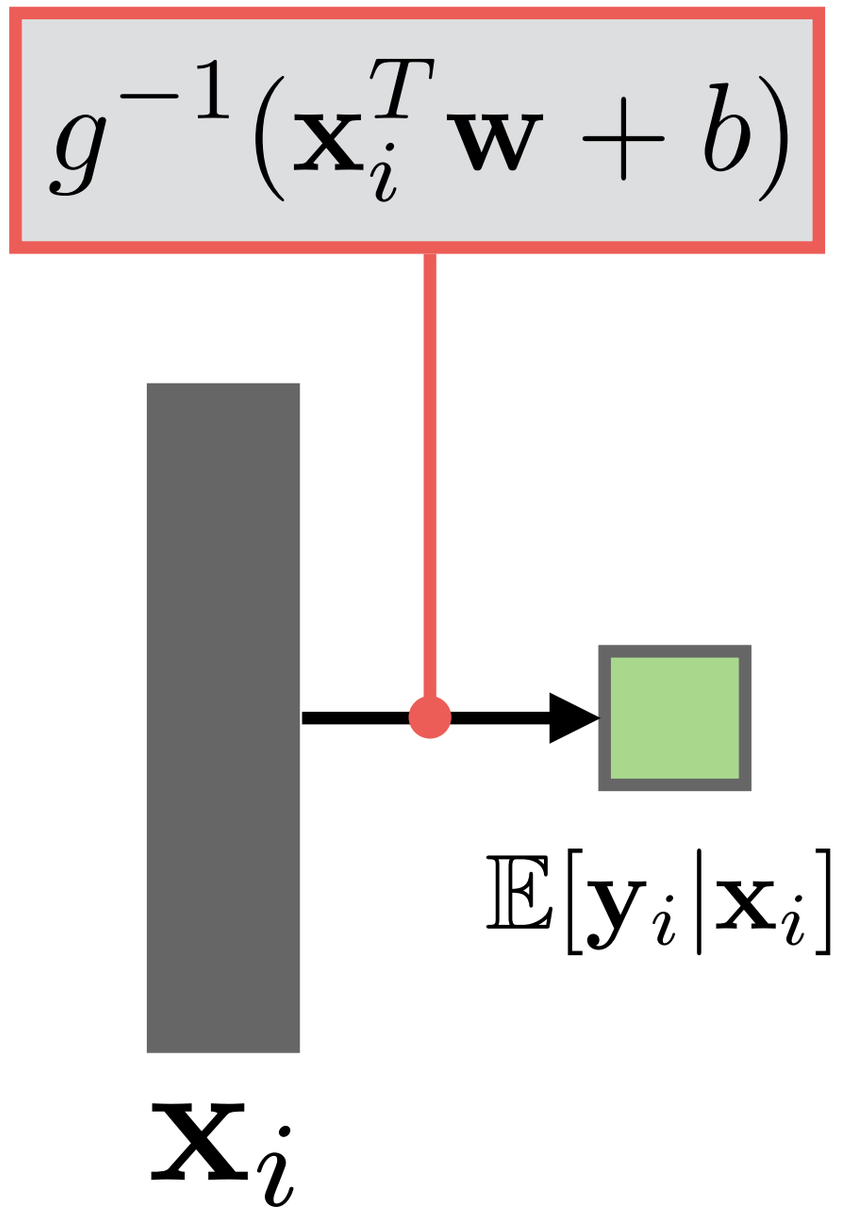
$$g^{-1}(\mathbf{x}_i^T \mathbf{w} + b) = \mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

Example: Logistic Regression

$$\mathbf{y}_i \in \{0, 1\} \qquad g^{-1}(z) = \frac{1}{1 + e^{-z}}$$

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:

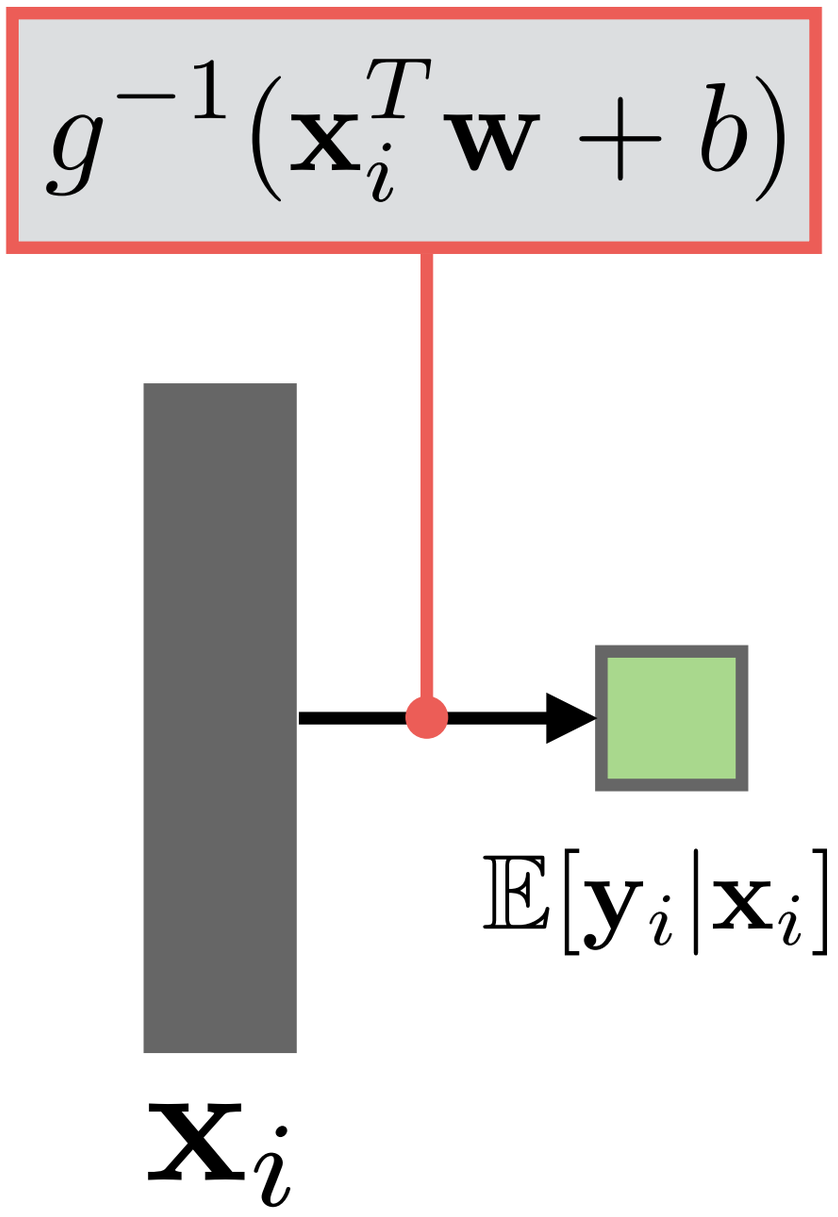
$$g_1^{-1}(\mathbf{x}_i^T \mathbf{w}_1 + b_1) = h_i$$

Define GLM on latent feature:

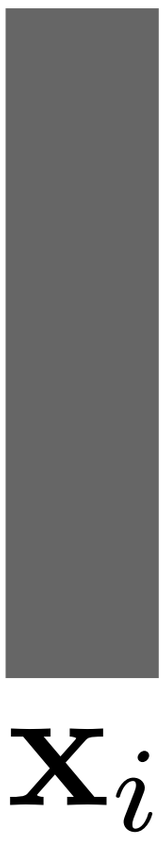
$$g_2^{-1}(h_i w_2 + b_2) = \mathbb{E}[\mathbf{y}_i | \mathbf{x}_i]$$

