Introduction to Machine Learning:

Lecture 2 – Intro to Neural Networks

Michael Kagan

SLAC NATIONAL ACCELERATOR LABORATORY

TRISEP Summer School
July 8-12, 2024

The Plan

- Lecture 1 – Machine Learning Fundamentals
- Lecture 2 – Intro to Neural Networks
- Lecture 3 – Intro to Deep Learning
- Lecture 4 – Intro to Unsupervised Learning
- Lecture 5 – Intro to Deep Generative Models

Reminder: Logistic Regression

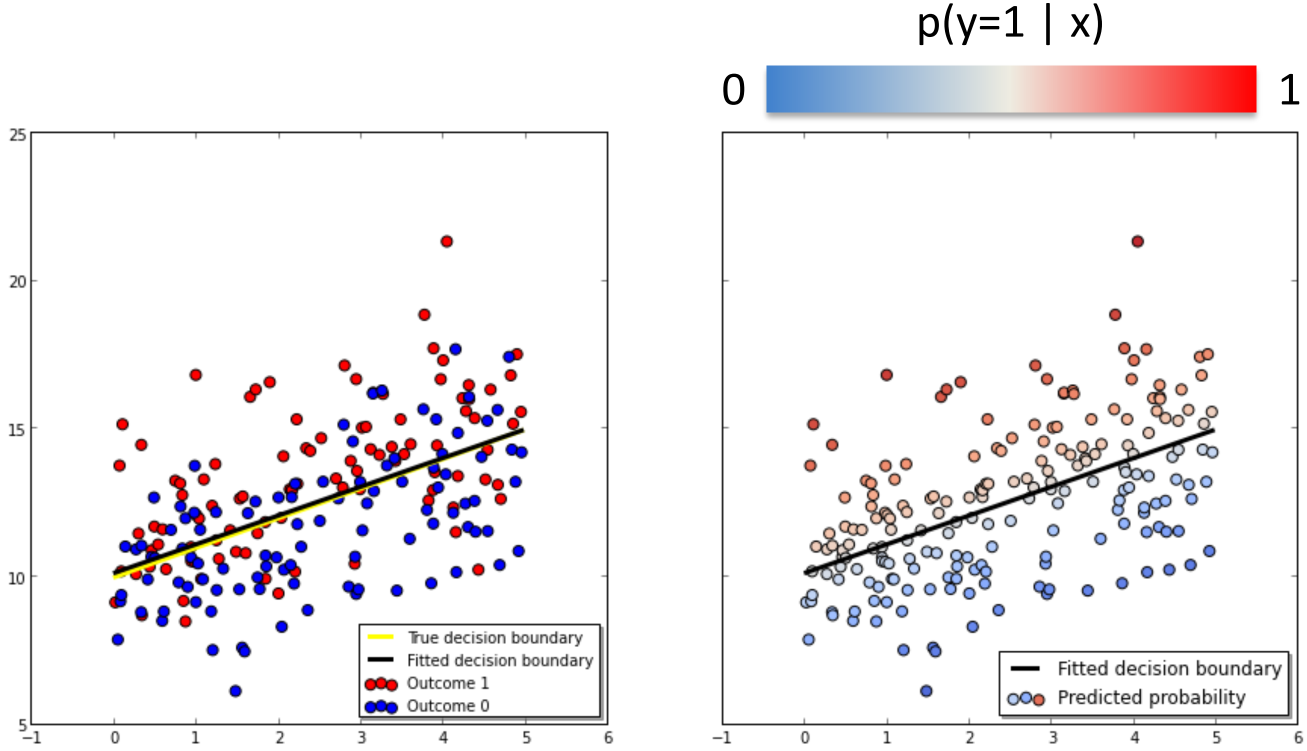
Logistic Regression

Linear decision boundary:

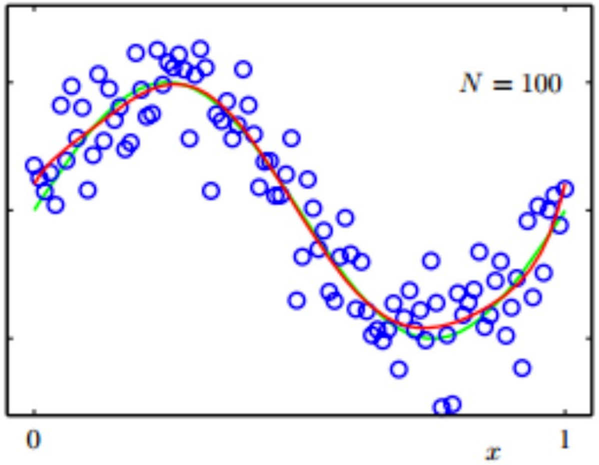
$$h(x; w) = \boldsymbol{w}^T \boldsymbol{x}$$

Class probability:

$$p(y = 1|\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

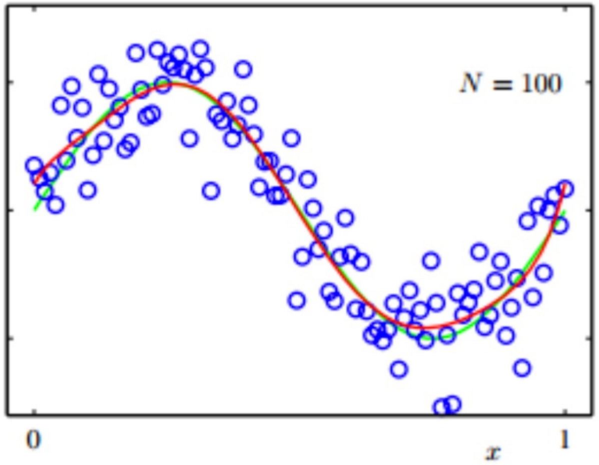


Adding non-linearity: Basis Functions

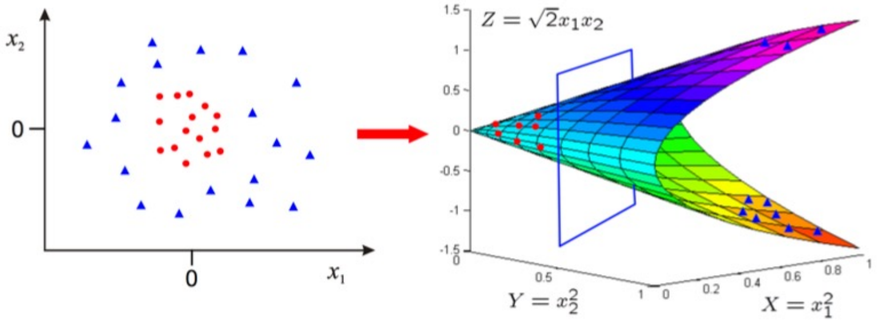


- What if non-linear relationship between **y** and **x**?

Adding non-linearity: Basis Functions



$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



- What if non-linear relationship between **y** and **x**?

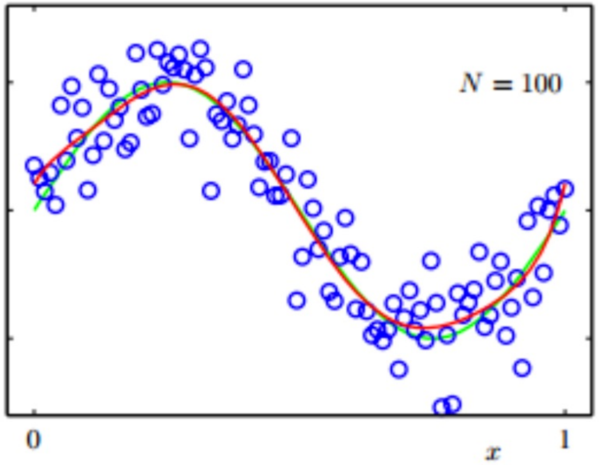
- Choose **basis functions $\phi(x)$** to form new features

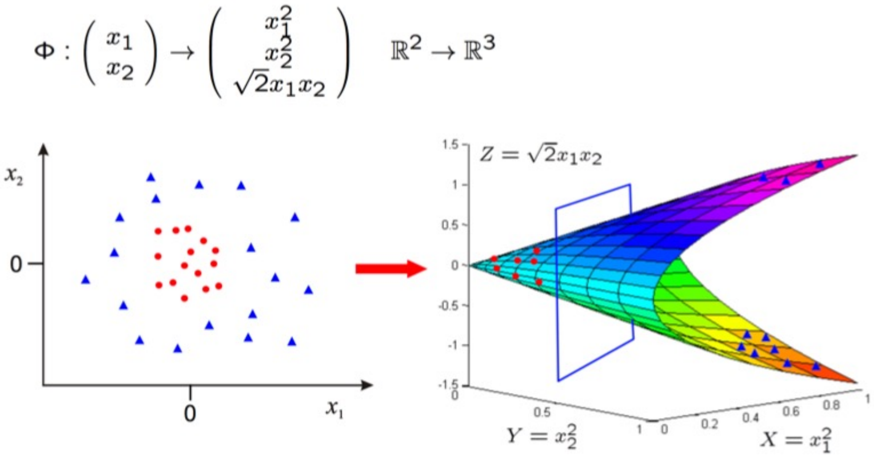
  – Example: Polynomial basis $\phi(x) \sim \{1, x, x^2, x^3, \dots\}$

  – Logistic regression on new features: $h(x; w) = \sigma\big(w^T \phi(x)\big)$

$$p(y = 1 | \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

Adding non-linearity: Basis Functions





$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix}$
$\mathbb{R}^2 \rightarrow \mathbb{R}^3$
x_2
0
0
x_1
$Z = \sqrt{2}x_1x_2$
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
$Y = x_2^2$
0
0.2
0.4
0.6
0.8
1
$X = x_1^2$

- What if non-linear relationship between **y** and **x**?

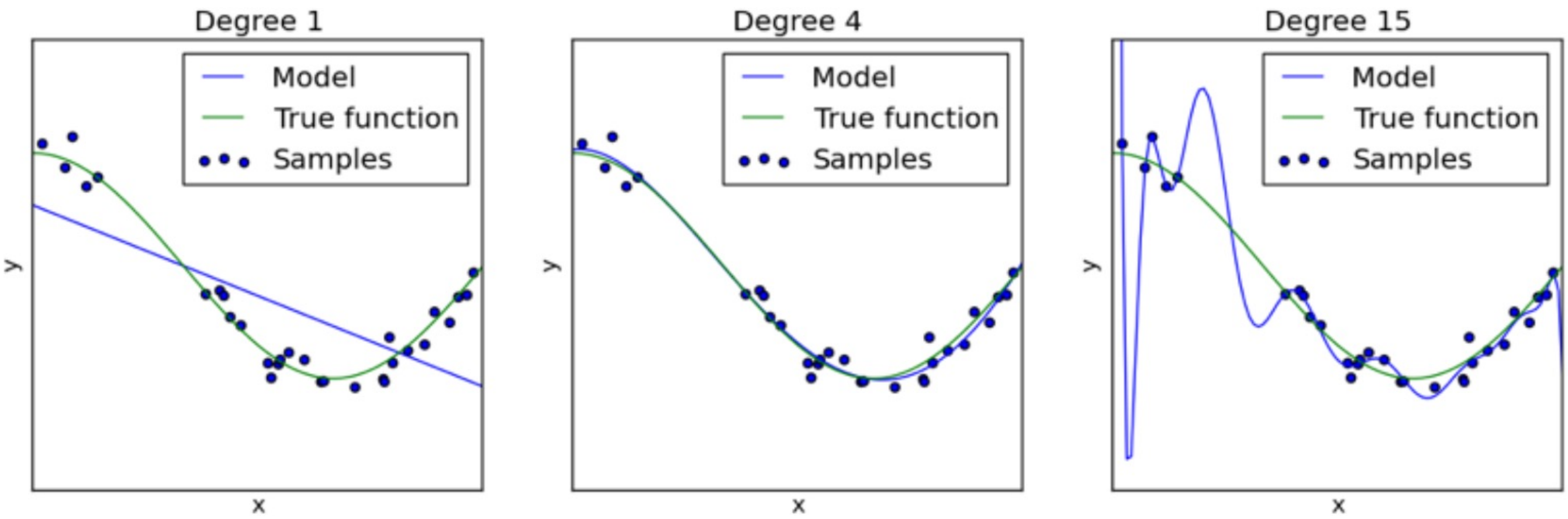
- Choose **basis functions $\boldsymbol{\phi(x)}$** to form new features

  – Example: Polynomial basis $\phi(x) \sim \{1, x, x^2, x^3, \ldots\}$

  – Logistic regression on new features: $h(x; w) = \sigma\big(w^T \phi(x)\big)$

- What basis functions to choose? *Overfit* with too much flexibility?

What is Overfitting



Underfitting

Overfitting

http://scikit-learn.org/

- Models allow us to **generalize** from data

- Different models generalize in different ways

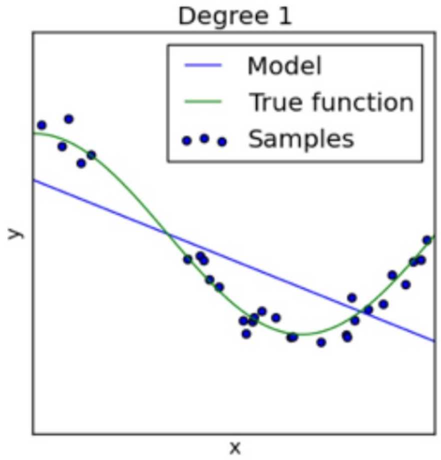
Bias Variance Tradeoff

- generalization error = systematic error + sensitivity of prediction
  (bias) (variance)

Bias Variance Tradeoff

- generalization error = systematic error + sensitivity of prediction
  (bias) (variance)

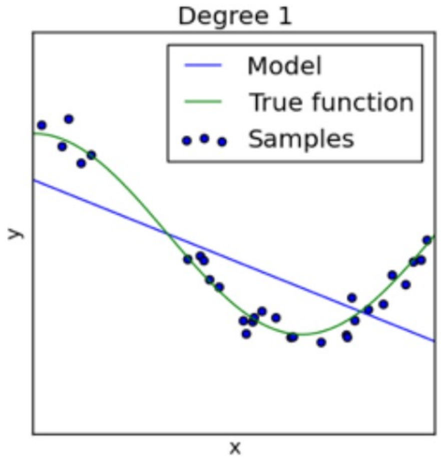
- Simple models under-fit:
  will deviate from data (high bias)
  but will not be influenced by
  peculiarities of data (low variance).



