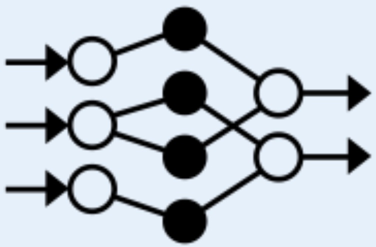
Introduction to Neural Networks

Qiyang Hu
UCLA Office of Advanced Research Computing

Oct 28th, 2022

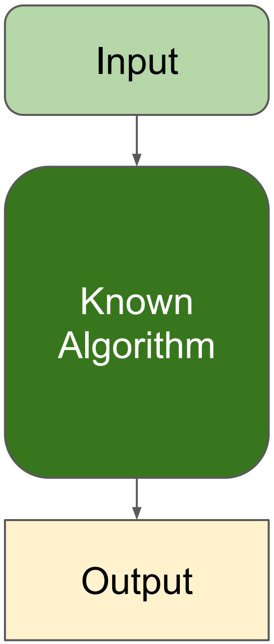
About this talk



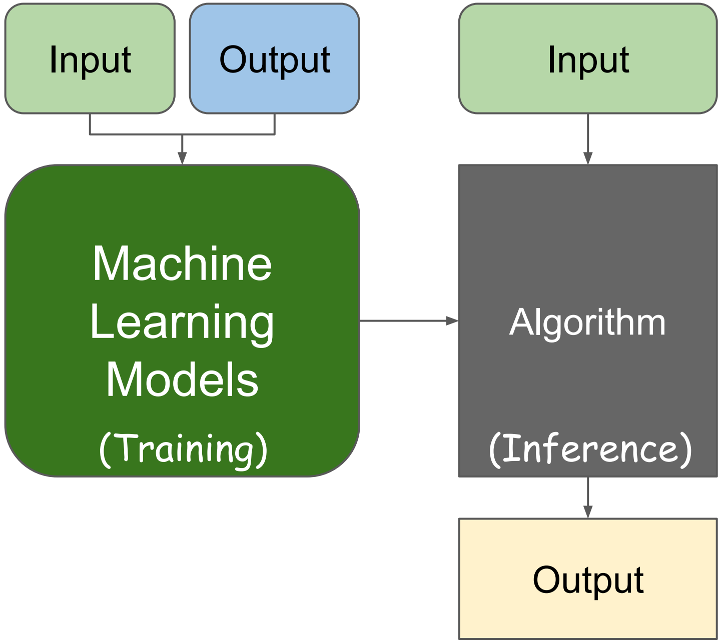
- An introduction, an overview
  - The intuitive explanations on basic concepts
  - The advanced technical developments
- The outline
  - Machine learning and Deep learning
  - Neural network modeling in a general ML/DL workflow
- My DL talks in this and next quarters
  - Introduction to NN (today)
  - Learning PyTorch (next Wednesday)
  - Deep learning, the GBU (next Friday)
  - Special NN topics, (conv, gans, transformer, lstm?) (next quarter)

What is Machine Learning?

Traditional Programming



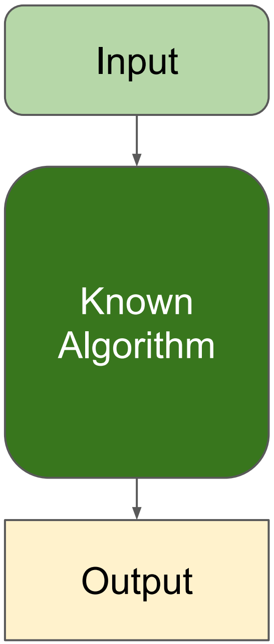
Machine Learning



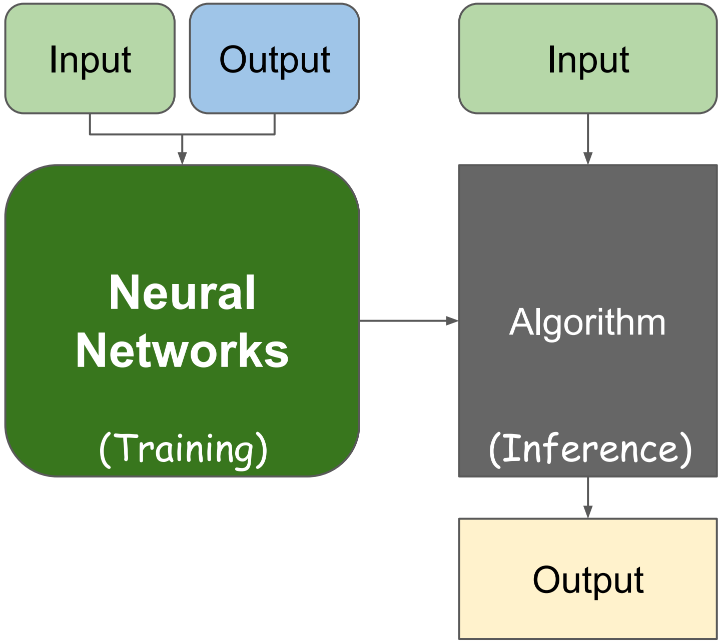
Qiyang Hu

What is Deep Learning?