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

Recursive GLMs



Recursive GLMs

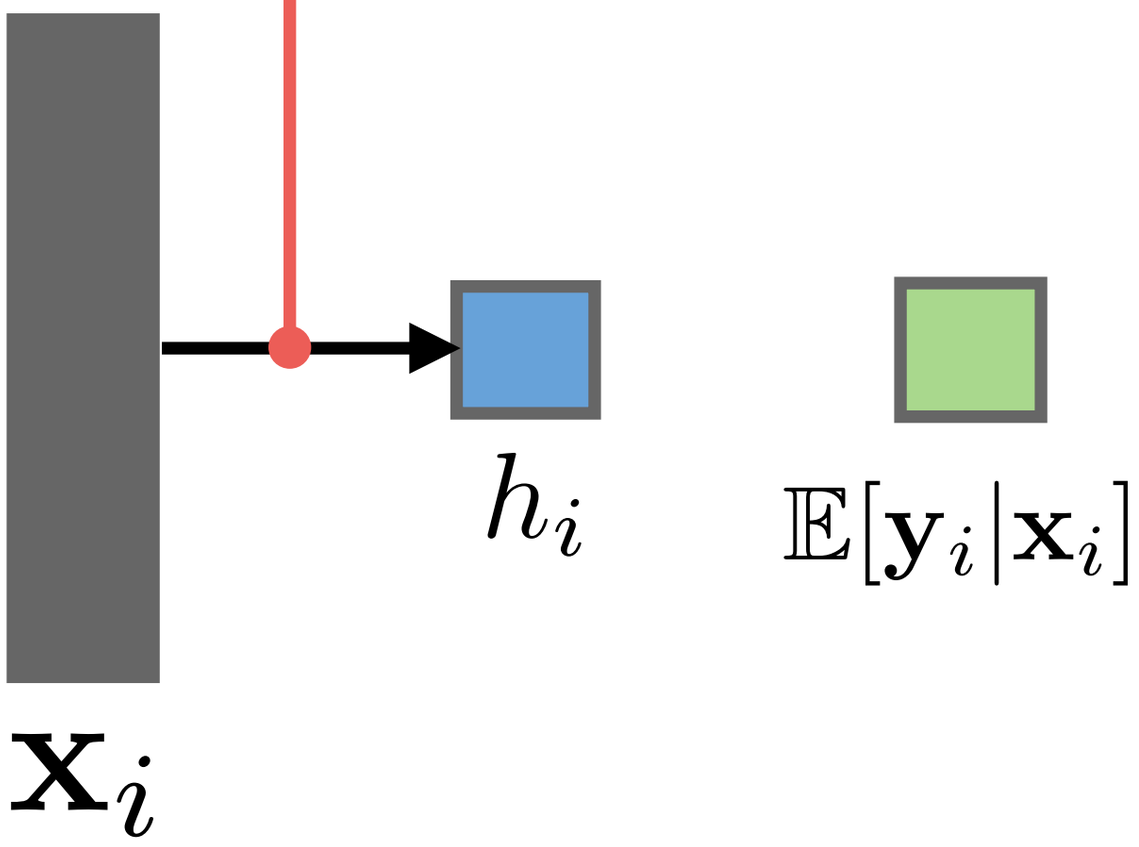


$\mathbf{x}_i$

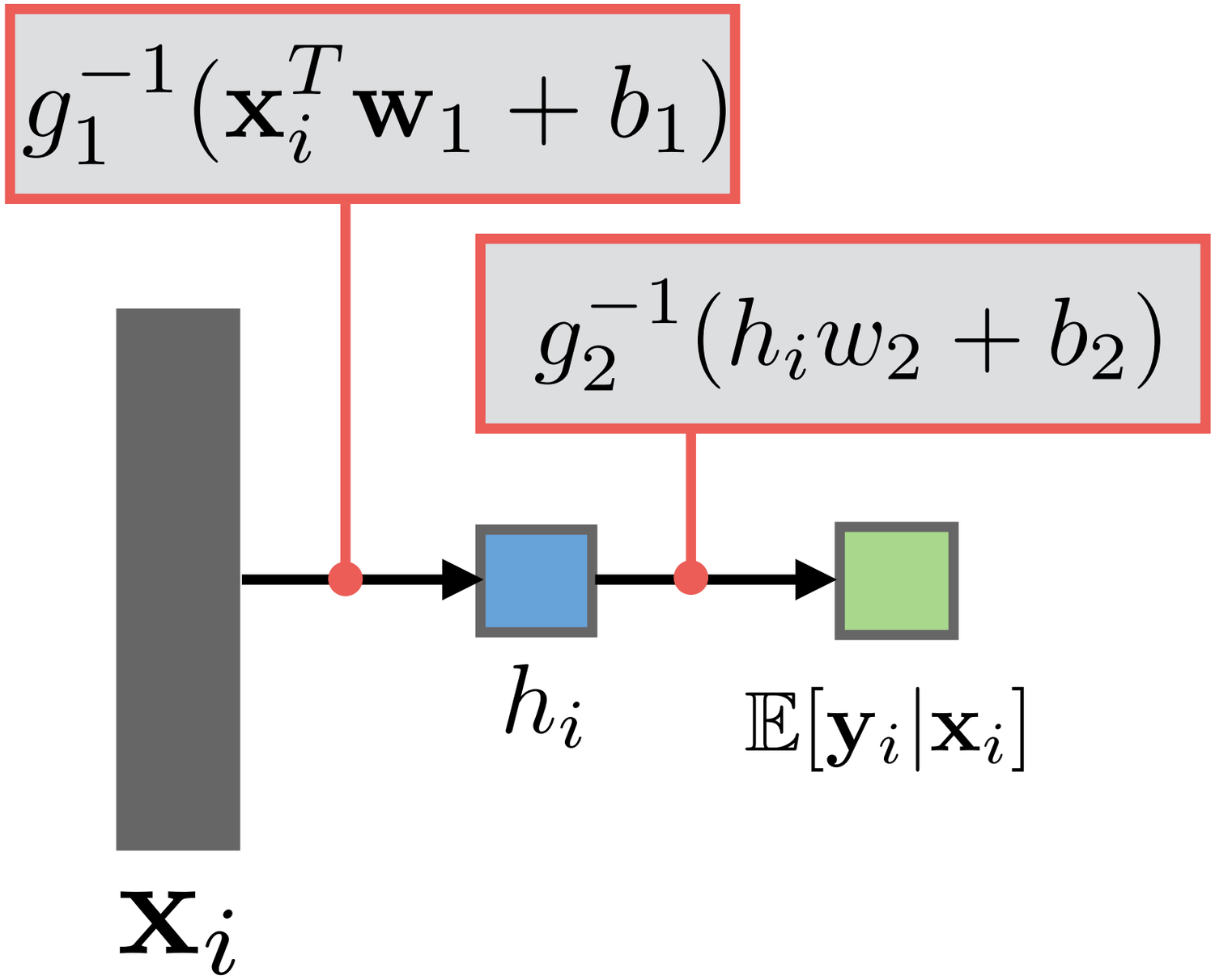
$\mathbb{E}[\mathbf{y}_i|\mathbf{x}_i]$

Recursive GLMs

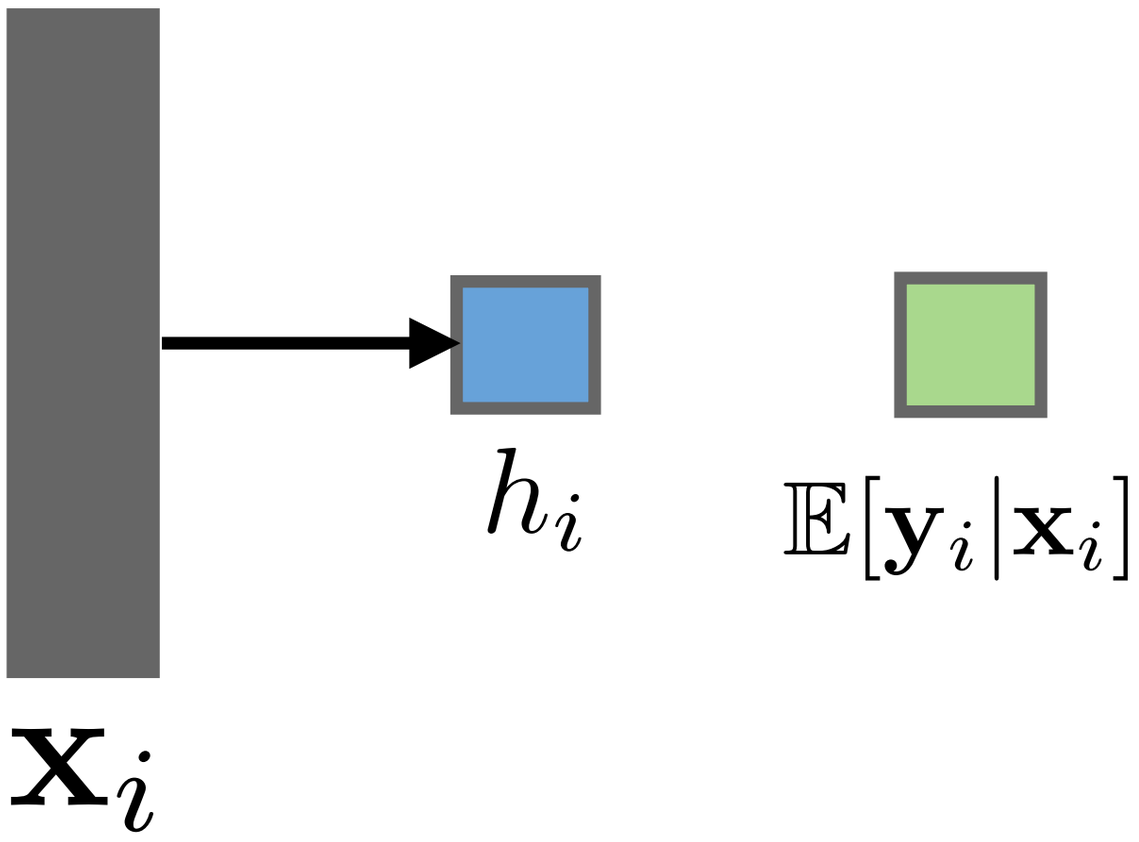
$$g_1^{-1}(\mathbf{x}_i^T \mathbf{w}_1 + b_1)$$



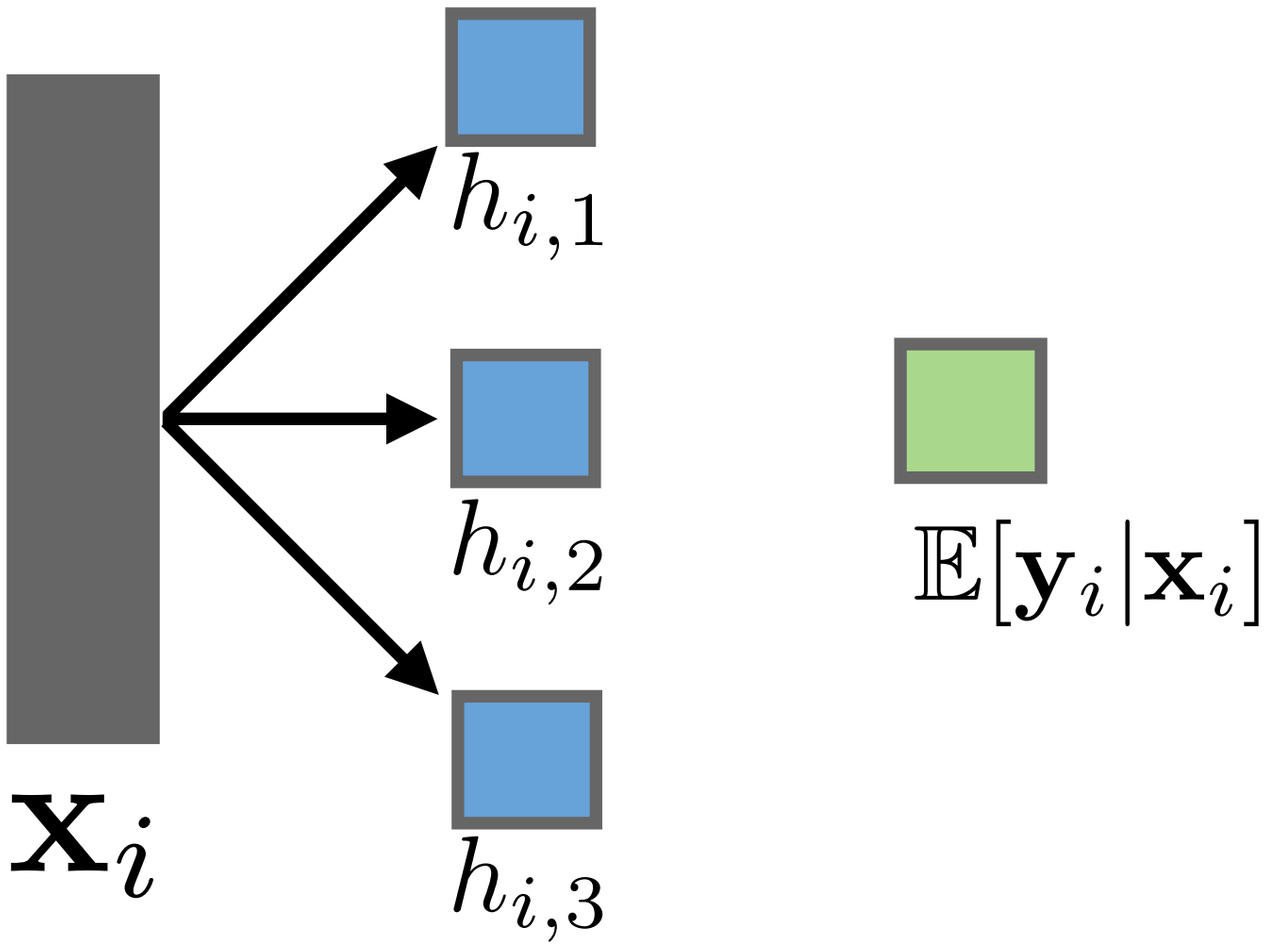
Recursive GLMs



Building Neural Nets from Recursive GLMs

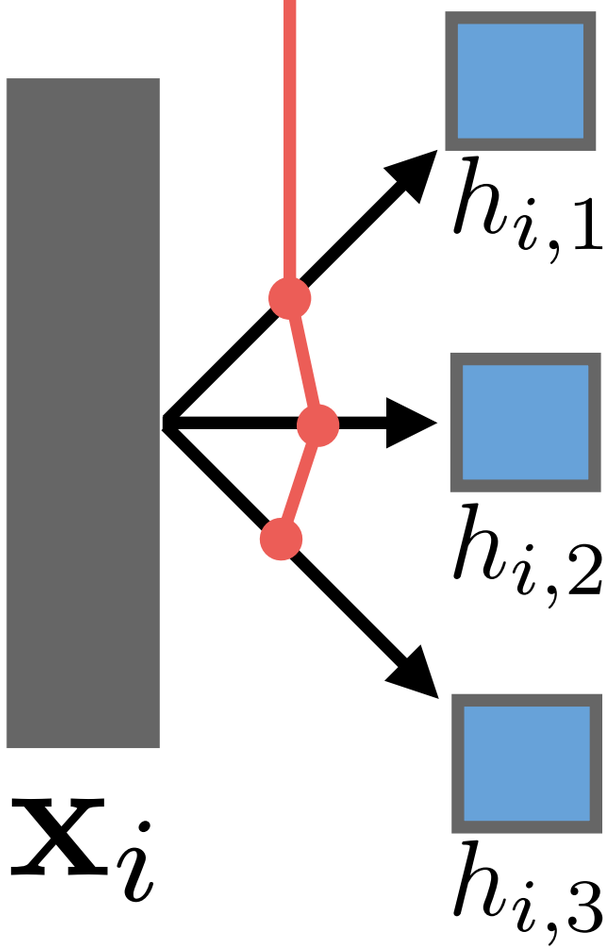


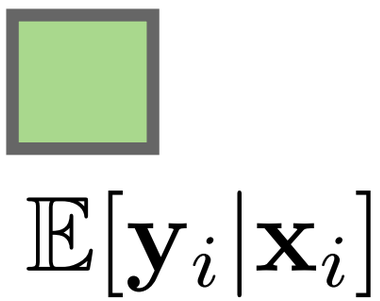
Building Neural Nets from Recursive GLMs



