Biostatistics 682: Applied Bayesian Inference
Lecture 13: Bayesian Neural Networks

Jian Kang

Department of Biostatistics
University of Michigan, Ann Arbor

Deep Learning

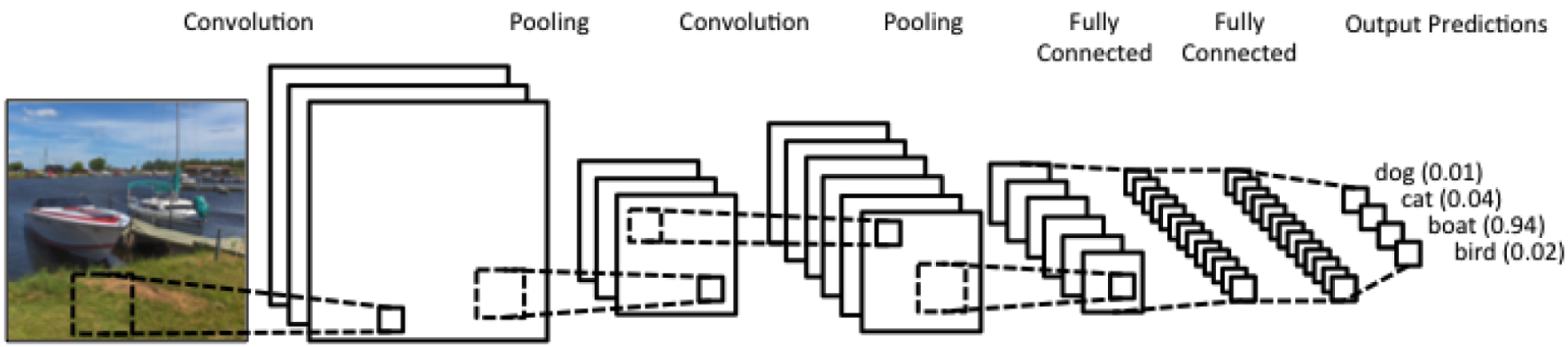


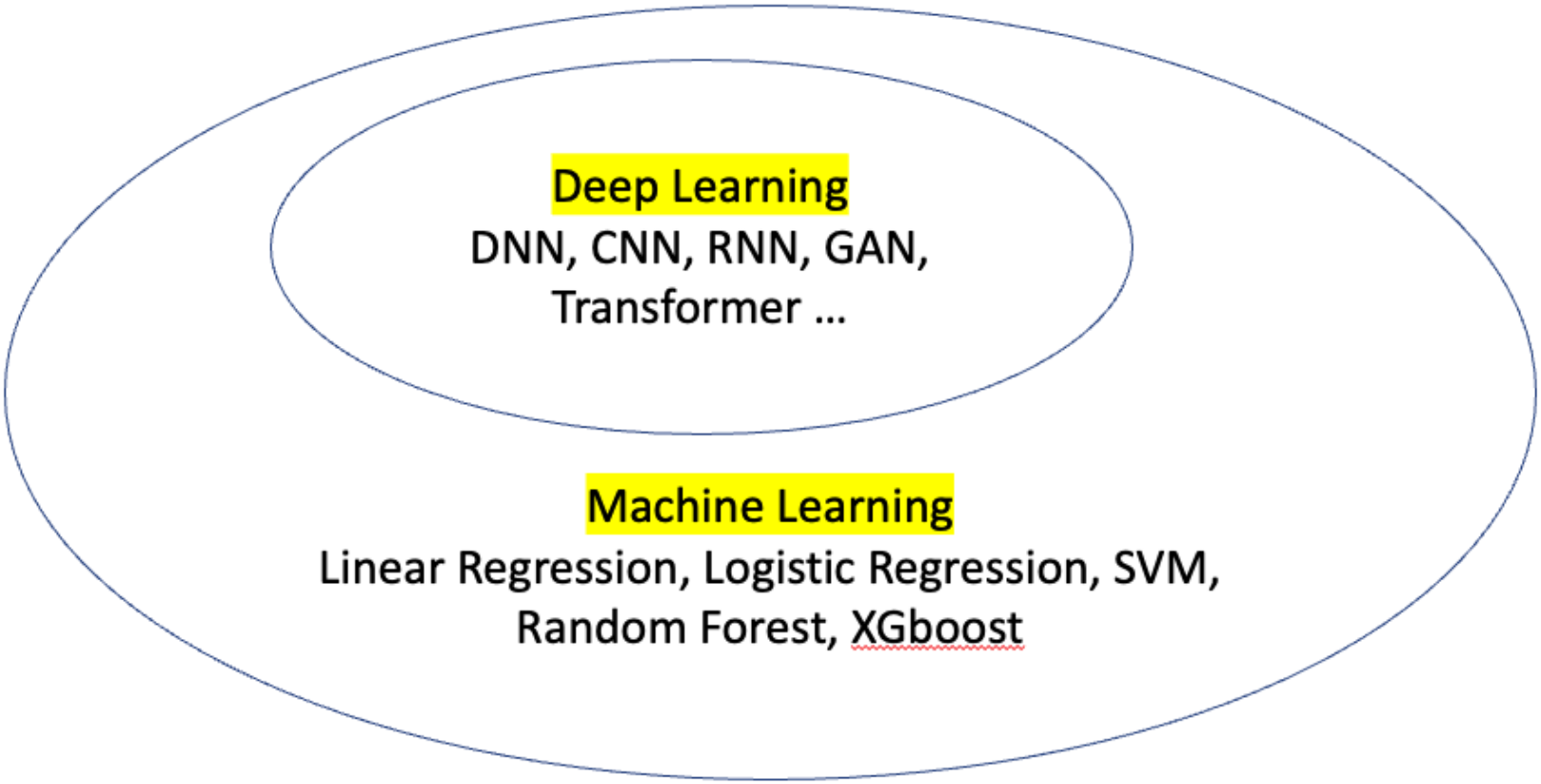
Figure from http://www.andreykurenkov.com

- Deep Learning systems are neural network models with
  - Some architectural and algorithmic innovations (e.g. many layers, different activation functions, better initialization and learning rates, dropout , ...)
  - larger data sets (web-scale) ...
  - larger-scale compute resources (GPU, cloud) ...
  - Increased industry investment and media type ...

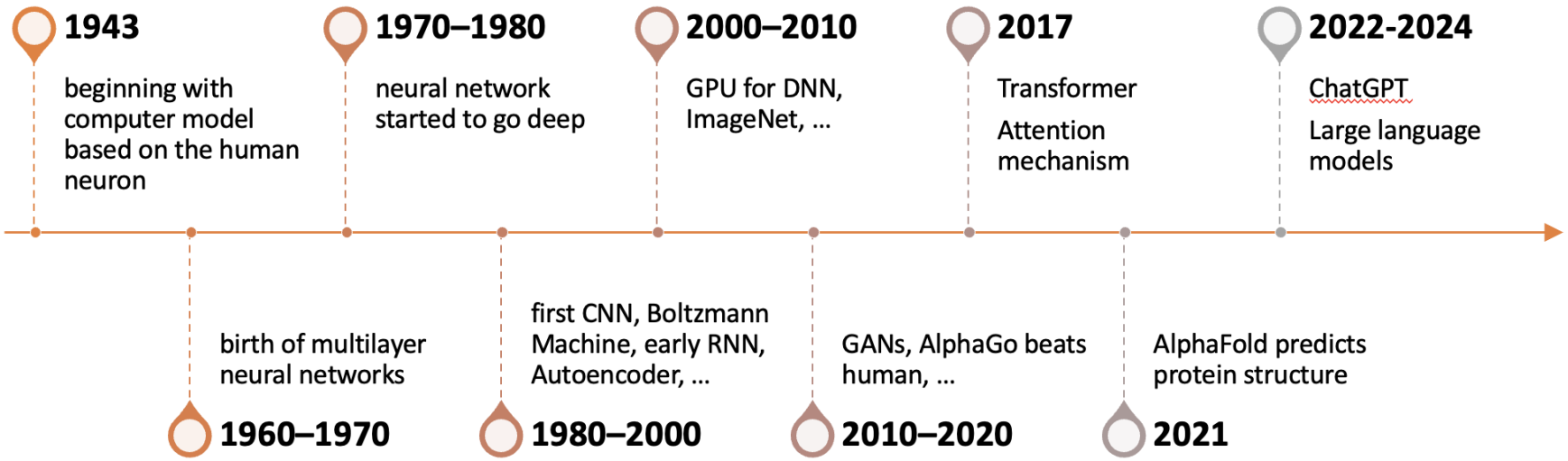
Different views about deep learning

- Computer Scientists: algorithms to process data and imitate the thinking process, or to develop abstractions
- Statisticians: composition of many nonlinear functions to model the complex dependency between input features and labels.
- Biomedical Researchers: tools to tackle problems in biomedical research for which classical methods are not well-suited.