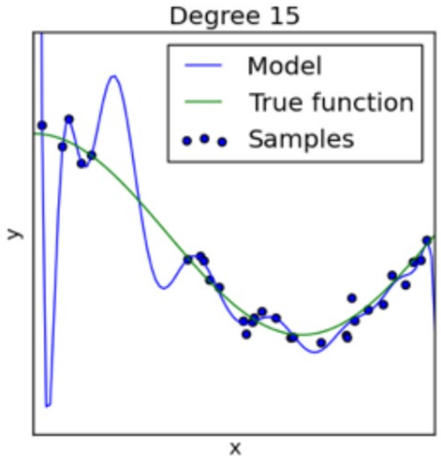
Bias Variance Tradeoff

- generalization error = systematic error + sensitivity of prediction
  (bias) (variance)

- Simple models under-fit:
  will deviate from data (high bias)
  but will not be influenced by
  peculiarities of data (low variance).



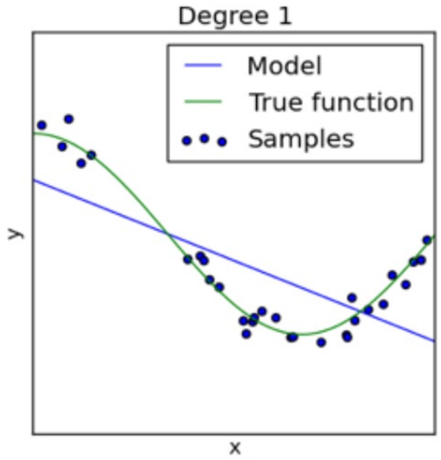
- Complex models over-fit:
  will not deviate systematically from
  data (low bias) but will be very
  sensitive to data (high variance).



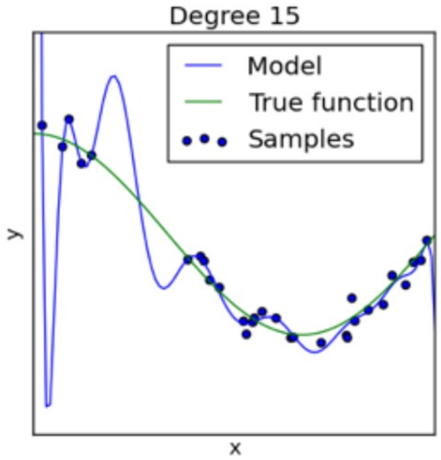
Bias Variance Tradeoff

- generalization error = systematic error + sensitivity of prediction
  (bias) (variance)

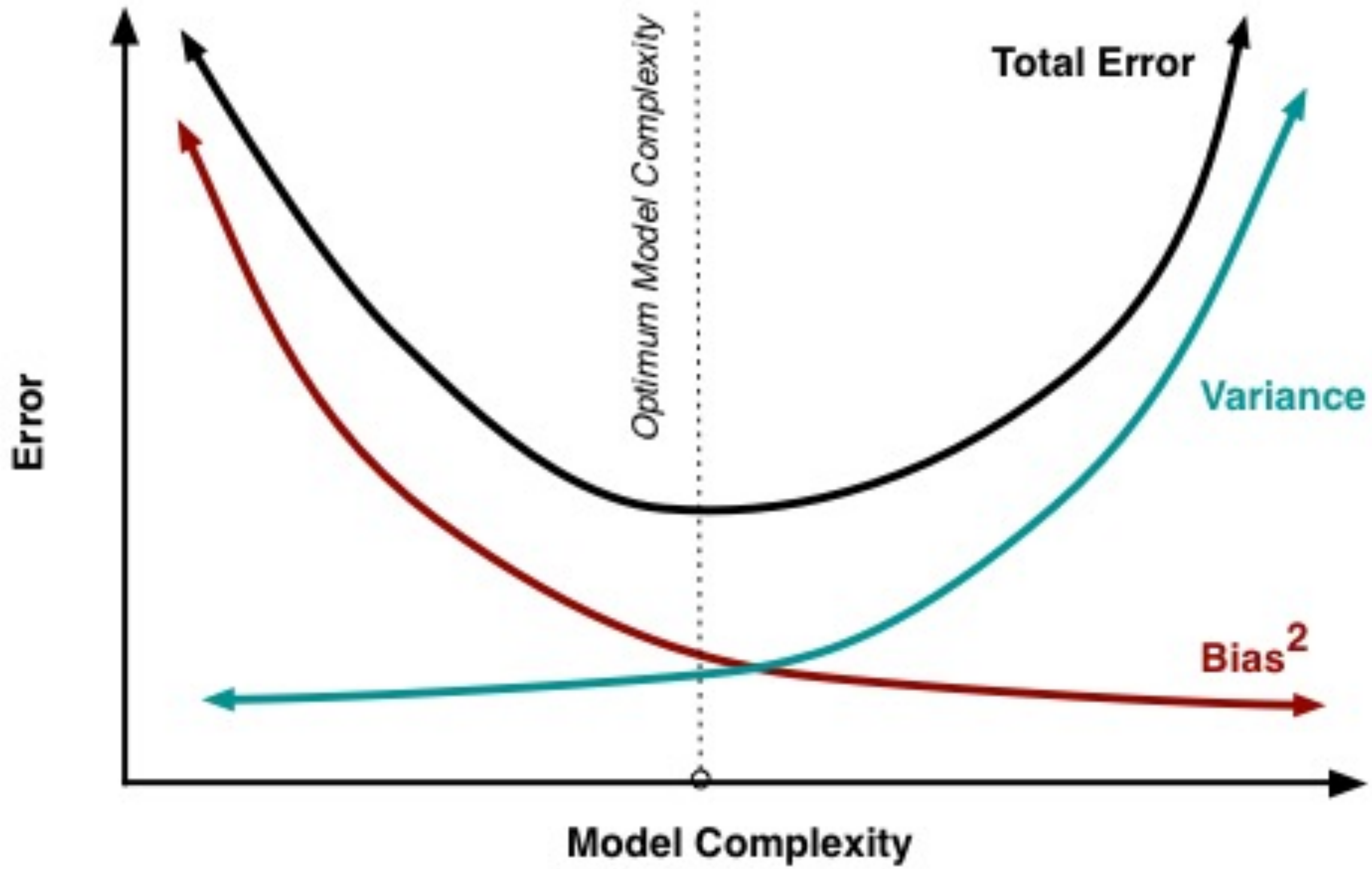
- Simple models under-fit:
  will deviate from data (high bias) but will not be influenced by peculiarities of data (low variance).



- Complex models over-fit:
  will not deviate systematically from data (low bias) but will be very sensitive to data (high variance).
  - **As dataset size grows, can reduce variance! Use more complex model**



Bias Variance Tradeoff

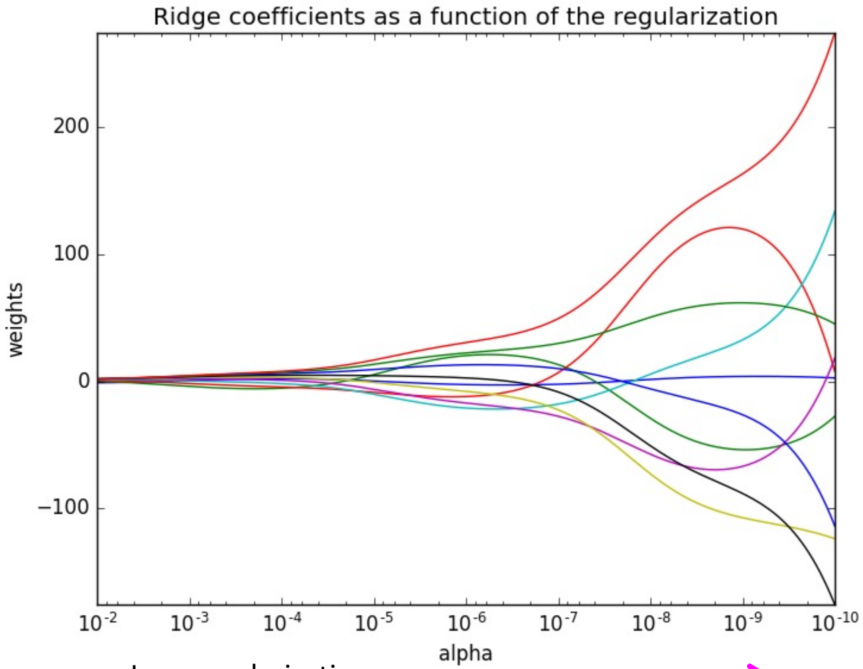


Regularization – Control Complexity

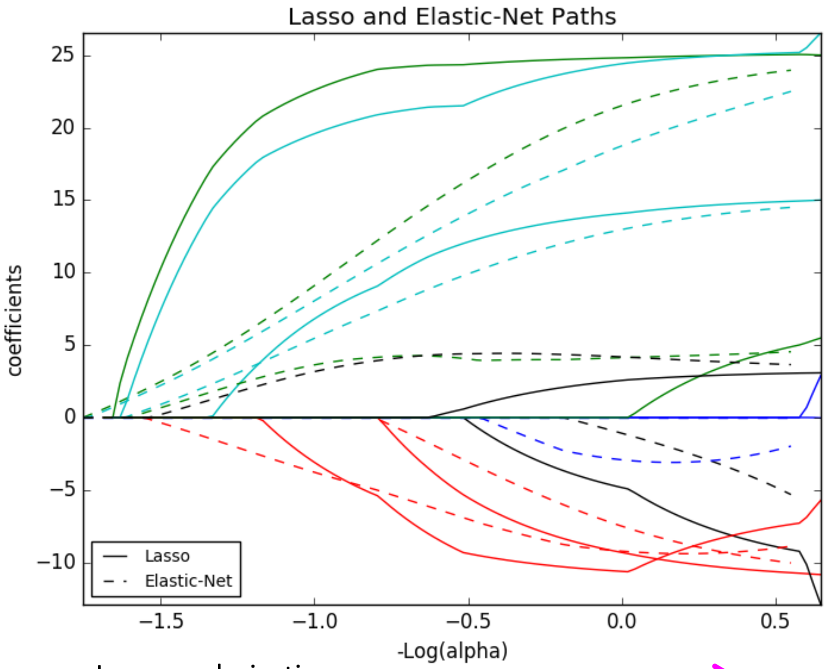
$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^2 + \alpha\Omega(\mathbf{w})$$

$$L2: \quad \Omega(\mathbf{w}) = ||\mathbf{w}||^2$$

$$L1: \quad \Omega(\mathbf{w}) = ||\mathbf{w}||$$



Less regularization →

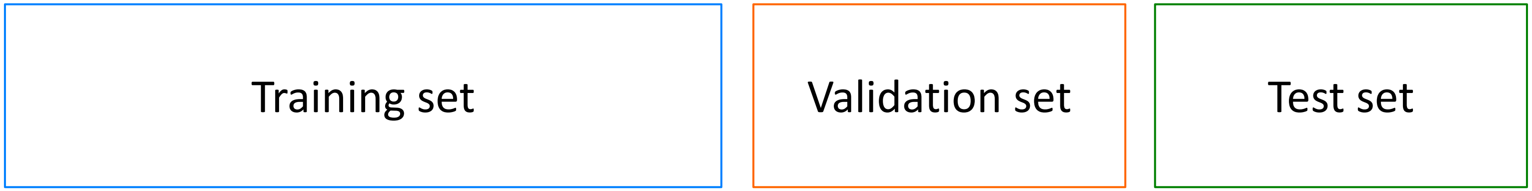


Less regularization →

- L2 keeps weights small, L1 keeps weights sparse!
- But how to choose hyperparameter α?

http://scikit-learn.org/

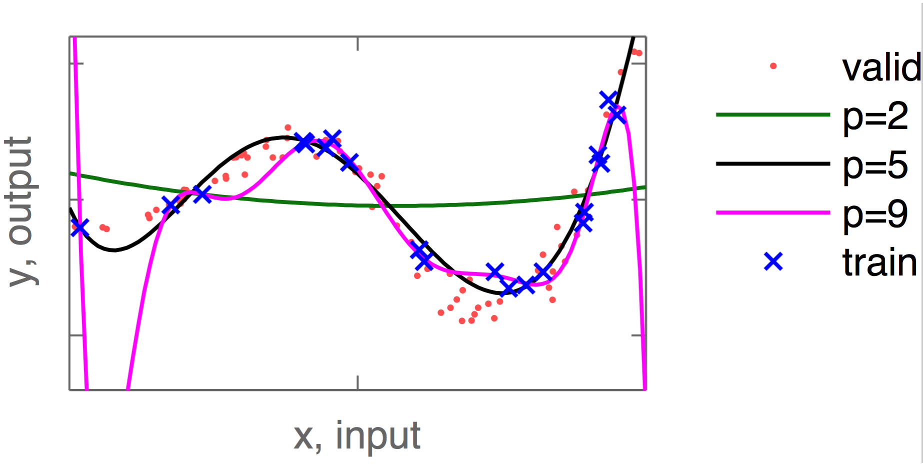
How to Measure Generalization Error?