Building Neural Nets from Recursive GLMs

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

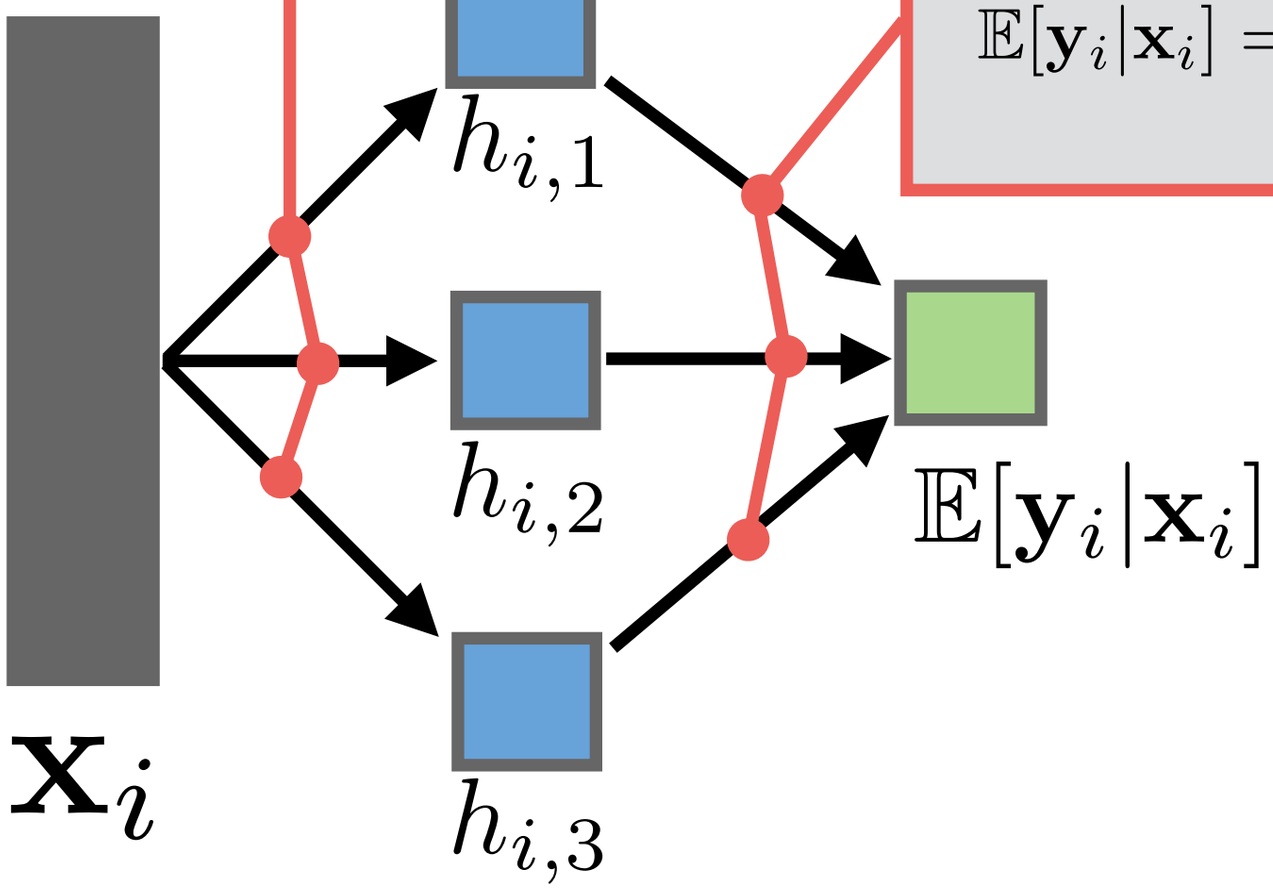




Building Neural Nets from Recursive GLMs

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

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

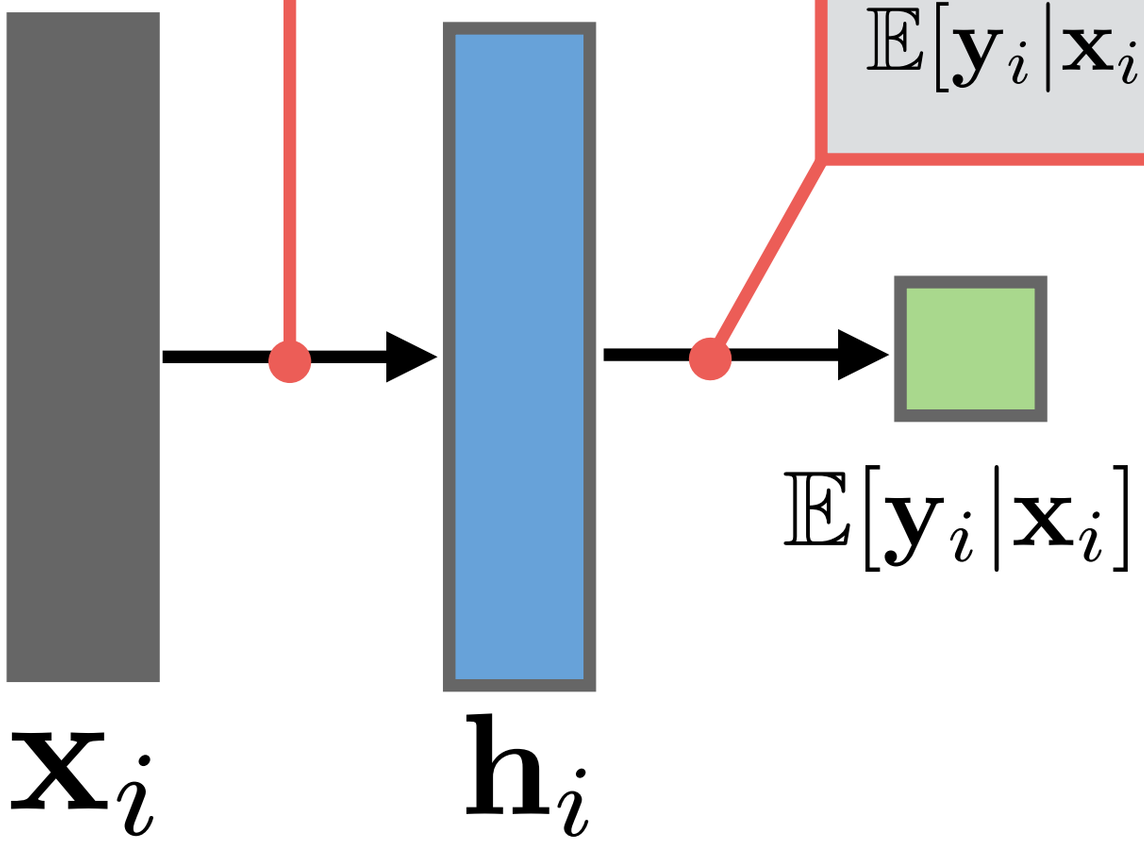


Building Neural Nets from Recursive GLMs

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

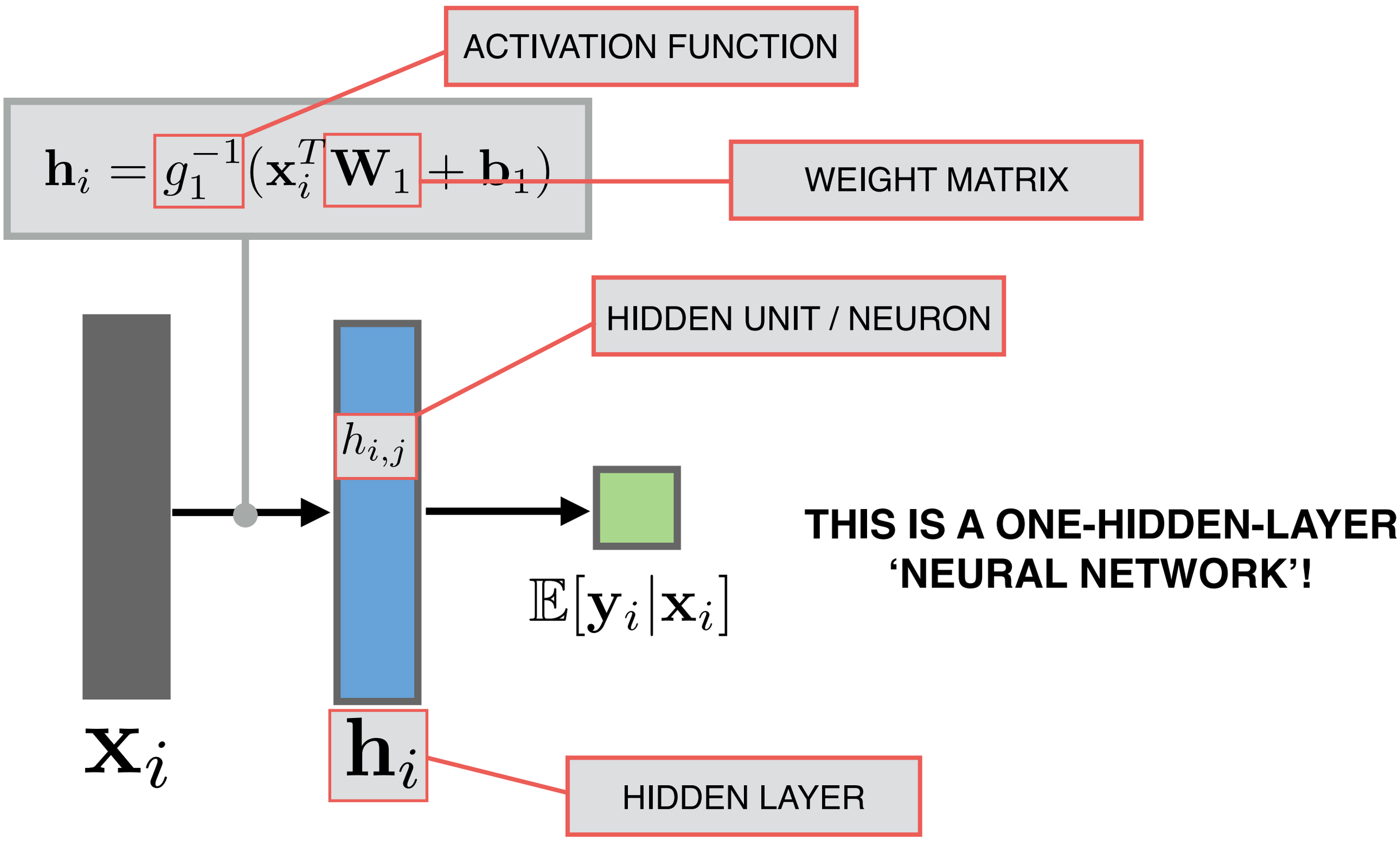
$$\mathbf{h}_i = g_1^{-1}(\mathbf{x}_i^T \mathbf{W}_1 + \mathbf{b}_1)$$

$$\mathbb{E}[\mathbf{y}_i|\mathbf{x}_i] = g_2^{-1}(\mathbf{h}_i \mathbf{w}_2 + b_2)$$



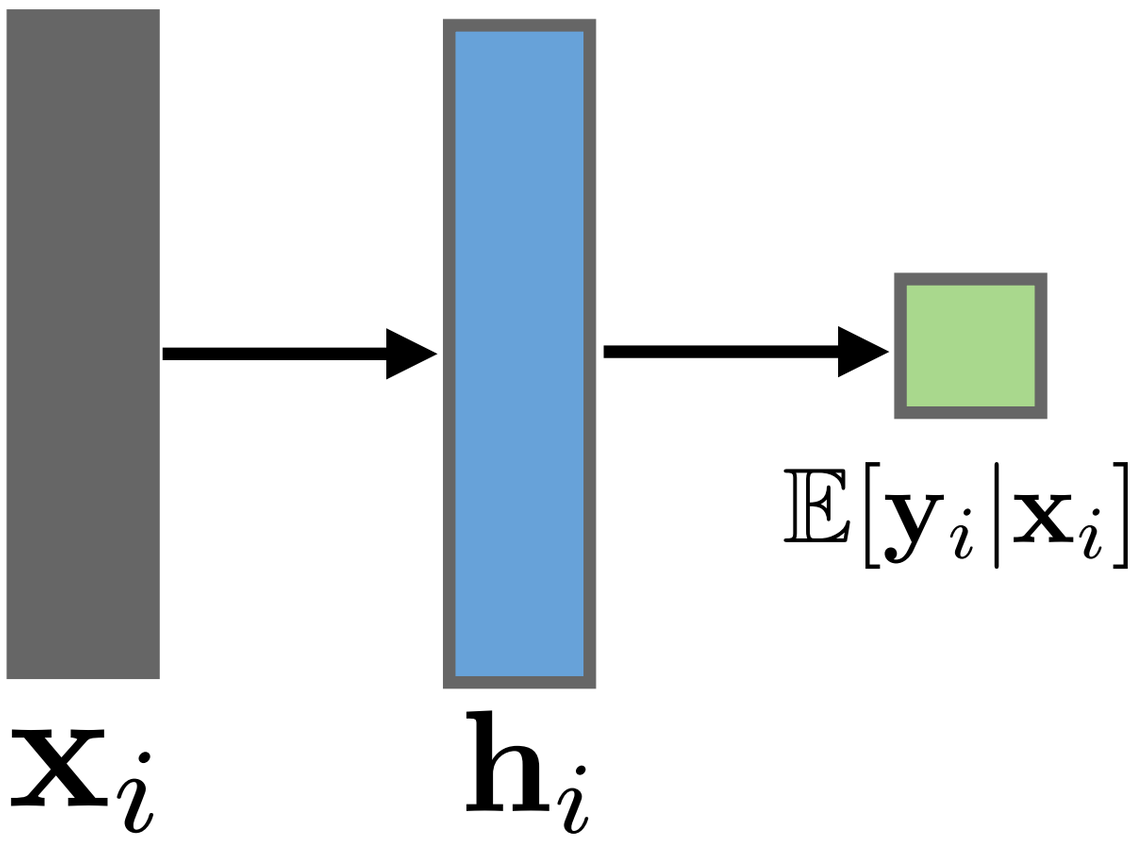
THIS IS A ONE-HIDDEN-LAYER ‘NEURAL NETWORK’!

Building Neural Nets from Recursive GLMs

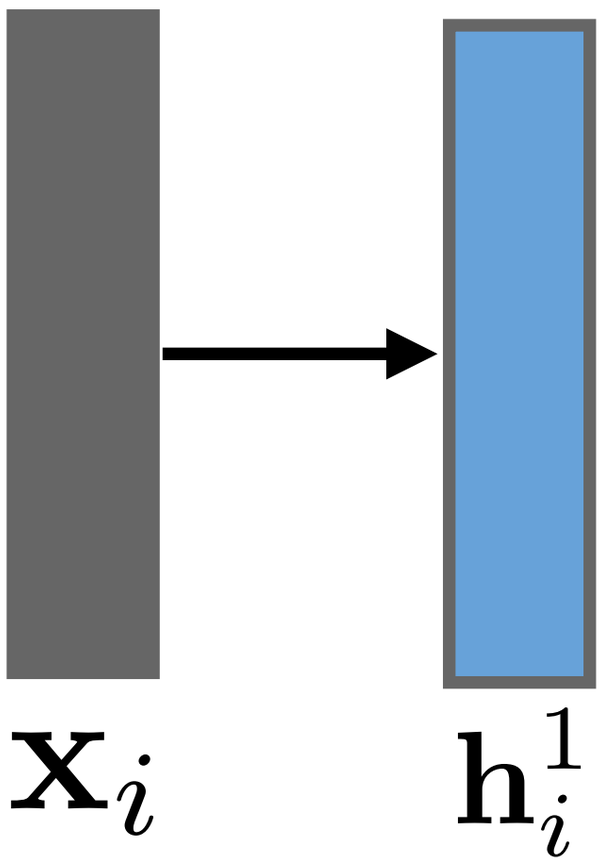


THIS IS A ONE-HIDDEN-LAYER ‘NEURAL NETWORK’!