Deep Learning and Machine Learning

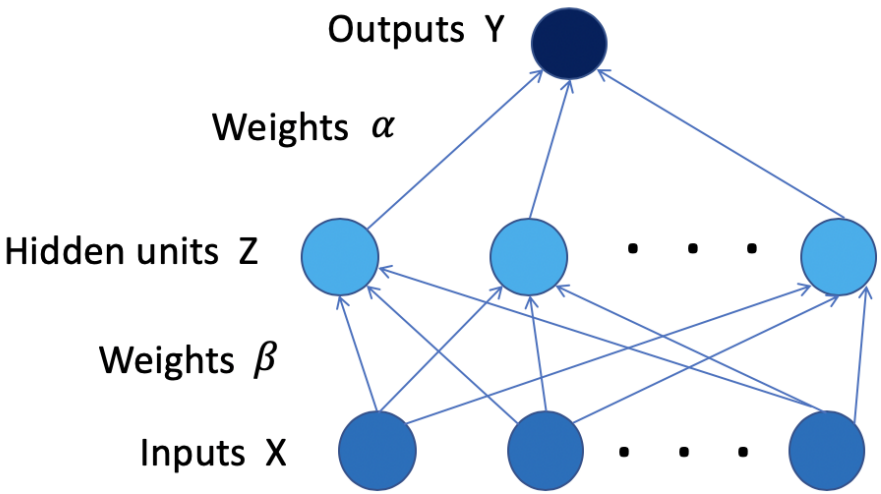


A Brief History



What is a Neural Network?

- Neural network is a parameterized non-linear function
  - Inputs: $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ with $\mathbf{x}_i = (x_{i,j})_{j=1}^p$.
  - Output: $\mathbf{Y} = (y_i)_{i=1}^n$.
  - Parameters: weights of neural net
- For example, one layer neural network model:



$$y_i = \alpha_0 + \sum_{k=1}^{K} \alpha_k z_{i,k} + \epsilon_i$$

$$z_{i,k} = \sigma\left(\beta_{k,0} + \sum_{j=1}^{p} \beta_{k,j} x_{i,j}\right)$$

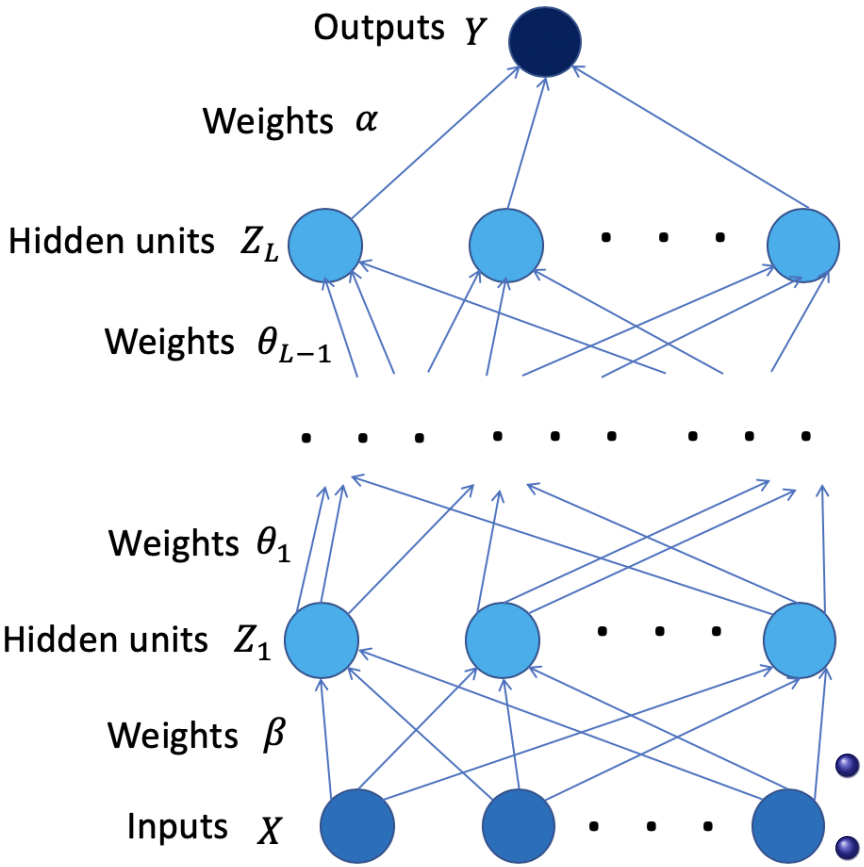
Many choices of $\sigma(\cdot)$, e.g.

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

- Classic approaches: find maximum likelihood estimates (penalized likelihood) using variants of stochastic gradient decent (SGD) optimization.

Multilayer neural networks

Multilayer neural networks model the overall function as multiple compositions (layers) of functions. e.g., L layer-neural networks with m_l units in the lth layer.



$$y_i = \alpha_0 + \sum_{k=1}^{K} \alpha_k z_{L,i,k} + \epsilon_i,$$

for $l = 1, \ldots, L-1$,

$$z_{l+1,i,k} = \sigma\left(\theta_{l,k,0} + \sum_{j=1}^{m_l} \theta_{l,k,j} z_{l,i,j}\right),$$

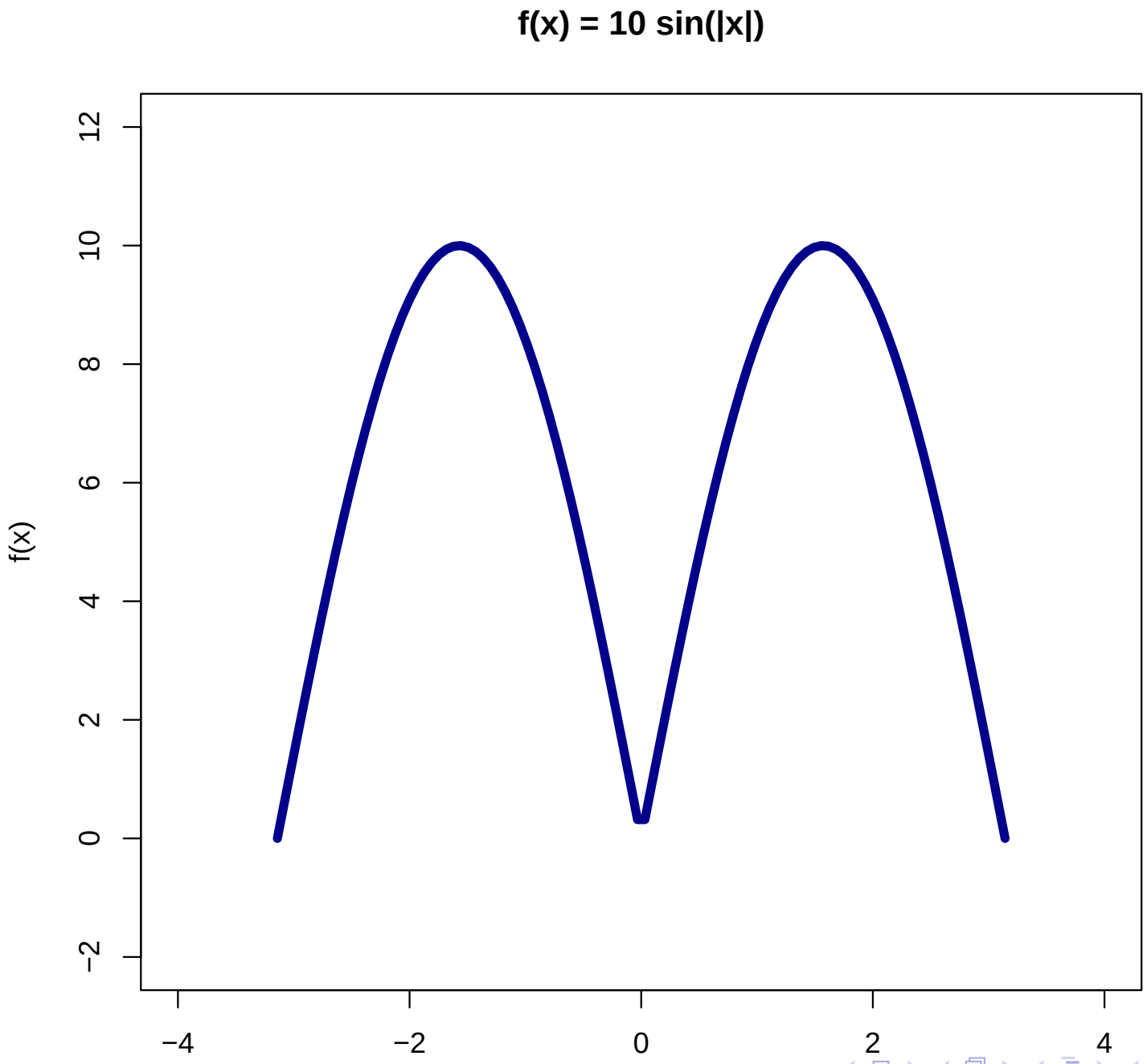
$$z_{1,i,k} = \sigma\left(\beta_{k,0} + \sum_{j=1}^{p} \beta_{k,j} x_{i,j}\right)$$