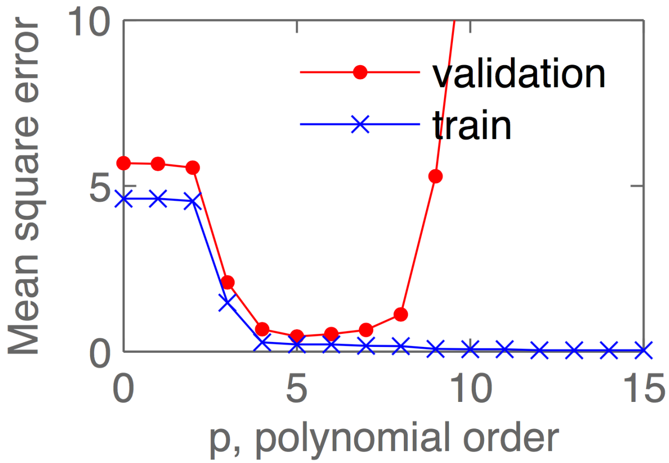

Training set

Validation set

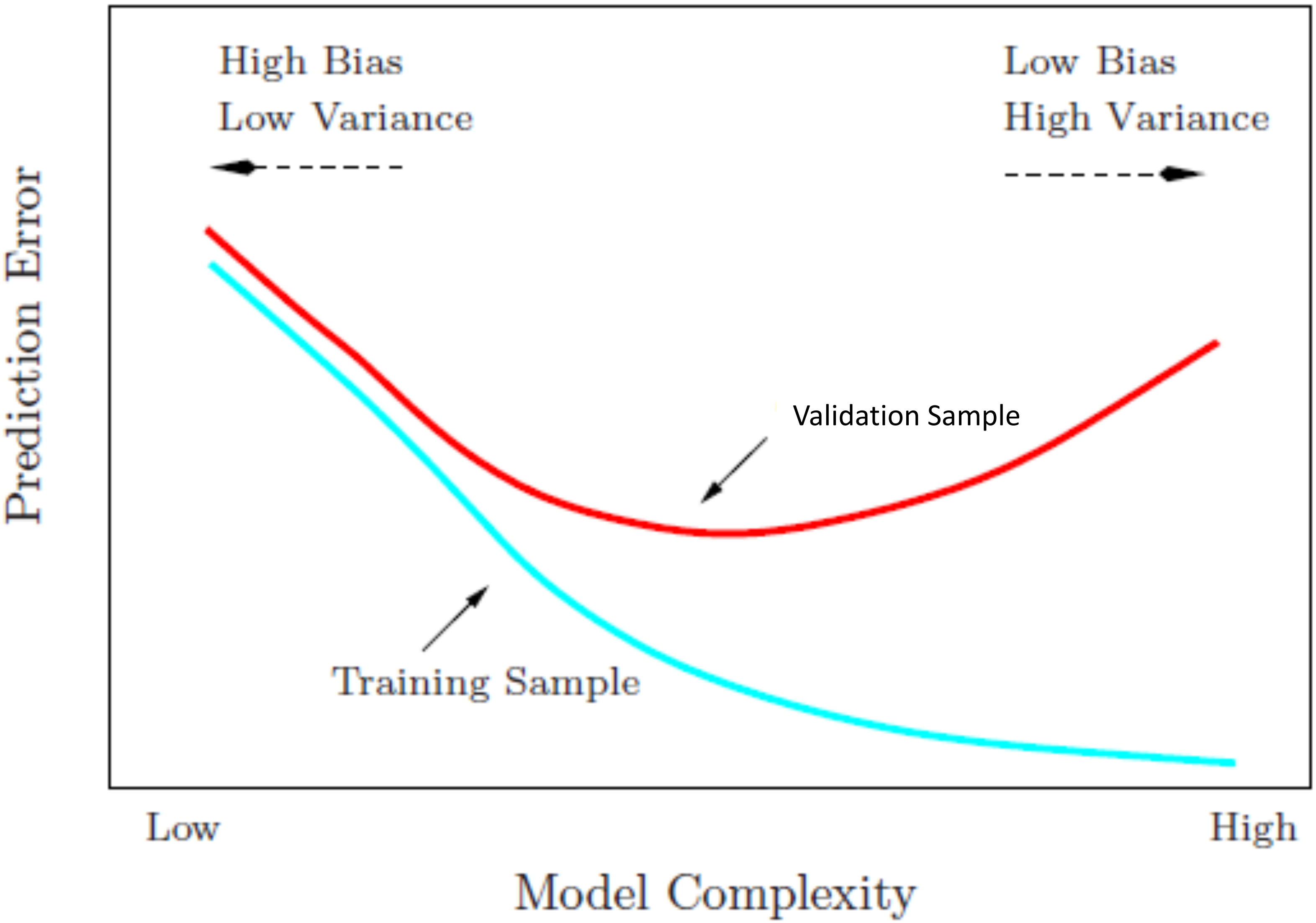
Test set

- Split dataset into multiple parts
- **Training set**
  - Used to fit model parameters
- **Validation set**
  - Used to check performance on independent data and tune hyper parameters
- **Test set**
  - final evaluation of performance after all hyper-parameters fixed
  - Needed since we tune, or "peek", performance with validation set





How to Measure Generalization Error?



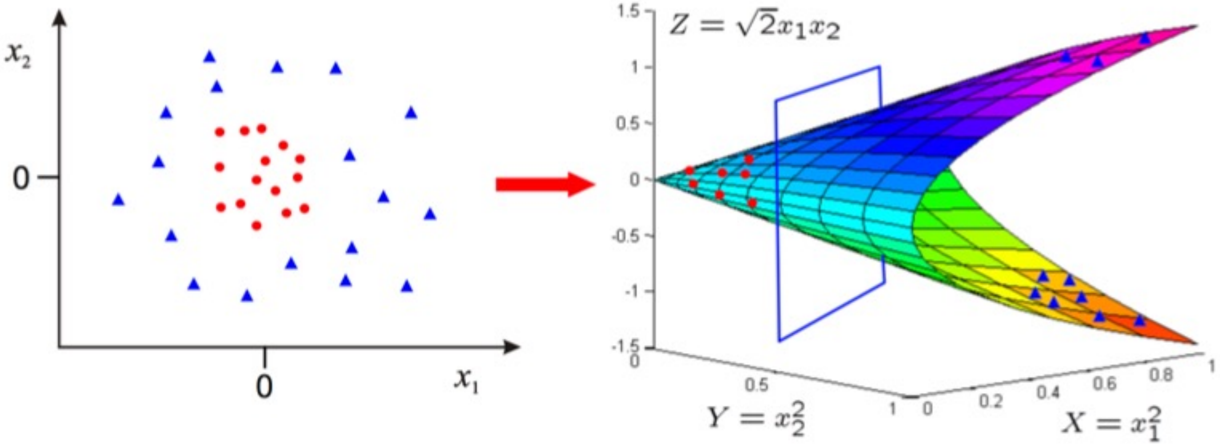
From Logistic Regression to Neural Networks

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\phi(\mathbf{x})}}$$

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \to \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \quad \mathbb{R}^2 \to \mathbb{R}^3$$



Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \dots\}$

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T \phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

- Learn the basis functions directly from data

$$\boldsymbol{\phi}(\boldsymbol{x};\ \boldsymbol{u}) \quad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation

Adding non-linearity

- What if we want a non-linear decision boundary?
  - Choose basis functions, e.g: $\phi(x) \sim \{x^2, \sin(x), \log(x), \dots\}$

$$p(y=1|\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\phi(\mathbf{x})}}$$

- What if we don't know what basis functions we want?

- Learn the basis functions directly from data

$$\boldsymbol{\phi}(\boldsymbol{x};\ \boldsymbol{u}) \quad \mathbb{R}^m \rightarrow \mathbb{R}^d$$

  - Where **u** is a set of parameters for the transformation

  - Combines basis selection & learning→*Representation Learning*
  - Several different approaches, focus here on neural networks
  - Learning / optimization becomes more difficult

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\boldsymbol{x};\ \boldsymbol{u}) = \sigma(\boldsymbol{u}_j^T \boldsymbol{x})$$

Neural Networks

- Define the basis functions $j = \{1 \ldots d\}$

$$\phi_j(\boldsymbol{x};\ \boldsymbol{u}) = \sigma(\boldsymbol{u}_j^T \boldsymbol{x})$$

- Put all $\boldsymbol{u}_j \in \mathbb{R}^{1\times m}$ vectors into matrix $\boldsymbol{U}$

$$\phi(\boldsymbol{x};\ \boldsymbol{U}) = \sigma(\boldsymbol{U}\boldsymbol{x}) = \begin{bmatrix} \sigma(u_1^T x) \\ \sigma(u_2^T x) \\ \vdots \\ \sigma(u_d^T x) \end{bmatrix} \quad \in \mathbb{R}^d$$

– σ is a point-wise non-linearity acting on each vector element

Neural Networks

- Define the basis functions $j \;=\; \{1 \dots d\}$

$$\phi_j(\boldsymbol{x};\ \boldsymbol{u}) \;= \sigma(\boldsymbol{u}_j^T \boldsymbol{x})$$

- Put all $\boldsymbol{u}_j \in \mathbb{R}^{1\times m}$ vectors into matrix $\boldsymbol{U}$

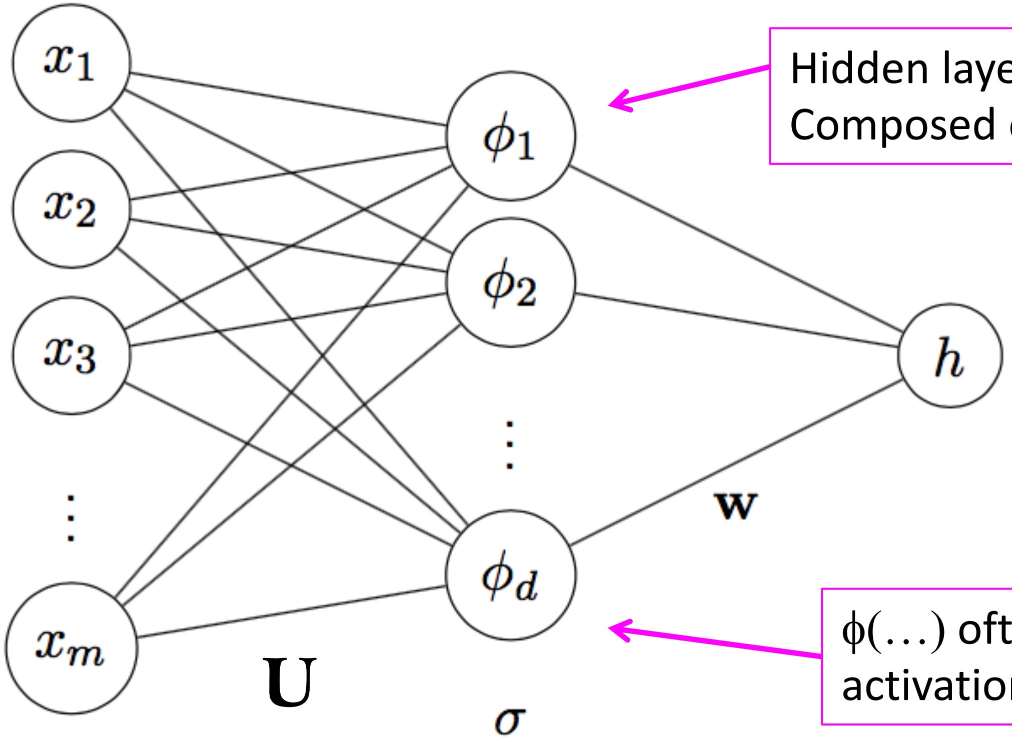
$$\phi(\boldsymbol{x};\ \boldsymbol{U}) = \sigma(\boldsymbol{U}\boldsymbol{x}) \;= \begin{bmatrix} \sigma(u_1^T x) \\ \sigma(u_2^T x) \\ \vdots \\ \sigma(u_d^T x) \end{bmatrix} \quad \in \mathbb{R}^d$$

  – σ is a point-wise non-linearity acting on each vector element

- Full model becomes

$$h(\boldsymbol{x};\ \boldsymbol{w}, \boldsymbol{U}) \;= w^T \phi(\boldsymbol{x};\ \boldsymbol{U})$$

Feed Forward Neural Network



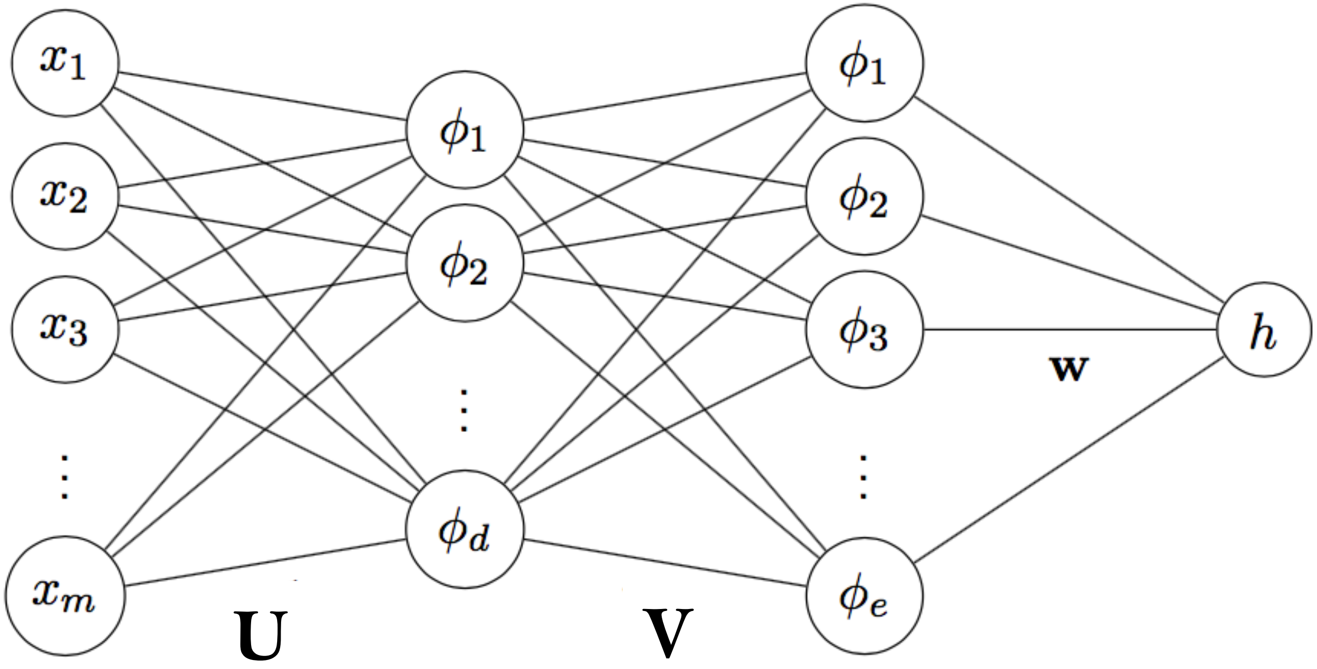
Hidden layer
Composed of *neurons*

ϕ(…) often called the activation function

$$\phi(\mathbf{x}) = \sigma(\mathbf{U}\mathbf{x})$$

$$h(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$$

Multi-layer Neural Network



- Multilayer NN
  - – Each layer adapts basis functions based on previous layer

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1|\mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1 | \mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

Neural Network Optimization Problem

- Neural Network Model: $h(\mathbf{x}) = \mathbf{w}^T \sigma(\mathbf{U}\mathbf{x})$

- **Classification**: Cross-entropy loss function

$$p_i = p(y_i = 1 | \mathbf{x}_i) = \sigma(h(\mathbf{x}_i))$$

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)$$

- **Regression**: Square error loss function

$$L(\mathbf{w}, \mathbf{U}) = \frac{1}{2} \sum_i (y_i - h(\mathbf{x}_i))^2$$

- Minimize loss with respect to weights **w, U**

Minimizing loss with gradient descent:

- Parameter update:

$$w \leftarrow w - \eta \frac{\partial L(w, U)}{\partial w}$$

$$U \leftarrow U - \eta \frac{\partial L(w, U)}{\partial U}$$

- How to compute gradients?