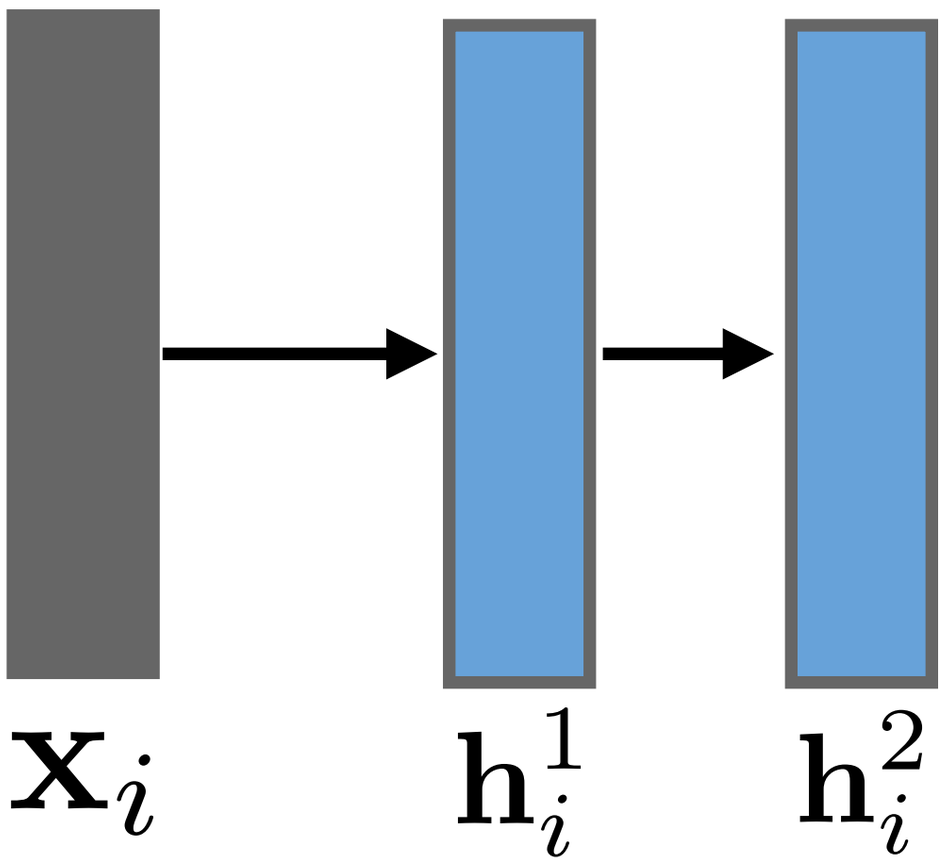
Deep Neural Networks



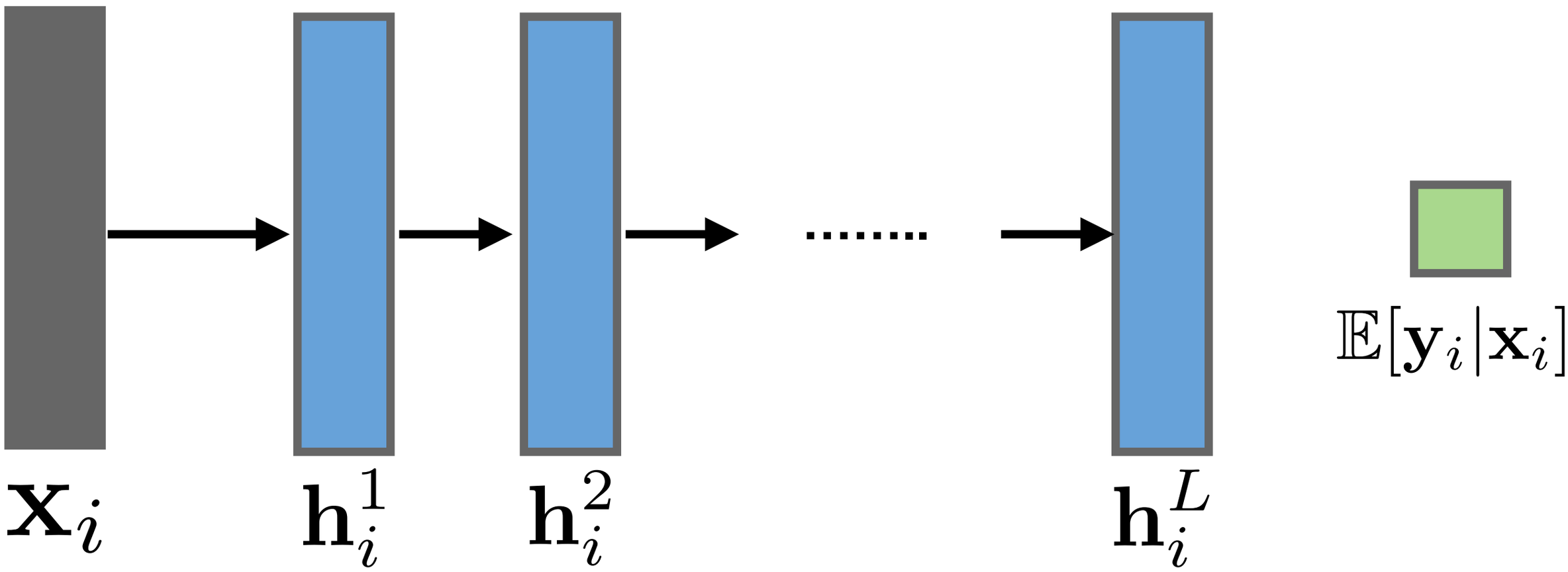
Deep Neural Networks



Deep Neural Networks



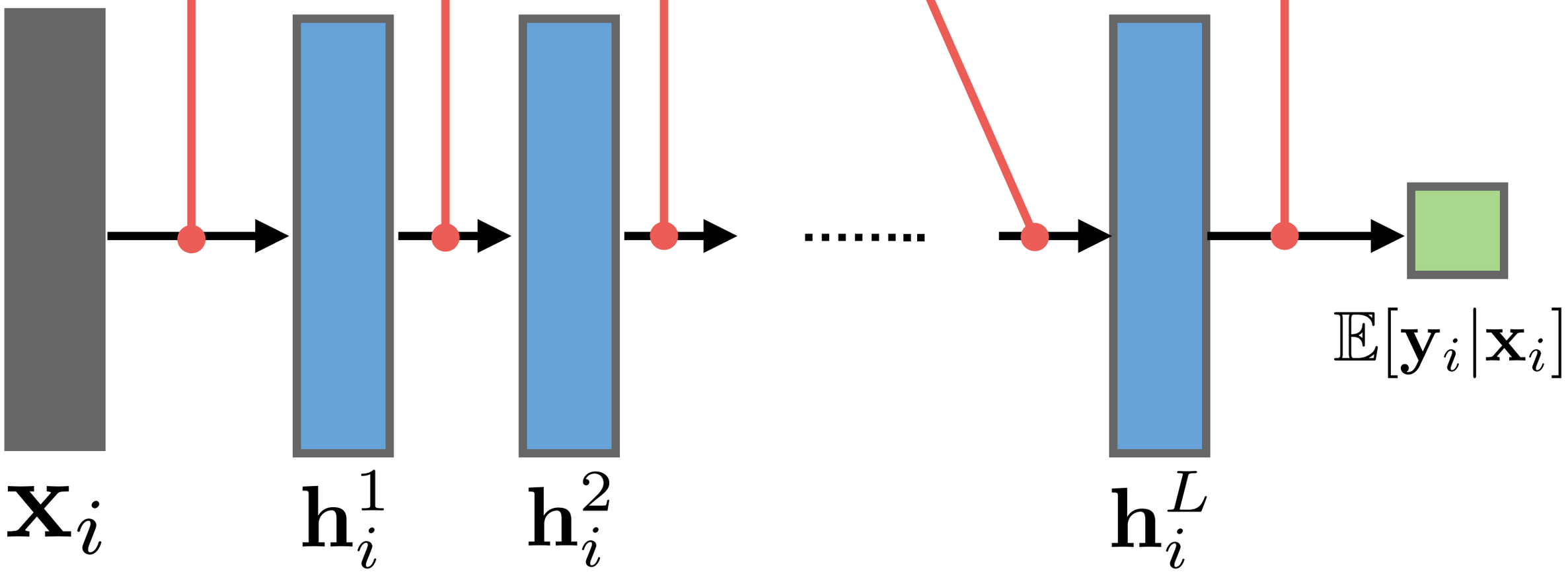
Deep Neural Networks



Deep Neural Networks

$$\mathbf{h}_i^l = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l), \quad \mathbf{h}_i^0 = \mathbf{x}_i$$

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



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

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

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

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

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 $K << N$

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.

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

NEW REPRESENTATION
OF FEATURES

Deep Neural Networks

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

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

NEW REPRESENTATION
OF FEATURES

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

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

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

SOLUTIONS:

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

SOLUTIONS:

- ☐ **More Data**: Many of the datasets being tested were too small, resulting in the NNs overfitting.
  - → **ImageNet Dataset**: 15 million images, 22,000 classes

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

SOLUTIONS:

- **More Data**: Many of the datasets being tested were too small, resulting in the NNs overfitting.
  - → **ImageNet Dataset**: 15 million images, 22,000 classes

- **Better Hardware**: And when the datasets were large, the hardware was not fast enough to make NN training practical.
  - → **Graphics Processing Units (GPUs)**: Can perform fast matrix multiplications (but with loss of precision).

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

SOLUTIONS:

- **More Data**: Many of the datasets being tested were too small, resulting in the NNs overfitting.
  - → **ImageNet Dataset**: 15 million images, 22,000 classes

- **Better Hardware**: And when the datasets were large, the hardware was not fast enough to make NN training practical.
  - → **Graphics Processing Units (GPUs)**: Can perform fast matrix multiplications (but with loss of precision).

- **Model Changes**: Small but crucial changes to the NN model architecture.

PROBLEM: Prior to 2000's, NNs with 4+ layers were hard to train. SVMs worked just as well as shallow NNs.

SOLUTIONS:

- **More Data**: Many of the datasets being tested were too small, resulting in the NNs overfitting.
  - → **ImageNet Dataset**: 15 million images, 22,000 classes

- **Better Hardware**: And when the datasets were large, the hardware was not fast enough to make NN training practical.
  - → **Graphics Processing Units (GPUs)**: Can perform fast matrix multiplications (but with loss of precision).

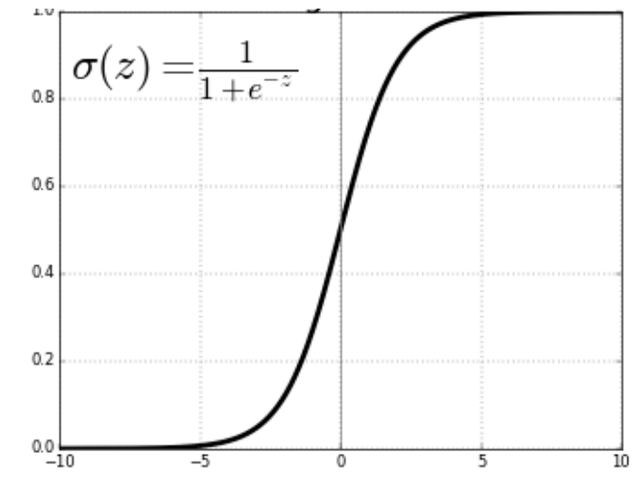
- **Model Changes**: Small but crucial changes to the NN model architecture.

- **Better Understanding of Gradient Ascent**: Gradient-based optimization is better in non-convex settings than previously thought.

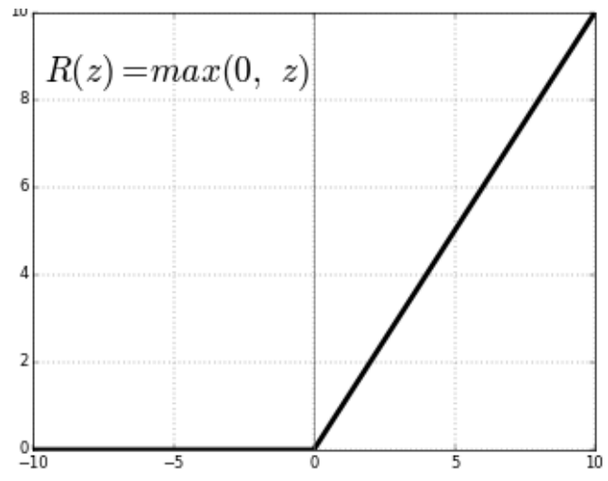
Deep Learning Breakthroughs: Model

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

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



VS



LOGISTIC FUNCTION

OLD

RECTIFIED LINEAR UNIT (ReLU)

NEW

Deep Learning Breakthroughs: Model

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

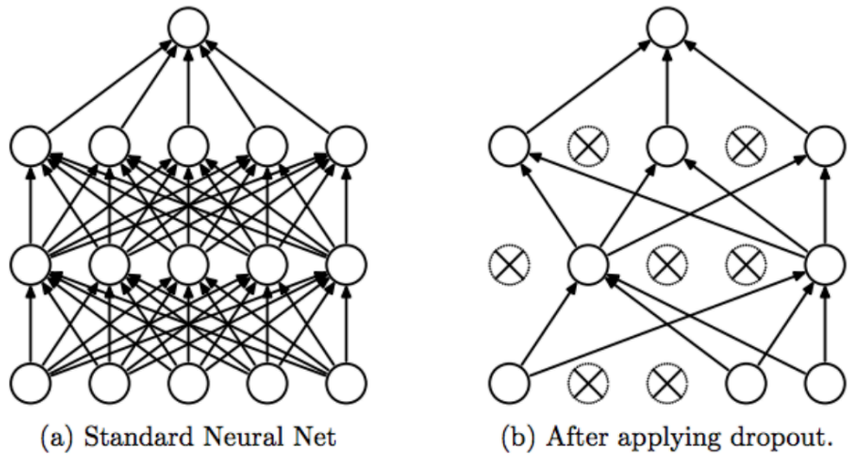
2. Better Regularization: Stochastic regularization strategies [Srivastava et al., 2014].

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

RIDGE PENALTY

OLD

VS



(a) Standard Neural Net (b) After applying dropout.

DROPOUT

NEW

Deep Learning Breakthroughs: Model

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

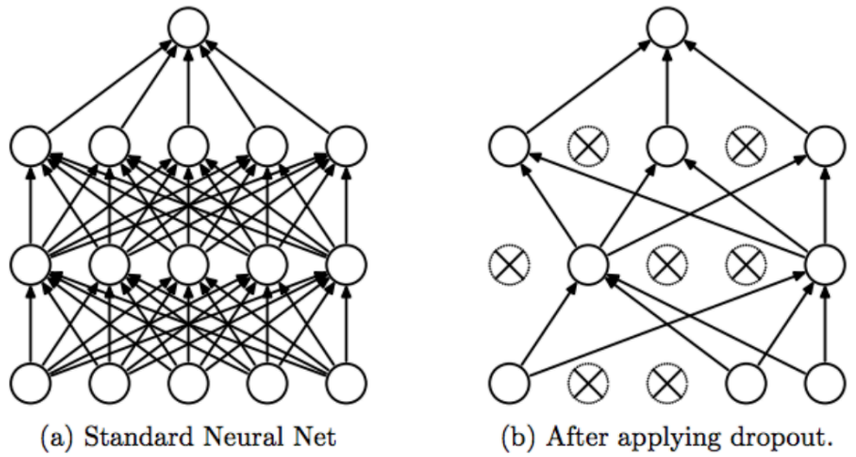
2. Better Regularization: Stochastic regularization strategies [Srivastava et al., 2014].

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

RIDGE PENALTY

OLD

VS



(a) Standard Neural Net (b) After applying dropout.

