Introduction to Machine Learning and Artificial Intelligence: Lecture II

Michael Kagan

SLAC NATIONAL ACCELERATOR LABORATORY

2nd COFI Advanced Instrumentation and Analysis Techniques School

December 9, 2023

The Plan

- Lecture 1
  - Introduction to Machine Learning fundamentals
  - Linear Models

- Lecture 2
  - Neural Networks
  - Deep Neural Networks
  - Inductive Bias and Model Architectures

- Lecture 3
  - Unsupervised Learning
  - Autoencoders
  - Towards Generative Models: Variation Autoencoders

Deep Learning Resourses

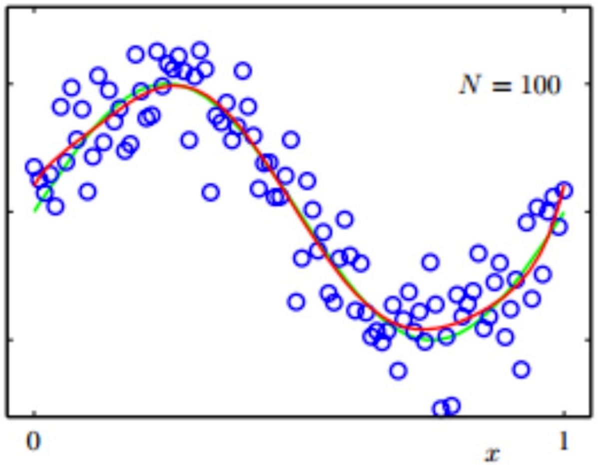
- Deep Learning is a HUGE field
  - O(10,000) papers submitted to conferences

- I only condensed *some* parts of what you would find in *some lectures* of a Deep Learning course
  - More details from other lecturers!

- Highly recommend Online-available lectures:
  - Francois Fleuret course at University of Geneva
  - Gilles Louppe course at University of Liege
  - Yann LeCun & Alfredo Canziani course at NYU

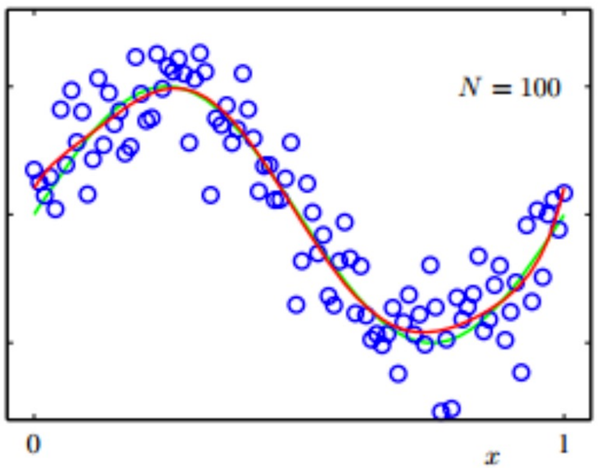
Bias-Variance Tradeoff

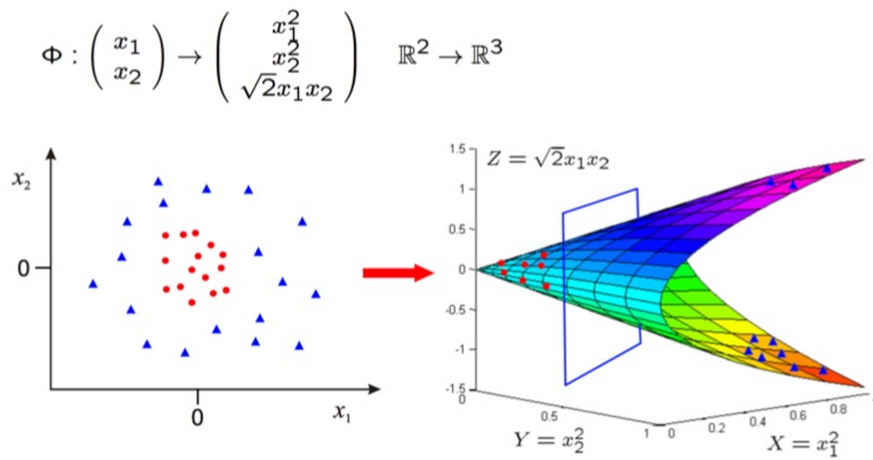
Basis Functions



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

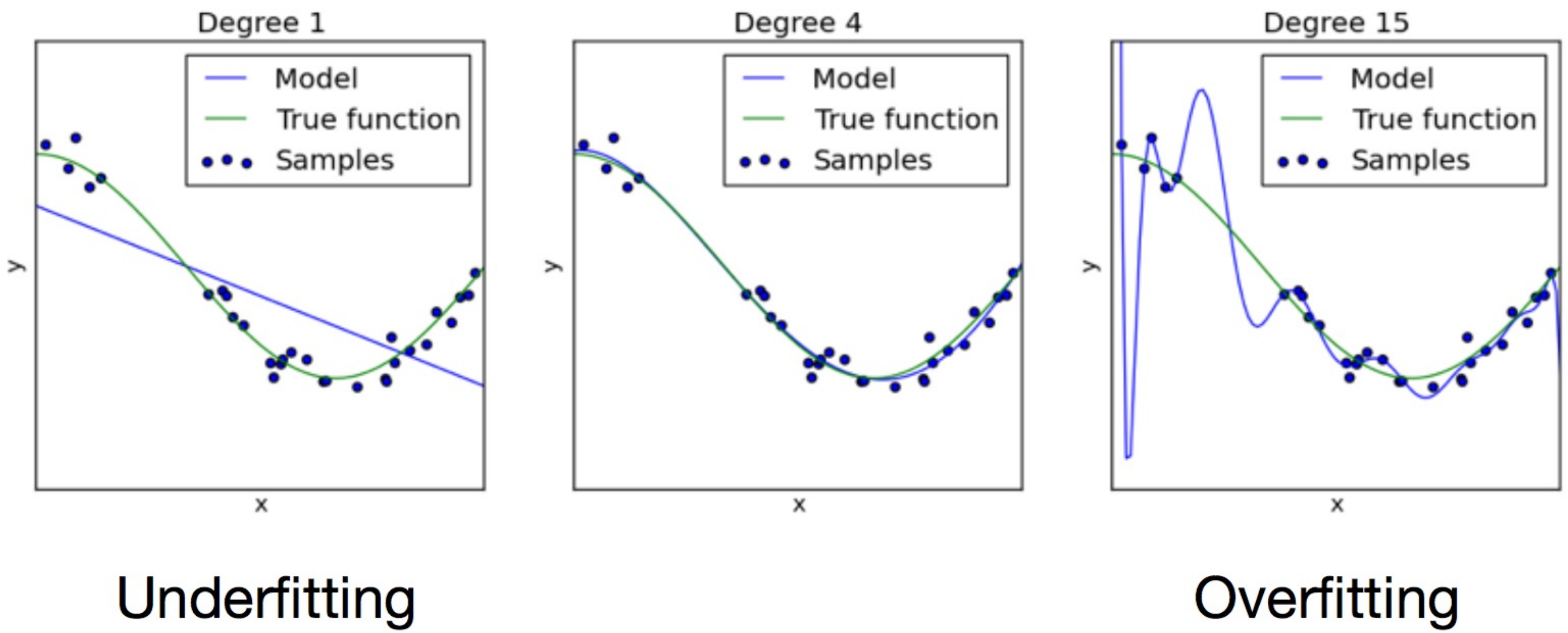
Basis Functions





- What if non-linear relationship between **y** and **x**?
- Choose **basis functions $\phi(x)$** to form new features
  - Example: Polynomial basis $\quad \phi(x) \sim \{1, x, x^2, x^3, \dots\}$
  - Logistic regression on new features: $\quad h(x; w) = \sigma\left(w^T \phi(x)\right)$
- What basis functions to choose? *Overfit* with too much flexibility?

What is Overfitting



http://scikit-learn.org/

- Models allow us to **generalize** from data
- Different models generalize in different ways

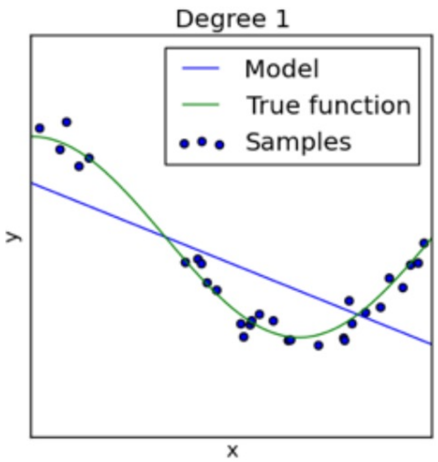
Bias Variance Tradeoff

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

Bias Variance Tradeoff

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

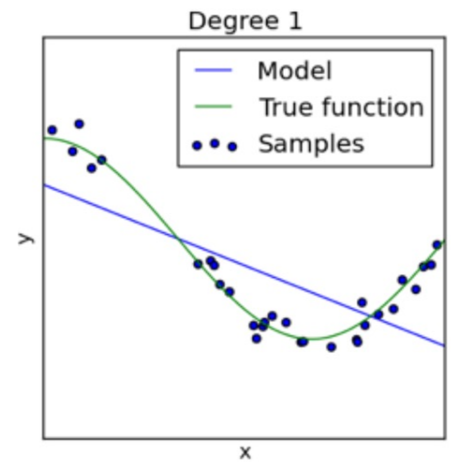
- Simple models under-fit:
  will deviate from data (high bias)
  but will not be influenced by
  peculiarities of data (low variance).



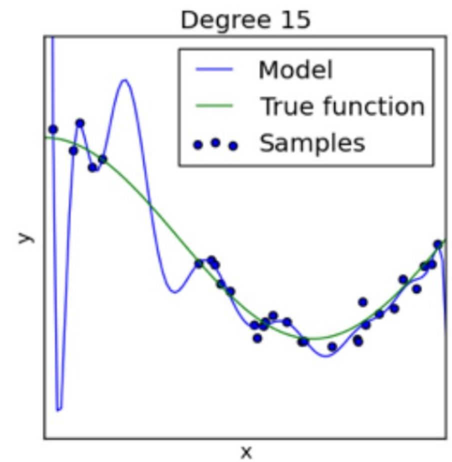
Bias Variance Tradeoff

- 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).



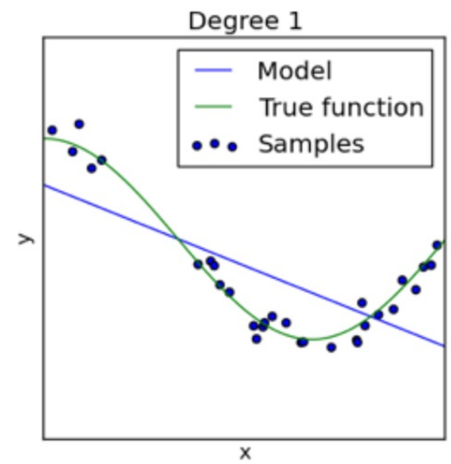
- Complex models over-fit:
  will not deviate systematically from
  data (low bias) but will be very
  sensitive to data (high variance).