Traditional Programming

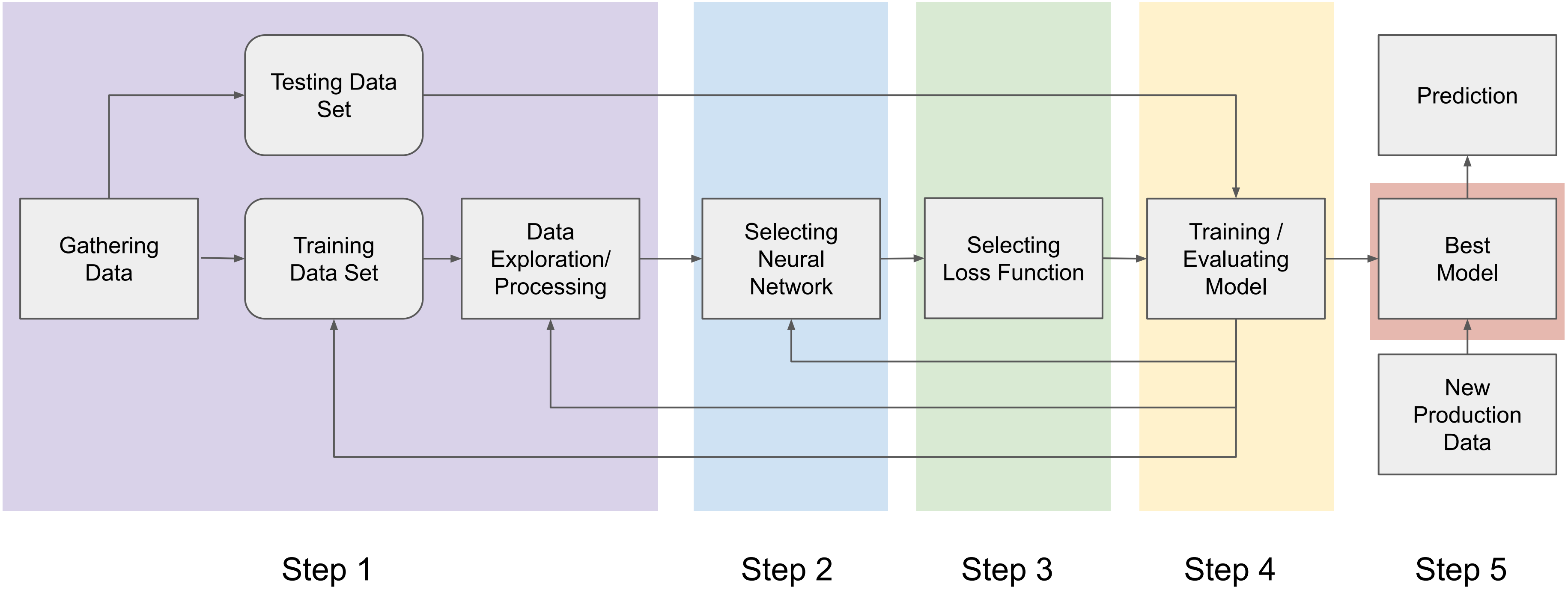


Deep Learning



Qiyang Hu

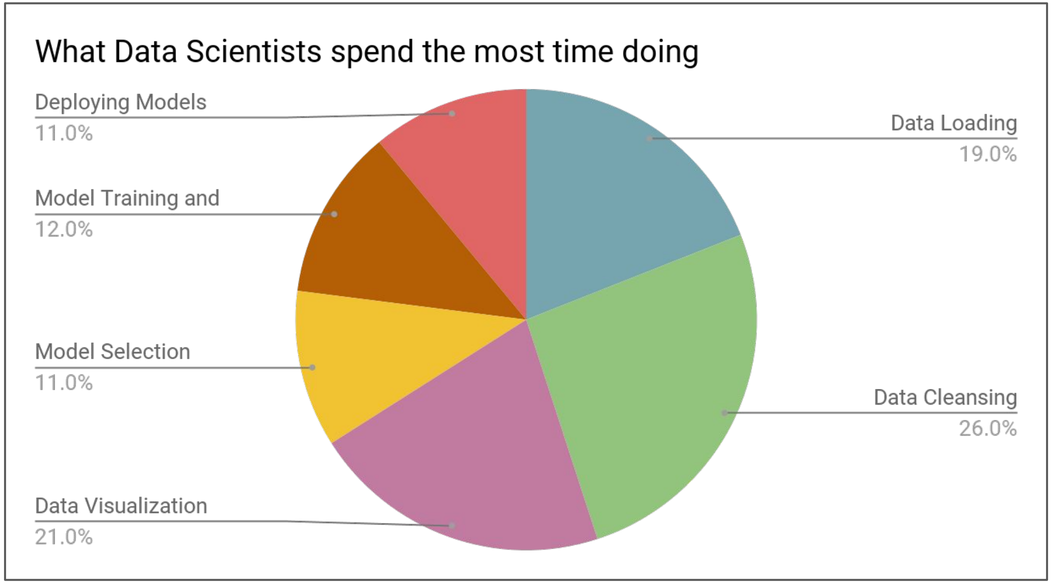
Simplified workflow for a deep learning project