DROPOUT

NEW

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

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

Deep Learning Breakthroughs: Model

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

3. **Skip Connections**: Identity connections to previous layer [He et al., 2015].

$$\mathbf{h}_i^l = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l) \quad \textbf{vs} \quad \mathbf{h}_i^l = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l) + \mathbf{h}_i^{l-1}$$

NONLINEAR TRANSFORM

SKIP CONNECTION

OLD

NEW

Deep Learning Breakthroughs: Model

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

3. **Skip Connections**: Identity connections to previous layer [He et al., 2015].

$$\mathbf{h}_i^l = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l) \quad \textbf{vs} \quad \mathbf{h}_i^l = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l) + \mathbf{h}_i^{l-1}$$

NONLINEAR TRANSFORM

SKIP CONNECTION

OLD

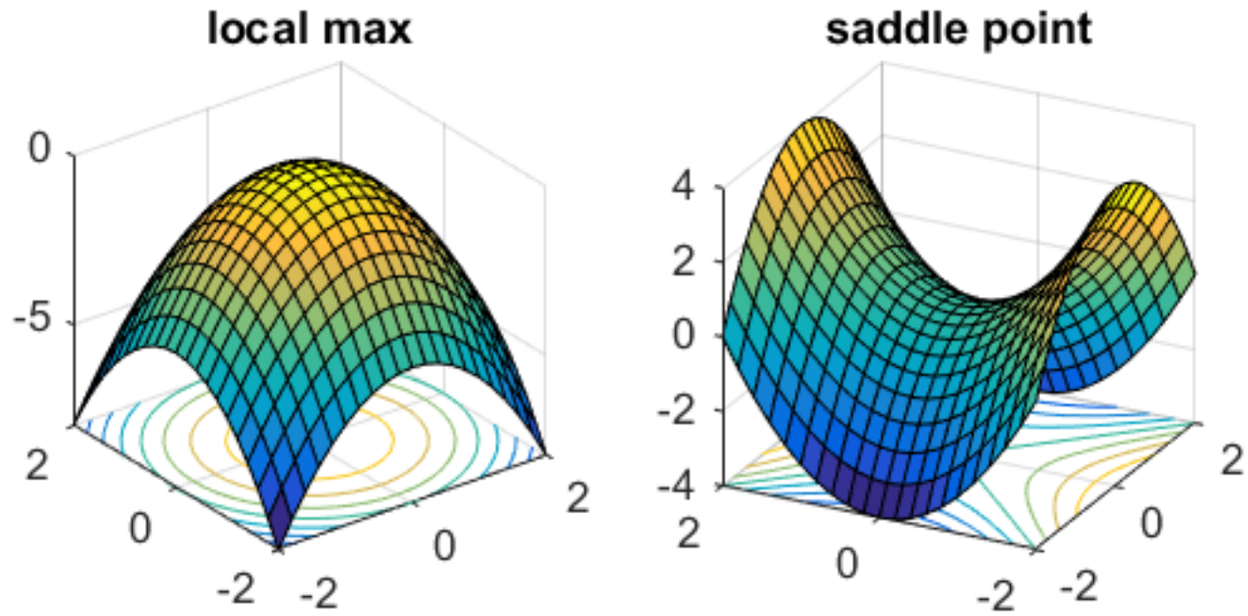
NEW

Can think of skip-connections as modeling the residual transformation:

$$\mathbf{h}_i^l - \mathbf{h}_i^{l-1} = f_l(\mathbf{h}_i^{l-1}\mathbf{W}_l + \mathbf{b}_l)$$

Deep Learning Breakthroughs: Optimization

Previously, optimization of deep NNs was thought to be hopelessly non-convex. However, many of the critical points are not maxima but rather *saddle points.* [Pascanu et al., 2014]

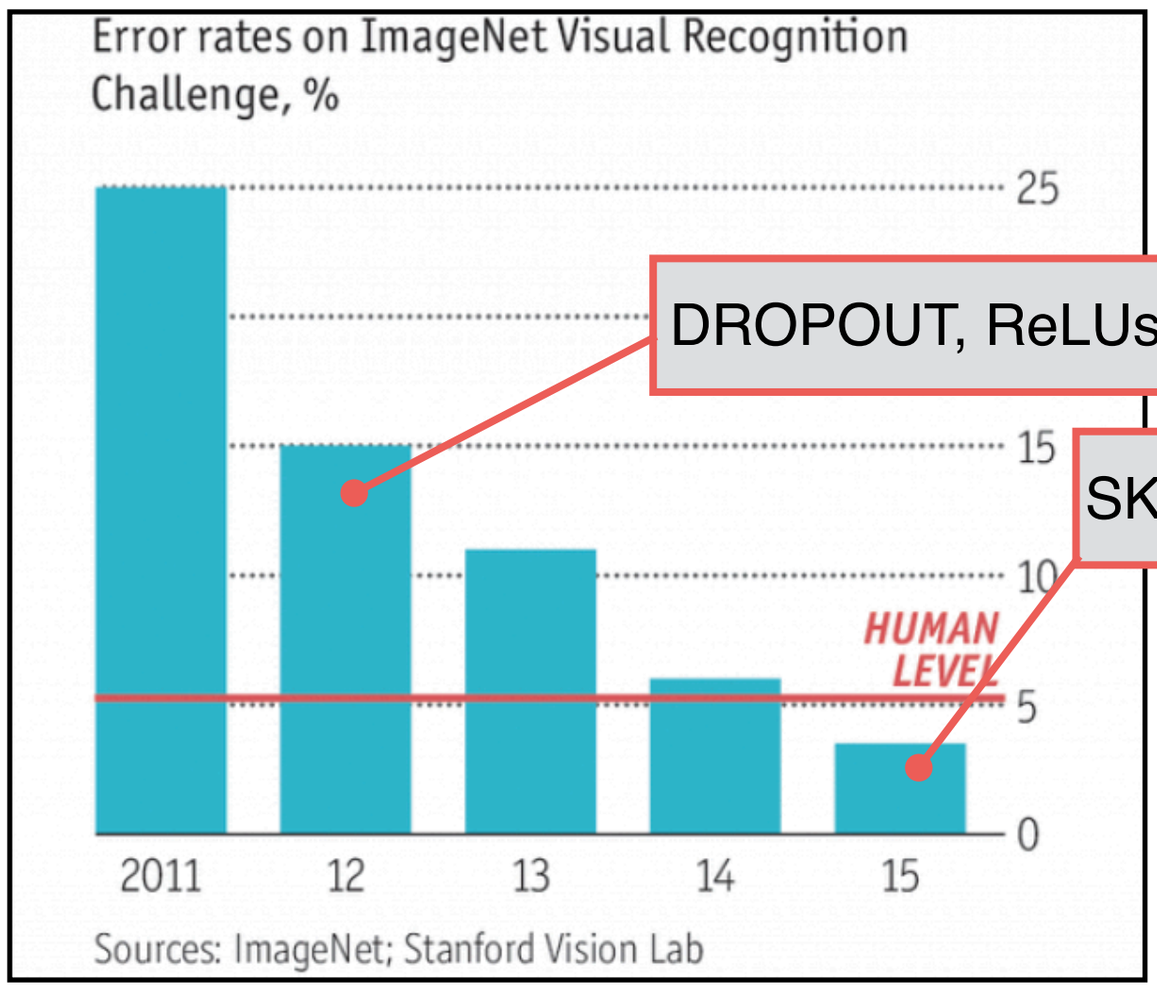


http://www.offconvex.org/2016/03/22/saddlepoints/

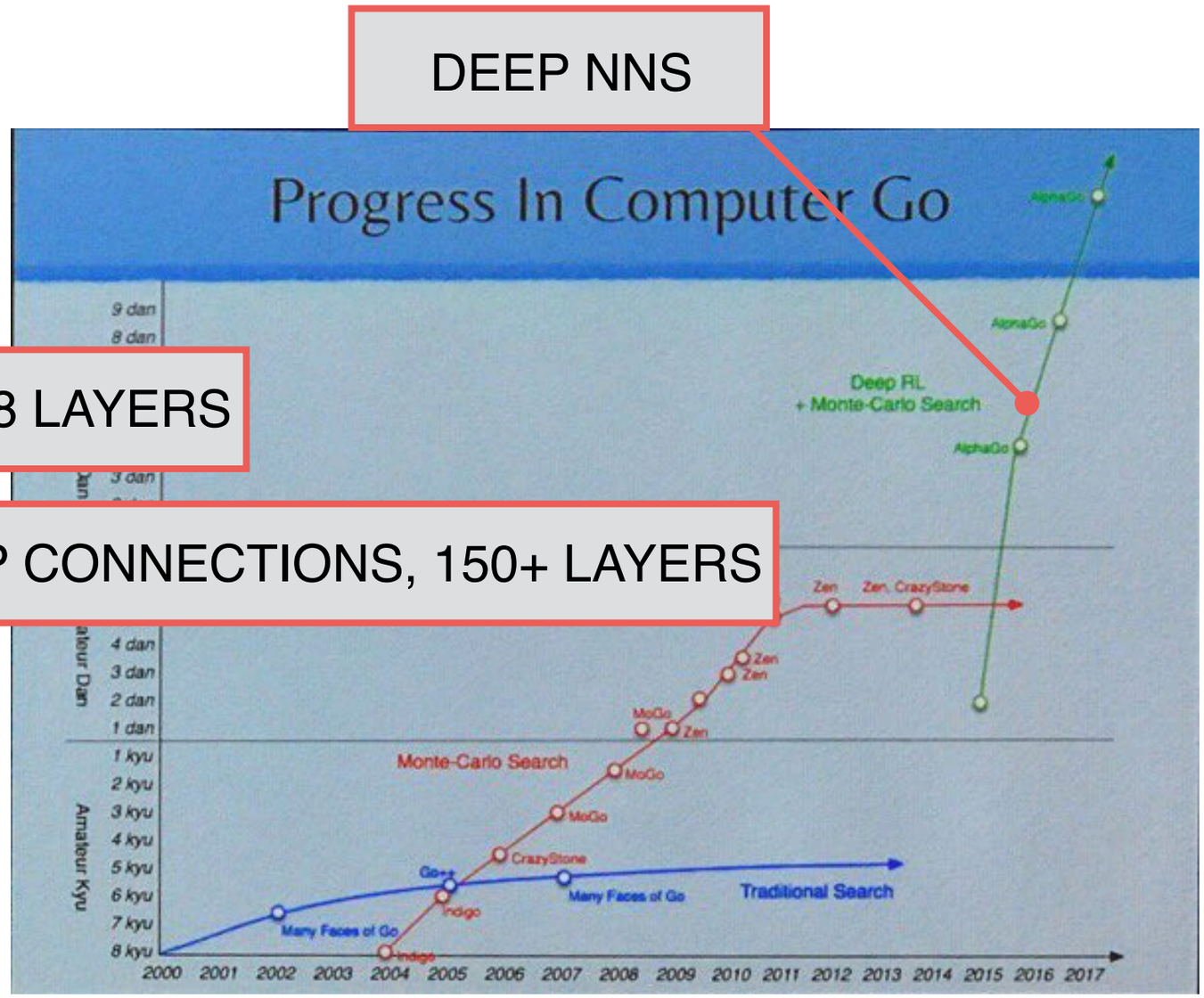
The intuition is that it's hard to build a high-dimensional max / min because the surface must be going in the same direction in all dimensions.

Deep Learning Results

- **Computer Vision**: Results on ImageNet object classification dataset.



- **Reinforcement Learning**: Results in playing Go.



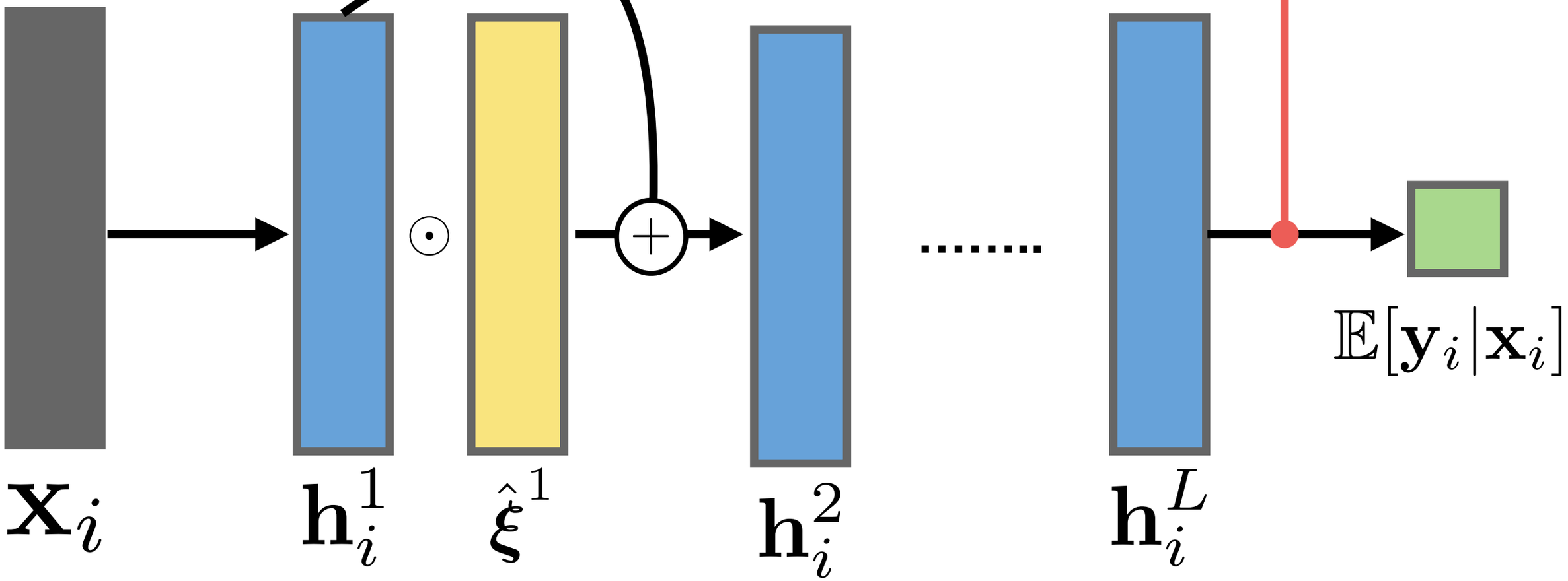
https://www.economist.com/news/special-report/21700756-artificial-intelligence-boom-based-old-idea-modern-twist-not

Modern Deep Neural Networks

ReLU

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

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



$\hat{\boldsymbol{\xi}}^1 \sim \text{Bernoulli}(p)$

Historical Perspective

DEEP LEARNING

(11+ YEARS)

17 YEAR GAP

EARLY WORK ON NNs

(32 YEARS)

2015 — A CNN by team from Microsoft beats the human benchmark (5% error rate) by bringing down the error rate to 3% in the ImageNet Challenge

2012 — A CNN designed by a team at the University of Toronto wins the ImageNet Challenge bringing dow the error rate to 16% (compared to 25% 2011)

2009 — A group around Andrew Ng introduce **Graphics Processing Units (GPUs)** for Deep Learning making them applicable on a large scale

2007 — Fei Fei Li and colleagues at Princeton University start to collect a large database of annotated images, the **ImageNet**

2006 — Hinton summarizes ideas of **multilayer neural networks** and training them to generate sensory data rather than to classify it

1989 — A group around Yann LeCun successfully applies a back-propagation algorithm to a multi-layer neural network, **recognizing handwritten ZIP codes**

1986 — Rumelhart, Hinton, and Williams introduce **backpropagation** as a learning procedure for "networks of neuron-like units"

1957 — Frank Rosenblatt develops the **Perceptron**, an early neural network enabling pattern recognition based on a two-layer learning network

https://micro-dimensions.com/blog/2017/2/13/whats-all-the-fuss-about-deep-learning-and-how-will-it-impact-pathology

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:

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

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

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]…

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

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]…

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

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

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]…

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

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

And therefore…

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

Infinitely Wide Neural Networks as GPs

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

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

Infinitely Wide Neural Networks as GPs

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

$$w^l_{m,n} \sim \mathrm{N}(0, \sigma^2/H)$$

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

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

Infinitely Wide Neural Networks as GPs

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

$$w^l_{m,n} \sim \mathrm{N}(0, \sigma^2/H)$$

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

ACTIVATION FUNCTION

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

Infinitely Wide Neural Networks as GPs

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

$$w^l_{m,n} \sim \mathrm{N}(0, \sigma^2/H)$$

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

ACTIVATION FUNCTION

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

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

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

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

So why use neural networks when we can just use GPs?