Chain Rule – Symbolic Differentiation

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(\sigma(h(\mathbf{x}_i))) + (1 - y_i) \ln(1 - \sigma(h(\mathbf{x}_i)))$$

- Derivative of sigmoid: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- Chain rule to compute gradient w.r.t. **w**

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial \mathbf{w}} = \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))\sigma(\mathbf{U}\mathbf{x}) + (1 - y_i)\sigma(h(\mathbf{x}))\sigma(\mathbf{U}\mathbf{x}_i)$$

- Chain rule to compute gradient w.r.t. **u**$_j$

$$\frac{\partial L}{\partial \mathbf{u}_j} = \frac{\partial L}{\partial h} \frac{\partial h}{\partial \sigma} \frac{\partial \sigma}{\partial \mathbf{u}_j} =$$

$$= \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))w_j \sigma(\mathbf{u}_j \mathbf{x}_i)(1 - \sigma(\mathbf{u}_j \mathbf{x}_i))\mathbf{x}_i$$

$$+ (1 - y_i)\sigma(h(\mathbf{x}_i))w_j \sigma(\mathbf{u}_j \mathbf{x}_i)(1 - \sigma(\mathbf{u}_j \mathbf{x}_i))\mathbf{x}_i$$

Chain Rule – Symbolic Differentiation

$$L(\mathbf{w}, \mathbf{U}) = -\sum_i y_i \ln(\sigma(h(\mathbf{x}_i))) + (1 - y_i) \ln(1 - \sigma(h(\mathbf{x}_i)))$$

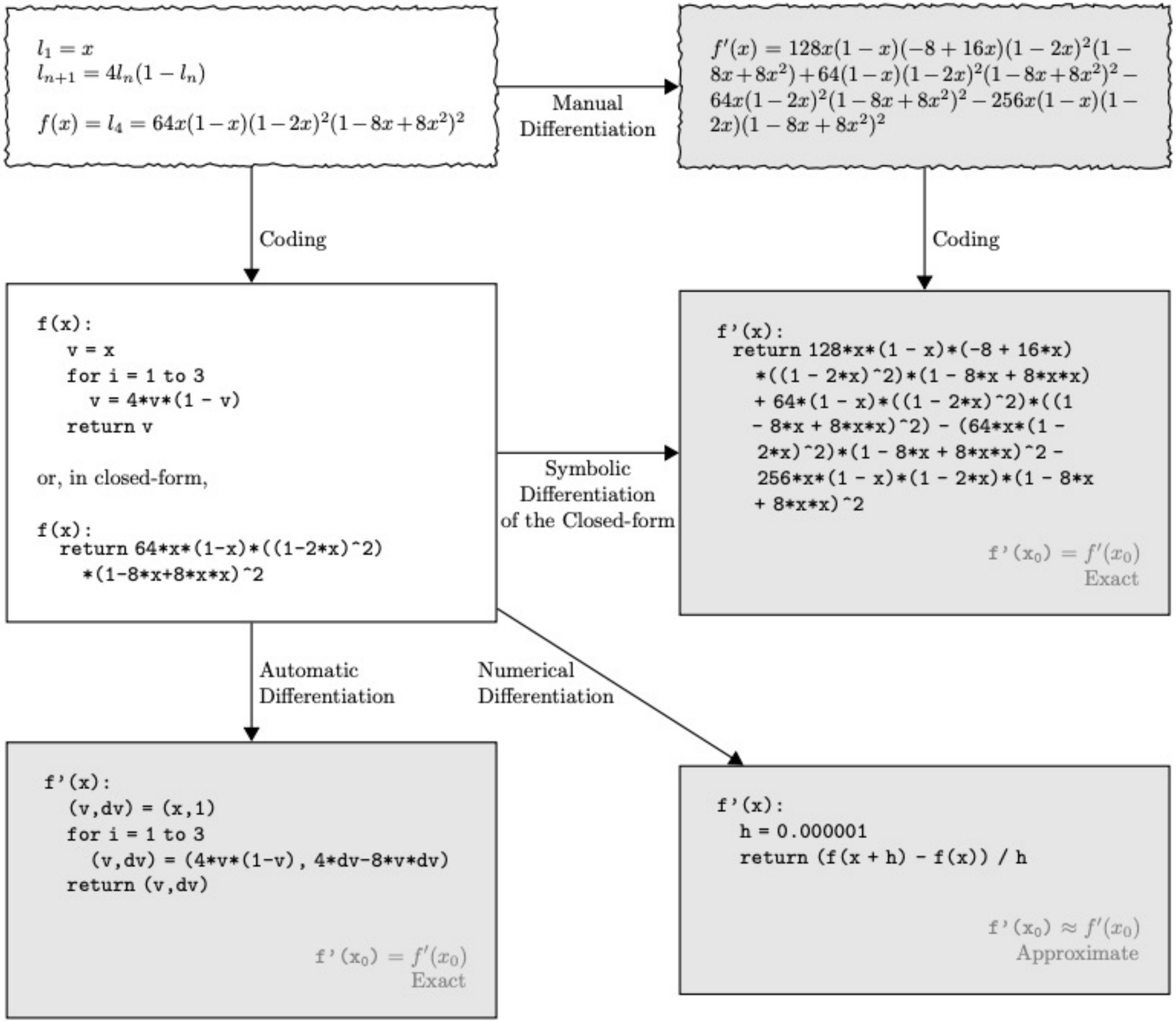
- Derivative of sigmoid: $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$
- Chain rule to compute gradient w.r.t. **w**

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \mathbf{w}} = \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))\sigma(\mathbf{U}\mathbf{x}) + (1 - y_i)\sigma(h(\mathbf{x}))\sigma(\mathbf{U}\mathbf{x}_i)$$

- Chain rule to compute gradient w.r.t. $\mathbf{u}_j$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{u}_j} = \frac{\partial L}{\partial h}\frac{\partial h}{\partial \sigma}\frac{\partial \sigma}{\partial \mathbf{u}_j} = \\ = \sum_i y_i(1 - \sigma(h(\mathbf{x}_i)))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i \\ + (1 - y_i)\sigma(h(\mathbf{x}_i))w_j\sigma(\mathbf{u}_j\mathbf{x}_i)(1 - \sigma(\mathbf{u}_j\mathbf{x}_i))\mathbf{x}_i\end{aligned}$$

Differentiation in Code



$l_1 = x$
$l_{n+1} = 4l_n(1-l_n)$

$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$

Manual Differentiation

$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$

Coding

```
f(x):
   v = x
   for i = 1 to 3
     v = 4*v*(1 - v)
   return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

Coding

```
f'(x):
 return 128*x*(1 - x)*(-8 + 16*x)
   *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
   + 64*(1 - x)*((1 - 2*x)^2)*((1
   - 8*x + 8*x*x)^2) - (64*x*(1 -
   2*x)^2)*(1 - 8*x + 8*x*x)^2 -
   256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
   + 8*x*x)^2
```

$\texttt{f'(x}_0\texttt{)} = f'(x_0)$
Exact

Symbolic Differentiation of the Closed-form

Automatic Differentiation

Numerical Differentiation

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

$\texttt{f'(x}_0\texttt{)} = f'(x_0)$
Exact

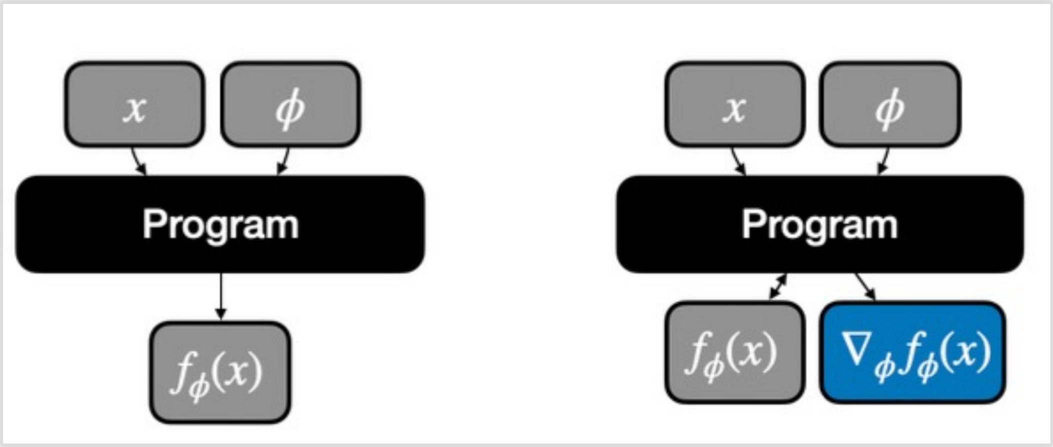
```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

$\texttt{f'(x}_0\texttt{)} \approx f'(x_0)$
Approximate

Baydin, Pearlmutter, Radul, Siskind. 2018. “Automatic Differentiation in Machine Learning: a Survey.” Journal of Machine Learning Research (**JMLR**)

Automatic Differentiation

Exact derivatives for gradient-based optimization come from running **differentiable code** via **automatic differentiation**



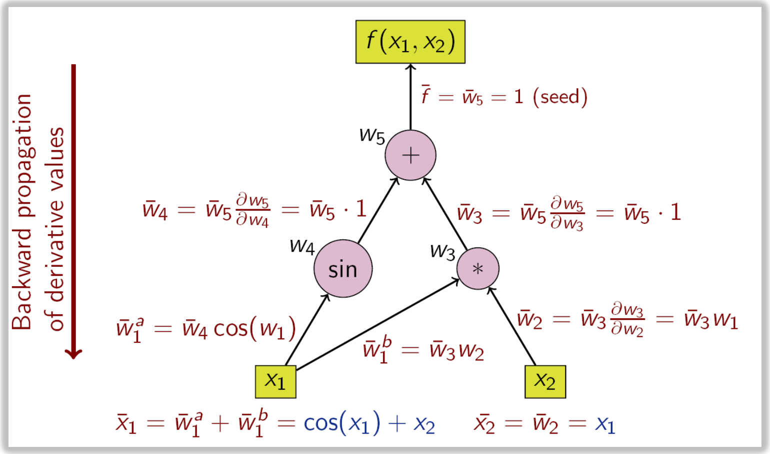

Backward propagation of derivative values
$f(x_1, x_2)$
$\bar{f} = \bar{w}_5 = 1$ (seed)
w_5
$+$
$\bar{w}_4 = \bar{w}_5 \frac{\partial w_5}{\partial w_4} = \bar{w}_5 \cdot 1$
$\bar{w}_3 = \bar{w}_5 \frac{\partial w_5}{\partial w_3} = \bar{w}_5 \cdot 1$
w_4
sin
w_3
$*$
$\bar{w}_1^a = \bar{w}_4 \cos(w_1)$
$\bar{w}_2 = \bar{w}_3 \frac{\partial w_3}{\partial w_2} = \bar{w}_3 w_1$
$\bar{w}_1^b = \bar{w}_3 w_2$
x_1
x_2
$\bar{x}_1 = \bar{w}_1^a + \bar{w}_1^b = \cos(x_1) + x_2$
$\bar{x}_2 = \bar{w}_2 = x_1$

Image credit: Wikipedia

TensorFlow

PyTorch

Backpropagation – Reverse Mode AD

- Loss function composed of layers of nonlinearity

$$L\left(\phi^{N}\left(\dots\phi^{1}(x)\right)\right)$$

- Forward step (f-prop)
  - Compute and save intermediate computations

$$\phi^{N}\left(\dots\phi^{1}(x)\right)$$

- Backward step (b-prop)

$$\frac{\partial L}{\partial \phi^{a}} = \sum_{j} \frac{\partial \phi_{j}^{(a+1)}}{\partial \phi_{j}^{a}} \frac{\partial L}{\partial \phi_{j}^{(a+1)}}$$

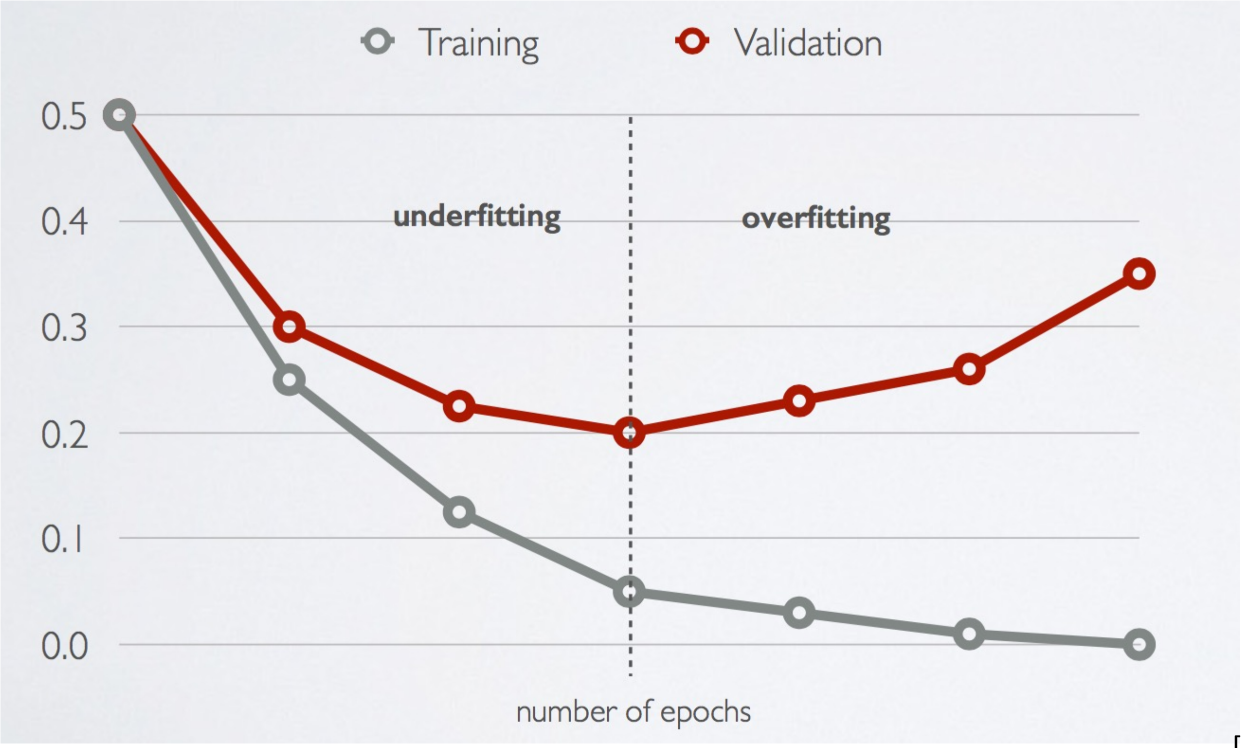
- Compute parameter gradients

$$\frac{\partial L}{\partial \mathbf{w}^{a}} = \sum_{j} \frac{\partial \phi_{j}^{a}}{\partial \mathbf{w}^{a}} \frac{\partial L}{\partial \phi_{j}^{a}}$$