Qiyang Hu

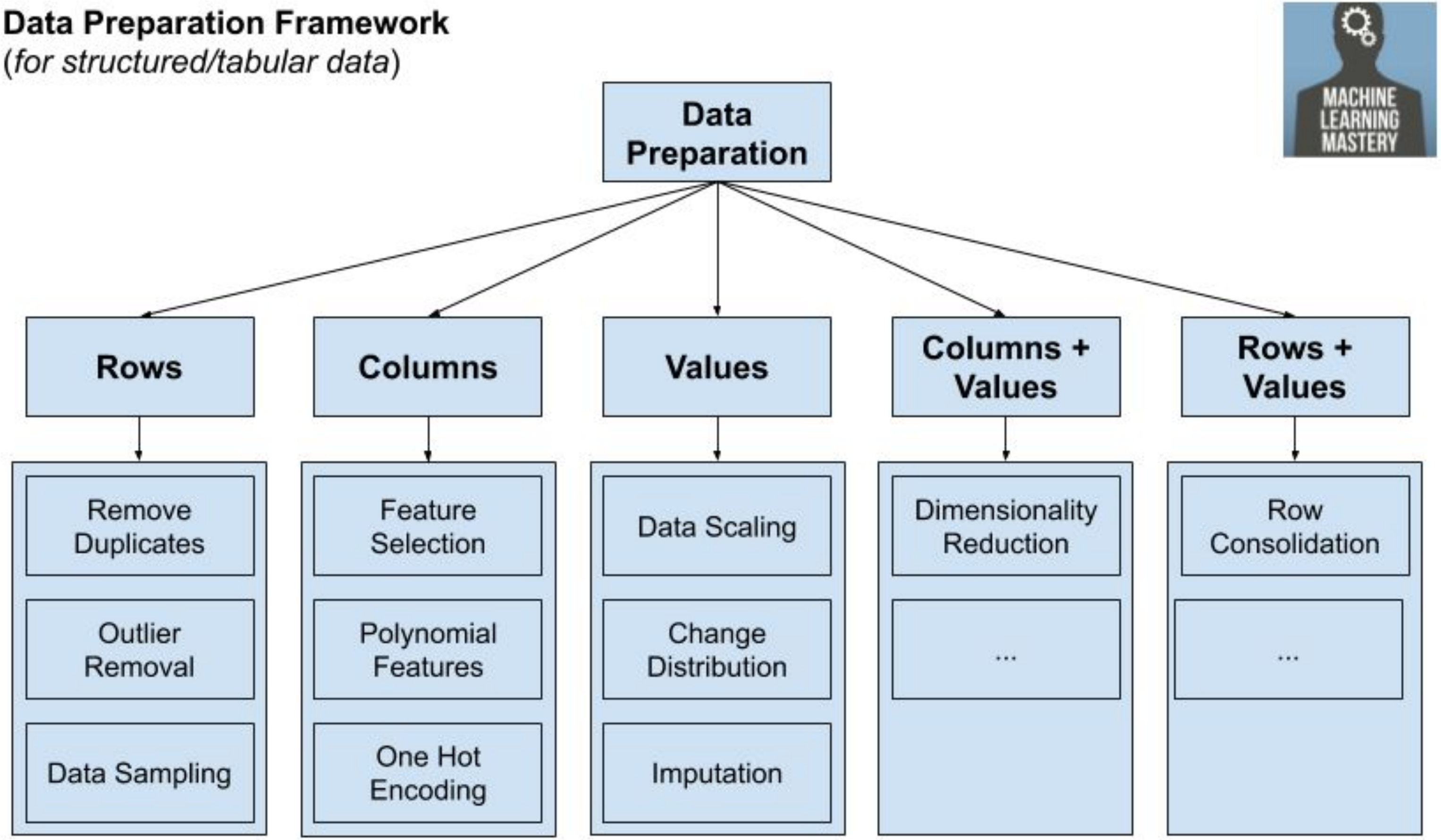
Step 1. Data Preparation and Processing

- The most time-consuming but the most *creative* job
  - Take ~66% time
  - Require experience
  - May need domain expertise
- Determines the upper limit for the goodness of DL
  - Models/Algorithms: just approach the upper limit



Anaconda’s State of Data Science Report, 2020 (Source)

Data Preparation Framework
(*for structured/tabular data*)



Qiyang Hu

More data-prep tasks might be needed

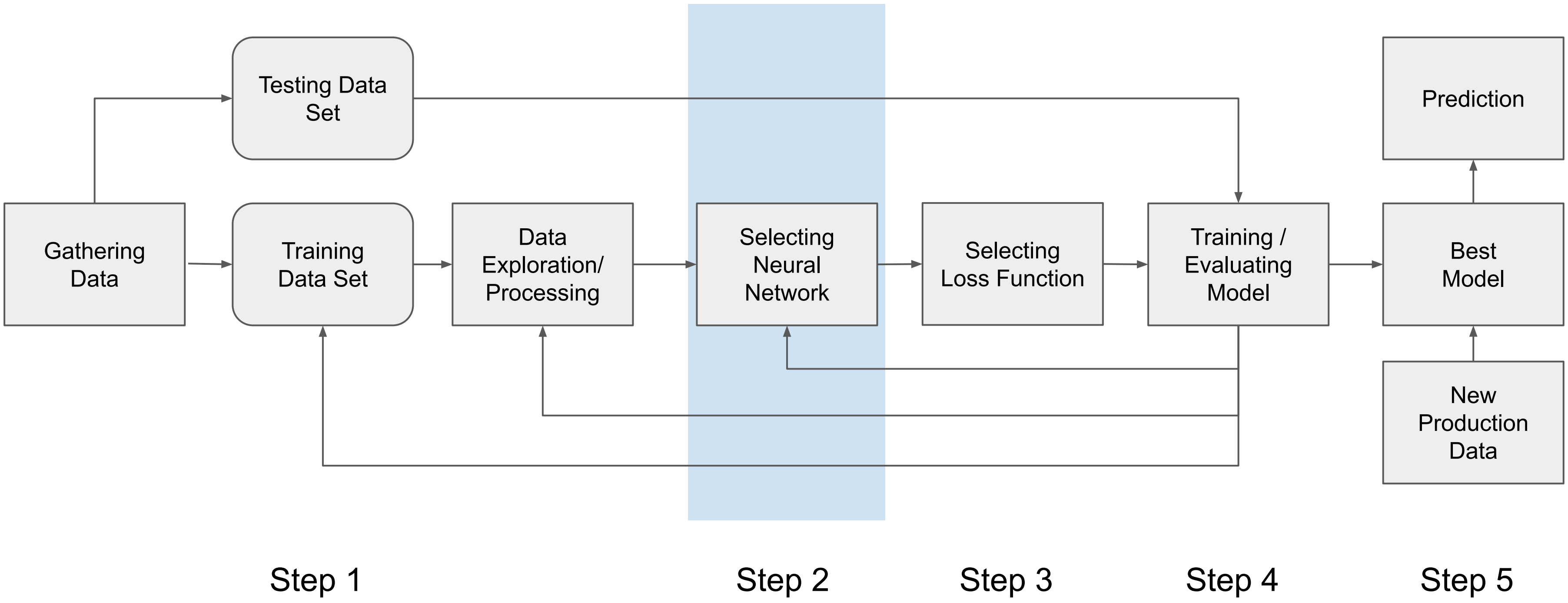
- Image Data Processing
  - Pixel scaling
  - Train-Time Augmentation
  - Test-Time Augmentation
  - Convolution and Flattening