So why use neural networks when we can just use GPs?

“Have we thrown the baby out with the bath water?”

~ David MacKay on GPs (1998)

Disconnecting GPs and Neural Nets

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]

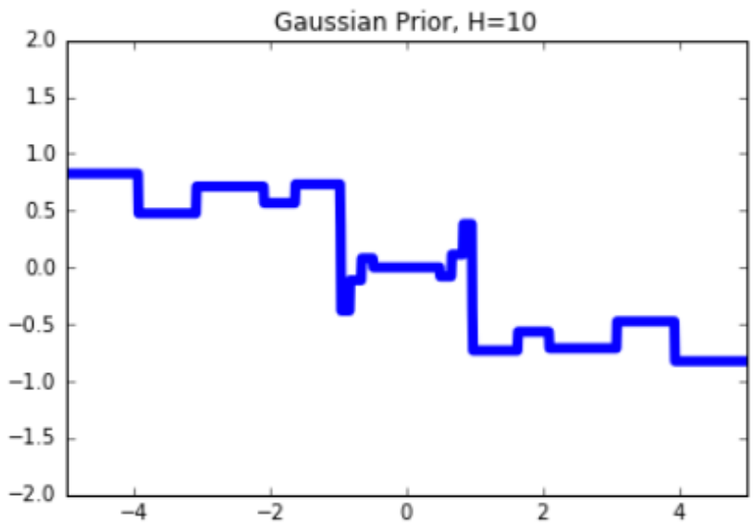
Function draws from a NN with H hidden units and a Gaussian prior:

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:

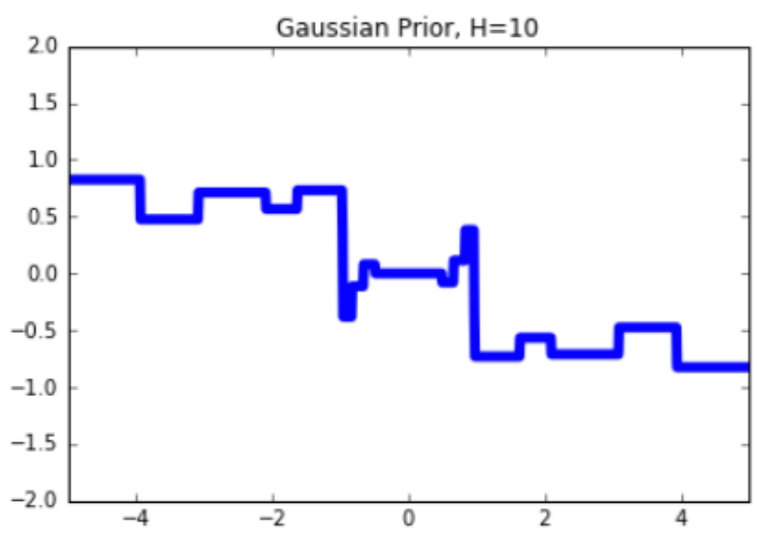


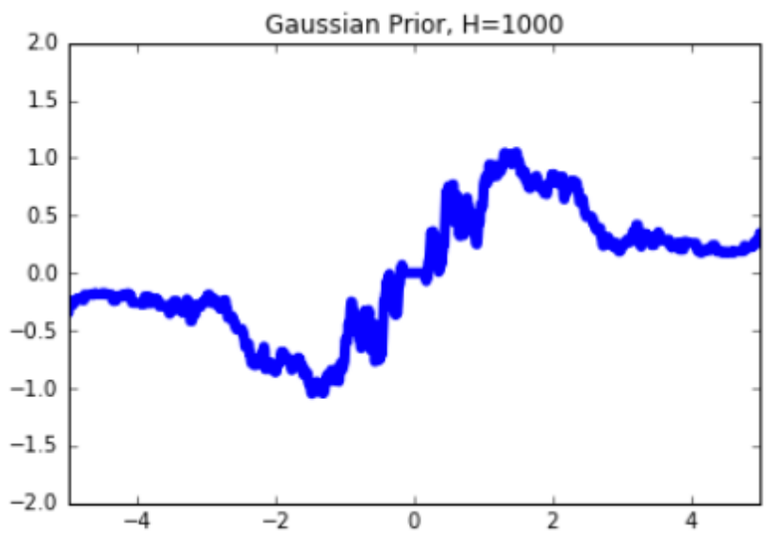
GPs Don't Learn Latent Representations

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

"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:



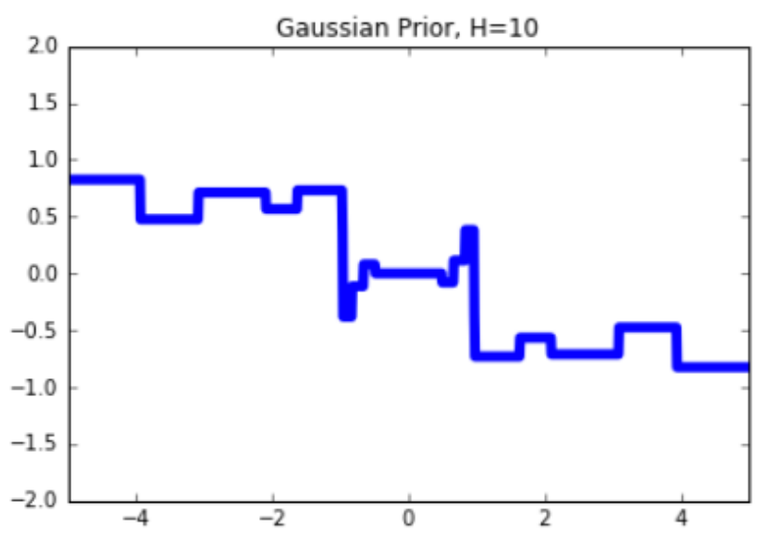


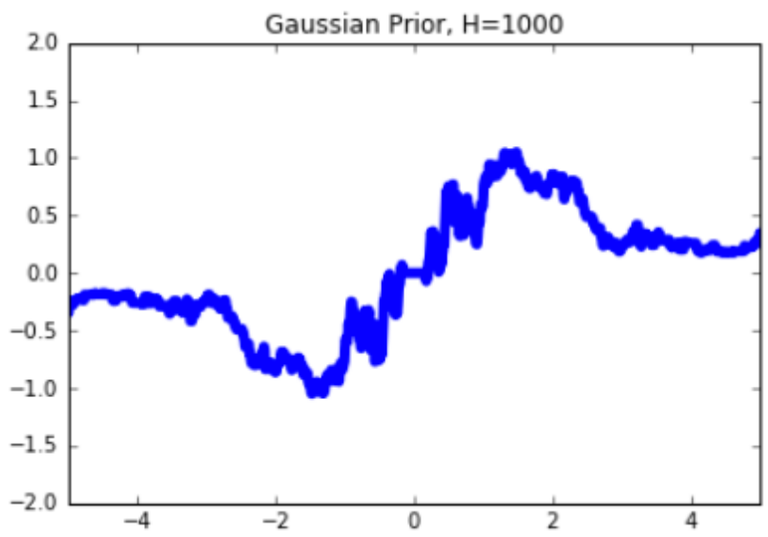
GPs Don't Learn Latent Representations

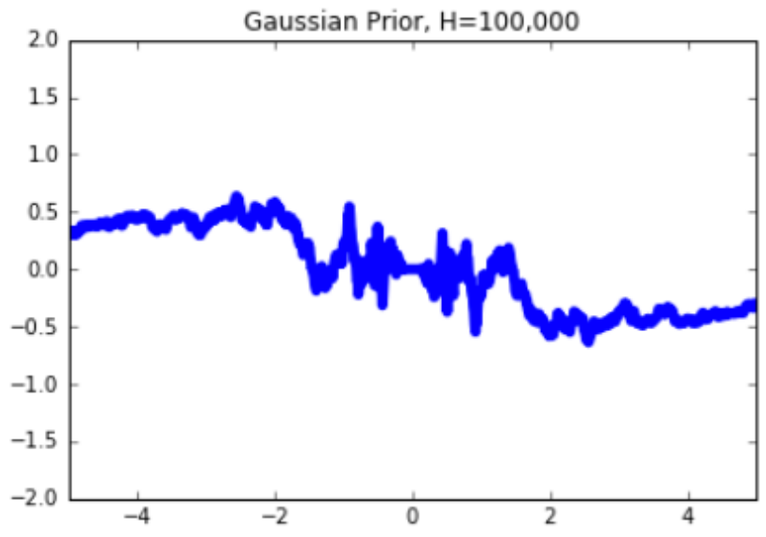
$$w_j^o \sim N\left(0, \frac{\sigma^2}{H}\right)$$

"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:

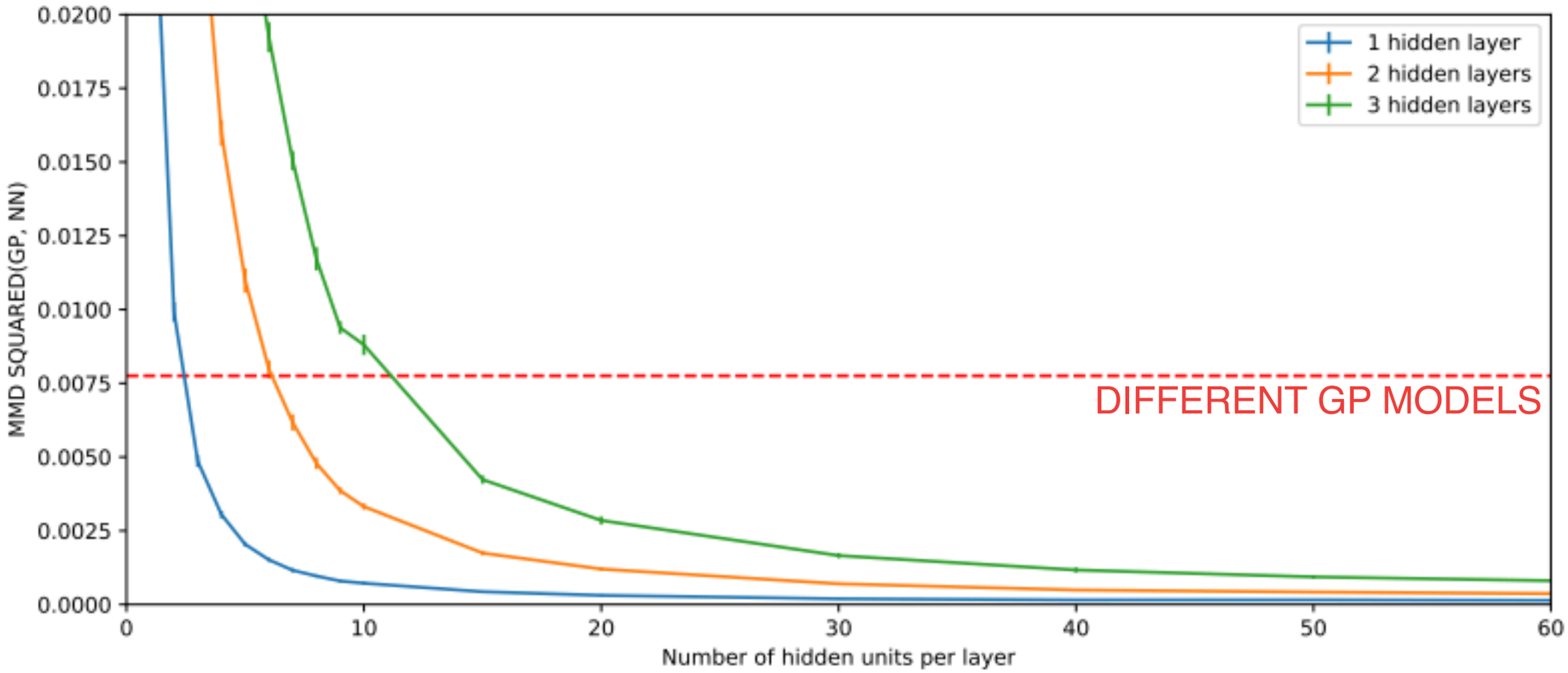






Bayesian NNs Behave Like GPs

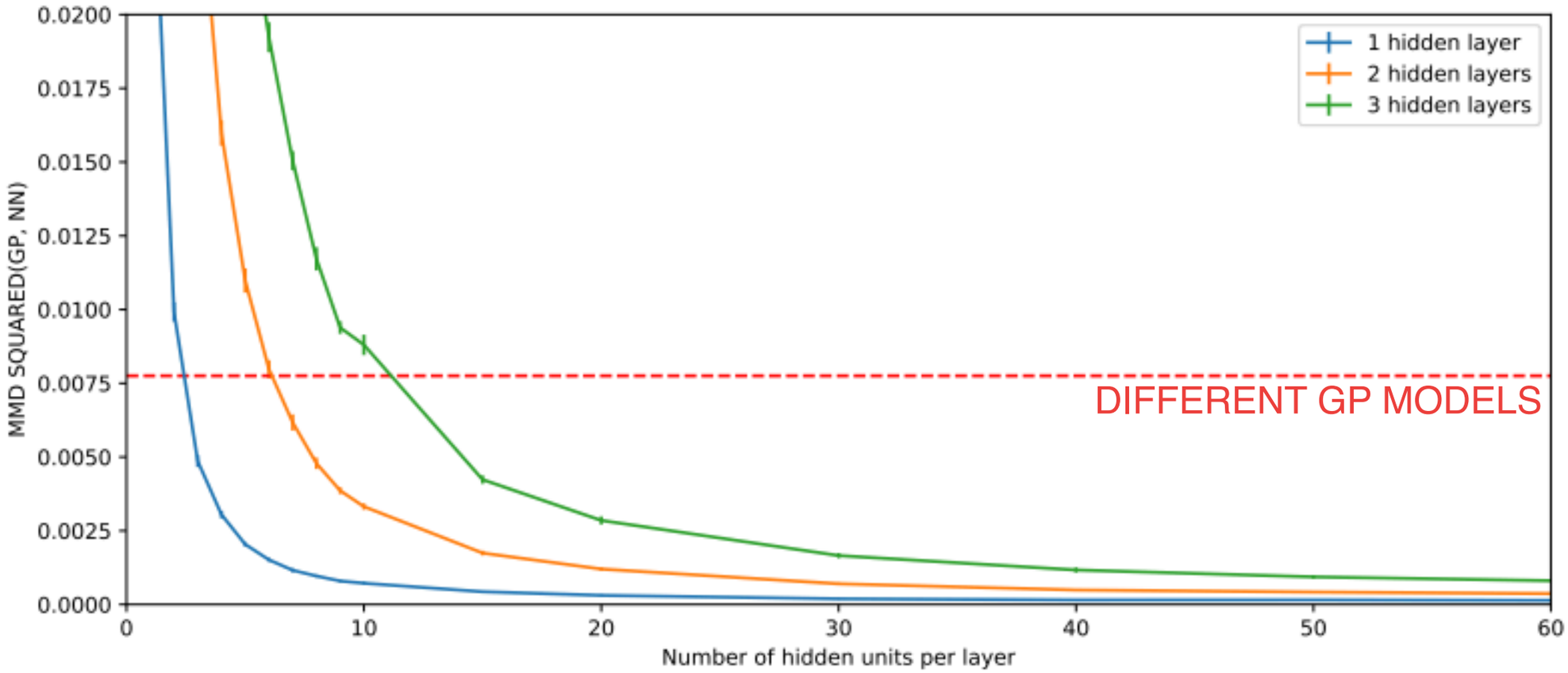
Empirical Investigation of GP behavior in Bayesian NNs:

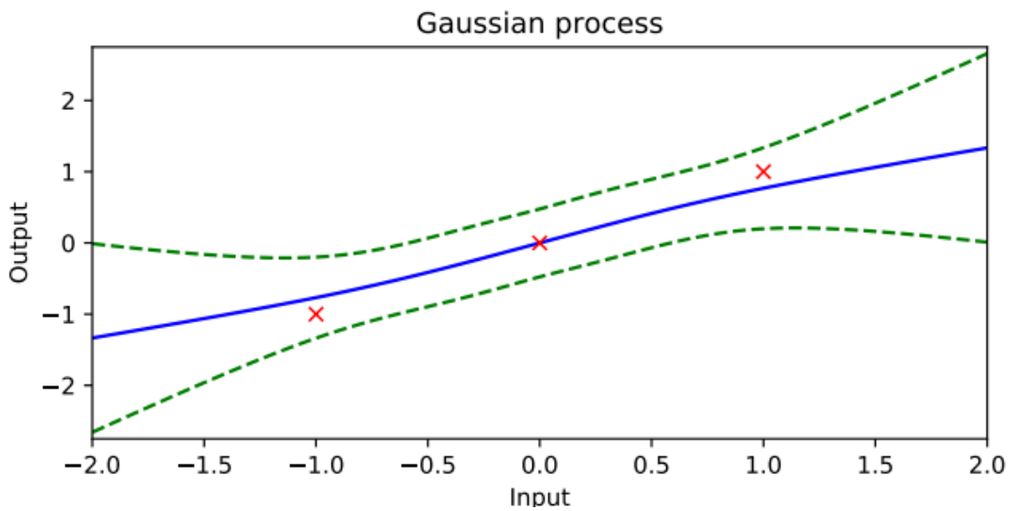


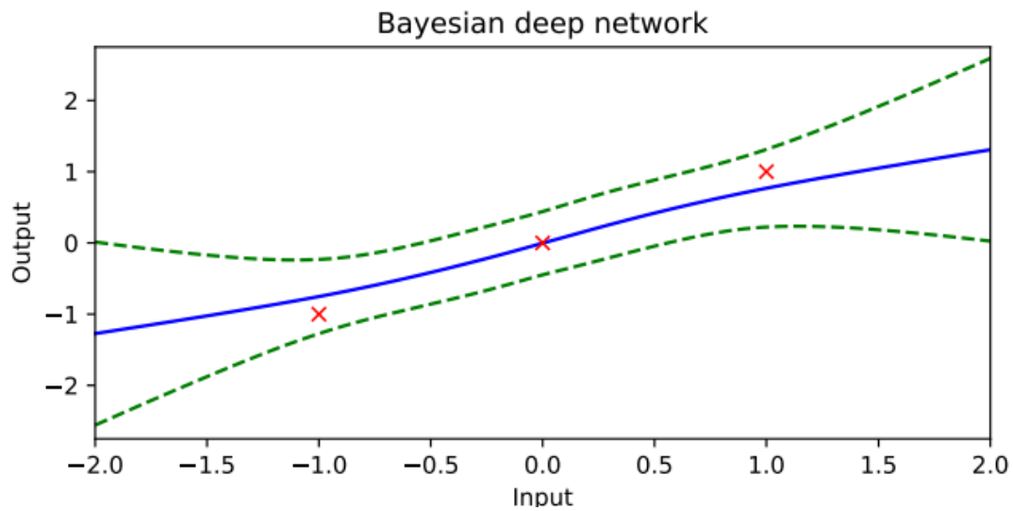
[Matthews et al., 2018]

Bayesian NNs Behave Like GPs

Empirical Investigation of GP behavior in Bayesian NNs:







[Matthews et al., 2018]

Bayesian NNs Behave Like GPs

This isn't necessarily a bad thing since Bayesian NNs are more computationally efficient as the dataset size grows.

GP MARGINAL LIKELIHOOD

$$\log p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta}) = \mathcal{O}(N^3)$$

$$\frac{-1}{2}\log\boxed{|\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbb{I}|}+\frac{-1}{2}\mathbf{y}^T\boxed{(\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbb{I})^{-1}}\mathbf{y}+\text{const}$$

Bayesian NNs Behave Like GPs

This isn't necessarily a bad thing since Bayesian NNs are more computationally efficient as the dataset size grows.

GP MARGINAL LIKELIHOOD

$$\log p(\mathbf{y}|\mathbf{X};\boldsymbol{\theta}) = \mathcal{O}(N^3)$$

$$\frac{-1}{2}\log\boxed{|\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbb{I}|}+\frac{-1}{2}\mathbf{y}^T\boxed{(\mathbf{K}_{\boldsymbol{\theta}}+\sigma^2\mathbb{I})^{-1}}\mathbf{y}+\text{const}$$

Bayesian NNs have $\mathcal{O}(N)$ dependence on dataset size.

Scalable Bayesian Optimization Using Deep Neural Networks

Jasper Snoek[*]	JSNOEK@SEAS.HARVARD.EDU
Oren Rippel[†*]	RIPPEL@MATH.MIT.EDU
Kevin Swersky[§]	KSWERSKY@CS.TORONTO.EDU
Ryan Kiros[§]	RKIROS@CS.TORONTO.EDU
Nadathur Satish[‡]	NADATHUR.RAJAGOPALAN.SATISH@INTEL.COM
Narayanan Sundaram[‡]	NARAYANAN.SUNDARAM@INTEL.COM
Md. Mostofa Ali Patwary[‡]	MOSTOFA.ALI.PATWARY@INTEL.COM
Prabhat[*]	PRABHAT@LBL.GOV
Ryan P. Adams[*]	RPA@SEAS.HARVARD.EDU

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

$$w_j^o \sim \mathcal{F}^{-1}\left[\exp\{it\mu - |ct|^\alpha(1 - i\beta\mathrm{sgn}(t)\tan(\pi\alpha/2)\}\right]$$

”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]

Priors with Infinite Variance

$$w_j^o \sim \mathcal{F}^{-1}\left[\exp\{it\mu - |ct|^\alpha(1 - i\beta\mathrm{sgn}(t)\tan(\pi\alpha/2)\}\right]$$

”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]

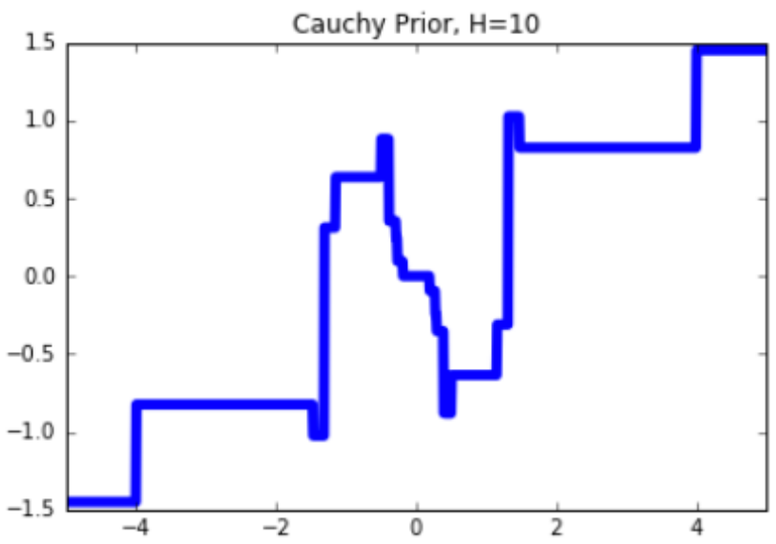
Function draws from a NN with H hidden units and a Cauchy prior:

Priors with Infinite Variance

$$w_j^o \sim \mathcal{F}^{-1}\left[\exp\{it\mu - |ct|^\alpha(1 - i\beta\mathrm{sgn}(t)\tan(\pi\alpha/2)\}\right]$$

”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]

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

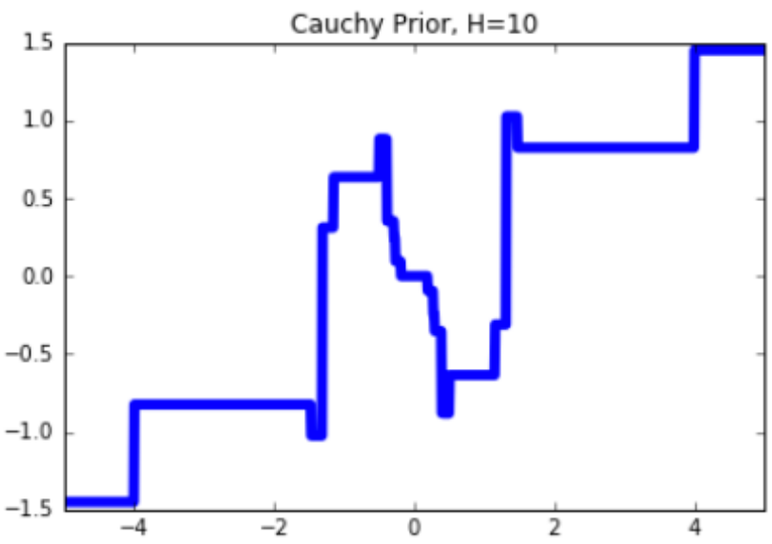


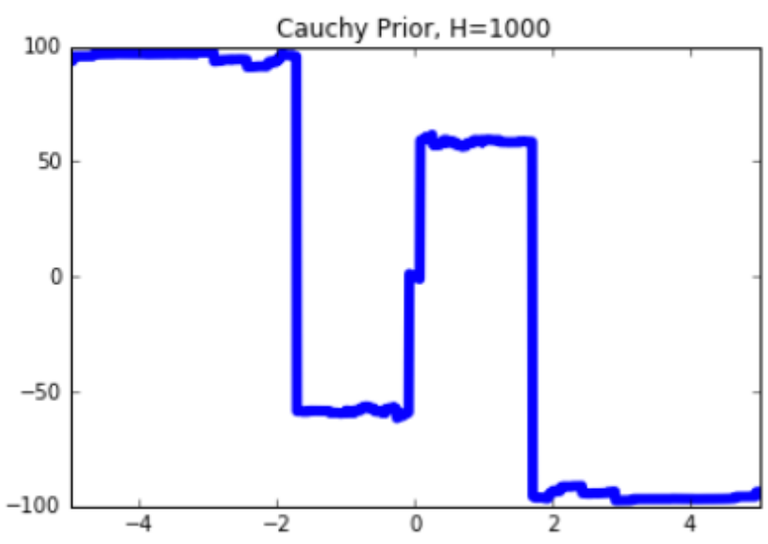
Priors with Infinite Variance

$$w_j^o \sim \mathcal{F}^{-1}\left[\exp\{it\mu - |ct|^\alpha(1 - i\beta\mathrm{sgn}(t)\tan(\pi\alpha/2)\}\right]$$