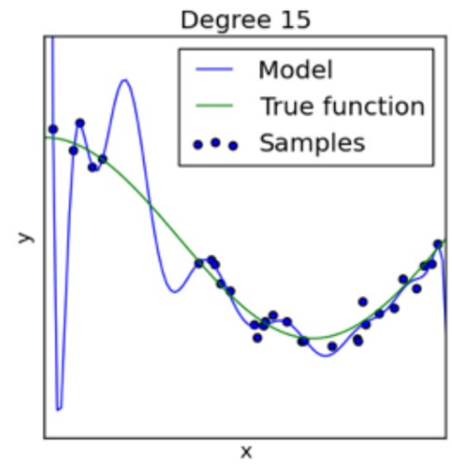
Bias Variance Tradeoff

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

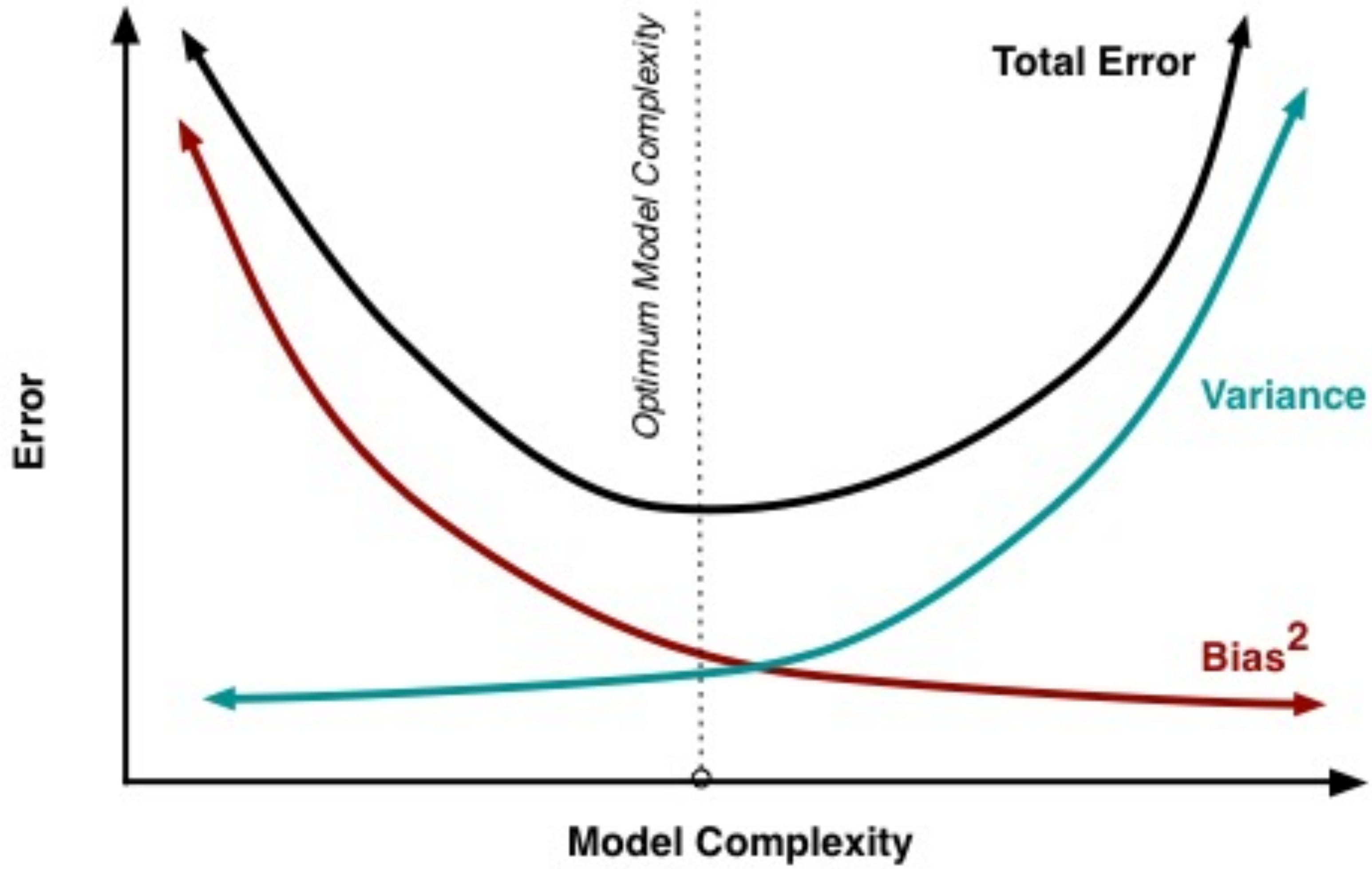
- Simple models under-fit:
  will deviate from data (high bias)
  but will not be influenced by
  peculiarities of data (low variance).



- 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

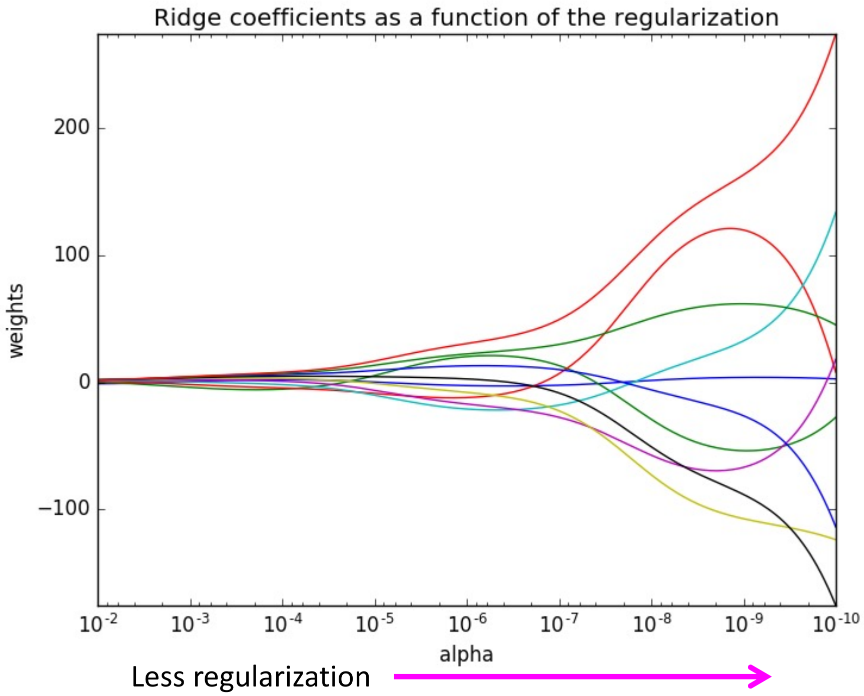


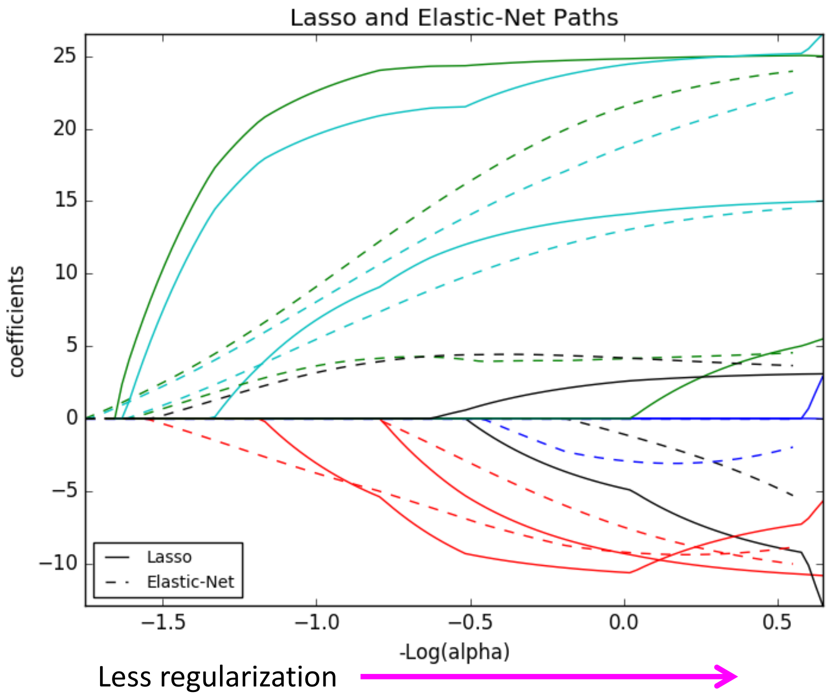
Regularization – Control Complexity

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

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

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





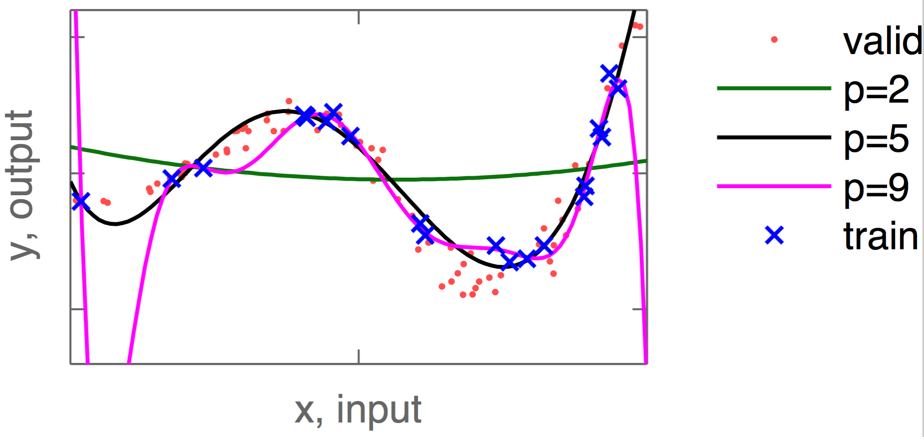
- L2 keeps weights small, L1 keeps weights sparse!
- But how to choose hyperparameter α?

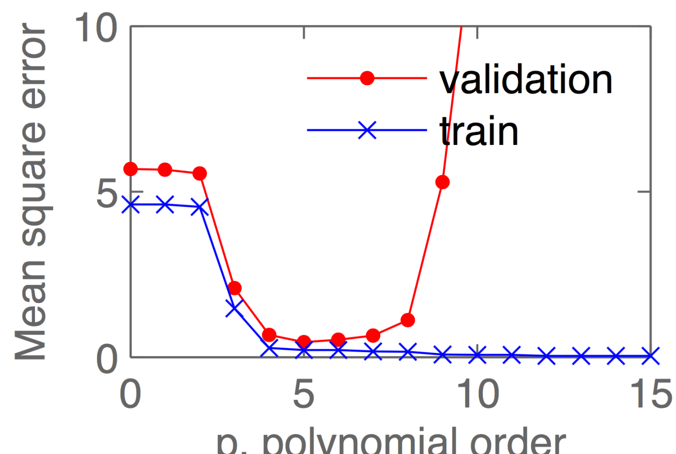
http://scikit-learn.org/

How to Measure Generalization Error?

Training set | Validation set | Test set

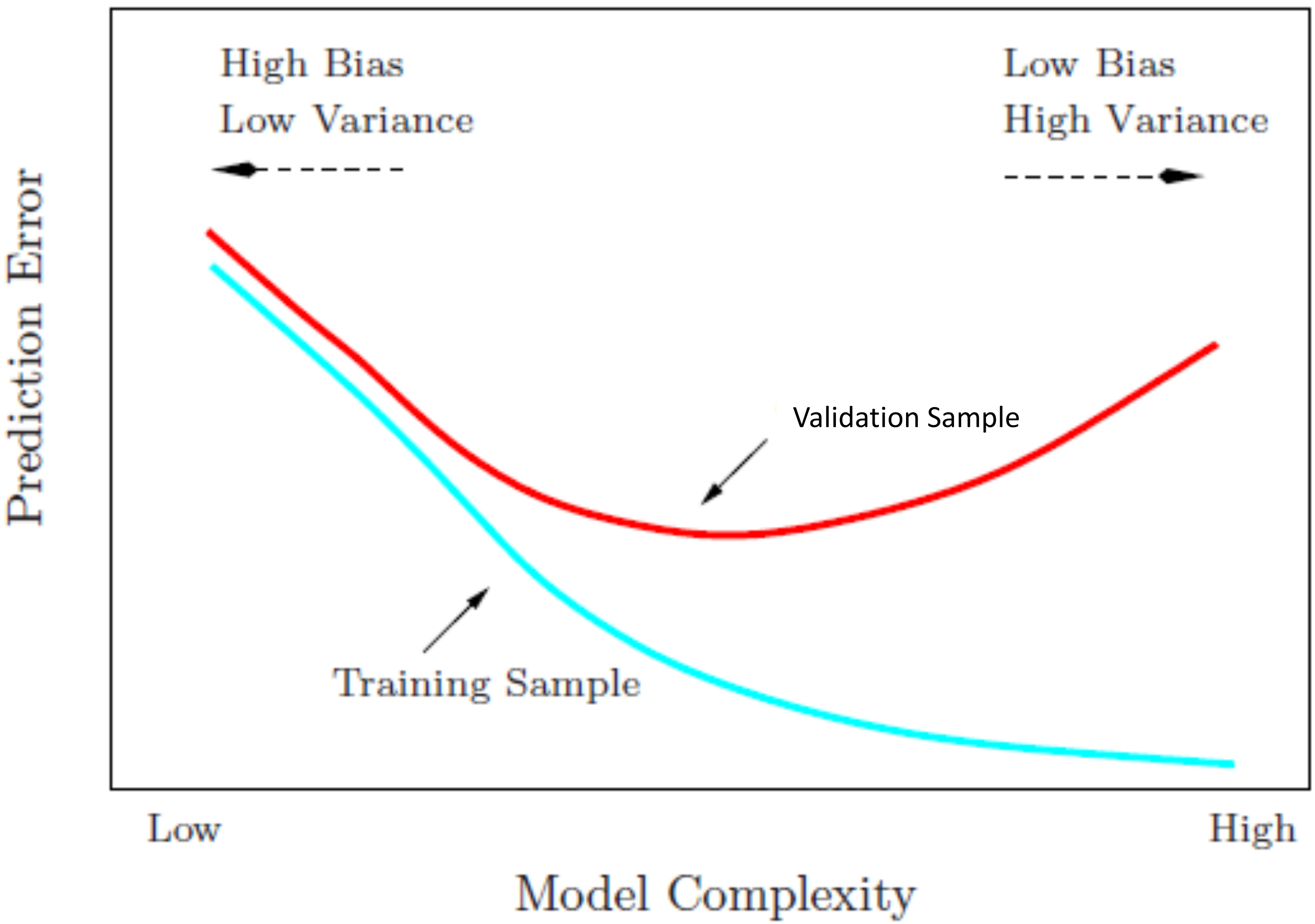
- Split dataset into multiple parts
- **Training set**
  - Used to fit model parameters
- **Validation set**
  - Used to check performance on independent data and tune hyper parameters
- **Test set**
  - final evaluation of performance after all hyper-parameters fixed
  - Needed since we tune, or “peek”,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





[Murray]

How to Measure Generalization Error?