Training

- Repeat gradient update of weights to reduce loss
  - Each iteration through dataset is called an epoch

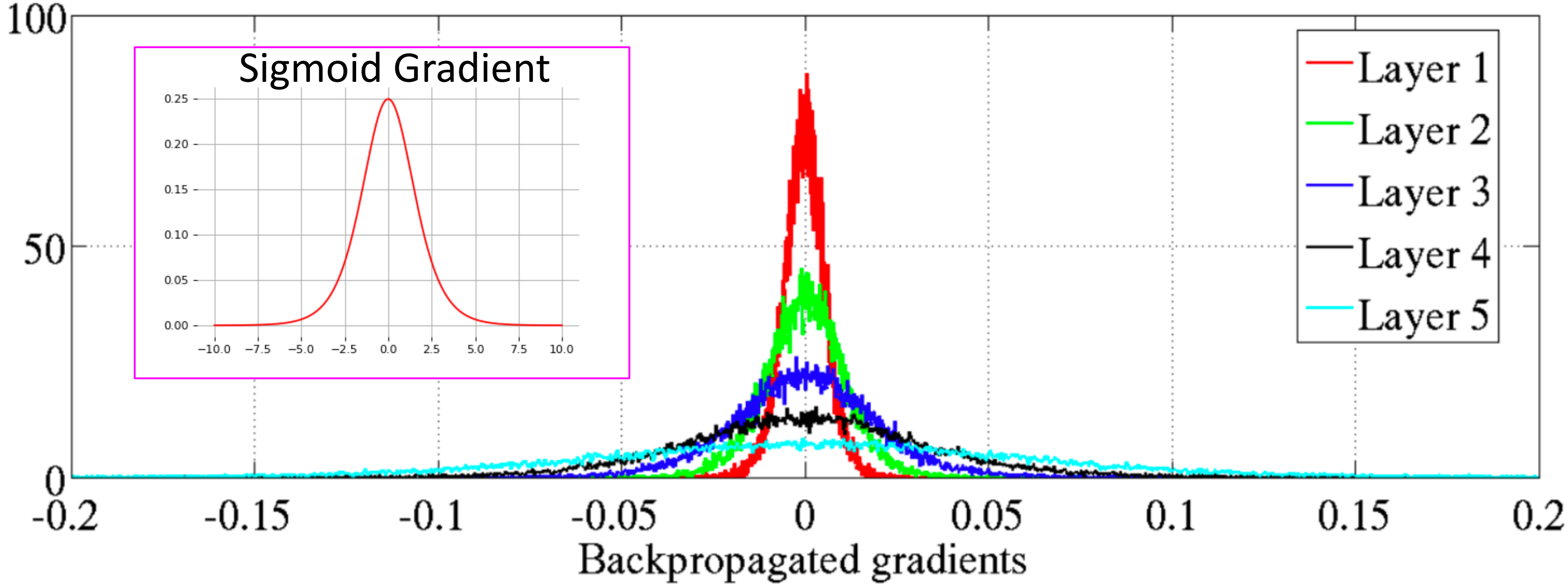
- Use validation set to examine for overtraining, and determine when to stop training



[graphic from H. Larochelle]

Vanishing Gradients

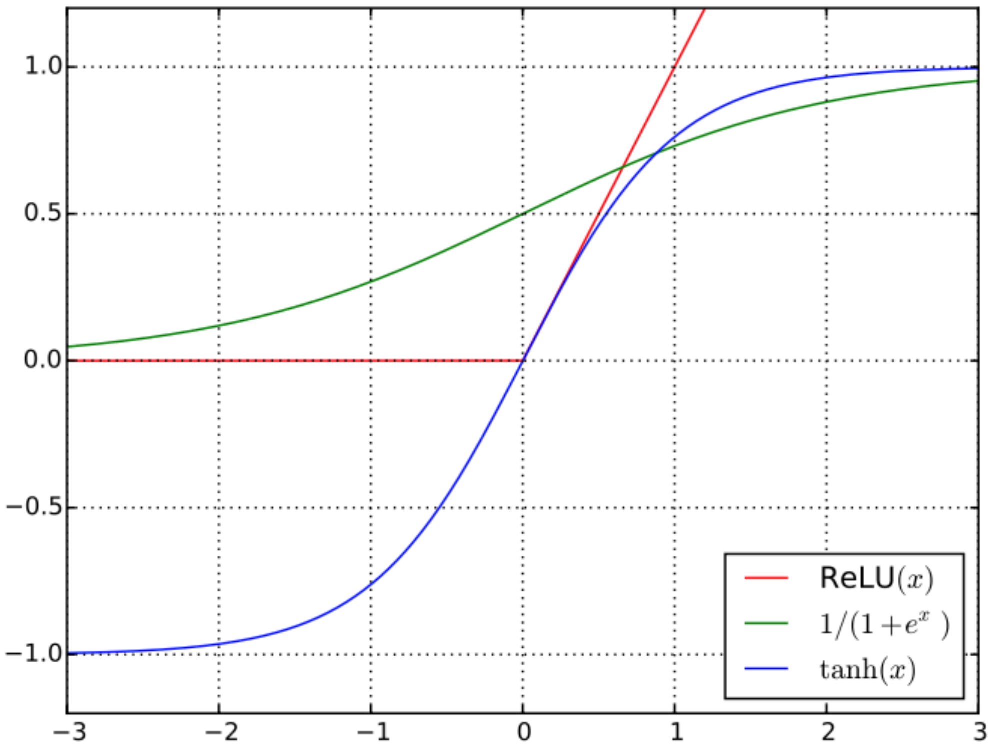
- Major challenge in DL: Vanishing Gradients
- Small gradients slow down / block, stochastic gradient descent → Limits ability to learn!



Backpropagated gradients normalized histograms (Glorot and Bengio, 2010).
Gradients for layers far from the output vanish to zero.

Slide credit: G. Louppe

Activation Functions



- **Vanishing gradient problem**
  - Derivative of sigmoid
    Nearly 0 when x is far from 0!
  - Can make gradient descent hard!

- **Rectified Linear Unit (ReLU)**
  - ReLU(x) = max{0, x}
  - Derivative is constant!

$$\frac{\partial \mathrm{Re}LU(x)}{\partial x} = \begin{cases} 1 & when\ x > 0 \\ 0 & otherwise \end{cases}$$

  - ReLU gradient doesn't vanish

Neural Network Decision Boundaries

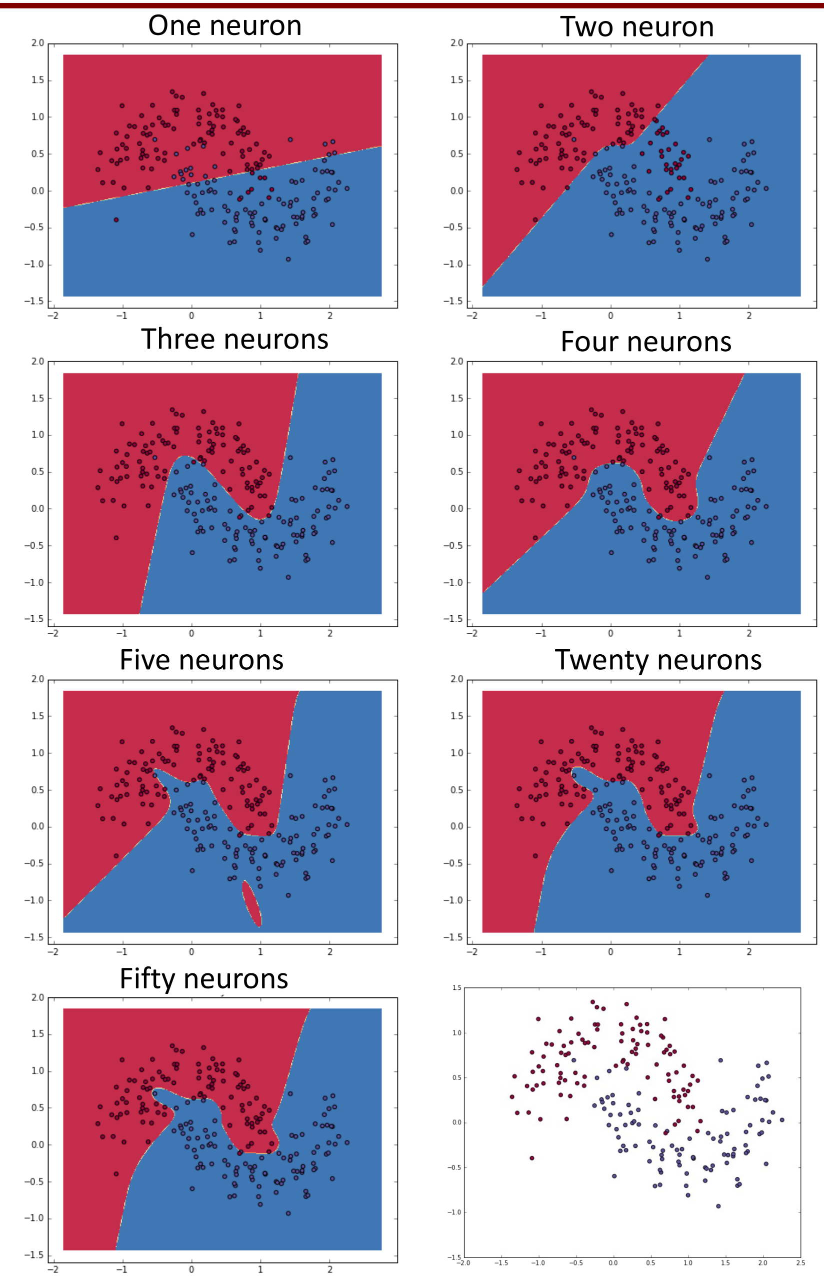
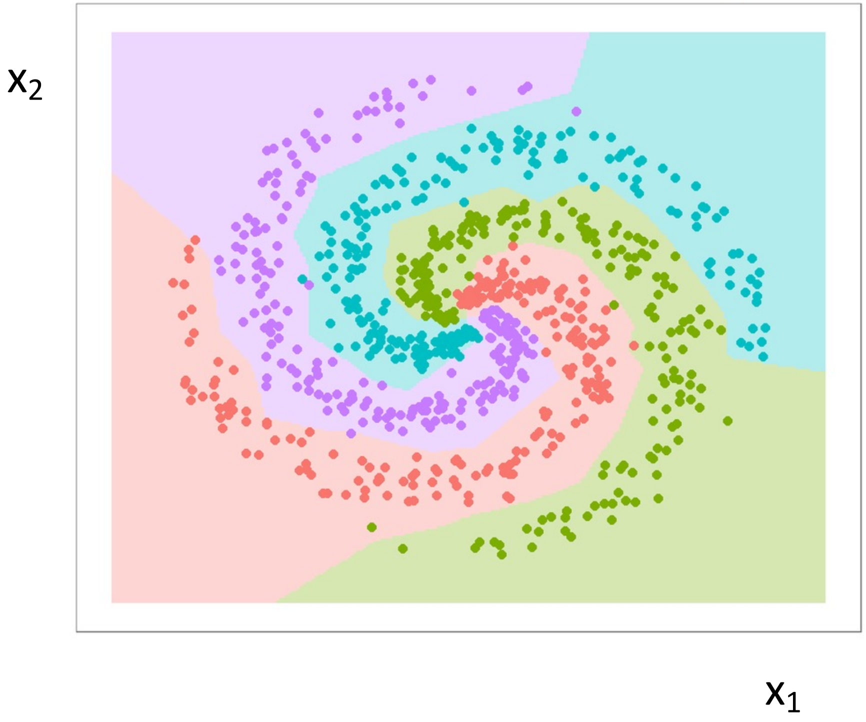


Image source

4-class classification
2-hidden layer NN
ReLU activations
L2 norm regularization



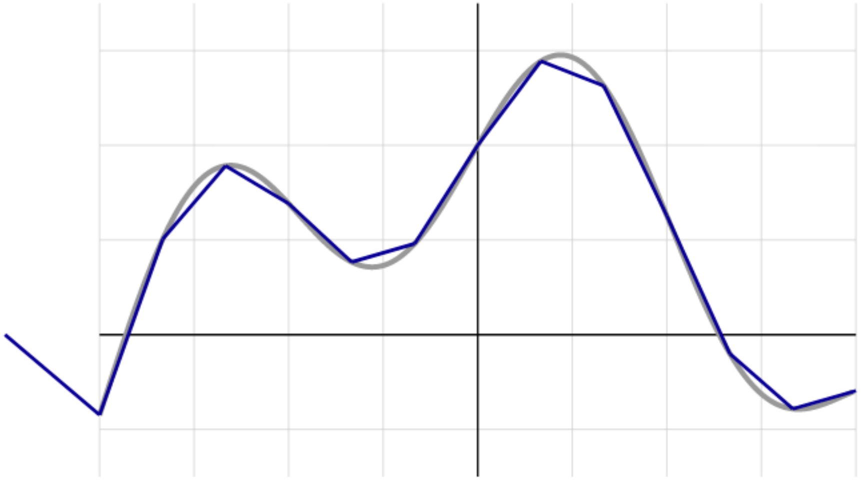
2-class classification
1-hidden layer NN
L2 norm regularization

Image source

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

$$f(x) = \sigma(w_1 x + b_1) + \sigma(w_2 x + b_2) + \sigma(w_3 x + b_3) + \ldots$$



Fleuret, Deep Learning Course

Universal approximation theorem

- Feed-forward neural network with a single hidden layer containing a finite number of non-linear neurons (ReLU, Sigmoid, and others) can approximate continuous functions arbitrarily well on a compact space of $\mathbb{R}^n$

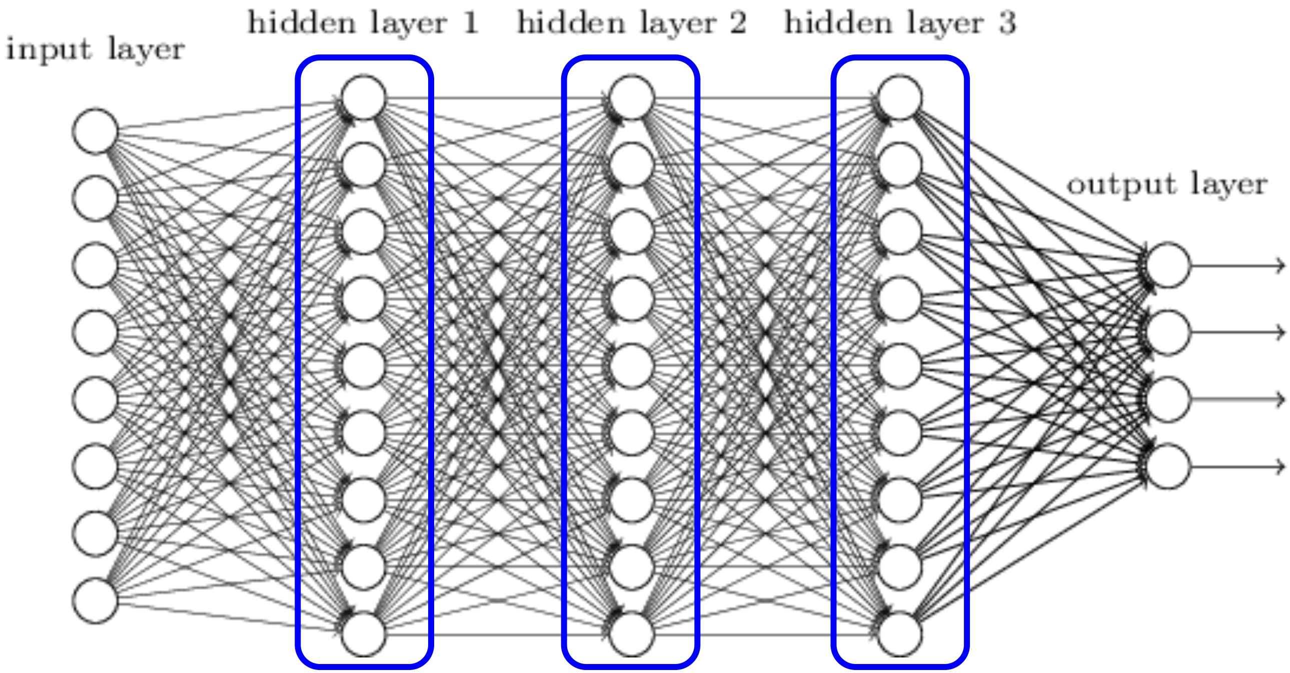
- Better approximation requires larger hidden layer, this theorem says nothing about relation between the two.