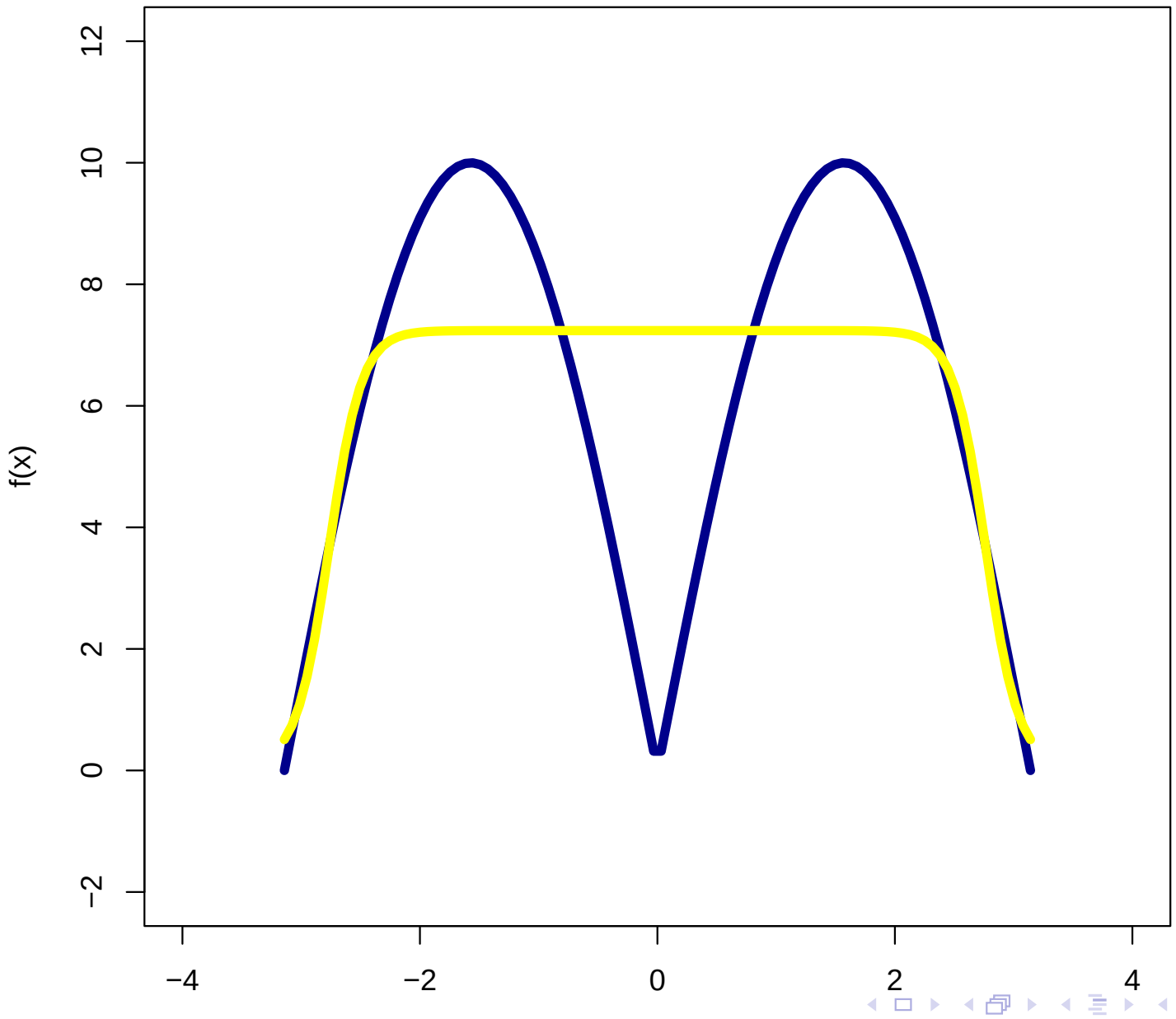
- $\mathbf{Z}_l = \{z_{l,i,k}\}$ of dimension $n \times m_l$;
- $\boldsymbol{\theta}_l = \{\theta_{l,k,j}\}$ of dimension $m_{l+1} \times (m_l + 1)$;