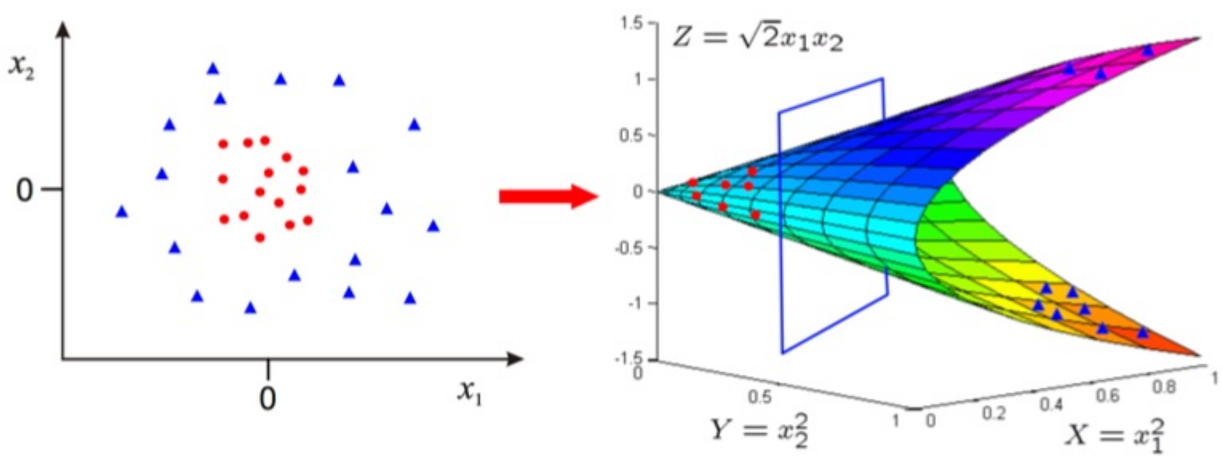
From Logistic Regression to Neural Networks

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

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

  - 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: $\phi(x) \sim \{x^2, \sin(x), \log(x), \ldots\}$

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

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

- Learn the basis functions directly from data

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

  - 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 $j = \{1 \dots d\}$

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

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 $\boldsymbol{U}$

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

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

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 $\boldsymbol{U}$

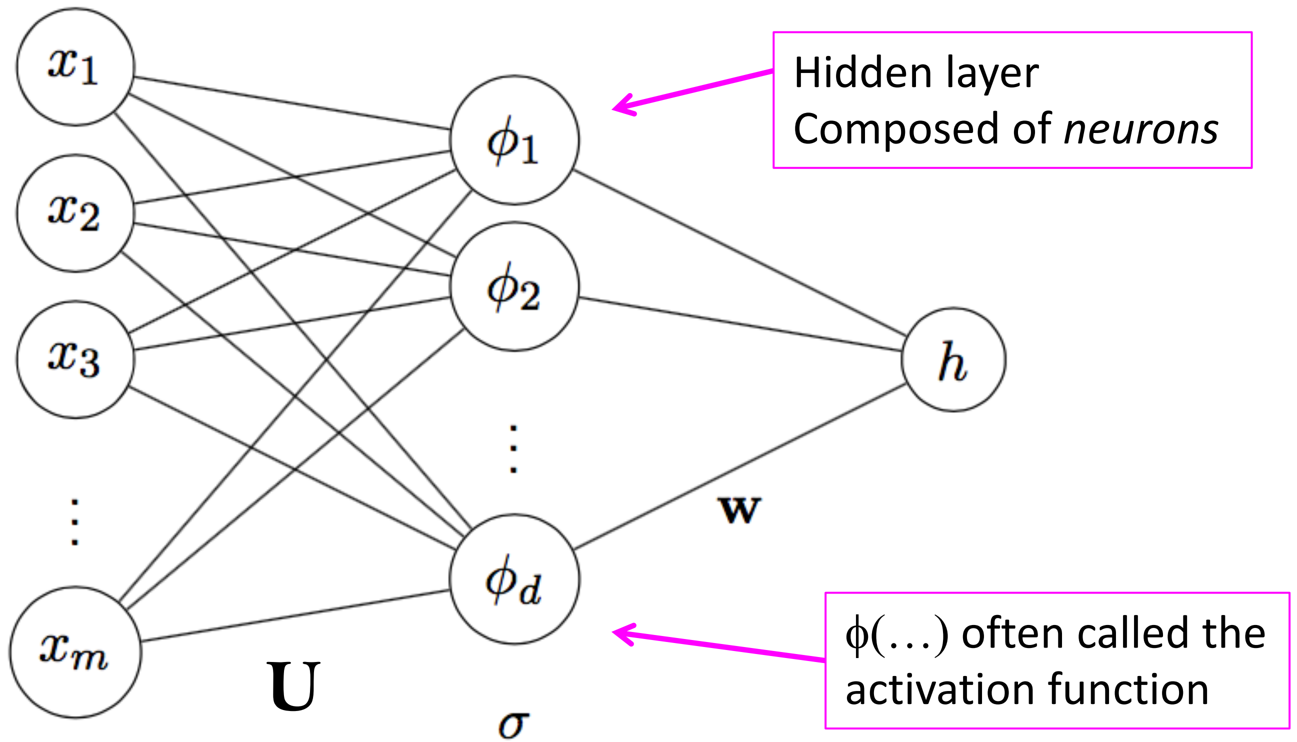
$$\phi(\boldsymbol{x};\ \boldsymbol{U}) = \sigma(\boldsymbol{U}\boldsymbol{x}) = \begin{bmatrix} \sigma(u_1^T x) \\ \sigma(u_2^T x) \\ \vdots \\ \sigma(u_d^T x) \end{bmatrix} \in \mathbb{R}^d$$

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

- Full model becomes

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

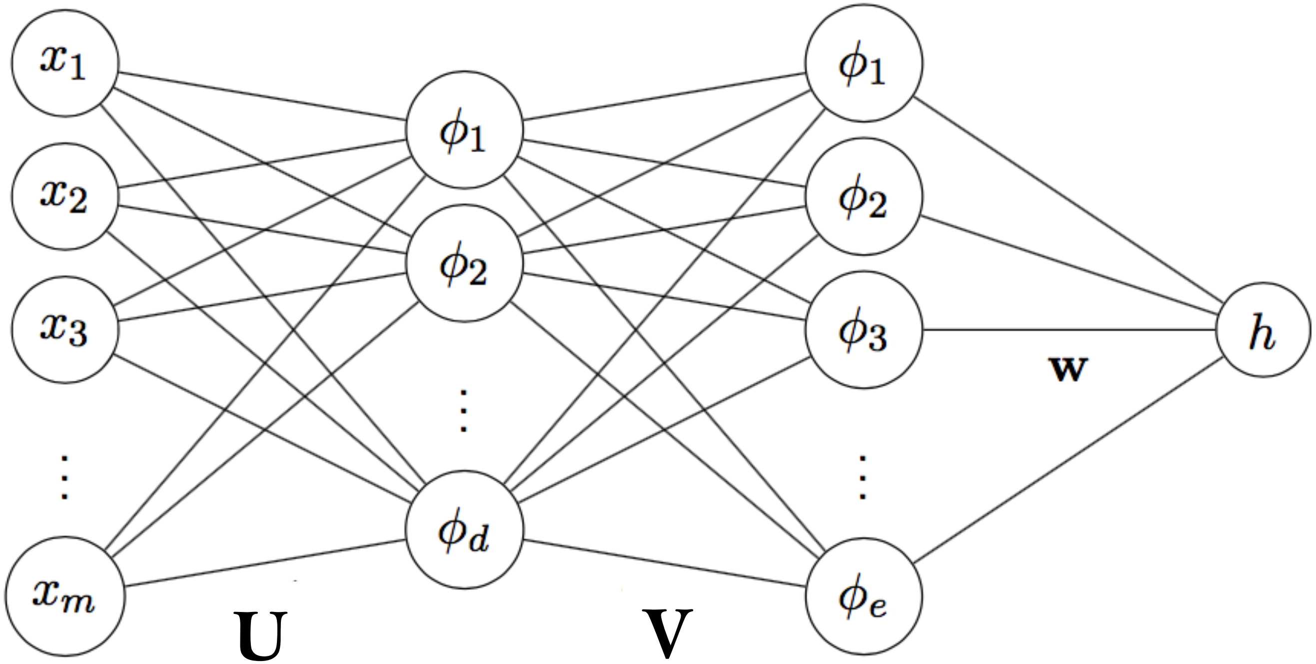
Feed Forward Neural Network



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

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

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. $\mathbf{u}_j$

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

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

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

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**

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

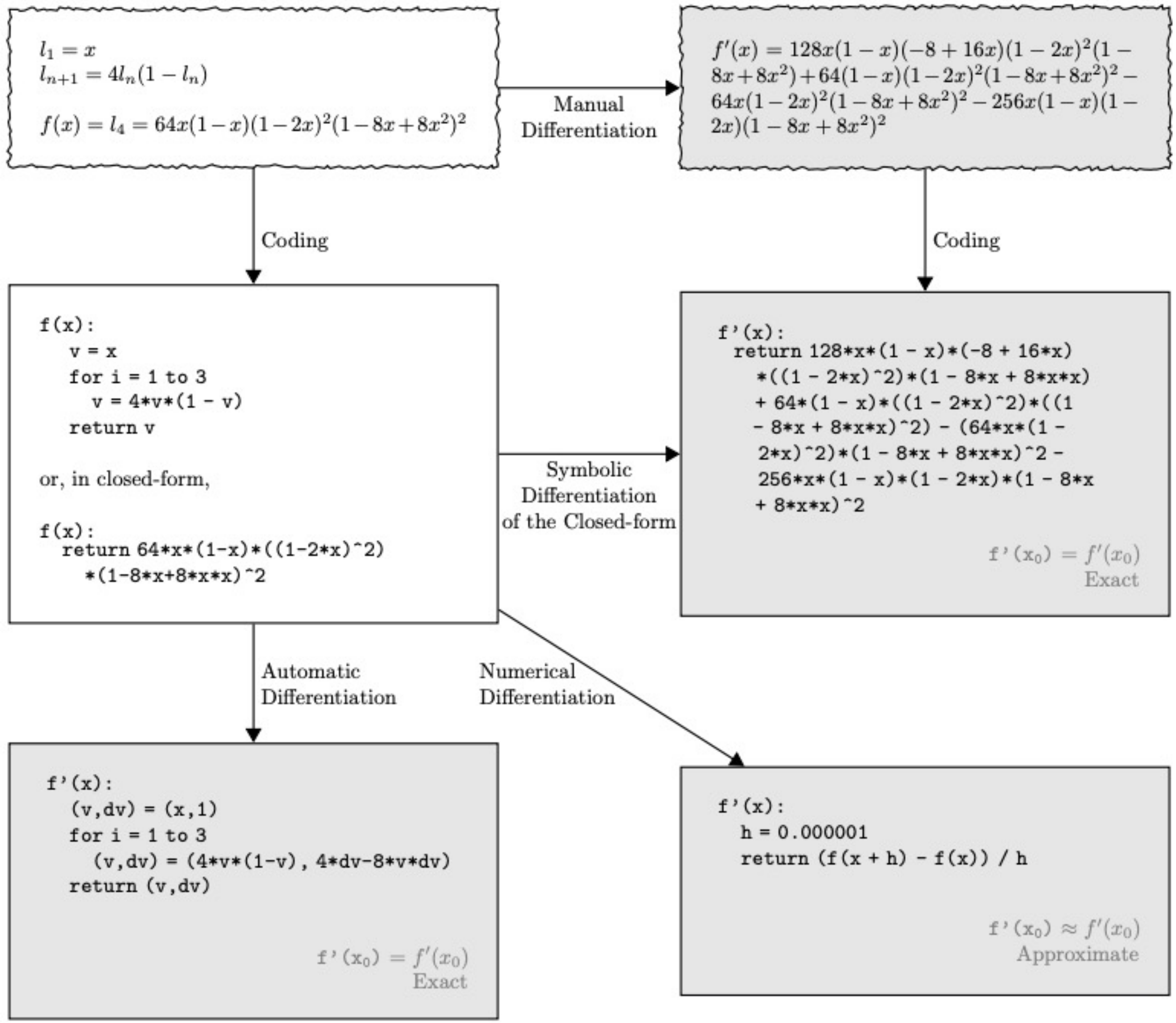
- Chain rule to compute gradient w.r.t. $\mathbf{u}_j$

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

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

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

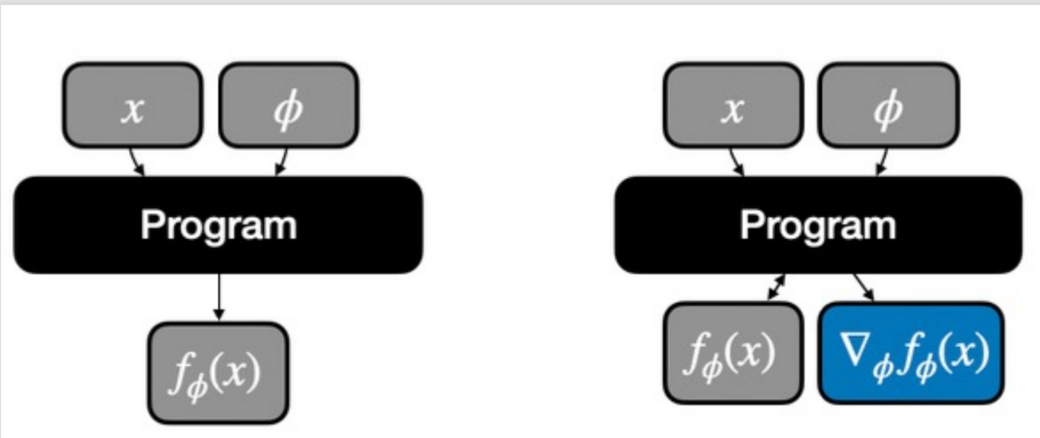
Differentiation in Code



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**



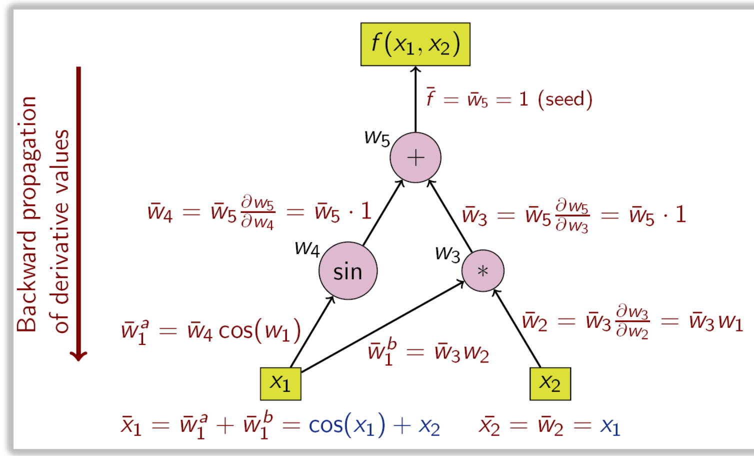


Image credit: Wikipedia

TensorFlow

PyTorch

Backpropagation – Reverse Mode AD

- Loss function composed of layers of nonlinearity

$$L\left(\phi^N\left(\ldots\phi^1(x)\right)\right)$$

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

$$\phi^N\left(\ldots\phi^1(x)\right)$$

- Backward step (b-prop)

$$\frac{\partial L}{\partial \phi^a} = \sum_j \frac{\partial \phi_j^{(a+1)}}{\partial \phi_j^a} \frac{\partial L}{\partial \phi_j^{(a+1)}}$$