- Can make training error as low as we want by using a larger hidden layer. Result states nothing about test error

- Doesn't say how to find parameters for this approximation

Fleuret, Deep Learning Course

Deep Neural Networks

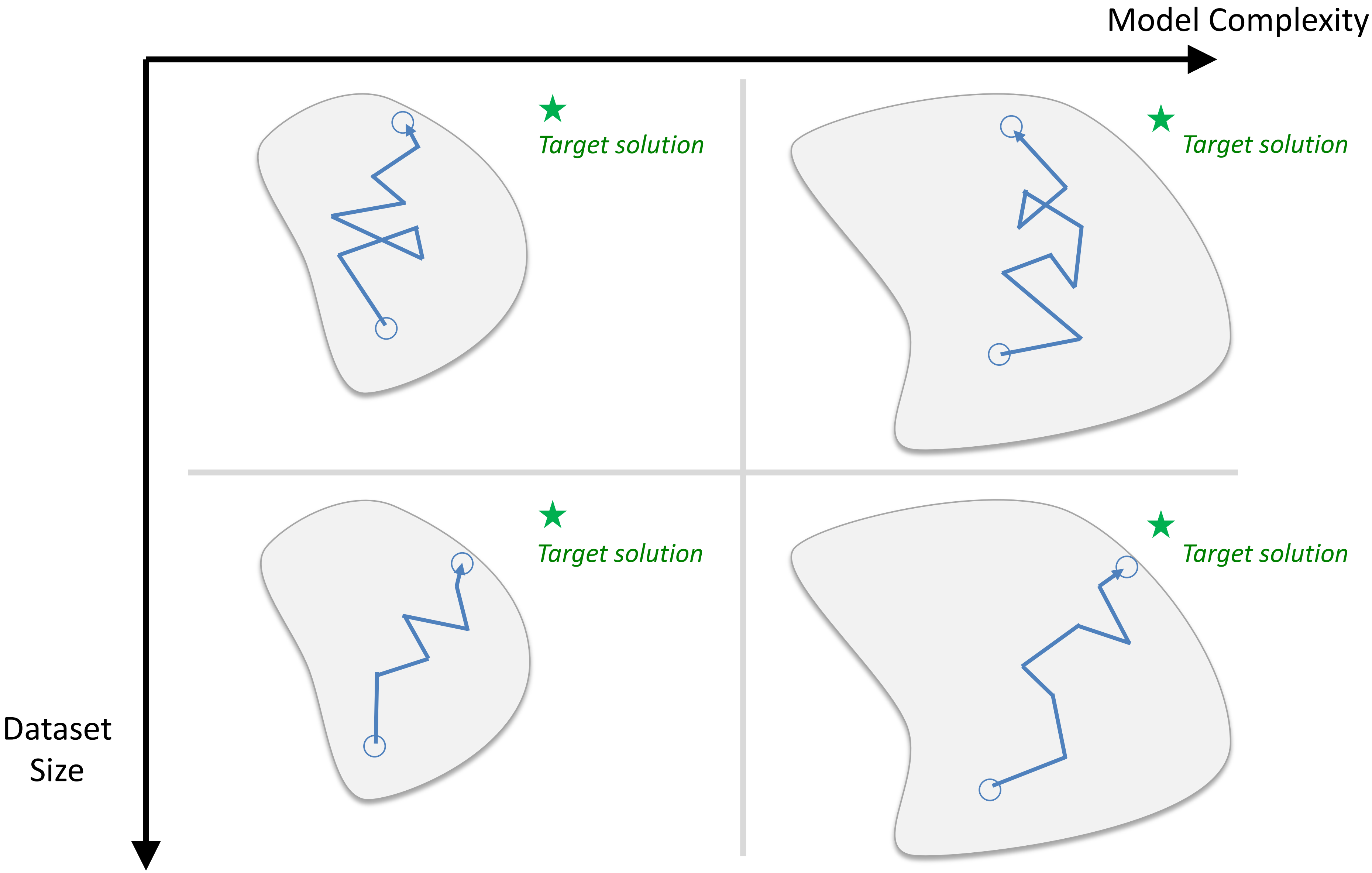


- As data complexity grows, need exponentially large number of neurons in a single-layer network to capture all structure in data

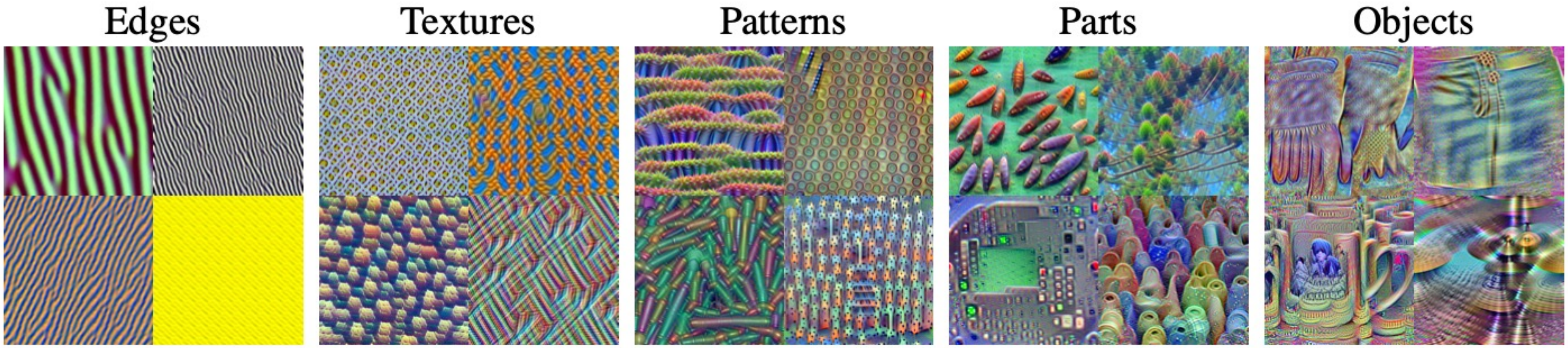
- Deep networks ***factorize the learning*** of structure across layers

- Difficult to train, recently possible with large datasets, fast computing (GPU/TPU) & new training algs. / network structures

More Complex Models – Bigger Search Space
More Data – Find Better Solutions



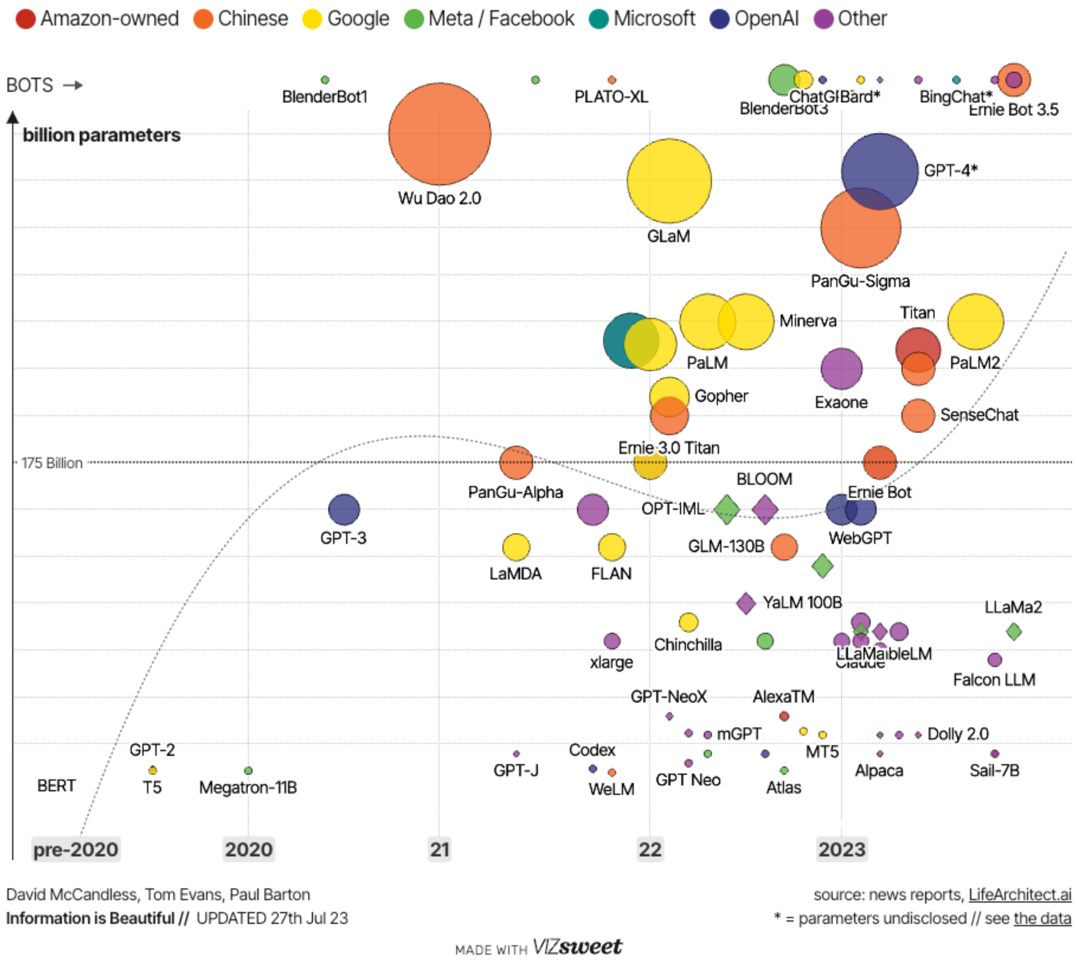
Hierarchical Learning of Features



Depth

2212.06727

Benefits of Depth



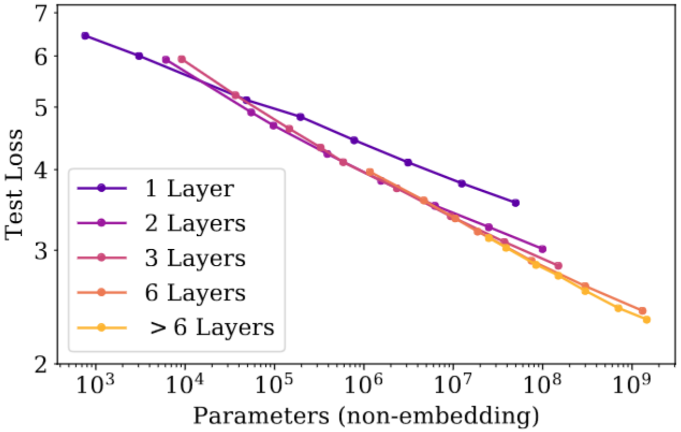
David McCandless, Tom Evans, Paul Barton
Information is Beautiful // UPDATED 27th Jul 23

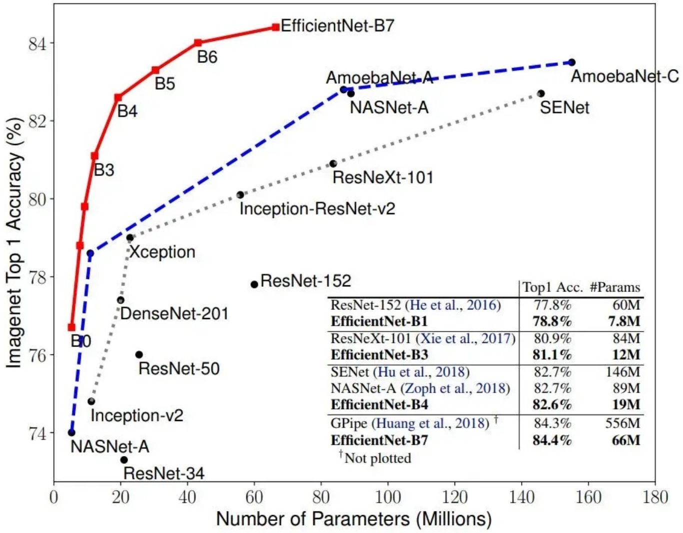
source: news reports, LifeArchitect.ai
* = parameters undisclosed // see the data

MADE WITH VIZsweet

Image credit: D. McCandless, T. Evans, P. Barton

2001.08361





	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

† Not plotted

1905.11946

Neural Network Zoo

- Structure of the networks, and the node connectivity can be adapted for problem at hand

- Moving inductive bias from feature engineering to model design

  - *Inductive bias*:
    Knowledge about the problem

  - *Feature engineering*:
    Hand crafted variables

  - *Model design*:
    The data representation and the structure of the machine learning model / network