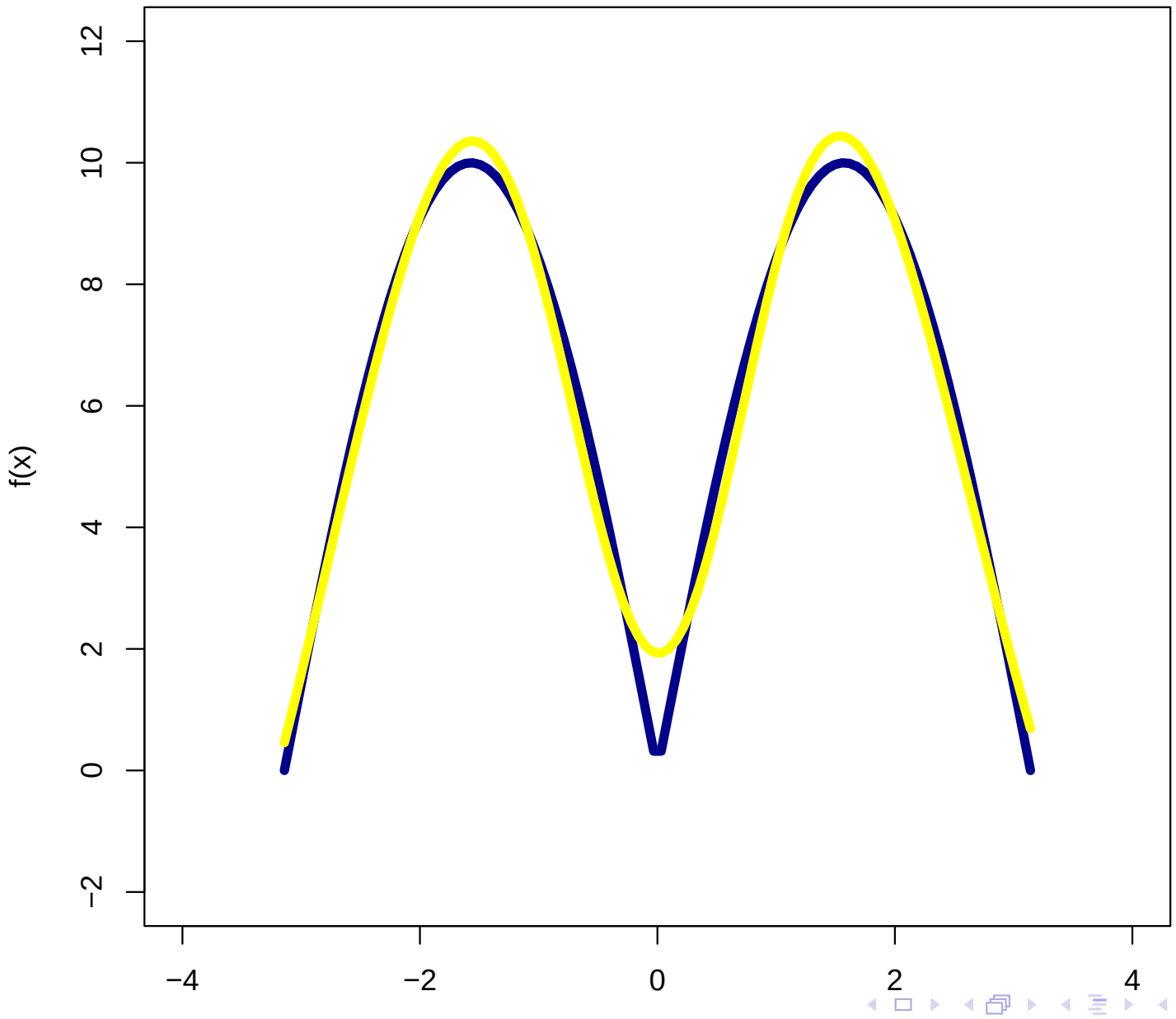
Why Neural Network Works

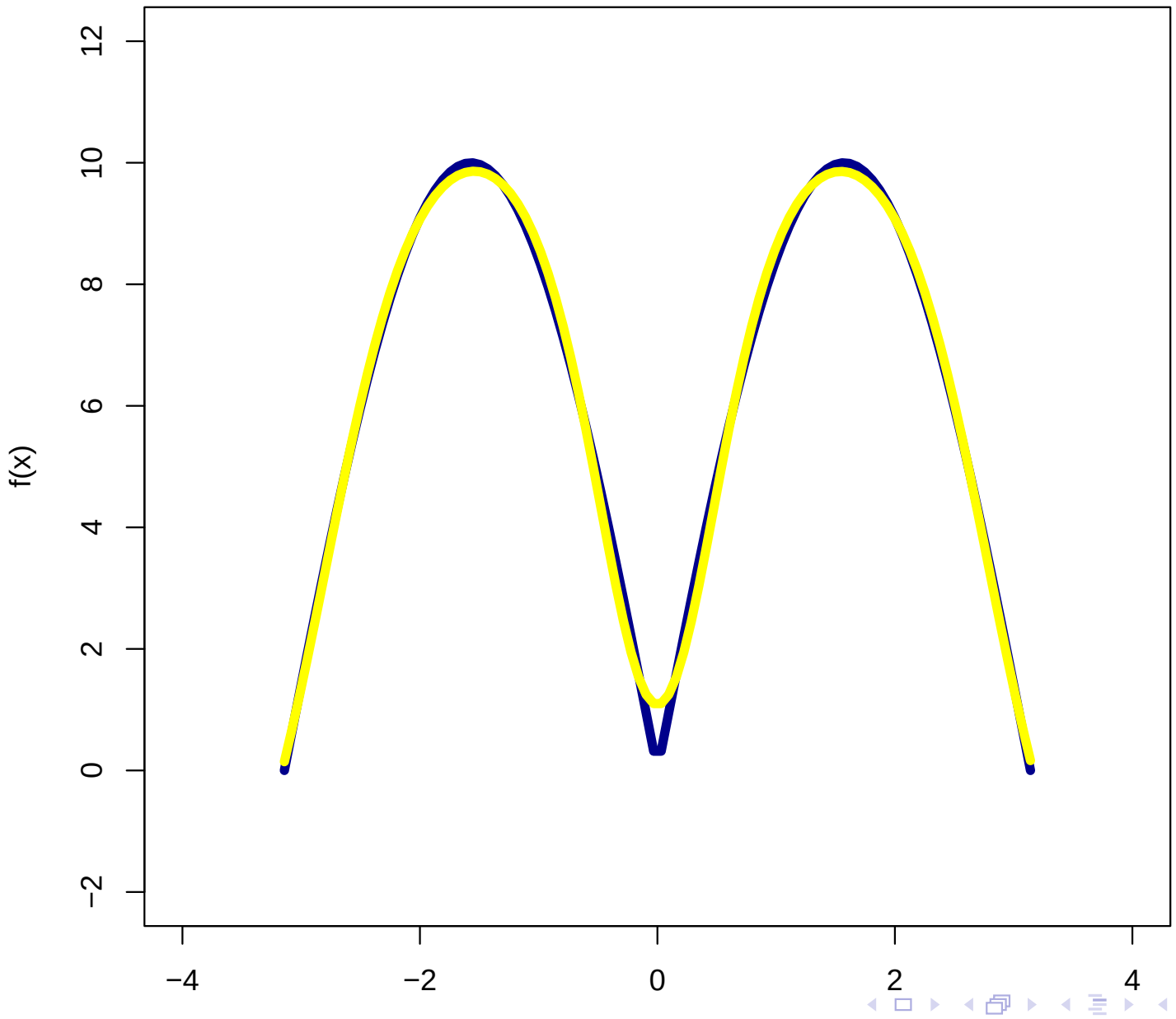
- Universal Approximation Theorem: with a sufficiently large number of hidden units, a single-layer neural network model can approximate any continuous function to an arbitrary degree of accuracy
- Toy Example:

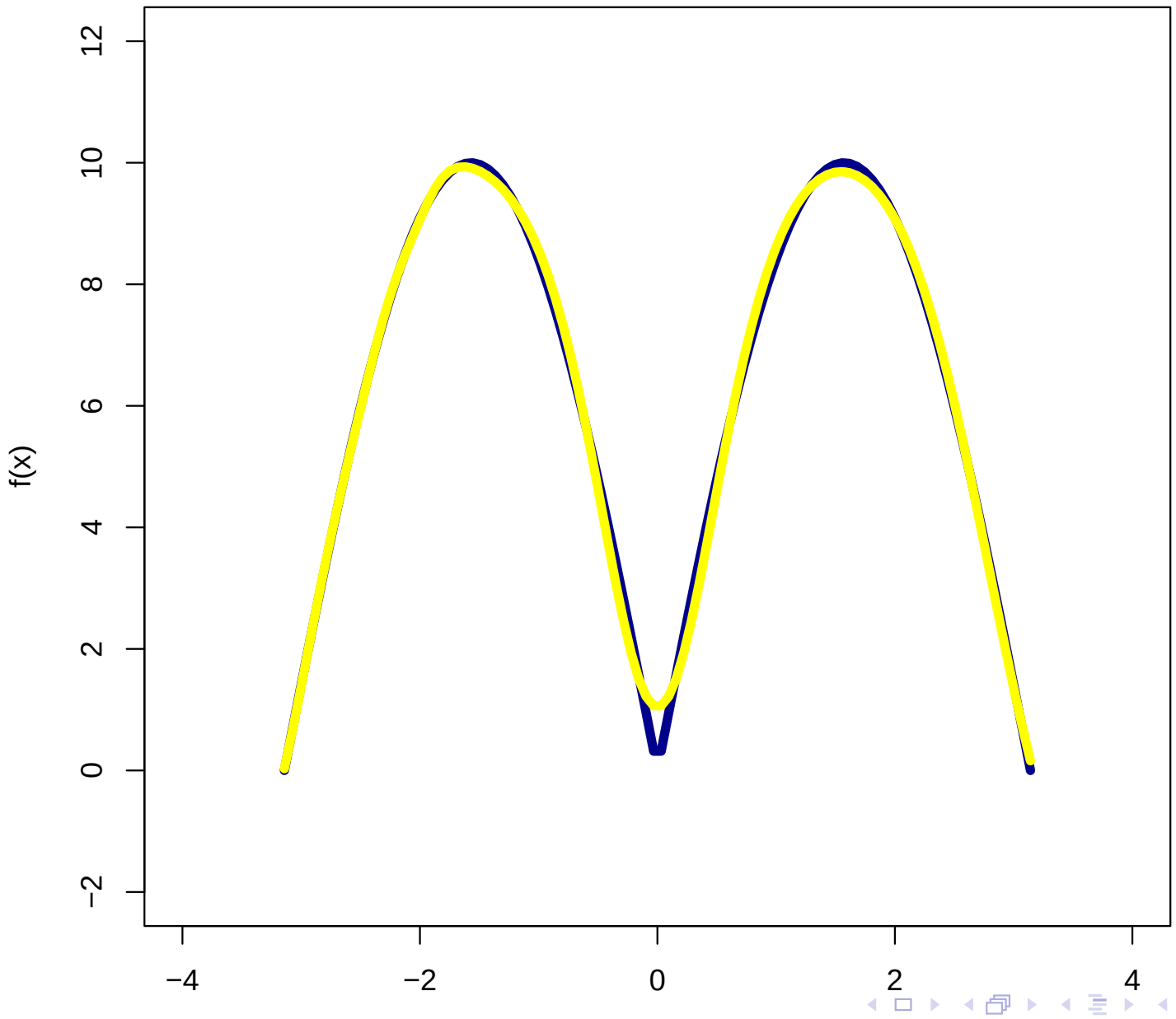
$$f(x) = 10\sin(|x|), -3 \leq x \leq 3$$