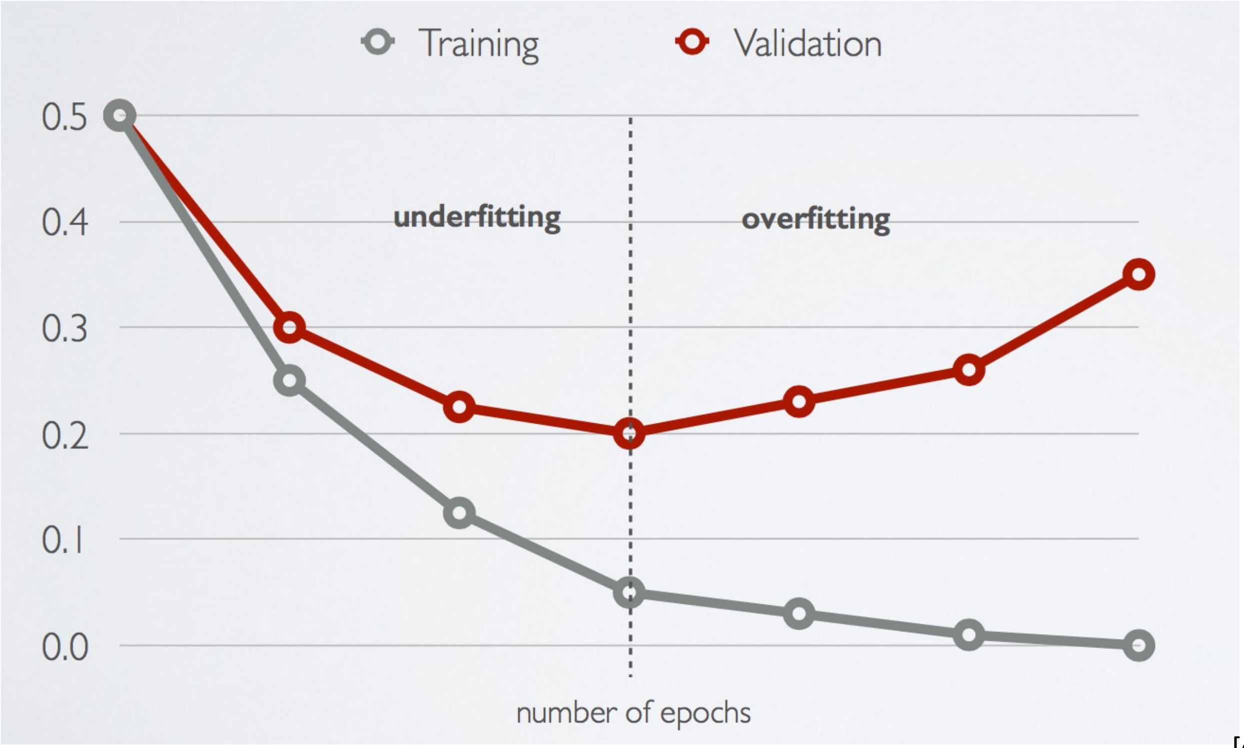
- Compute parameter gradients

$$\frac{\partial L}{\partial \mathbf{w}^a} = \sum_j \frac{\partial \phi_j^a}{\partial \mathbf{w}^a} \frac{\partial L}{\partial \phi_j^a}$$

Training

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

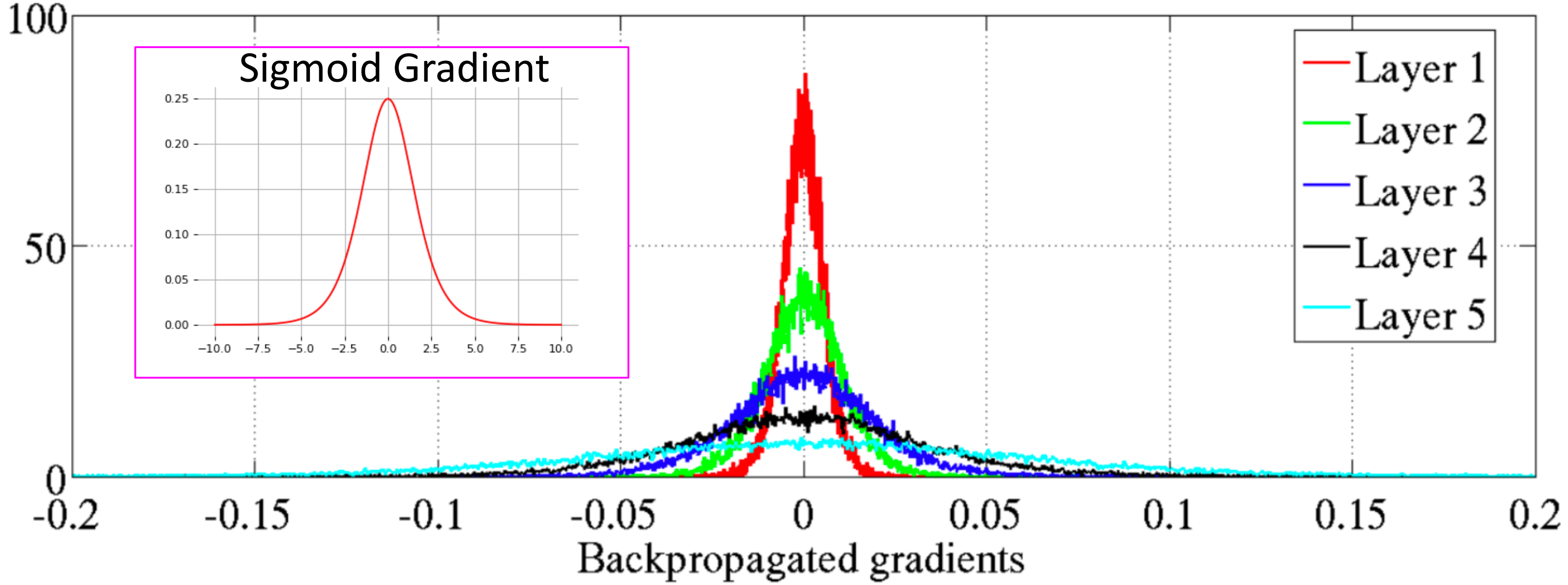
- Use validation set to examine for overtraining, and determine when to stop training



[graphic from H. Larochelle]

Vanishing Gradients

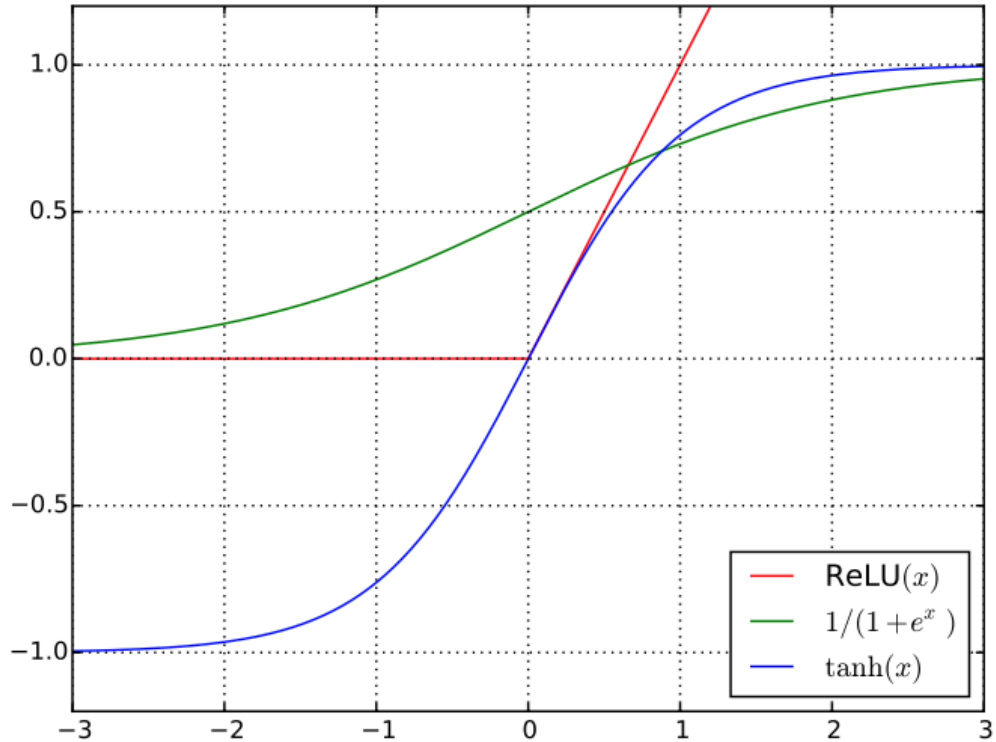
- Major challenge in DL: Vanishing Gradients
- Small gradients slow down / block, stochastic gradient descent → Limits ability to learn!