- Data Tokenization
  - Breaking the sequence data into units
  - Mapping units to vectors
  - Aligning & padding sequences

- Data Embedding
  - Map data to lower-dim vectors
    - Sparse to dense
    - Merging diverse data
    - Preserve relationship
  - Techniques
    - Std Dimensionality Reduction
    - Word2Vec
    - Be part of the model training
  - *Representation Learning*

$$\text{Embedding Dims} \approx \sqrt[4]{\text{Possible Values}}$$

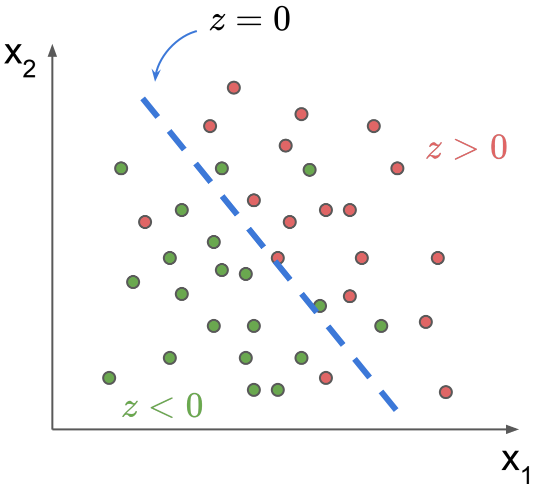
Workflow for a deep learning project



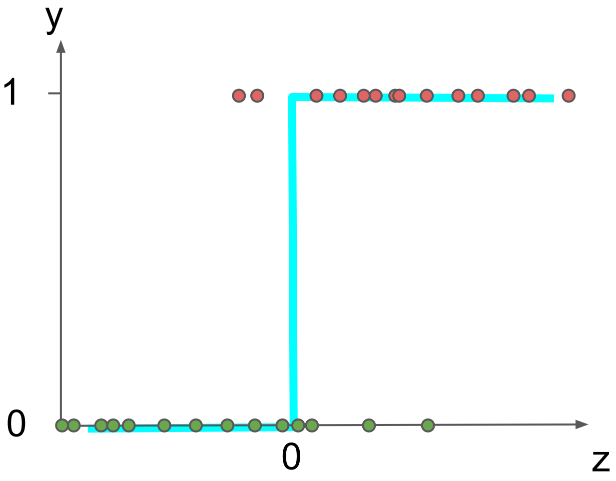
Qiyang Hu

What is Neural Network?

- Recap for simple linear classification problem



$$z = w_1 x_1 + w_2 x_2 + b$$

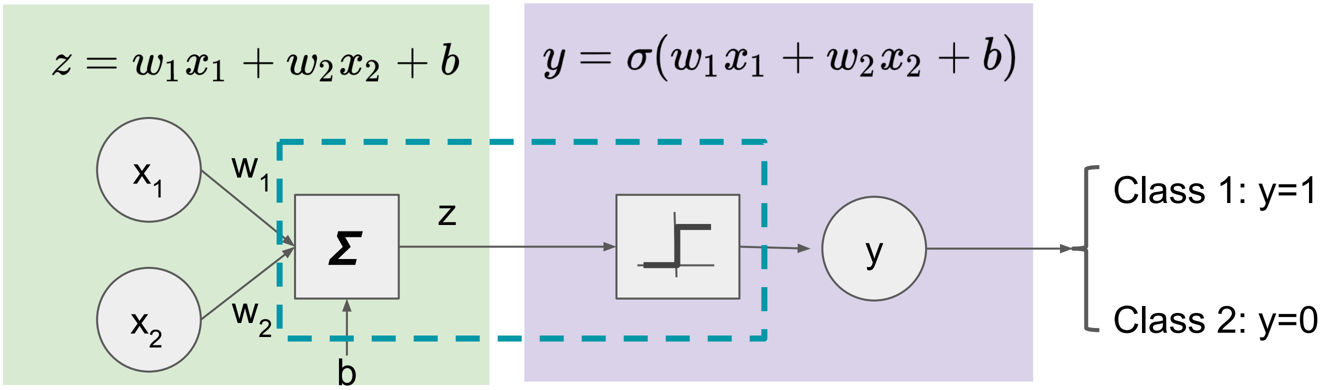


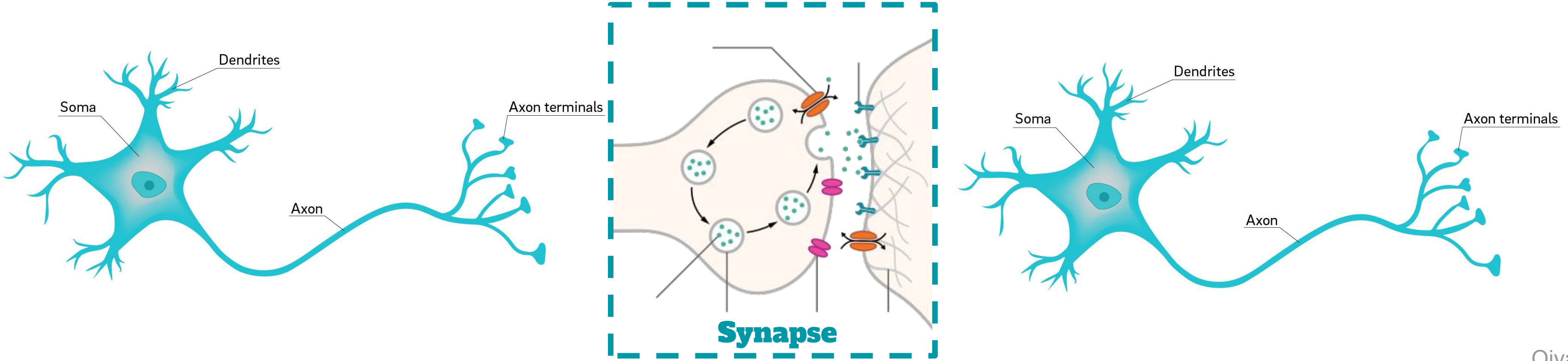
$$y = \sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Qiyang Hu