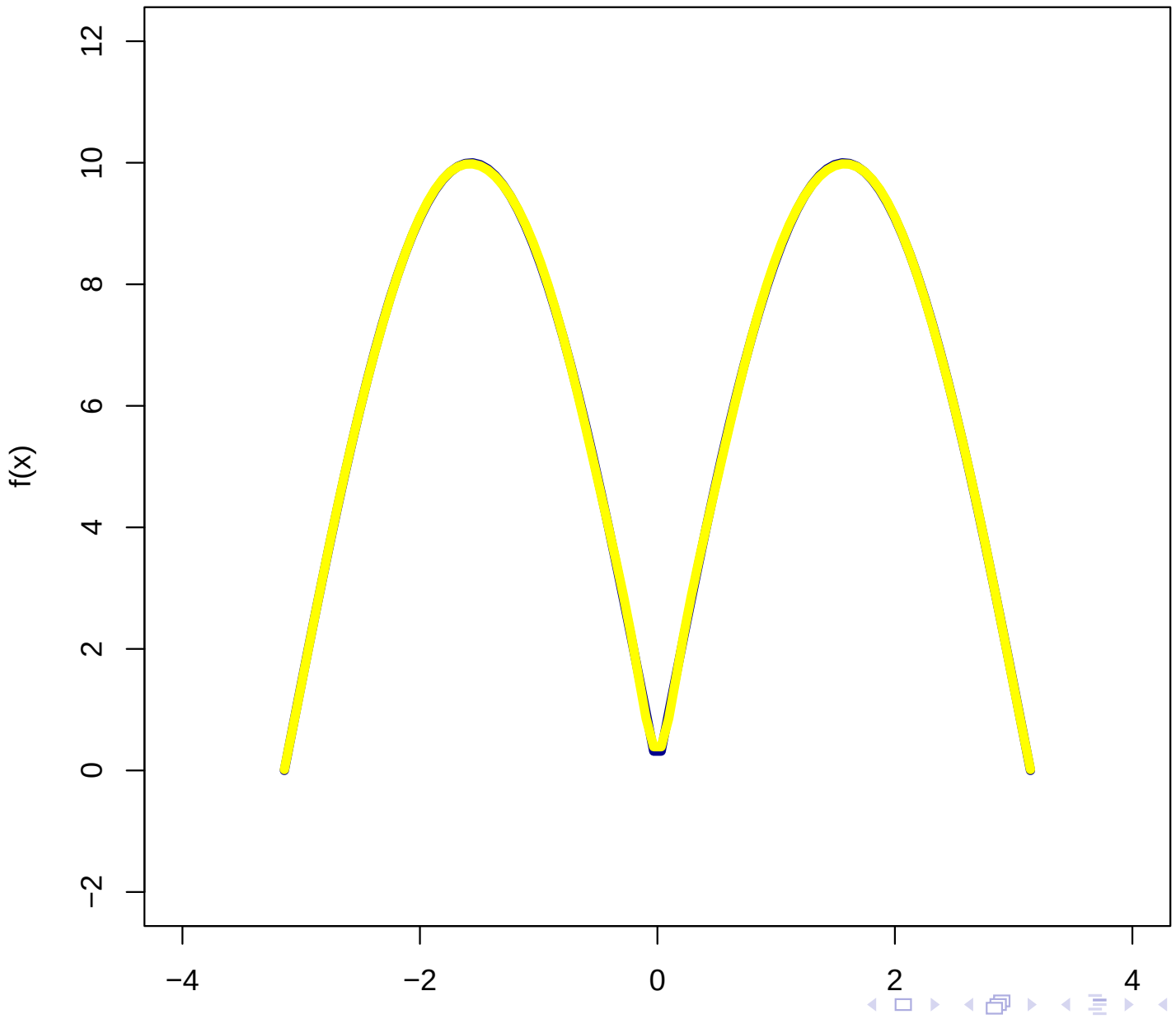

f(x) = 10 sin(|x|)
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4

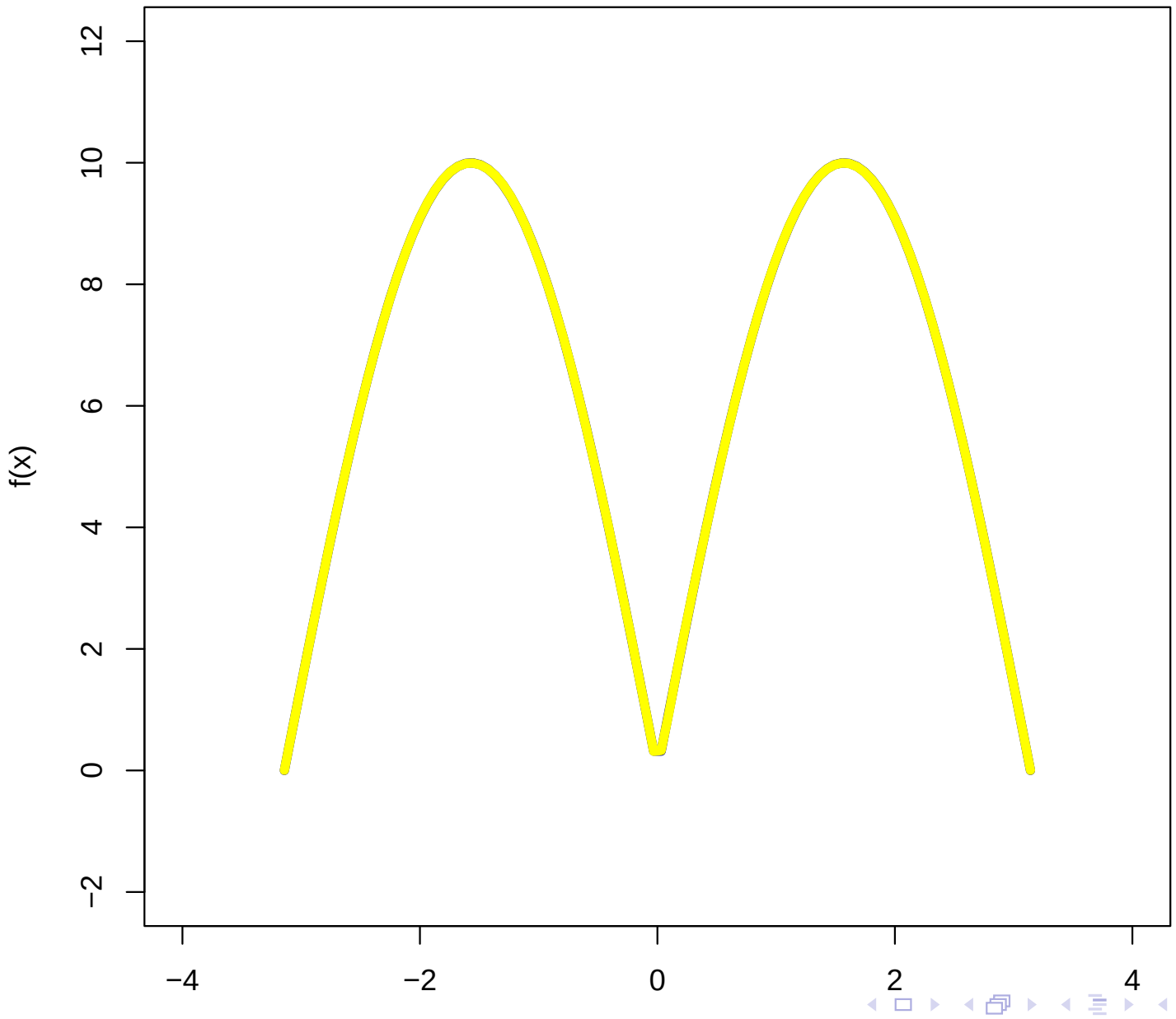

1 layer 2 neurons NN approx err = 6.06609
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4