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

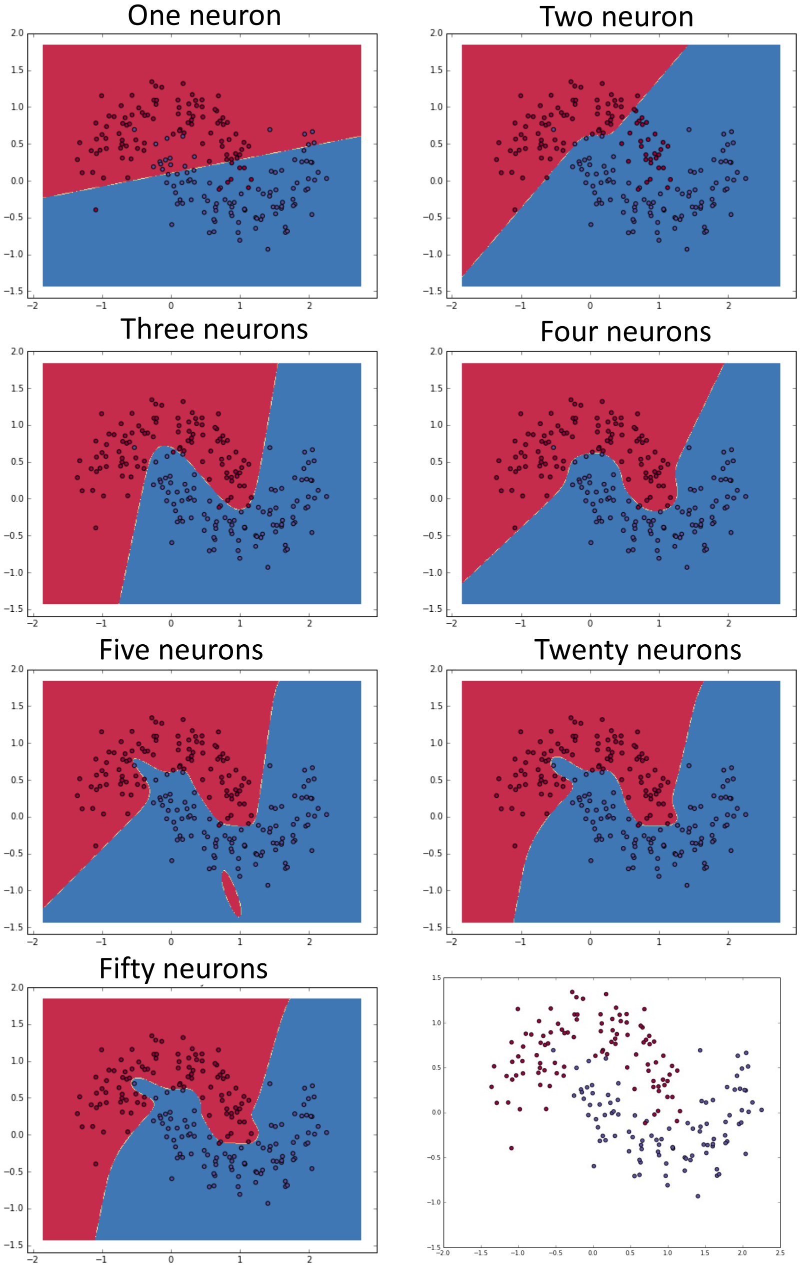


Image source

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

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

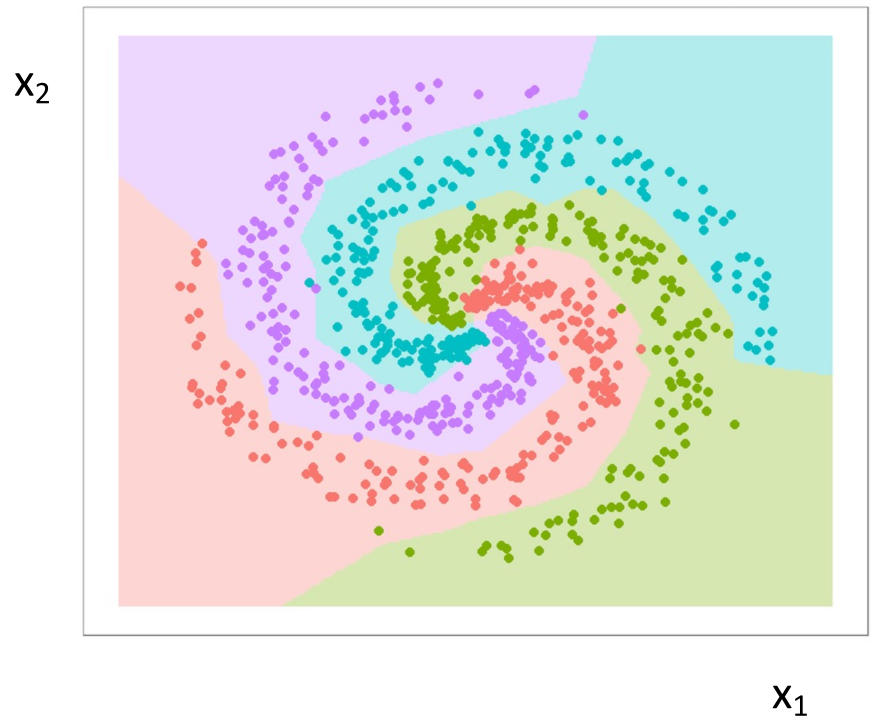
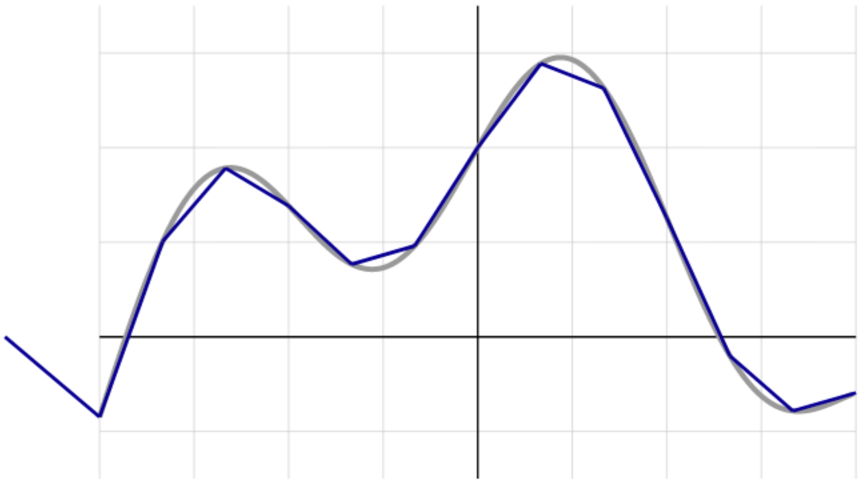


Image source

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 $\mathbb{R}^n$

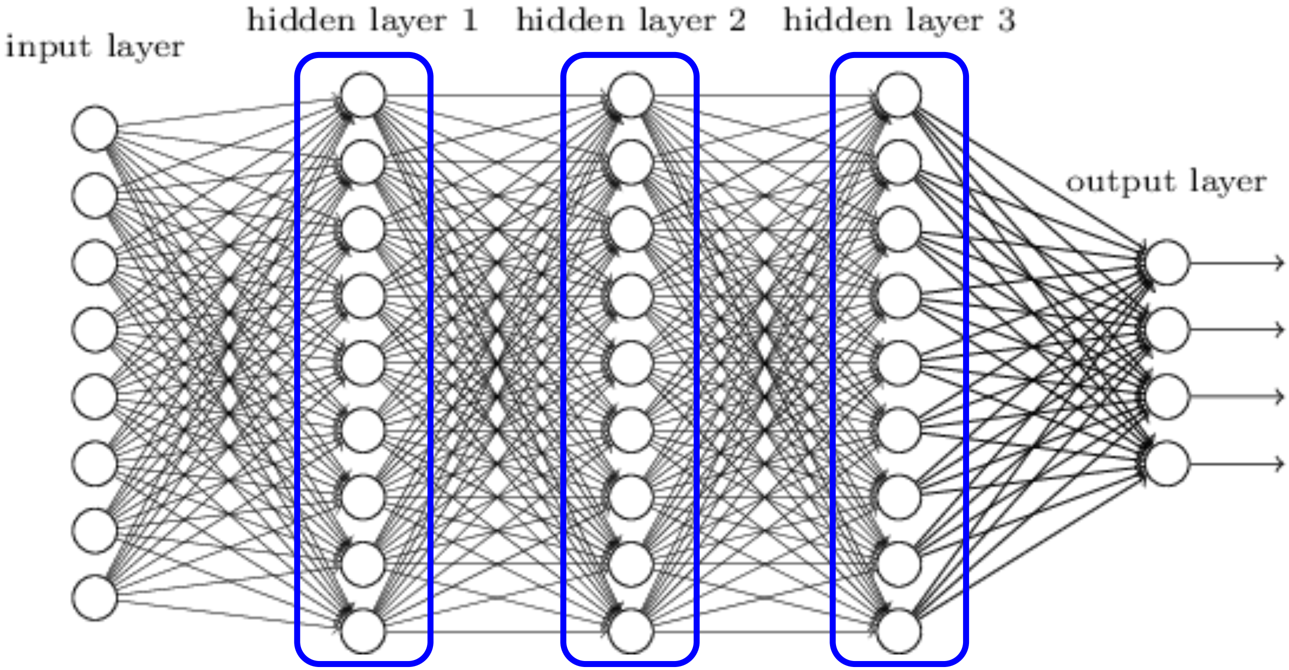
- Better approximation requires larger hidden layer, this theorem says nothing about relation between the two.

- 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

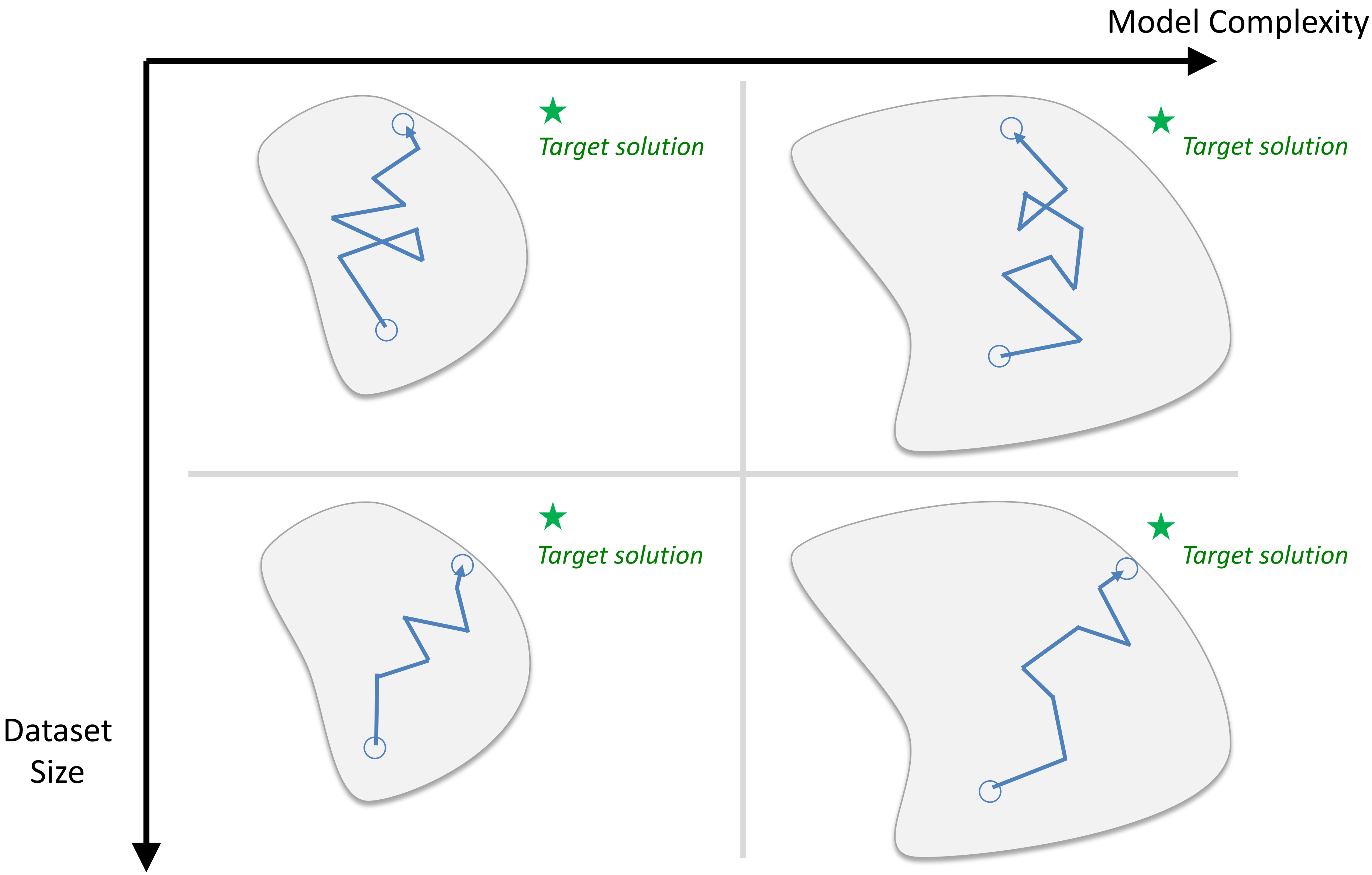
Fleuret, Deep Learning Course

Deep Neural Networks

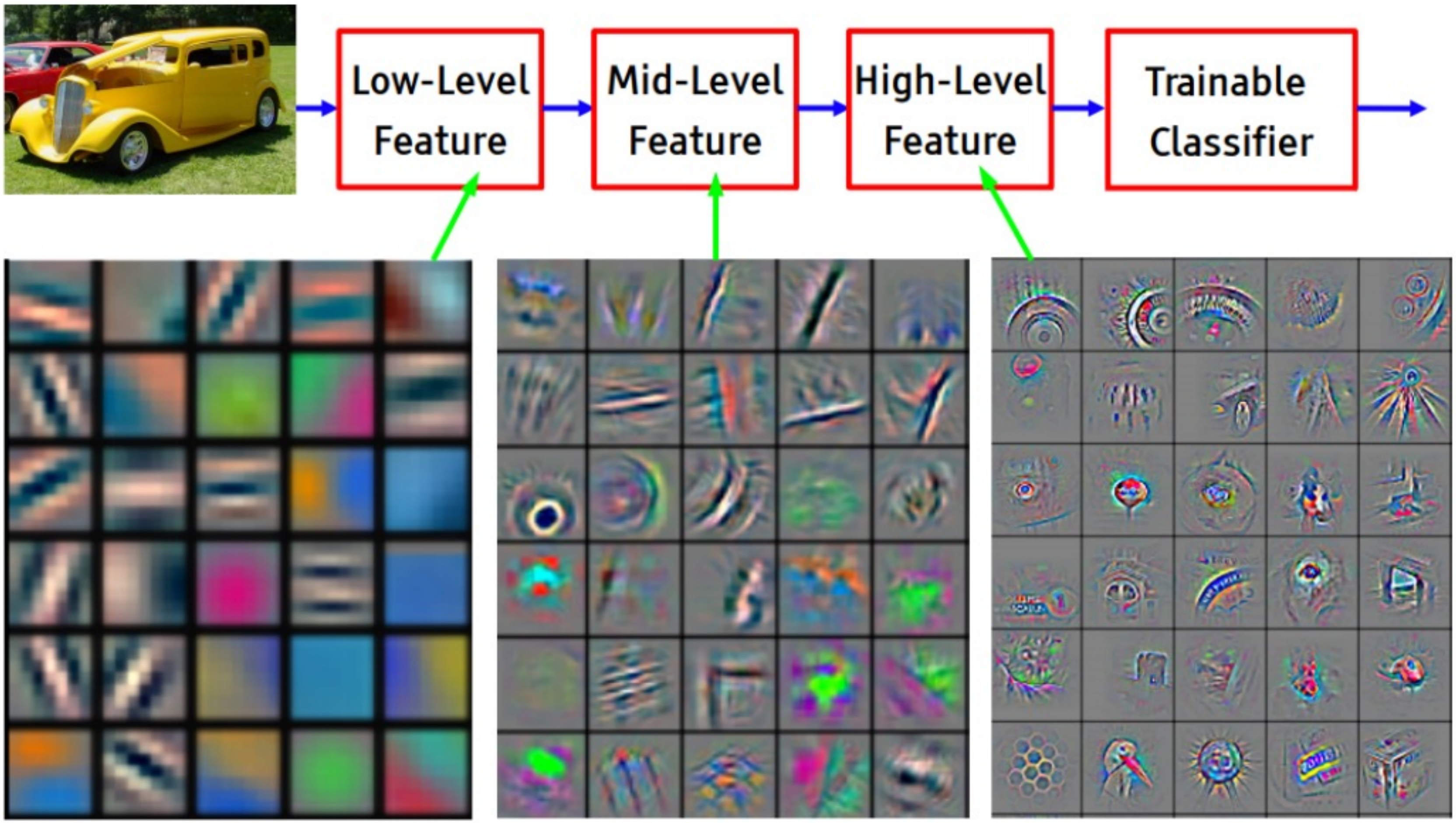


- As data complexity grows, need exponentially large number of neurons in a single-layer network to capture all structure in data
- Deep networks ***factorize the learning*** of structure across layers
- Difficult to train, recently possible with large datasets, fast computing (GPU/TPU) & new training algs. / network structures