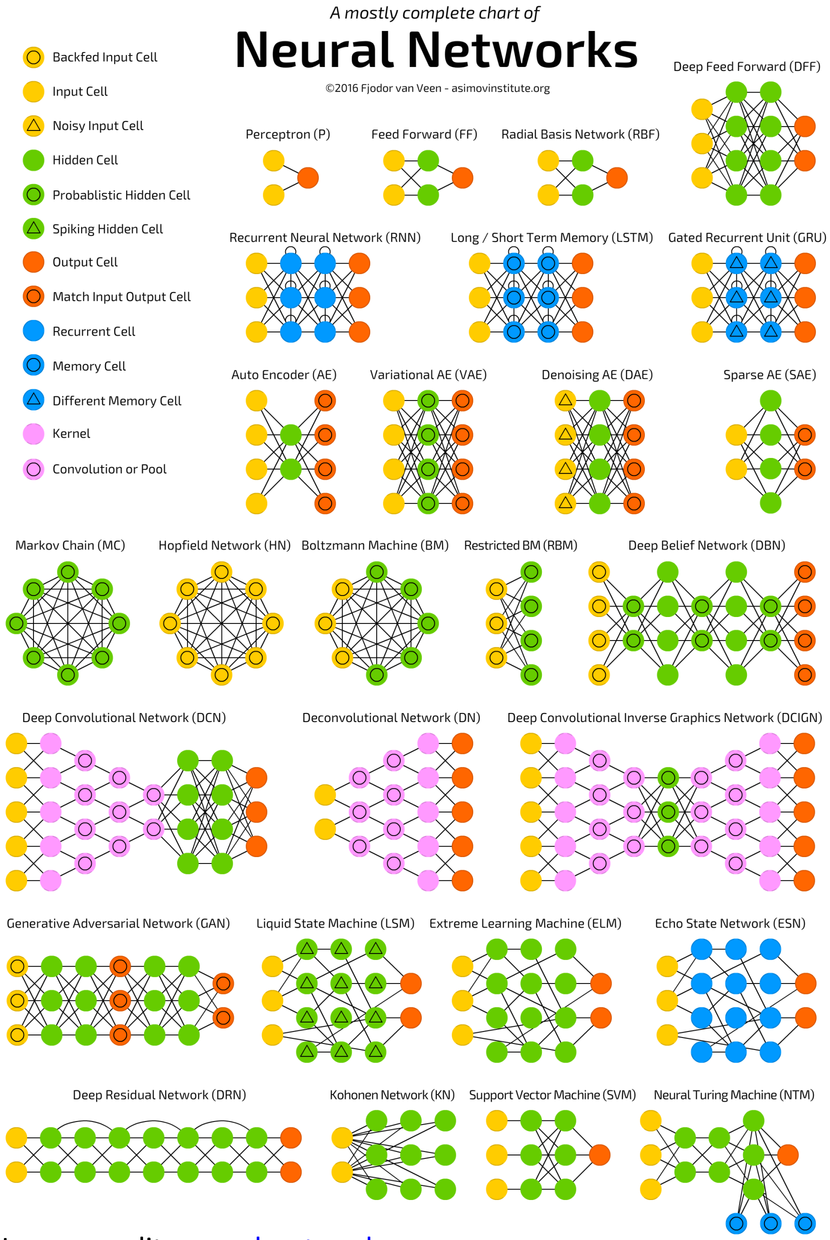

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Image credit: neural-network-zoo

Neural Network Zoo – "Optimization" Perspective

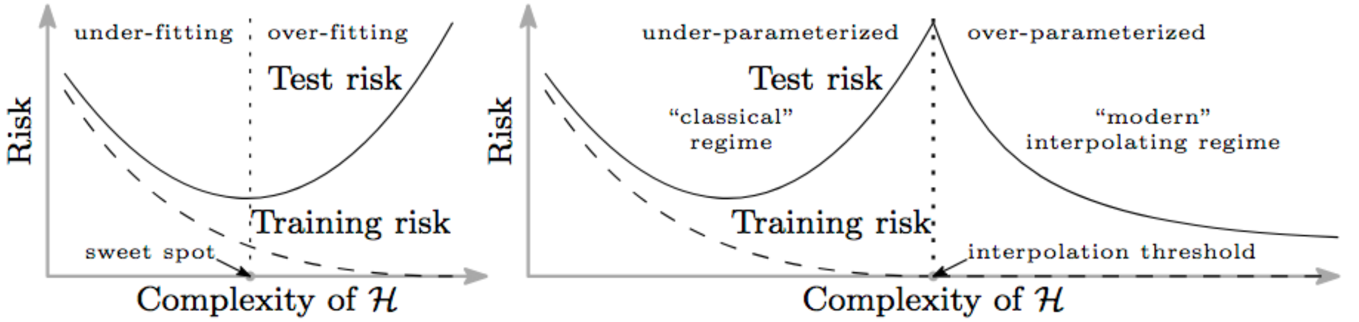
- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

Neural Network Zoo – “Optimization” Perspective

- A single layer network may need a width exponential in D to approximate a depth-D network’s output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

Belkin et. al. 2018



(a) U-shaped “bias-variance” risk curve

(b) “double descent” risk curve

Figure 1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (b) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from using high complexity function classes (i.e., the “modern” interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Fleuret, Deep Learning Course

Neural Network Zoo – "Optimization" Perspective

- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction
  - But we must control that:
    - Gradients don't vanish
    - Gradient amplitude is homogeneous across network
    - Gradients are under control when weights change

Fleuret, Deep Learning Course

Neural Network Zoo – “Optimization” Perspective

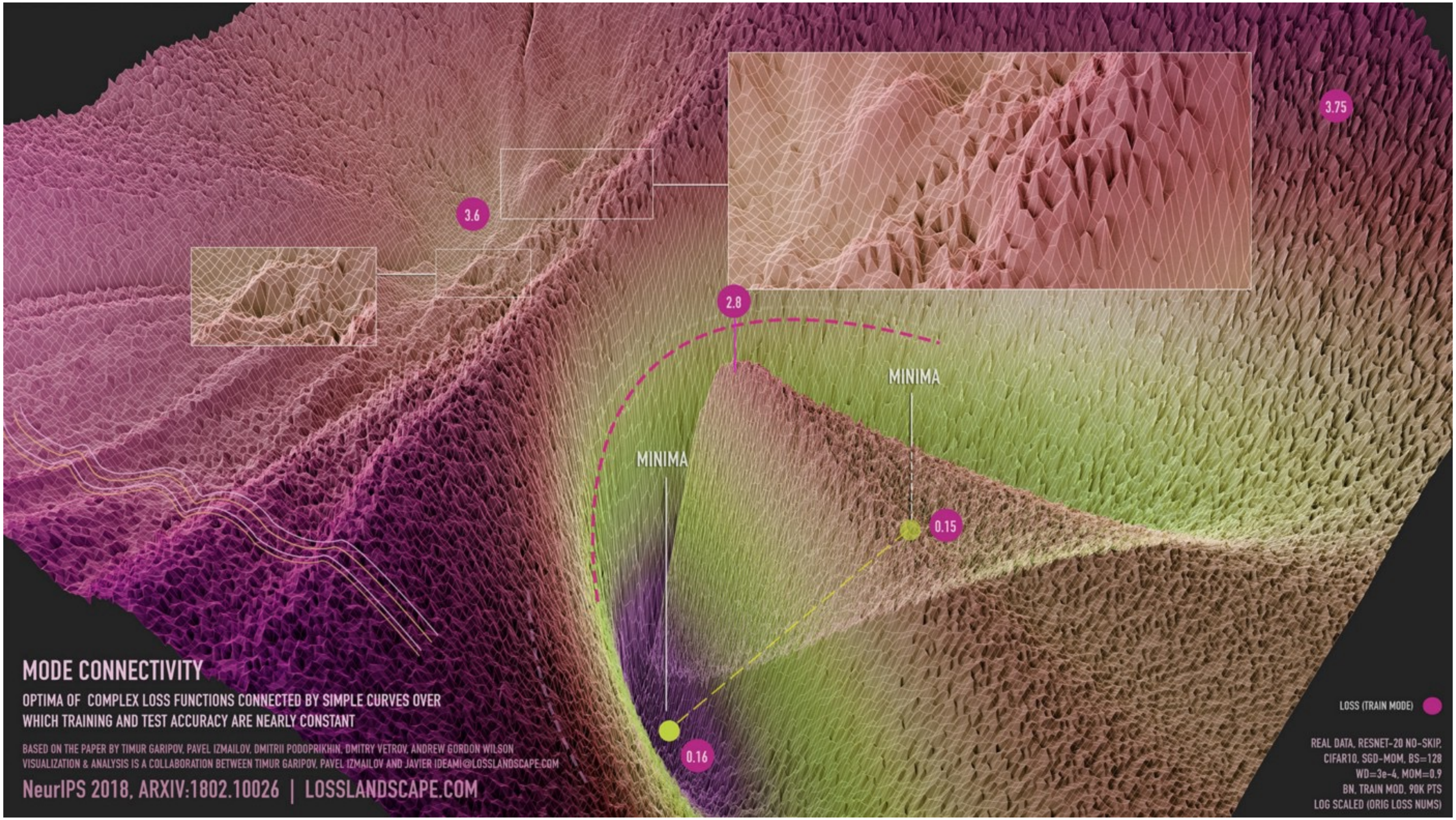
- A single layer network may need a width exponential in D to approximate a depth-D network’s output
  - Simplified version of Telgarsky (2015, 2016)

- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

- Major part of deep learning is choosing the right function
  - Need to make gradient descent work, even if substantial engineering required

Fleuret, Deep Learning Course

Deep Neural Networks Loss Landscape



MODE CONNECTIVITY

OPTIMA OF COMPLEX LOSS FUNCTIONS CONNECTED BY SIMPLE CURVES OVER WHICH TRAINING AND TEST ACCURACY ARE NEARLY CONSTANT

BASED ON THE PAPER BY TIMUR GARIPOV, PAVEL IZMAILOV, DMITRII PODOPRIKHIN, DMITRY VETROV, ANDREW GORDON WILSON
VISUALIZATION & ANALYSIS IS A COLLABORATION BETWEEN TIMUR GARIPOV, PAVEL IZMAILOV AND JAVIER IDEAMI@LOSSLANDSCAPE.COM

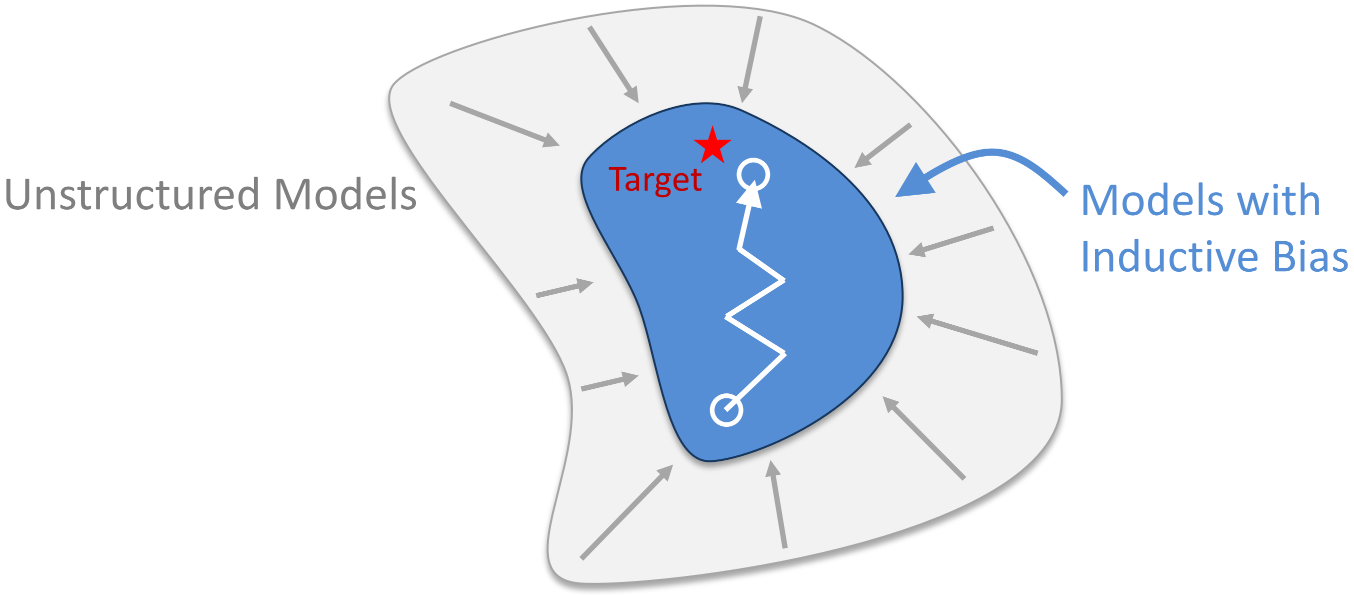
NeurIPS 2018, ARXIV:1802.10026 | LOSSLANDSCAPE.COM

REAL DATA, RESNET-20 NO-SKIP,
CIFAR10, SGD-MOM, BS=128
WD=3e-4, MOM=0.9
BN, TRAIN MOD, 90K PTS
LOG SCALED (ORIG LOSS NUMS)

https://arxiv.org/abs/1802.10026

Choosing the right function…

- We know a lot about our data
  - What transformations shouldn’t affect predictions
  - Symmetries, structures, geometry, …
- **Inductive Bias:** we can match models to this knowledge
  - Throw out irrelevant functions we know aren’t the solution
  - Bias the learning process towards good solutions