”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]

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



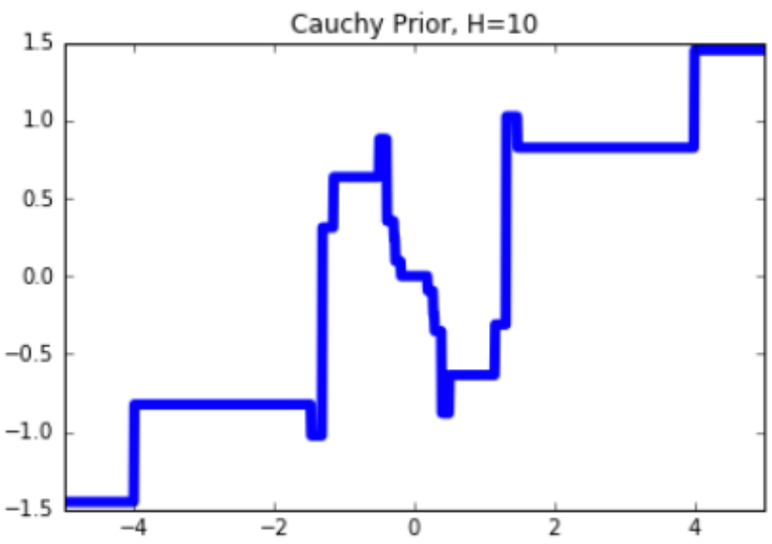


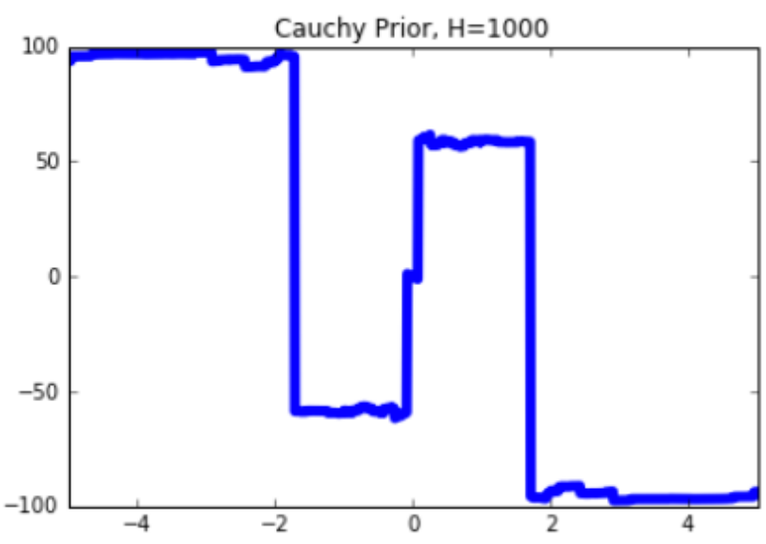
Priors with Infinite Variance

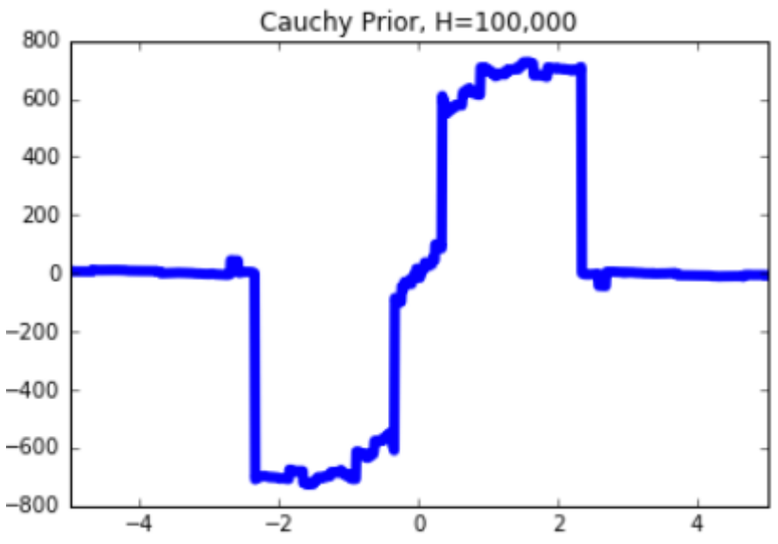
$$w_j^o \sim \mathcal{F}^{-1}\left[\exp\{it\mu - |ct|^\alpha(1 - i\beta\mathrm{sgn}(t)\tan(\pi\alpha/2)\}\right]$$

”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]

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







Neal's Recommendation: Student-t

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

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

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

Neal's Recommendation: Student-t

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

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

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:

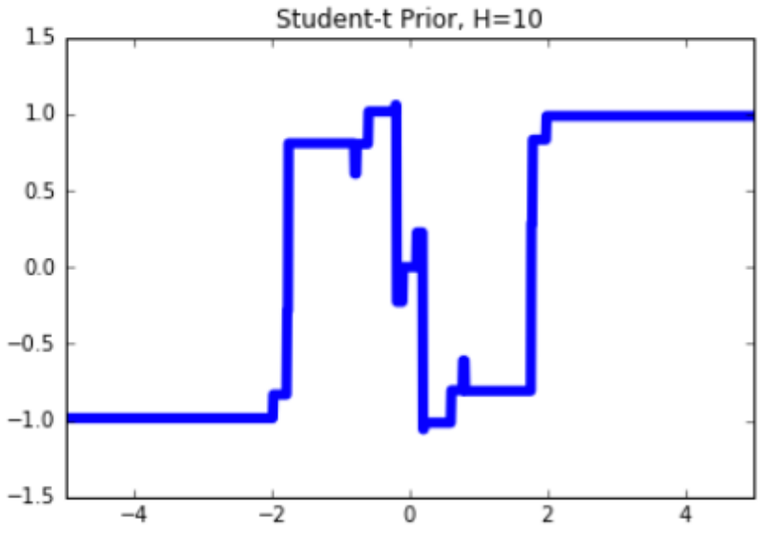
Neal's Recommendation: Student-t

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

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

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:



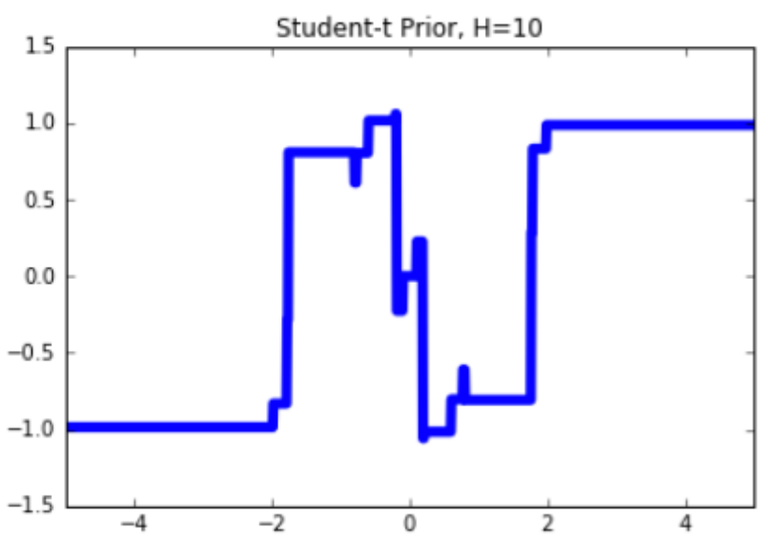
Neal's Recommendation: Student-t

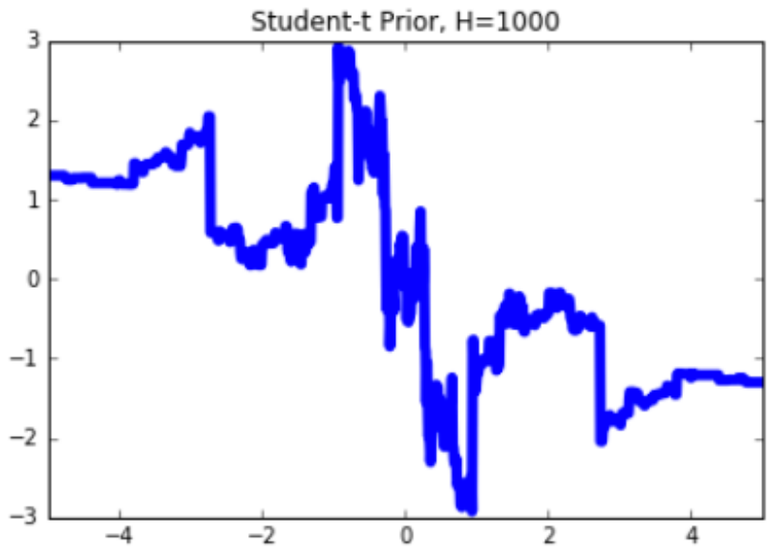
$$w_j^o \sim \mathrm{N}(0, \sigma^2)$$

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

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:





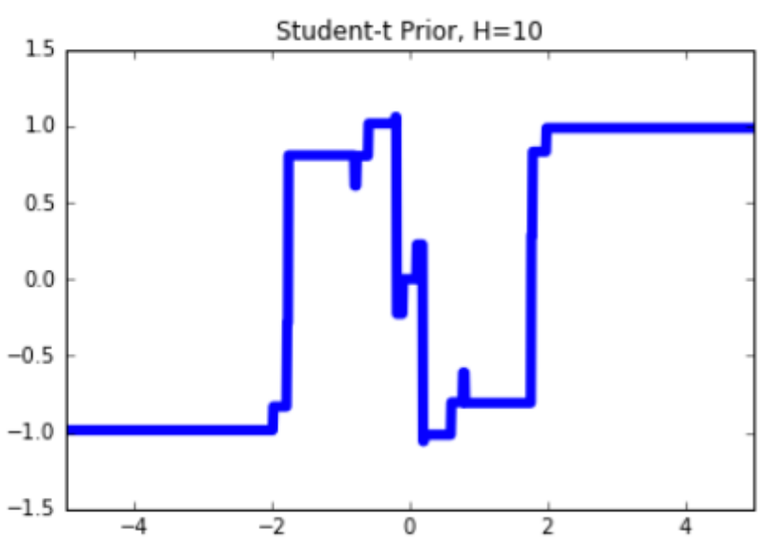
Neal's Recommendation: Student-t

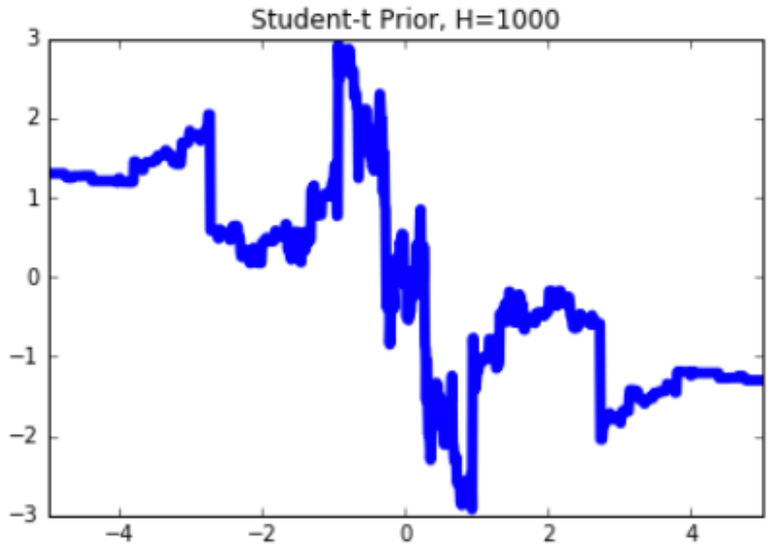
$$w_j^o \sim \mathrm{N}(0, \sigma^2)$$

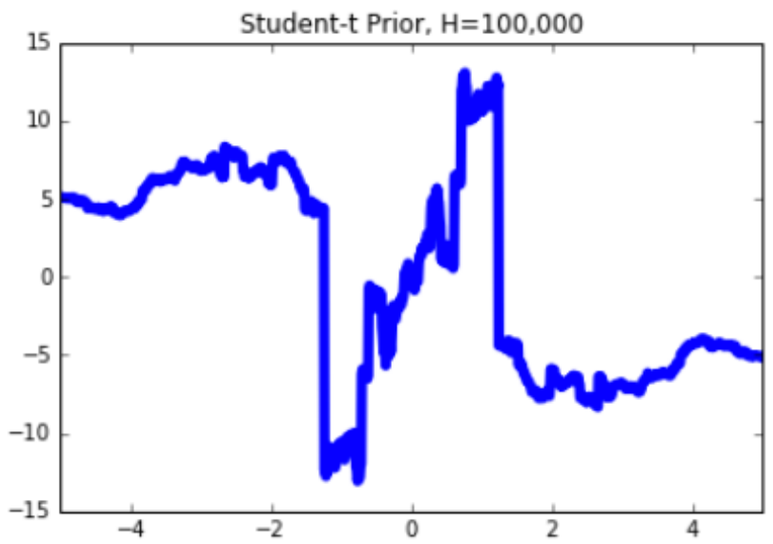
$$\sigma^2 \sim \Gamma^{-1}(\nu/2, \nu/2)$$

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:







Summary

- **Gaussian Processes** are flexible regression models with closed-form posterior quantities, making them good for uncertainty quantification.

Summary

- **Gaussian Processes** are flexible regression models with closed-form posterior quantities, making them good for uncertainty quantification.

- **Deep neural networks** can be thought of as regressors with adaptive basis functions, which jointly learn a predictor and latent representation. They learn powerful functions but uncertainty quantification is hard.

-

Summary

- **Gaussian Processes** are flexible regression models with closed-form posterior quantities, making them good for uncertainty quantification.

- **Deep neural networks** can be thought of as regressors with adaptive basis functions, which jointly learn a predictor and latent representation. They learn powerful functions but uncertainty quantification is hard.

- **Bayesian NNs with Gaussian priors** quickly take-on GP behavior, which makes them good plug-in GP approximations. However, if learning latent features is desired, then must break CLT to break GP connection.

Thank you. Questions?

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. Do, 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.

Jaehoon 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.