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

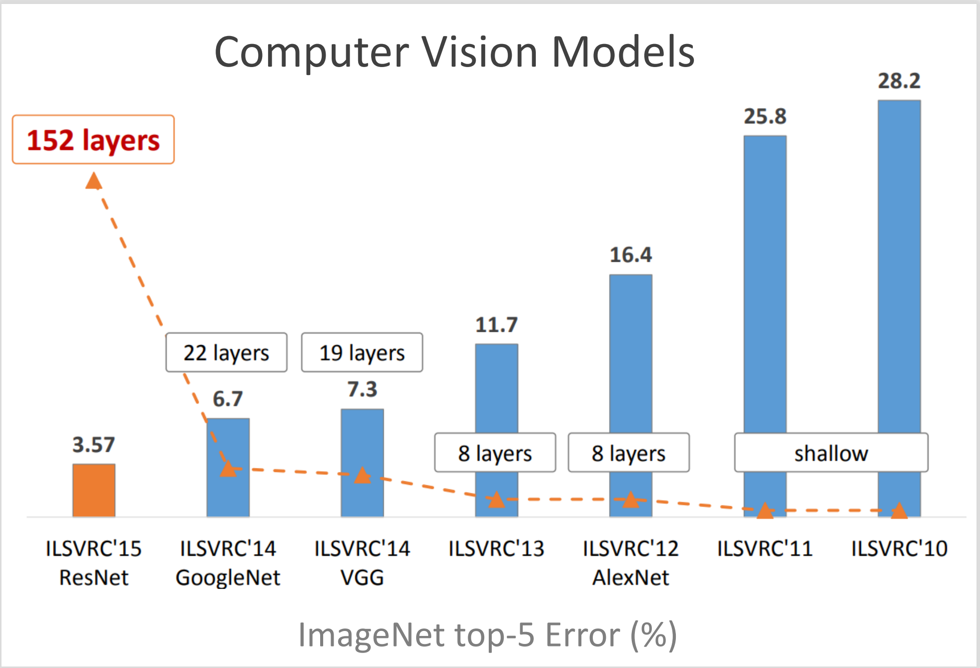


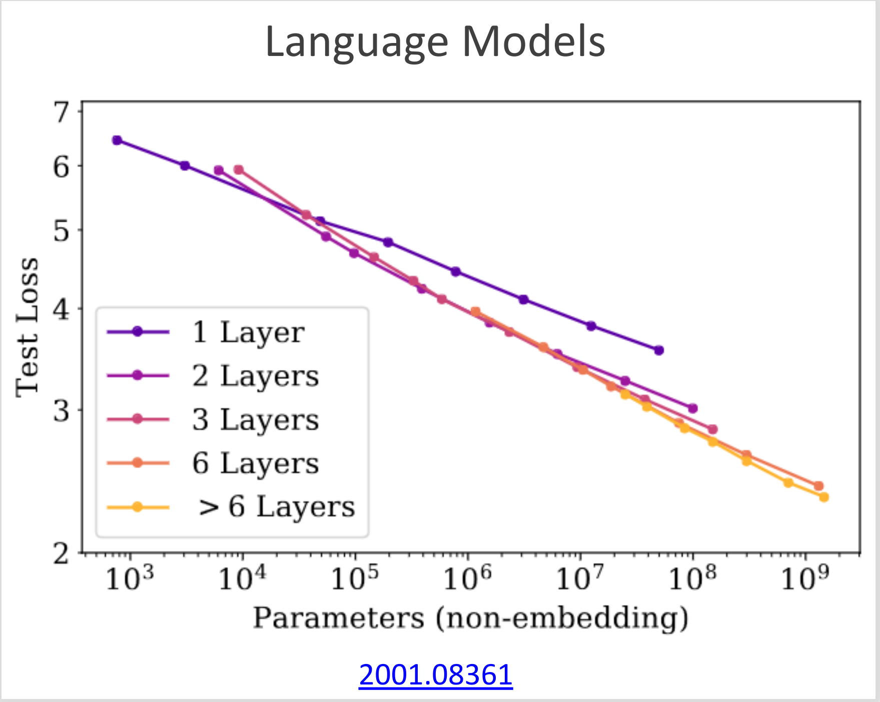
Hierarchical Composition of Features



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Benefits of Depth





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

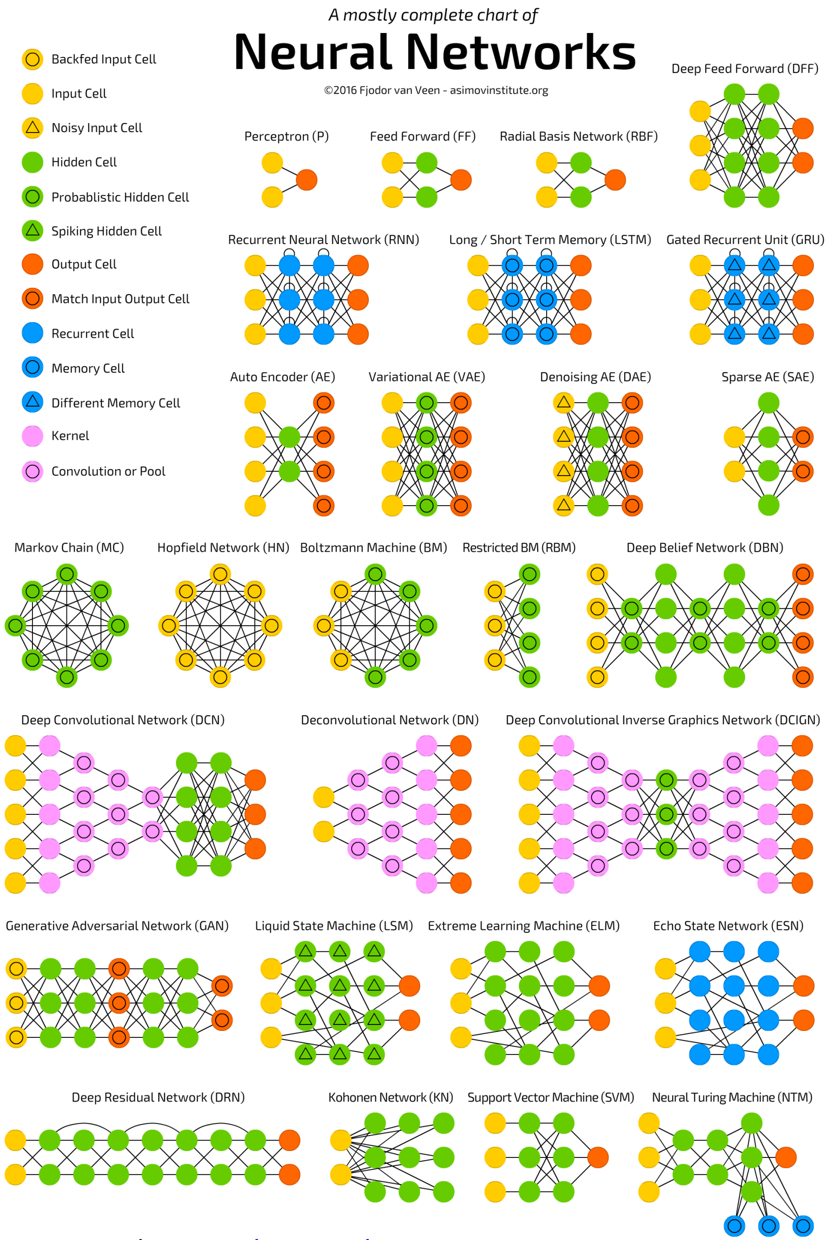


Image credit: neural-network-zoo

Neural Network Zoo – "Optimization" Perspective

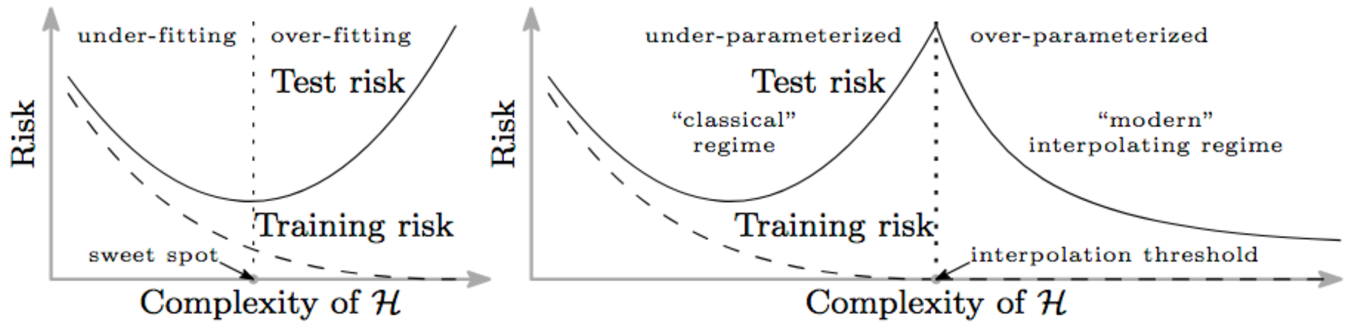
- A single layer network may need a width exponential in D to approximate a depth-D network's output
  - Simplified version of Telgarsky (2015, 2016)

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.

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)

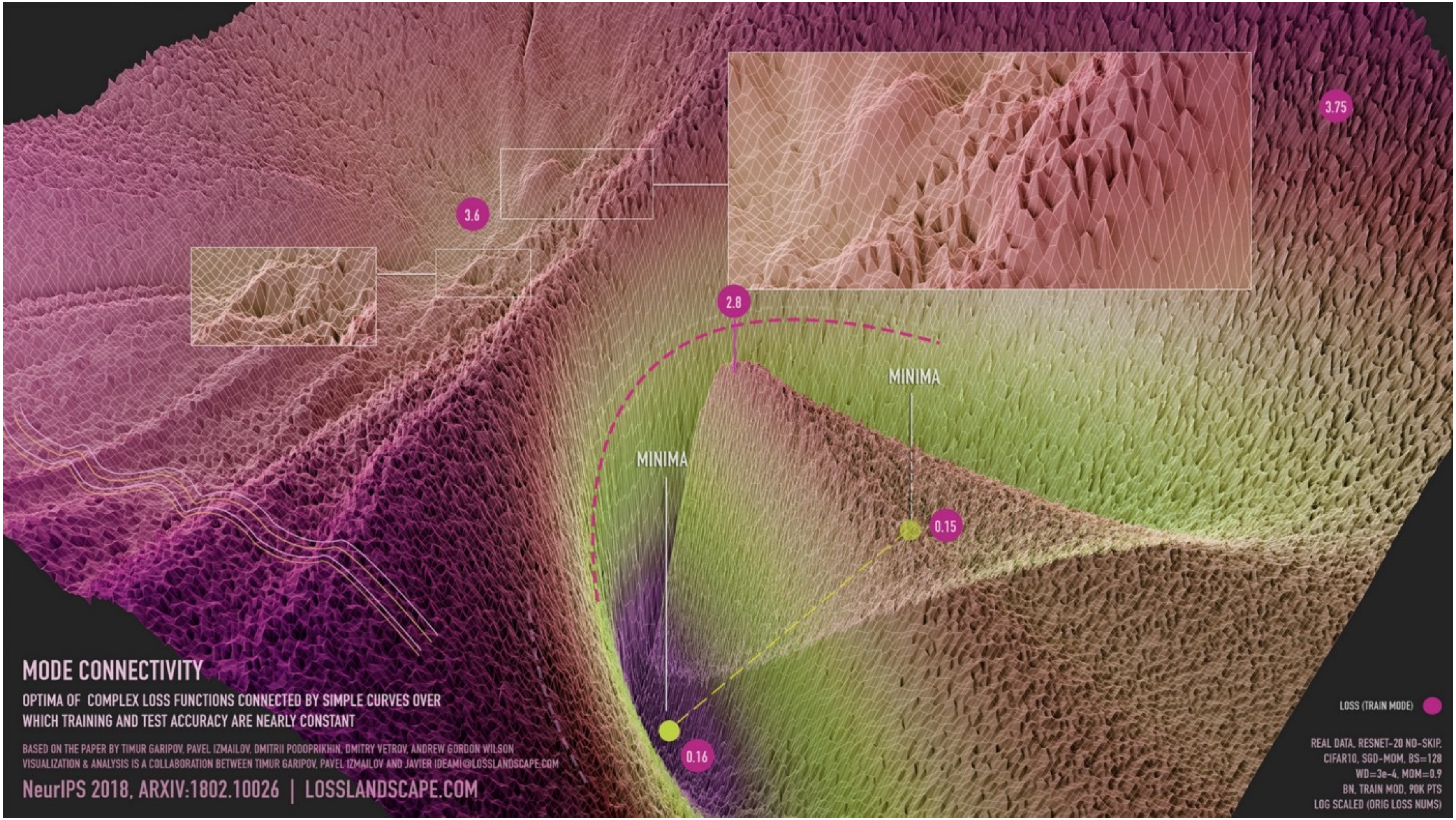
- Over-parametrizing a deep model often improves test performance, contrary to bias-variance tradeoff prediction