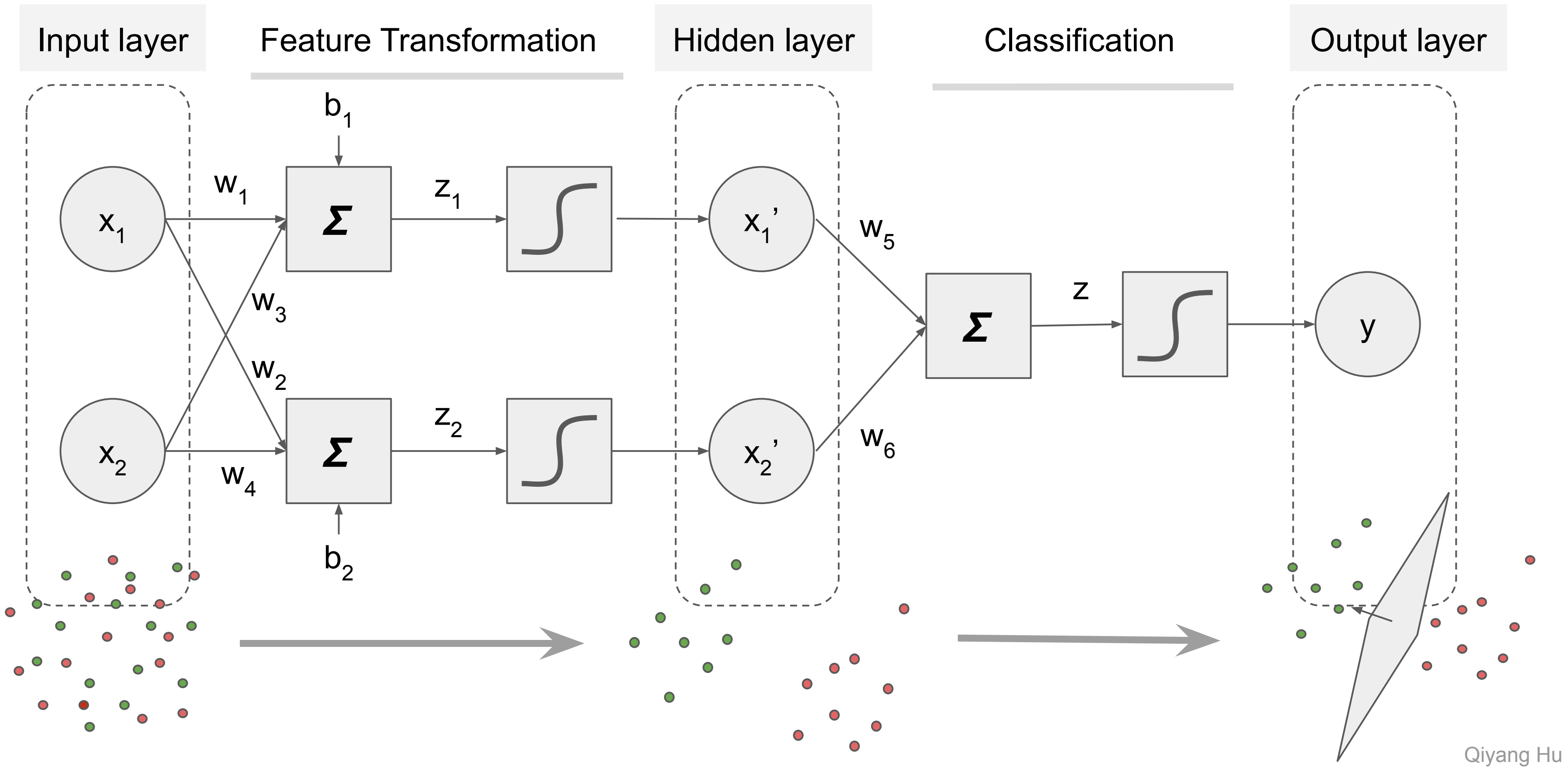
Artificial Neuron and Biological Neuron

McCulloch-Pitts (MCP) neuron model



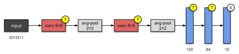


Neural Networks ~ piling/stacking logistic-regression classifiers



Deep neural networks

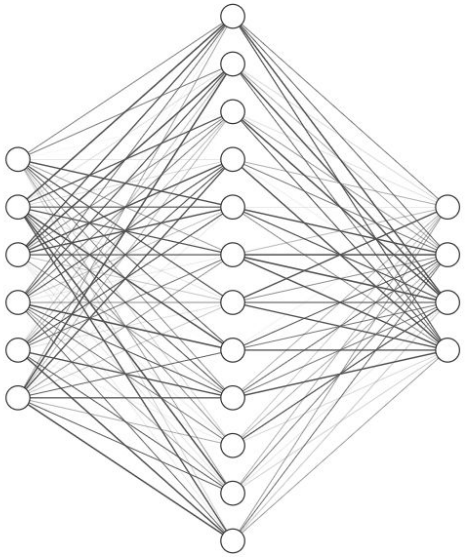
- LeNet-5 (1998)