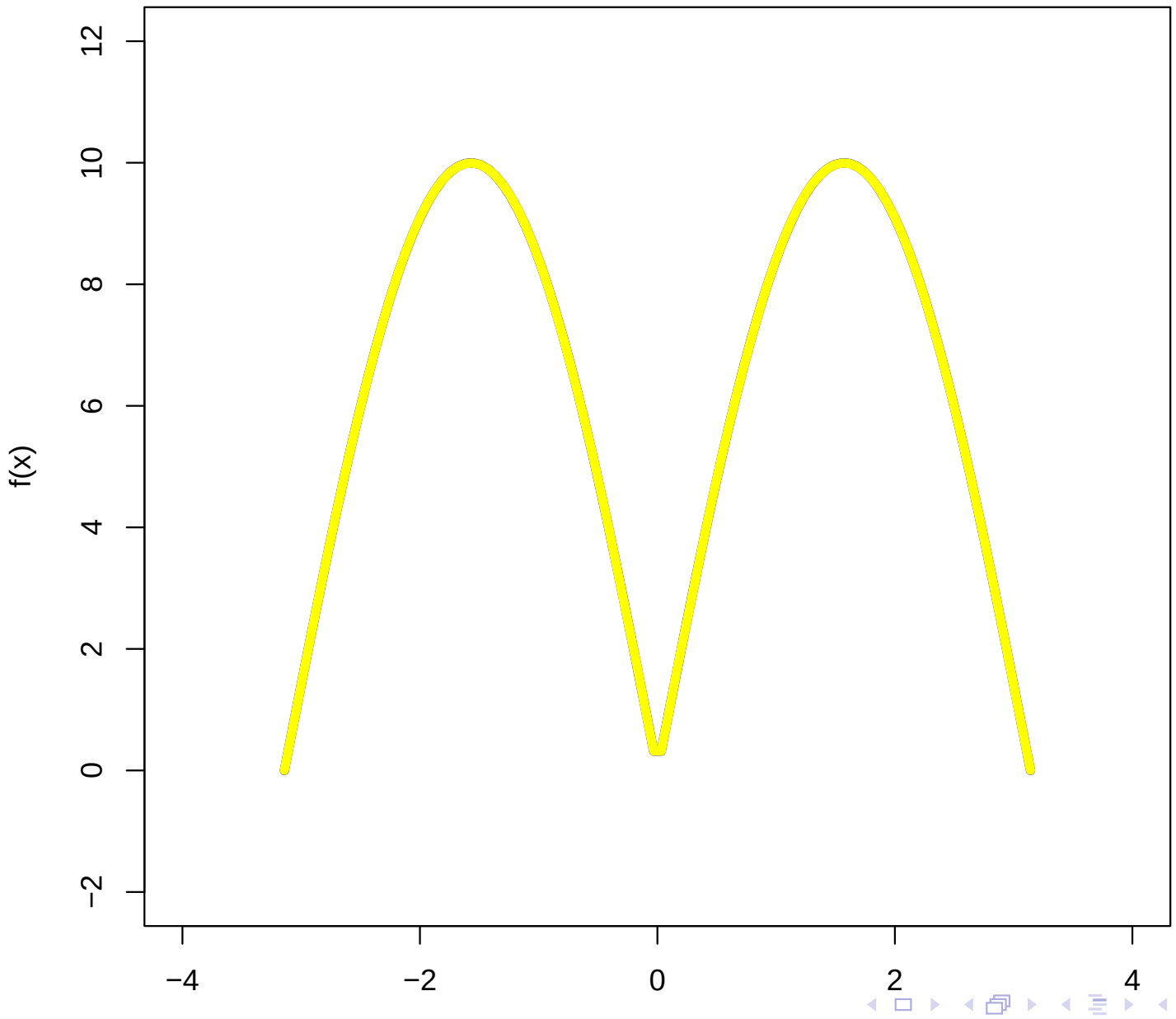

1 layer 3 neurons NN approx err = 0.20084
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4


1 layer 4 neurons NN approx err = 0.03494
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4


1 layer 6 neurons NN approx err = 0.03015
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4


1 layer 8 neurons NN approx err = 0.00064
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4


1 layer 10 neurons NN approx err = 2e−05
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4


1 layer 12 neurons NN approx err = 1e−05
f(x)
12
10
8
6
4
2
0
−2
−4
−2
0
2
4

Why go deep?

- Shallow net may need (exponentially) more width
- Shallow net may overfit more
- The deeper the better?
- Toy Examples: `https://playground.tensorflow.org`

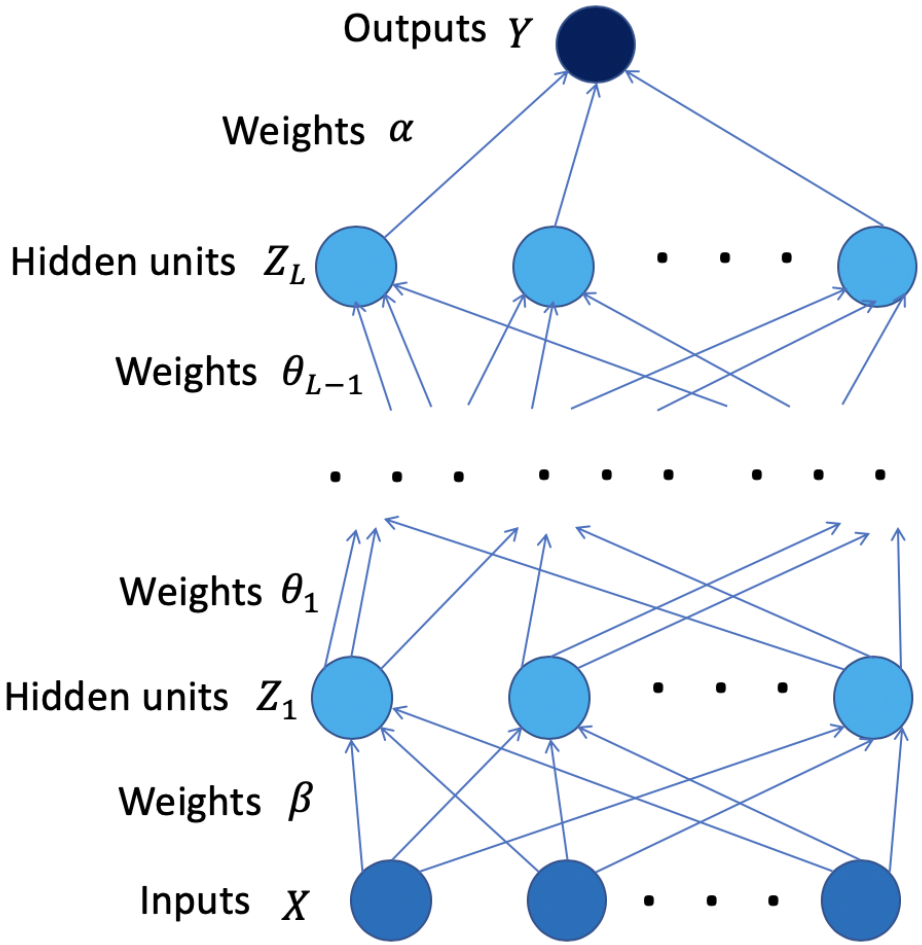
Limitations of Deep Learning

Neural networks and deep learning provide amazing performance on many benchmark tasks, but they are generally:

- very data hungry (e.g. often millions of examples)
- very compute-intensive to train and deploy (cloud GPU recourses)
- poor at representing uncertainty
- easily fooled by adversarial examples
- hard to optimize: non-convex + choice of architecture, learning procedure, initialisation, etc, require expert knowledge and experimentation
- uninterpretable black-boxes, lacking in transparency difficult to trust

Why Bayesian neural networks?