  - But we must control that:
    - Gradients don't vanish
    - Gradient amplitude is homogeneous across network
    - Gradients are under control when weights change

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

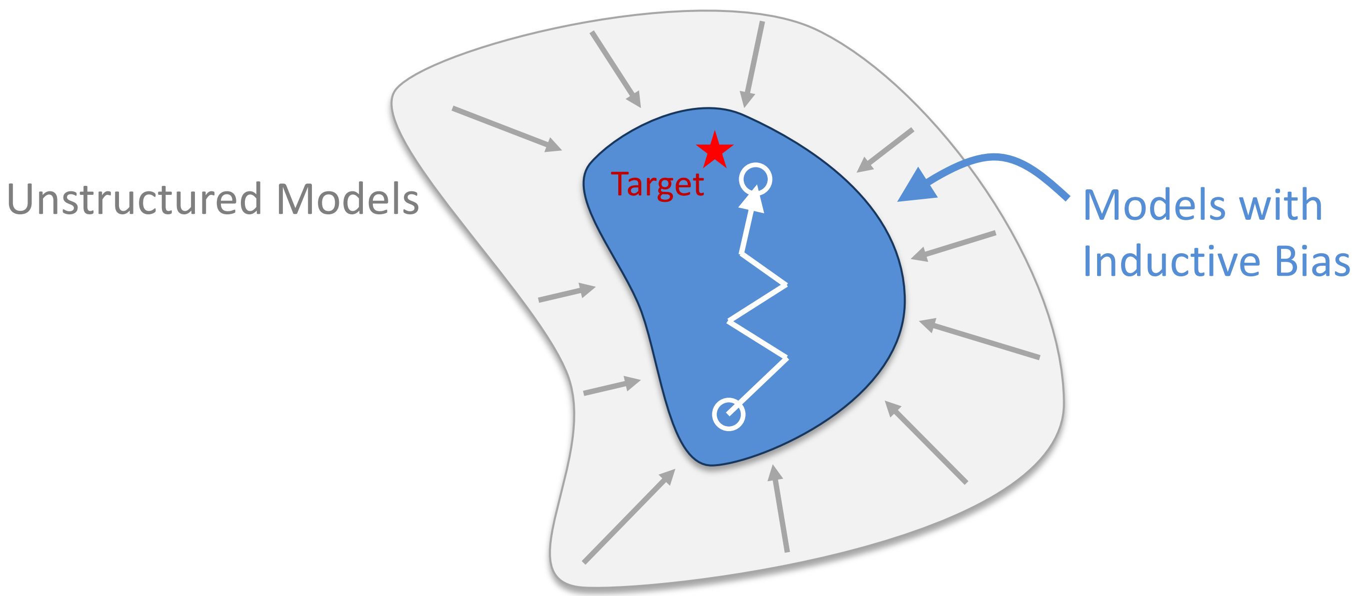
Deep Neural Networks Loss Landscape



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



Choosing the right function…

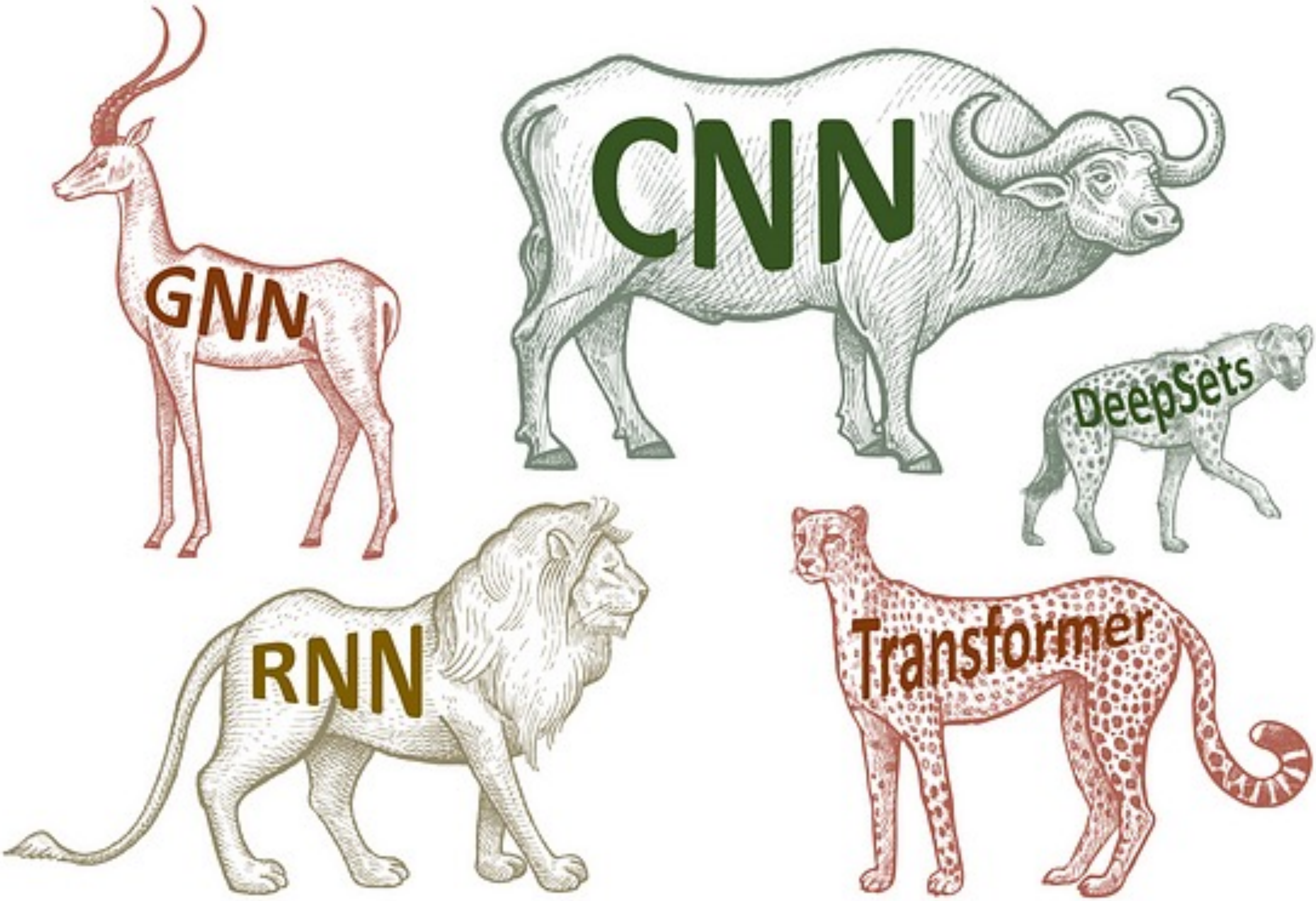
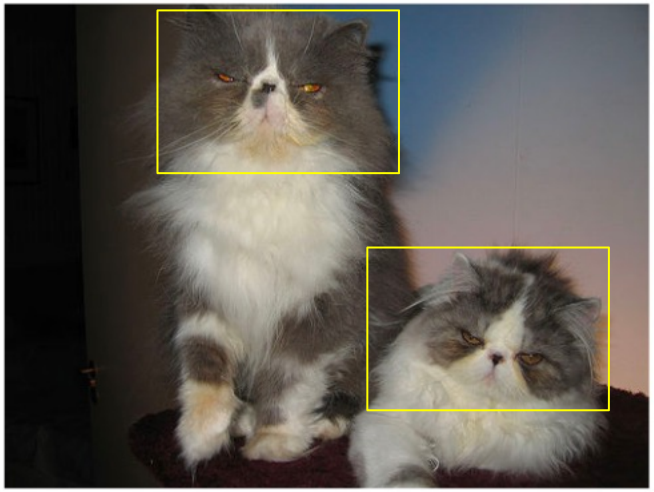
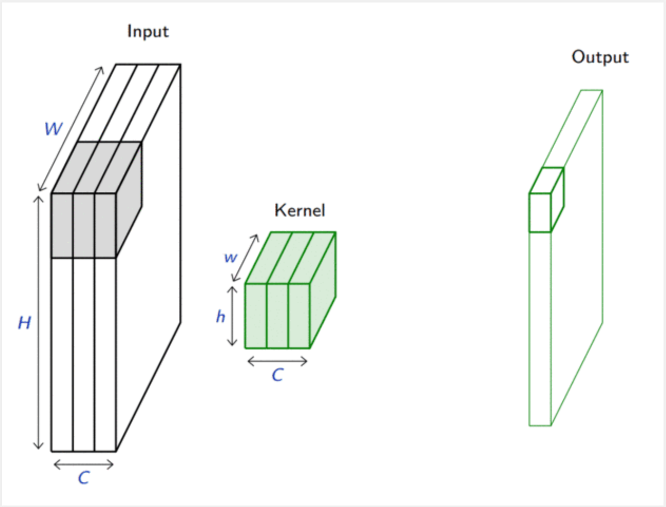


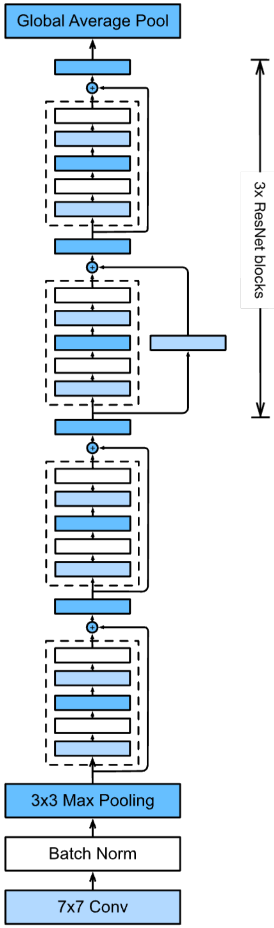
Image credit: Michael Bronstein

Example: Images & Convolutional Neural Networks

- When structure of data includes ***translation invariance***, a representation meaningful at one location should be used everywhere







ResNet
(He et al, 2015)

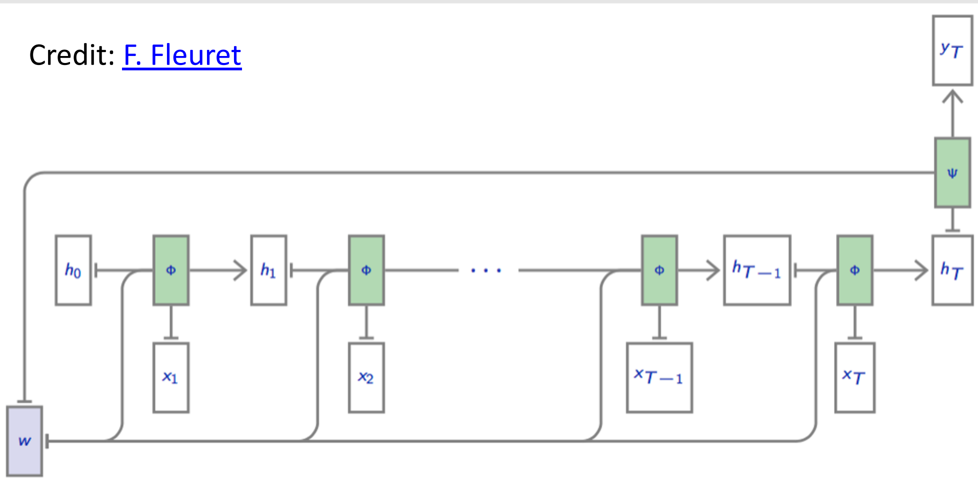
- Convolutional layers build on this idea: same "local" transformation applied everywhere and preserves signal structure

Fleuret, [Deep Learning Course]

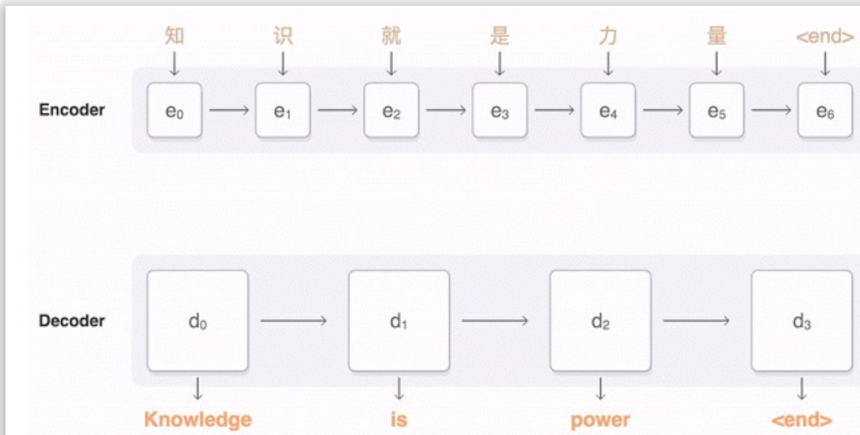
Example: Sequences & Recurrent Neural Networks

- Many data have temporal / sequence structure and are of variable length
  - Text, Video, Speech, DNA, …
  - Features can be local in time, but meaningful across time step: *a feature can happen any time*

- ***Recurrent layers*** allow sequential data processing, applying same transformations across time steps.