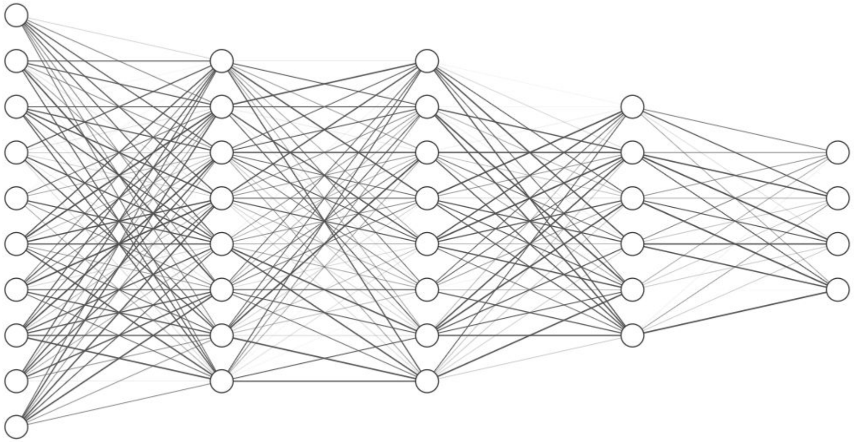
Year	CNN	Developed by	Place	Top-5 error rate	No. of parameters
1998	LeNet(8)	Yann LeCun et al			60 thousand
2012	AlexNet(7)	Alex Krizhevsky, Geoffrey Hinton, Ilya Sutskever	1st	15.3%	60 million
2013	ZFNet()	Matthew Zeiler and Rob Fergus	1st	14.8%	
2014	GoogLeNet(19)	Google	1st	6.67%	4 million
2014	VGG Net(16)	Simonyan, Zisserman	2nd	7.3%	138 million
2015	ResNet(152)	Kaiming He	1st	3.6%	

Why deep?

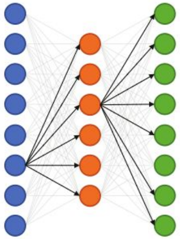
- Shallow network can fit any function
  - Has less number of hidden layers
  - Has to be really “fat”

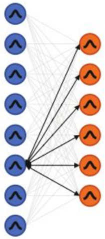


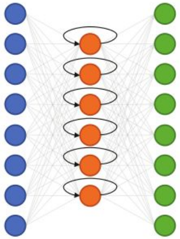
- Deep network is more efficient.
  - Exponentially fewer parameters (2017)
  - It can extract/build better features