- Dealing with all sources of parameter uncertainty
- Potentially dealing with structure uncertainty



- Feedforward neural nets model $\pi(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}, \mathcal{M})$
- Parameters θ:
  - $\boldsymbol{\theta} = \{\boldsymbol{\beta}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{L-1}, \boldsymbol{\alpha}\}$.
- Model structure $\mathcal{M}$:
  - The choice of architecture
  - Number of hidden units and layers
  - choice of activation functions

Bayesian Machine Learning

- Learning:

$$\pi(\boldsymbol{\theta} \mid \mathbf{Y}, \mathbf{X}, \mathcal{M}) = \frac{\pi(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\theta}, \mathcal{M})\pi(\boldsymbol{\theta} \mid \mathcal{M})}{\pi(\mathbf{Y} \mid X, \mathcal{M})}$$

  - $\pi(\mathbf{Y} \mid \mathbf{X}, \mathcal{M})$: likelihood of parameters $\boldsymbol{\theta}$ in model $\mathcal{M}$.
  - $\pi(\boldsymbol{\theta} \mid \mathcal{M})$: prior distribution of $\boldsymbol{\theta}$.
  - $\pi(\boldsymbol{\theta} \mid \mathbf{Y}, \mathbf{X}, \mathcal{M})$: posterior of $\boldsymbol{\theta}$ given data $\mathbf{Y}$ and $\mathbf{X}$.

- Prediction:

$$\pi(\tilde{Y} \mid \tilde{X}, \mathbf{Y}, \mathbf{X}, \mathcal{M}) = \int \pi(\tilde{Y} \mid \tilde{X}, \boldsymbol{\theta}, \mathbf{Y}, \mathbf{X}, \mathcal{M})\pi(\boldsymbol{\theta} \mid \mathbf{Y}, \mathbf{X}, \mathcal{M})d\boldsymbol{\theta}.$$

  - $(\tilde{Y}, \tilde{X})$: test data, $(\mathbf{Y}, \mathbf{X})$: training data
  - Conditional independence: $\pi(\tilde{Y} \mid \tilde{X}, \boldsymbol{\theta}, \mathbf{Y}, \mathbf{X}, \mathcal{M}) = \pi(\tilde{Y} \mid \tilde{X}, \boldsymbol{\theta}, \mathcal{M})$.

- Model Comparison:

$$\pi(\mathcal{M} \mid \mathbf{Y}, \mathbf{X}) = \frac{\pi(\mathbf{Y}, \mathbf{X} \mid \mathcal{M})\pi(\mathcal{M})}{\pi(\mathbf{Y}, \mathbf{X})}$$

Why should we care

- Calibrated model and prediction uncertainty: getting systems that know when they don't know
- Automatic control model complexity and structure learning

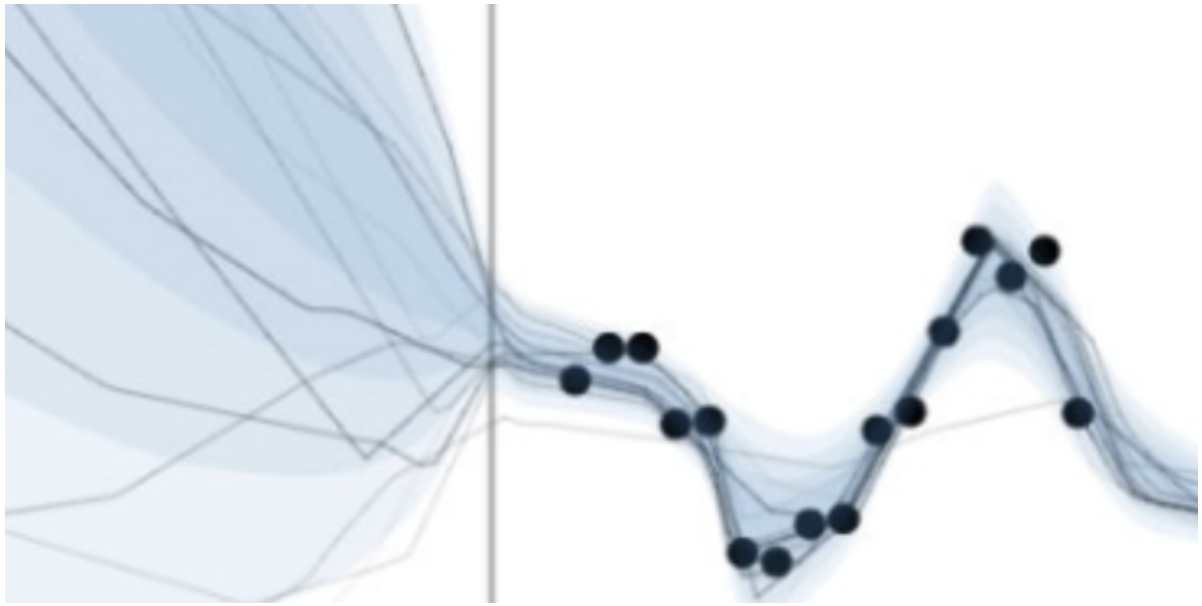
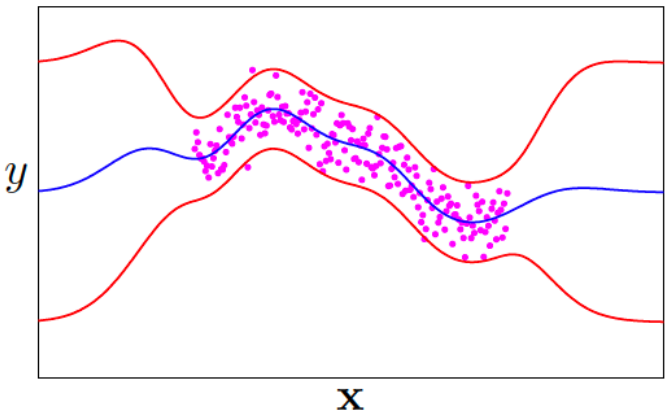


Figure from Yarin Gal's thesis "Uncertianty in Deep Learning" (2016)

Recall: Gaussian Processes

- Consider a nonlinear regression problem: $y = f(x) + \epsilon$.



  - Gaussian process priors:

$$f \sim \mathrm{GP}(\mu, \kappa),$$

    where $\mathrm{E}\{f(x)\} = \mu(x)$ and $\mathrm{Cov}\{f(x), f(x')\} = \kappa(x, x')$.

  - Gaussian processes: a stochastic process for which any finite linear combination of samples has a joint Gaussian

$$[f(x_1), \ldots, f(x_n)] \sim \mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

    where $\boldsymbol{\mu} = \{\mu(x_1), \ldots, \mu(x_n)\}$ and $\boldsymbol{\Sigma} = \{\kappa(x_i, x_j)\}_{1 \le i,j \le n}$.

- Gaussian Process defines a distribution over functions.
- GPs can be used for regression, classification, ranking, dim. reduct ...

Neural networks and Gaussian Processes

- A neural network with one hidden layer, infinitely many hidden units and Gaussian priors on the weights is equivalent to a Gaussian process (GP)
- How about deep neural networks?