Unstructured Models

Target

Models with Inductive Bias

Choosing the right function…

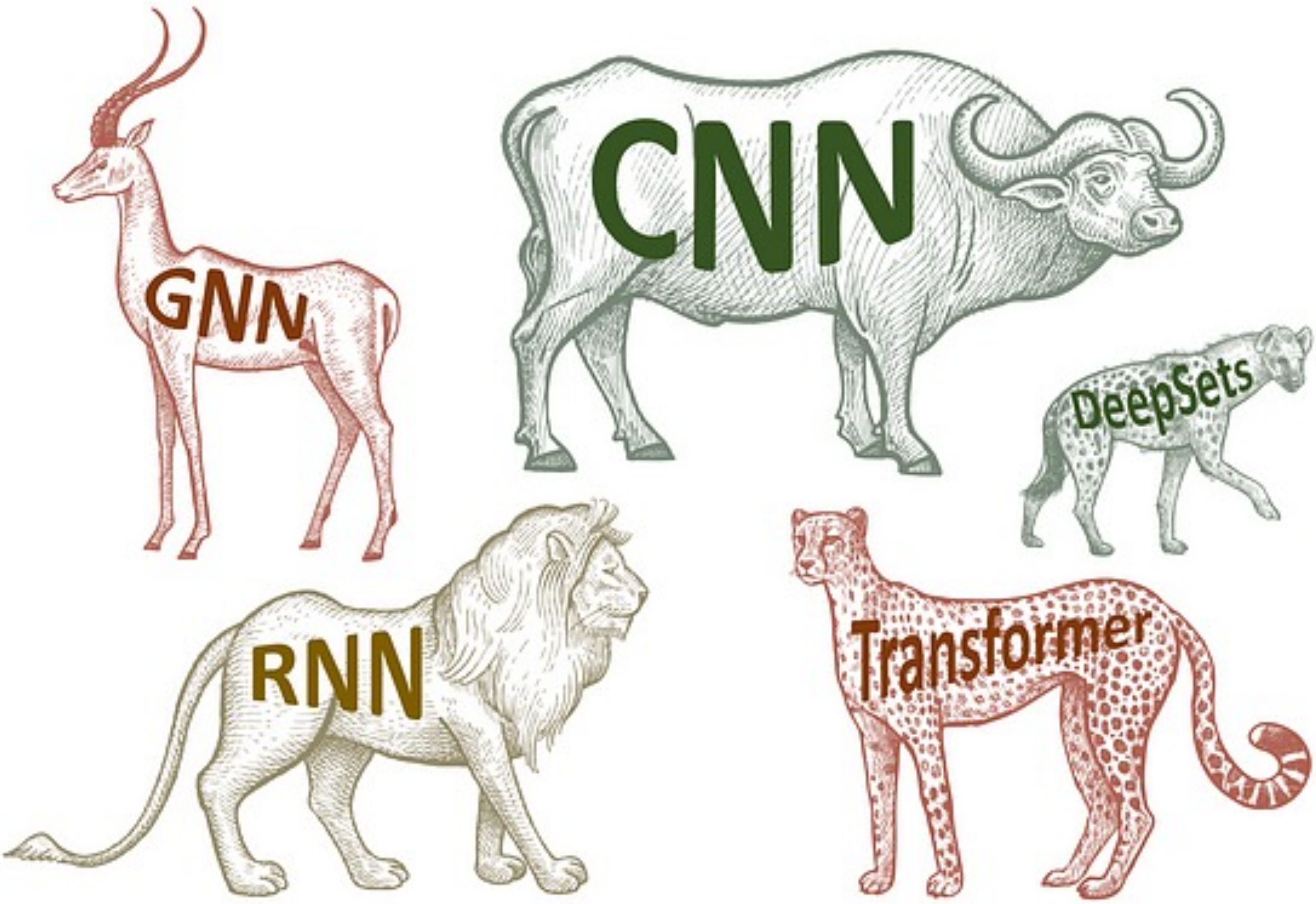


Image credit: Michael Bronstein

Summary

- Neural Networks allow us to combine non-linear basis selection with feature learning

- But must keep in mind the bias-variance tradeoff and how models will generalize

- Deep neural networks allow learning complex function by hierarchically structuring the feature learning, and we can use inductive bias (knowledge) to define models that are well adapted to our problem

Backup

Bias Variance Tradeoff

Bias Variance Tradeoff

- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$

$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$\begin{aligned} E[(y - h(x))^2] &= E[(y - \bar{y})^2] \quad + \quad (\bar{y} - \bar{h}(x))^2 \quad + \quad E[(h(x) - \bar{h}(x))^2] \\ &= \text{noise} \quad + \quad (\text{bias})^2 \quad + \quad \text{variance} \end{aligned}$$

Bias Variance Tradeoff

- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$
$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$\begin{aligned} E[(y-h(x))^2] &= \boxed{\begin{matrix} E[(y-\bar{y})^2] \\ \text{noise} \end{matrix}} \quad &+ \quad (\bar{y}-\bar{h}(x))^2 \quad &+ \quad E[(h(x)-\bar{h}(x))^2] \\ & &+ \quad (\text{bias})^2 \quad &+ \quad \text{variance} \end{aligned}$$

Intrinsic noise in system or measurements
Can not be avoided or improved with modeling
Lower bound on possible noise

Bias Variance Tradeoff

- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$

$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$\begin{aligned} E[(y-h(x))^2] &= \boxed{E[(y-\bar{y})^2]} + \boxed{(\bar{y}-\bar{h}(x))^2} + E[(h(x)-\bar{h}(x))^2] \\ &= \boxed{\text{noise}} + \boxed{(\text{bias})^2} + \text{variance} \end{aligned}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

Bias Variance Tradeoff

- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$

$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$E[(y - h(x))^2] = \boxed{\begin{array}{l} E[(y - \bar{y})^2] \\ \text{noise} \end{array}} + \boxed{\begin{array}{l} (\bar{y} - \bar{h}(x))^2 \\ (\text{bias})^2 \end{array}} + \boxed{\begin{array}{l} E[(h(x) - \bar{h}(x))^2] \\ \text{variance} \end{array}}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

- **More Complexity** will make the model "move" more to capture the data points, and hence its **variance will be larger**.

Bias Variance Tradeoff

- Model h(x), defined over dataset, modeling random variable output y

$$E[y] = \bar{y}$$

$$E[h(x)] = \bar{h}(x)$$

- Examining generalization error at x, w.r.t. possible training datasets

$$\begin{array}{r} E[(y-h(x))^2] \\ \end{array} \begin{array}{c} = \\ = \end{array} \boxed{\begin{array}{l} E[(y-\bar{y})^2] \\ \text{noise} \end{array}} \begin{array}{c} + \\ + \end{array} \boxed{\begin{array}{l} (\bar{y}-\bar{h}(x))^2 \\ (\text{bias})^2 \end{array}} \begin{array}{c} + \\ + \end{array} \boxed{\begin{array}{l} E[(h(x)-\bar{h}(x))^2] \\ \text{variance} \end{array}}$$

- The **more complex the model** h(x) is, the more data points it will capture, and **the lower the bias** will be.

- **More Complexity** will make the model "move" more to capture the data points, and hence its **variance will be larger**.
  - **As dataset size grows, can reduce variance! Can use more complex model**

Automatic Differentiation

Automatic Differentiation

Exact derivatives for gradient-based optimization come from running **differentiable code** via **automatic differentiation**

$$\begin{tikzcd}
f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R} \arrow[d, "\text{automatic differentiation}"] \\
\nabla f(\mathbf{x}) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)
\end{tikzcd}$$

$$\begin{tikzcd}
\texttt{f(x) \{…\};} \arrow[d] \\
\texttt{df(x) \{…\};}
\end{tikzcd}$$

Automatic Differentiation Example

- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

$$f(a, b) = \log(ab)$$

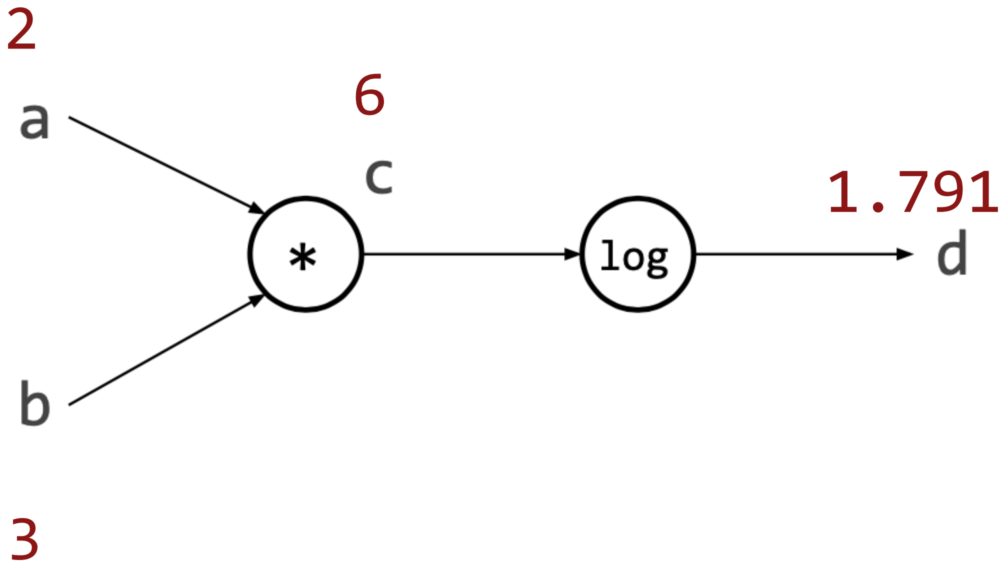
$$\nabla f(a, b) = \left(\frac{1}{a}, \frac{1}{b}\right)$$

Automatic Differentiation Example

- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

```
f(a, b):
    c = a * b
    d = log(c)
    return d
```

```
f(2, 3) = 1.791
```

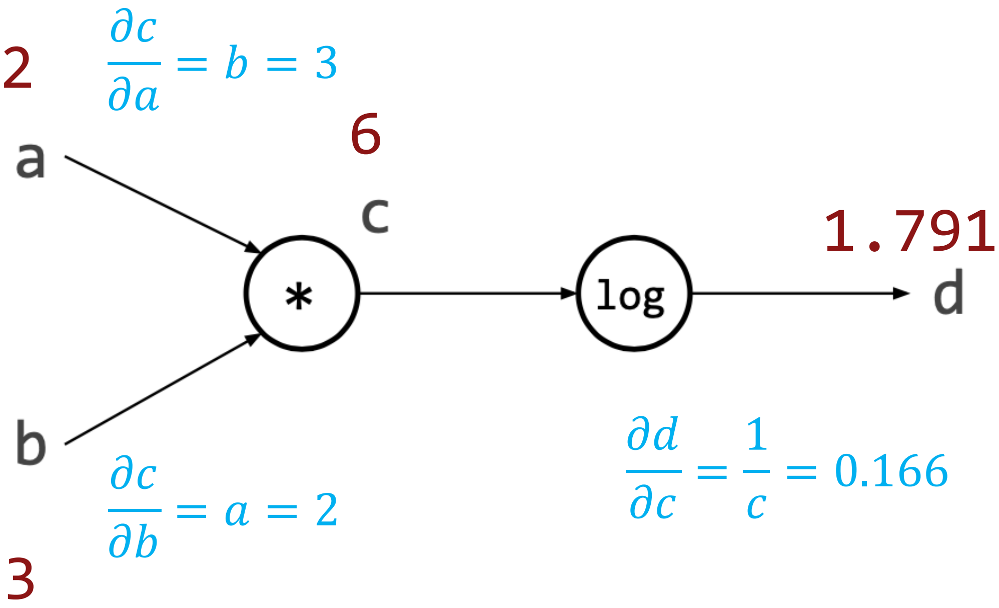


Primals

Automatic Differentiation Example

- All numerical algorithms, when executed, evaluate to compositions of a finite set of elementary operations with known derivatives
  - Represent as a **computational graph** showing dependencies

```
f(a, b):
    c = a * b
    d = log(c)
    return d
```



$$\frac{\partial c}{\partial a} = b = 3$$

$$\frac{\partial c}{\partial b} = a = 2$$

$$\frac{\partial d}{\partial c} = \frac{1}{c} = 0.166$$

```
f(2, 3) = 1.791
df(2,3) = [0.5, 0.333]
```

Chain Rule: $\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial a} = 0.166 * 3 = 0.5$

Forward and Reverse Mode

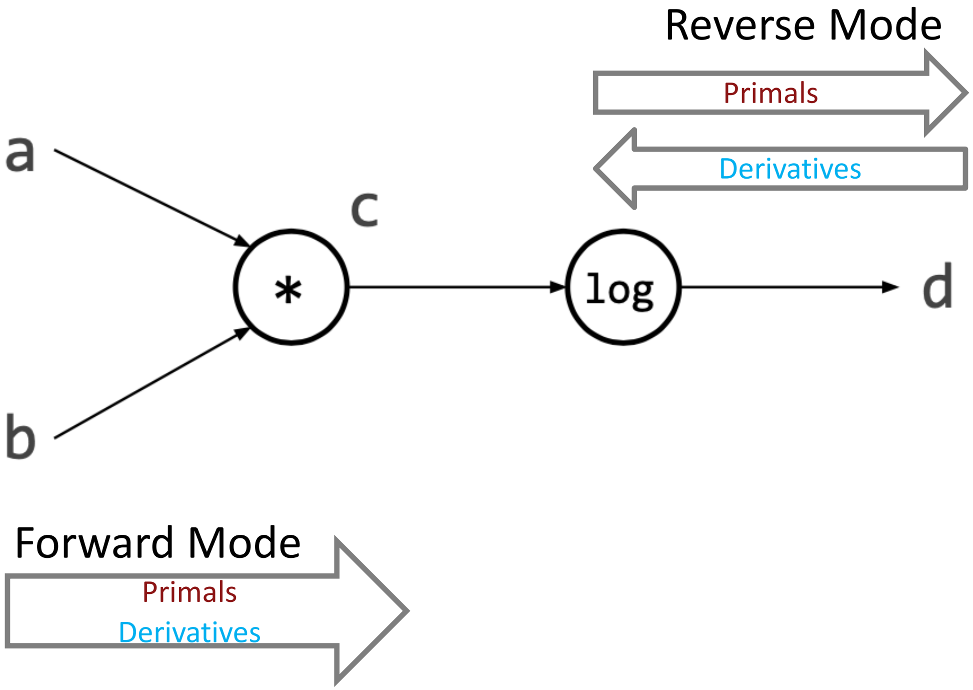
- Derivatives can be computed in **Forward Mode** and **Reverse Mode**

Forward Mode Single Evaluation: $\boldsymbol{f}(\boldsymbol{x})\colon \mathbb{R}^N \to \mathbb{R}^M$

$$\frac{d\boldsymbol{f}(\boldsymbol{x})}{d\boldsymbol{x}} = \begin{pmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{df_1}{dx_N} & \cdots & \frac{df_M}{dx_N} \end{pmatrix}$$

Reverse Mode Single Evaluation: $\boldsymbol{f}(\boldsymbol{x})\colon \mathbb{R}^N \to \mathbb{R}^M$

$$\frac{d\boldsymbol{f}(\boldsymbol{x})}{d\boldsymbol{x}} = \begin{pmatrix} \frac{df_1}{dx_1} & \cdots & \frac{df_M}{dx_1} \\ \vdots & \ddots & \vdots \\ \frac{df_1}{dx_N} & \cdots & \frac{df_M}{dx_N} \end{pmatrix}$$



Chain Rule: $\frac{\partial d}{\partial a} = \frac{\partial d}{\partial c}\frac{\partial c}{\partial a}$