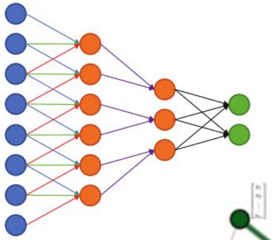


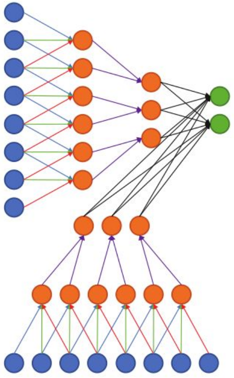
Deep neural *networks*

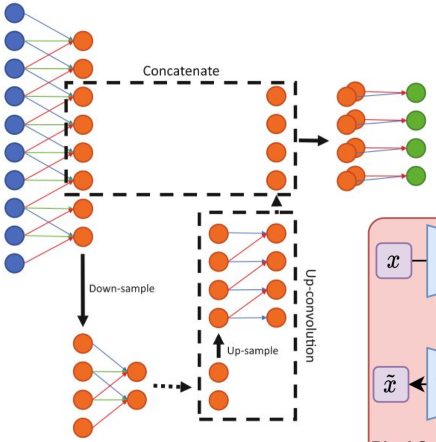


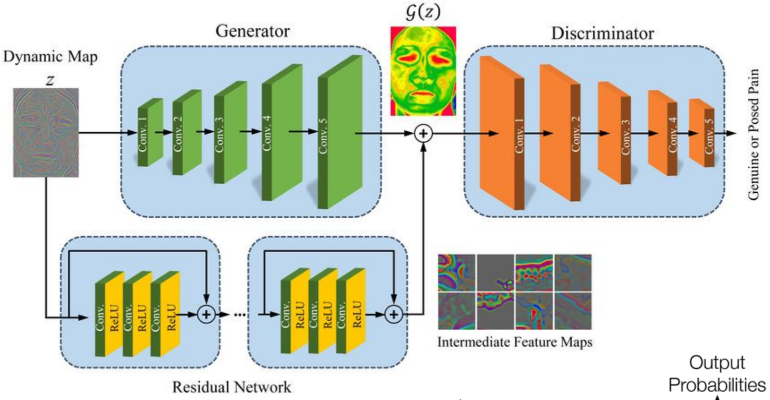


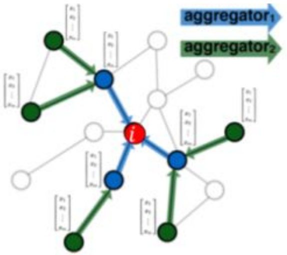


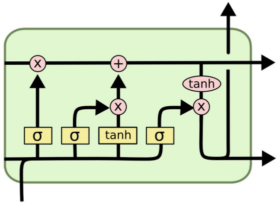


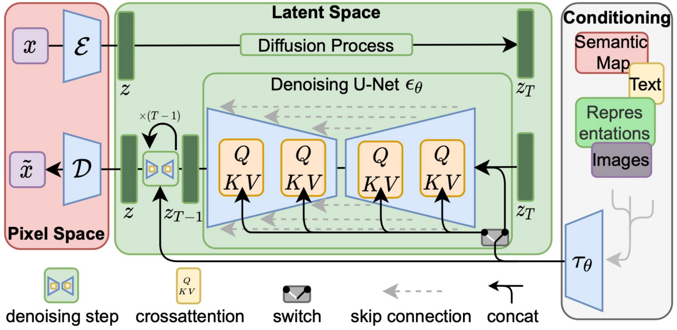


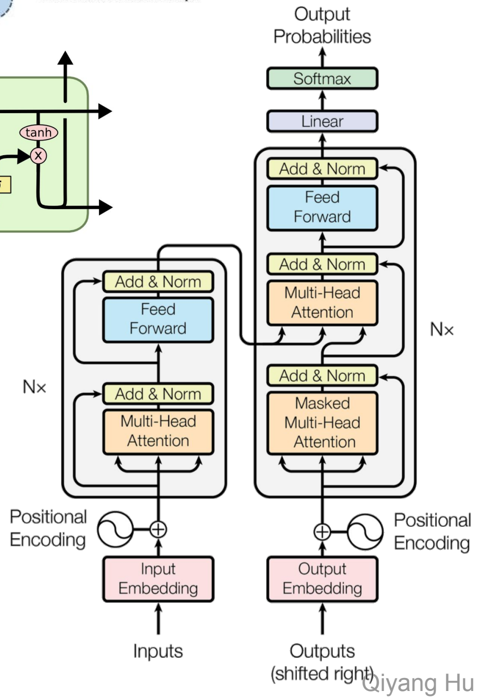








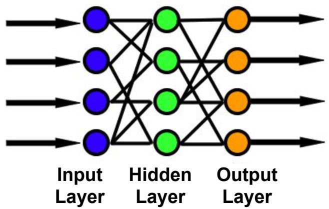




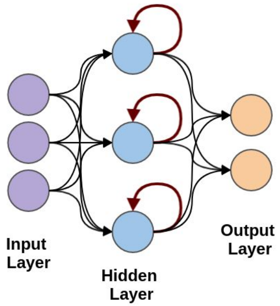
Qiyang Hu

A simpler classification of neural network types

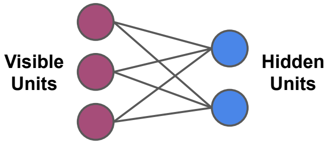
- Feed forward neural networks (No cycle in node connections)
  - Fully connected network
  - Convolutional networks (CNNs)



- Recurrent networks (w/ directed cycle in node connections)
  - Fully recurrent NN
  - Recursive NN
  - Long short-term memory (LSTM)
  - Hopfield network (w/o hidden nodes)

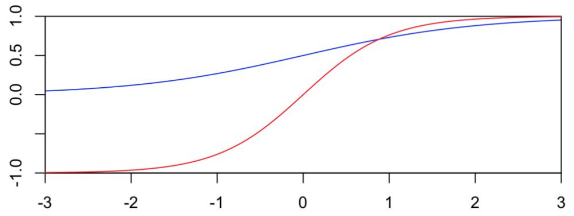


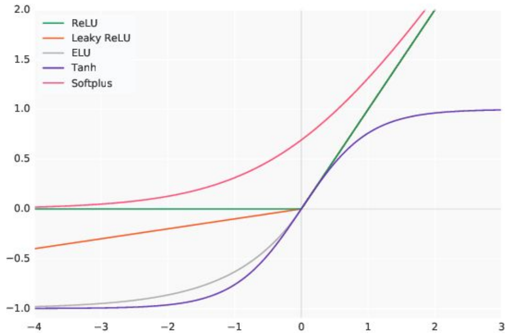
- Symmetric networks (no directions in node connections)
  - Boltzmann Machines
    - RBM, DBM, SOM



Activation Function

- Sigmoid function: $$\sigma(z) = \frac{1}{1+\exp(-z)}$$
- tanh function: $$tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$$
- Rectified linear unit (ReLU) $$f(x) = x^+ = \max(0, x)$$
  - Softplus
  - Leaky ReLU
  - Exponential LU (ELUs)
  - GELU
  - Dynamic ReLU
- Softmax function: $$y_i = \frac{e^{z^{(i)}}}{\sum_{j=0}^{K} e^{z^{(j)}}}$$
- Maxout Network:
  - *Learnable* activation function