Published as a conference paper at ICLR 2018

DEEP NEURAL NETWORKS AS GAUSSIAN PROCESSES

Jaehoon Lee[*†], **Yasaman Bahri**[*†], **Roman Novak**, **Samuel S. Schoenholz**, **Jeffrey Pennington**, **Jascha Sohl-Dickstein**

Google Brain
{jaehlee, yasamanb, romann, schsam, jpennin, jaschasd}@google.com

ABSTRACT

It has long been known that a single-layer fully-connected neural network with an i.i.d. prior over its parameters is equivalent to a Gaussian process (GP), in the limit of infinite network width. This correspondence enables exact Bayesian inference for infinite width neural networks on regression tasks by means of evaluating the corresponding GP. Recently, kernel functions which mimic multi-layer random neural networks have been developed, but only outside of a Bayesian framework. As such, previous work has not identified that these kernels can be used as covariance functions for GPs and allow fully Bayesian prediction with a deep neural network.

In this work, we derive the *exact equivalence* between infinitely wide *deep* networks and GPs. We further develop a computationally efficient pipeline to compute the covariance function for these GPs. We then use the resulting GPs to perform Bayesian inference for wide deep neural networks on MNIST and CIFAR-10. We observe that trained neural network accuracy approaches that of the corresponding GP with increasing layer width, and that the GP uncertainty is strongly correlated with trained network prediction error. We further find that test performance increases as finite-width trained networks are made wider and more similar to a GP, and thus that GP predictions typically outperform those of finite-width networks. Finally we connect the performance of these GPs to the recent theory of signal propagation in random neural networks.

Prior families for weights and biases

Setup. For layer ℓ with weight matrix $\theta_{\ell,k,j}$ and bias $\theta_{\ell,k,0}$.

- **Gaussian (ridge)**: $\theta_{\ell,k,j} \sim \mathcal{N}(0, \sigma^2)$; smooth function prior; GP link in wide limit.
- **Laplace (lasso)**: $\theta_{\ell,k,j} \sim \text{Laplace}(0, b)$; induces sparsity.
- **Student-t**: heavy tails; robust to outliers and occasional large weights.
- **Horseshoe (global-local)**: $\theta_{\ell,k,j} \mid \lambda_{k,j}, \tau \sim \mathcal{N}(0, \lambda_{k,j}^2 \tau^2)$, $\lambda_{k,j} \sim \text{Half-Cauchy}(0, 1)$, $\tau \sim \text{Half-Cauchy}(0, 1)$; strong shrinkage of most weights, little shrinkage on a few.
- **Spike-and-slab**: mixture of a point mass at 0 and a diffuse slab; structural sparsity (costly for large nets).

Recommendation. Start with Gaussian; use Horseshoe when you expect many near-zero weights; Laplace if you prefer convex sparsity.

Horseshoe Prior: Global–Local Shrinkage

Motivation. We want a prior that

- strongly shrinks most weights toward zero (to avoid overfitting), and
- allows a few large weights to escape shrinkage (to capture strong signals).

Hierarchical formulation:

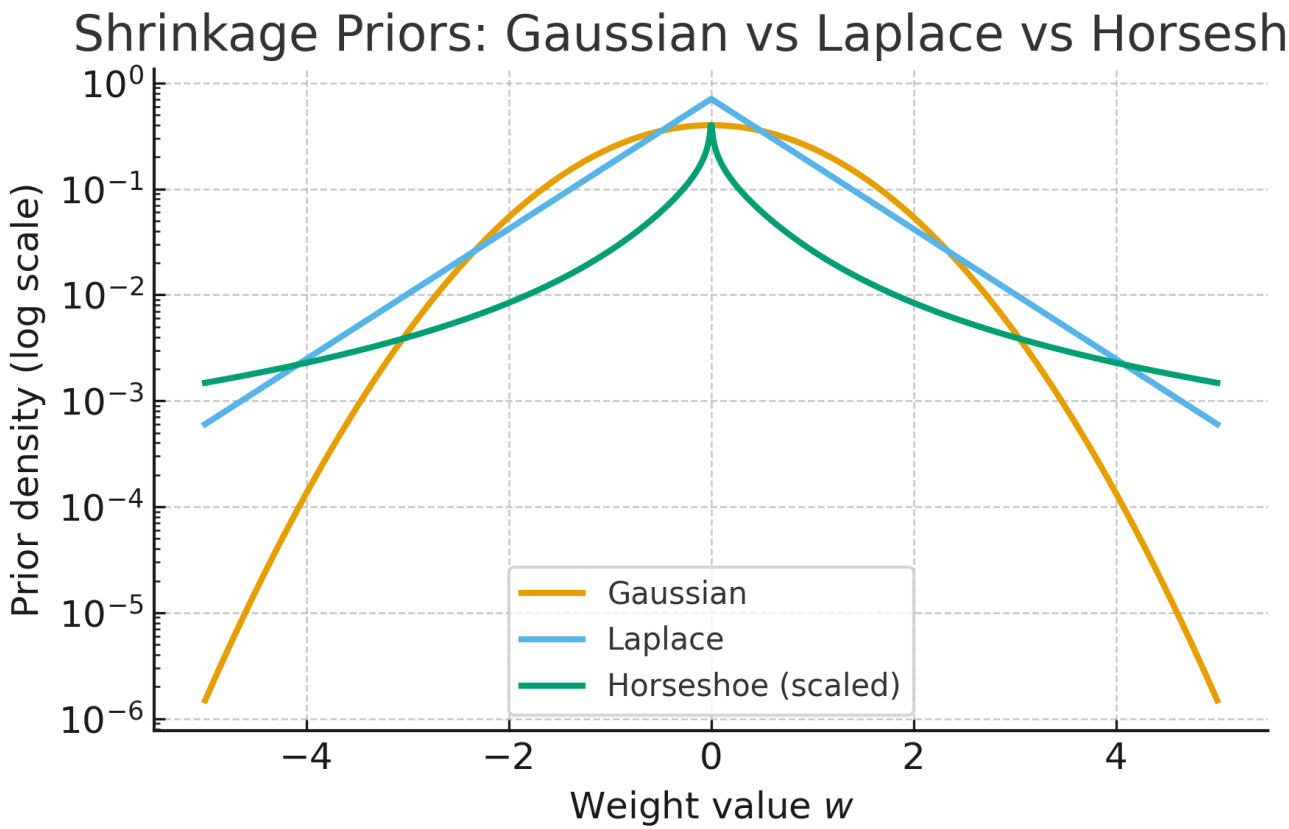
$$\theta_{\ell,k,j} \mid \lambda_{k,j}, \tau \sim \mathcal{N}(0, \lambda_{k,j}^2 \tau^2), \;\; \lambda_{k,j} \sim \text{Half-Cauchy}(0,1), \;\; \tau \sim \text{Half-Cauchy}(0,1)$$

Interpretation:

- τ: global shrinkage — controls overall regularization strength across the network.
- $\lambda_{k,j}$: local shrinkage — allows specific connections to remain large.

Effect:

- Heavy-tailed marginal prior $\Rightarrow$ strong shrinkage for noise, weak shrinkage for signal.
- Ideal for sparse high-dimensional settings, e.g., neural nets with redundant weights.



Comparison: Gaussian prior shrinks uniformly; Laplace shrinks linearly; Horseshoe adapts locally.

LOO Expected Log Predictive Density (elpd)

Motivation: We want to evaluate how well a Bayesian model predicts new data, while penalizing overfitting.

Definition: $\text{elpd}_{\text{LOO}} = \sum_{i=1}^{n} \log \pi(y_i \mid y_{-i})$ Each term assesses how probable observation y_i is, given the model fit on all other data y_{-i}.

Interpretation:

- Measures *out-of-sample predictive accuracy*.
- Higher $\text{elpd}_{\text{LOO}} \Rightarrow$ better predictive performance.
- Computed efficiently using **Pareto-smoothed importance sampling (PSIS)** rather than refitting n times.

In PyMC:

- `pm.loo(idata)` estimates elpd_{LOO} and its standard error.
- Compare models via $\Delta\text{elpd}_{\text{LOO}}$ (difference) and SE for uncertainty.

In context: We use LOO elpd to compare Gaussian, Laplace, and Horseshoe BNNs — a higher elpd_{LOO} means more reliable predictive uncertainty calibration.

Example: Simulated Motorcycle Accident Data

- The dataset consists of a series of measurements of head acceleration in a simulated motorcycle accident, used to test crash helmets.
  - Y_i: head acceleration
  - X_i: time (in milliseconds) after the impact.

  After standardization of Y_i and normalization of X_i, we split the data into two sets training set (100) and test set (33).