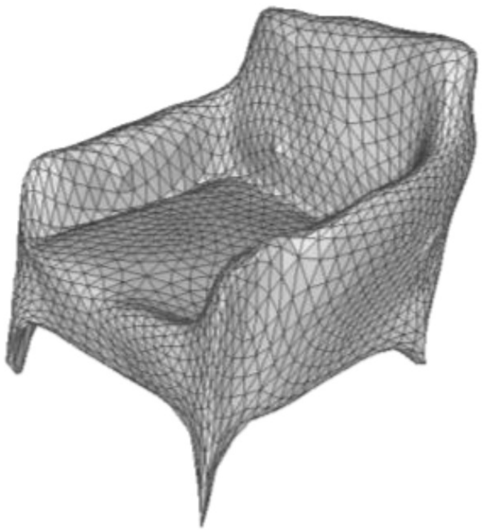
Credit: F. Fleuret



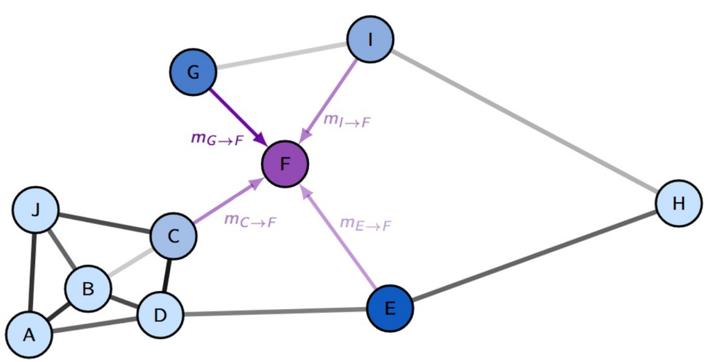
Y. Wu et al, 2016

Example: Geometric Data & Graph Neural Networks

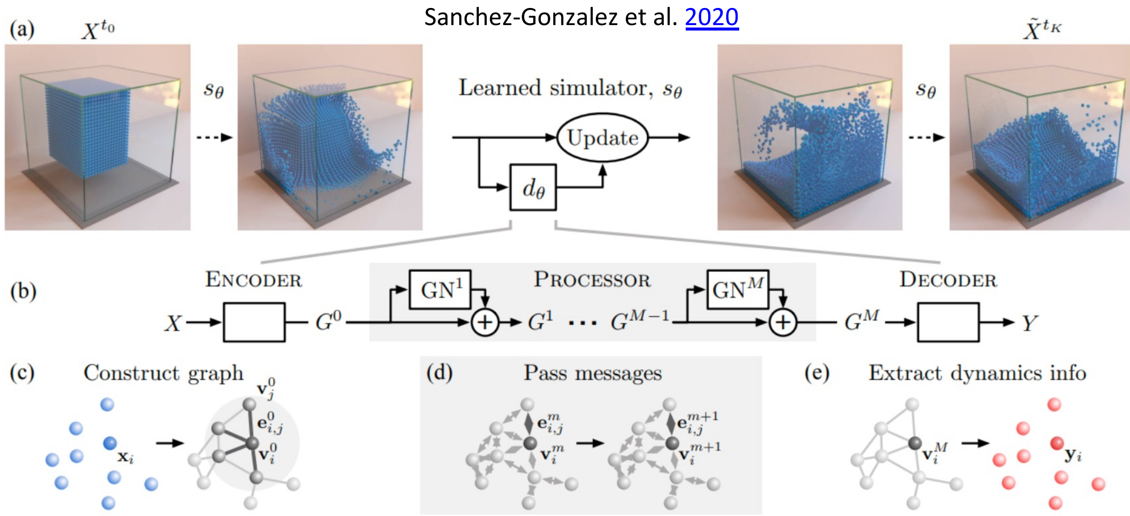
- Permutation invariant data with geometric relationships
  - Features can be local on graph, but meaningful anywhere on graph



- Graph layers can encode these relationships on nodes & edges



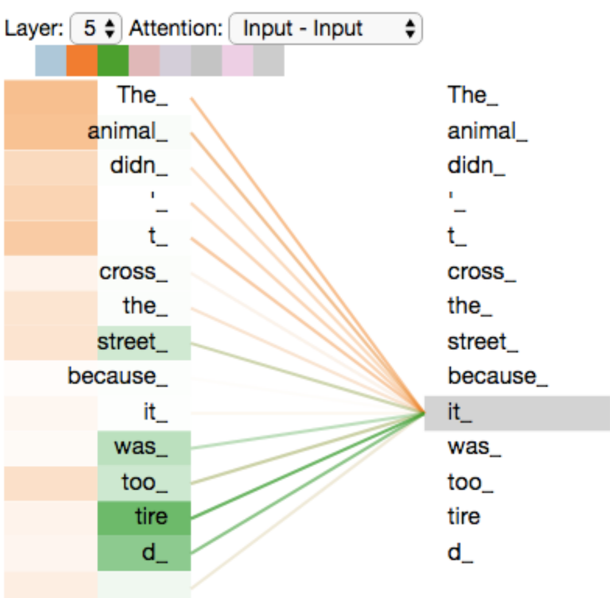
Sanchez-Gonzalez et al. 2020

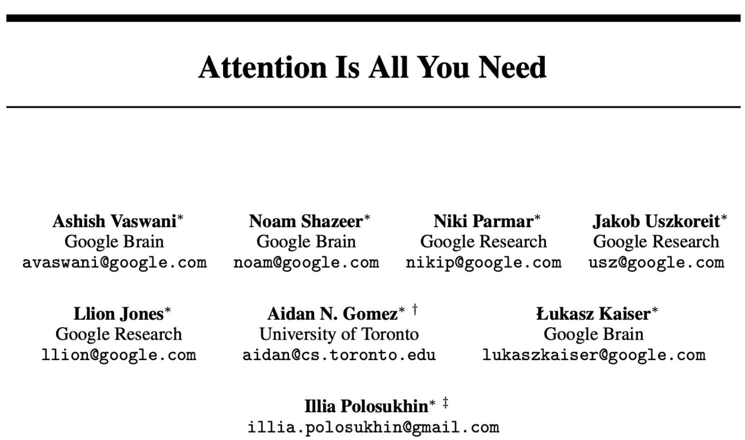


Examples: Transformers and Deep Sets

- **Deep Sets** and **Transformers** can process permutation invariant sets of data

- *Transformers are very adaptable*:
  Built using layers of ***attention***, they can also process sequences, images, and other data





Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

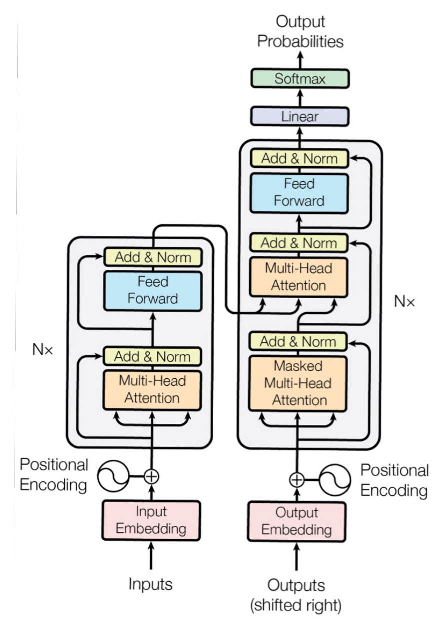
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

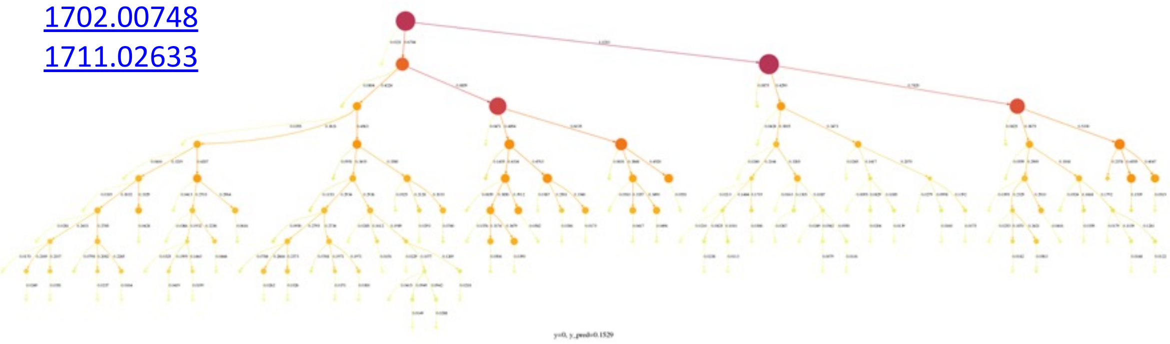
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



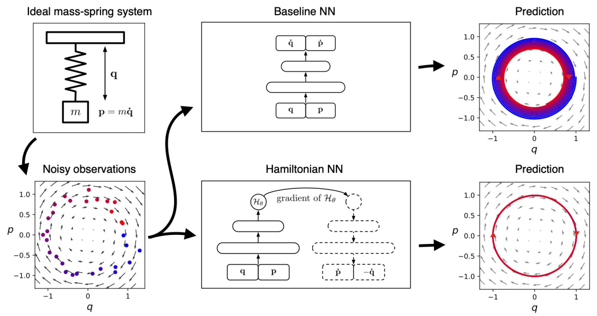
Physics Inspired Models

QCD Structured Neural Nets

1702.00748
1711.02633

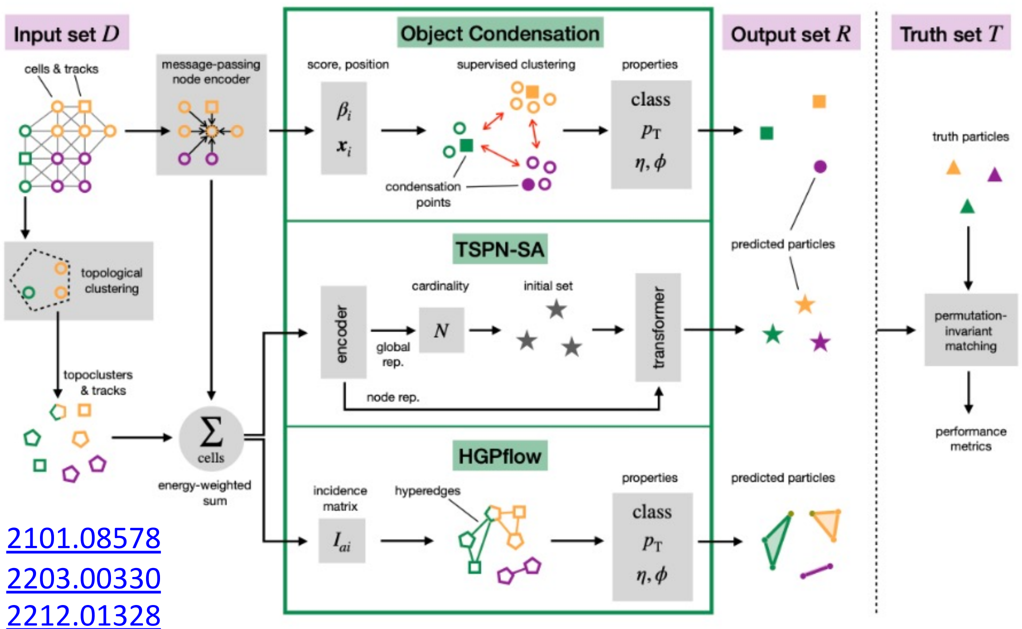


Hamiltonian Neural Nets



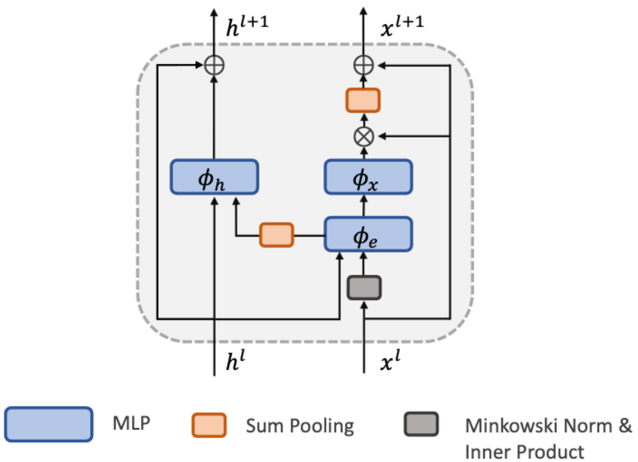
1906.01563

Neural Net Clustering for Particle Flow



2101.08578
2203.00330
2212.01328

Lorentz Equivariance



Lorentz Group Equivariant Block (LGEB)

2201.08187

Summary

- Neural Networks allow us to combine non-linear basis selection with feature learning
- Deep neural networks allow learning complex function by hierarchically structuring the feature learning
- We can use our inductive bias (knowledge) to define models that are well adapted to our problem
- Many neural networks structures are available for training models on a wide array of data types.
- More details in talks this week by:
  K. Terao, C. Adams, M. Liu