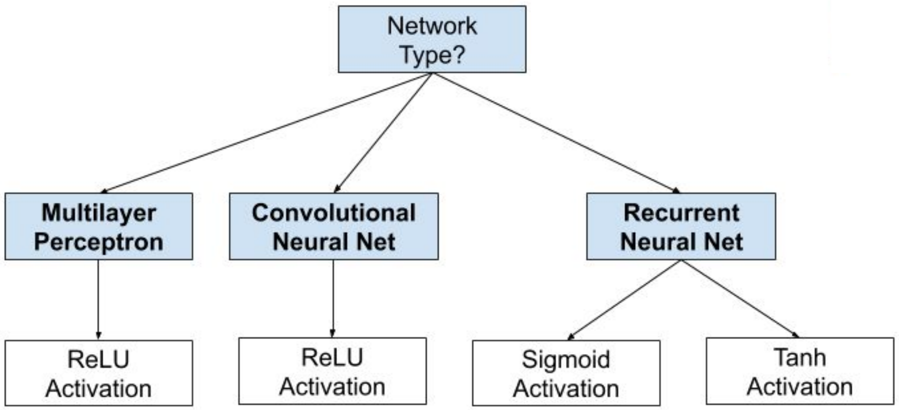




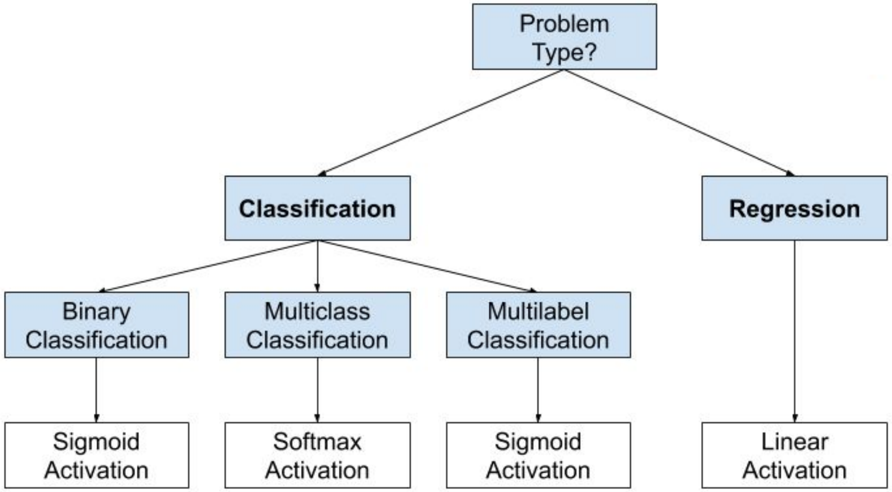
Qiyang Hu

How to choose activation functions?

For hidden layers

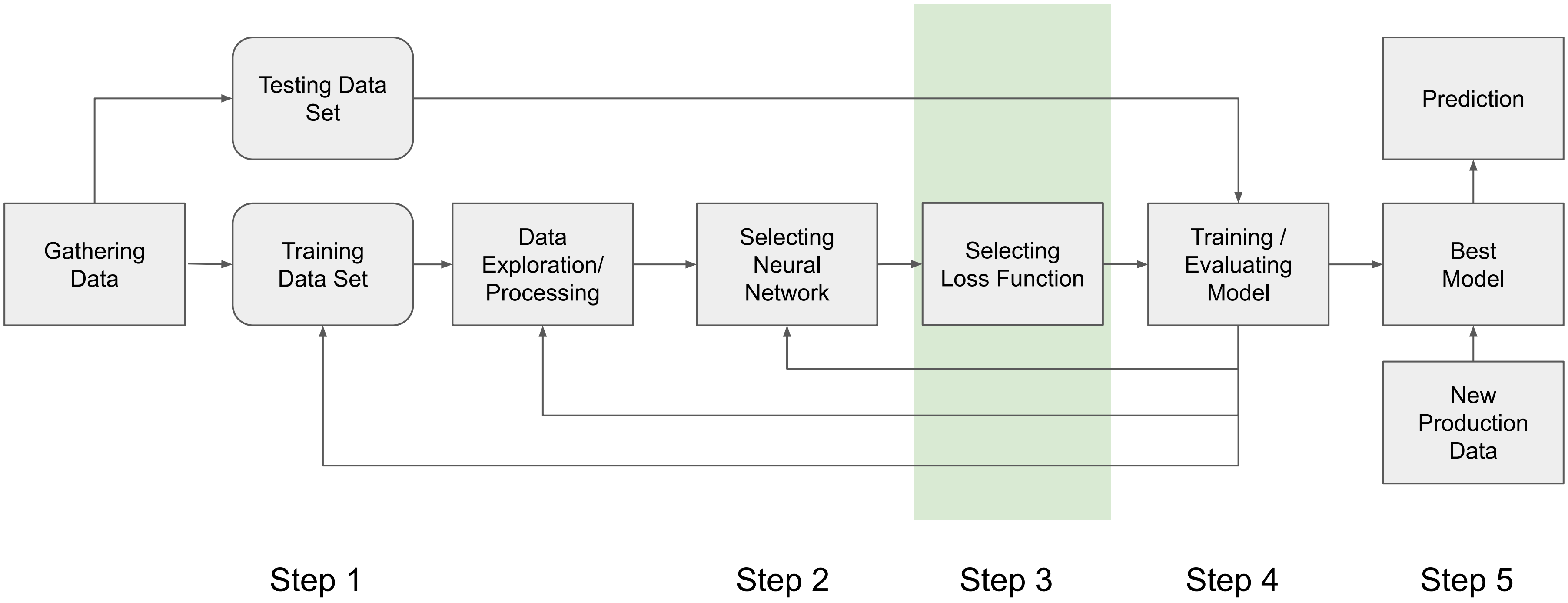


For output layers



From [Machine Learning Mastery Blog Post]

Workflow for a deep learning project



Qiyang Hu

How to measure the performance of the model?

- General name: objective function

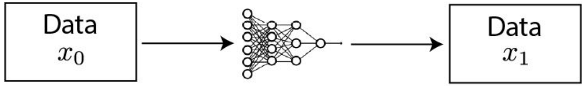
- Measure the misfit of the model as a function of parameters
  - Criterion is to *minimize* the error functions
  - <u>Loss Function</u>, <u>Cost Function</u>: a penalty on difference between predictions and labels

- Evaluate the probability of *generating* training set
  - Criterion is to *maximize* the distribution likelihood as a function of parameters
  - <u>Maximum (log)-likelihood estimation</u>: minimize the divergence of distributions

- Regression losses and classification losses

Loss functions

- Generative/Predictive:



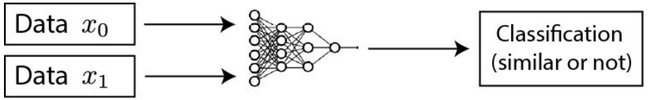
  - Regression Loss
    - Mean Square Error / Quadratic Loss / L2 Loss:
    - Mean Absolute Error / L1 Loss:
    - Huber Loss
    - Quantile Loss
  - Cross-Entropy Loss and variations
    - Log Loss / Negative Log Likelihood
    - Weighted CE / Balanced CE / Focal Loss
    - Dice Loss / IOU Loss / Tversky Loss

$$L_{MSE} = \frac{1}{n} \sum_{i}^{n} (t_i - s_i)^2$$

$$L_{MAE} = \frac{1}{n} \sum_{i}^{n} |t_i - s_i|$$

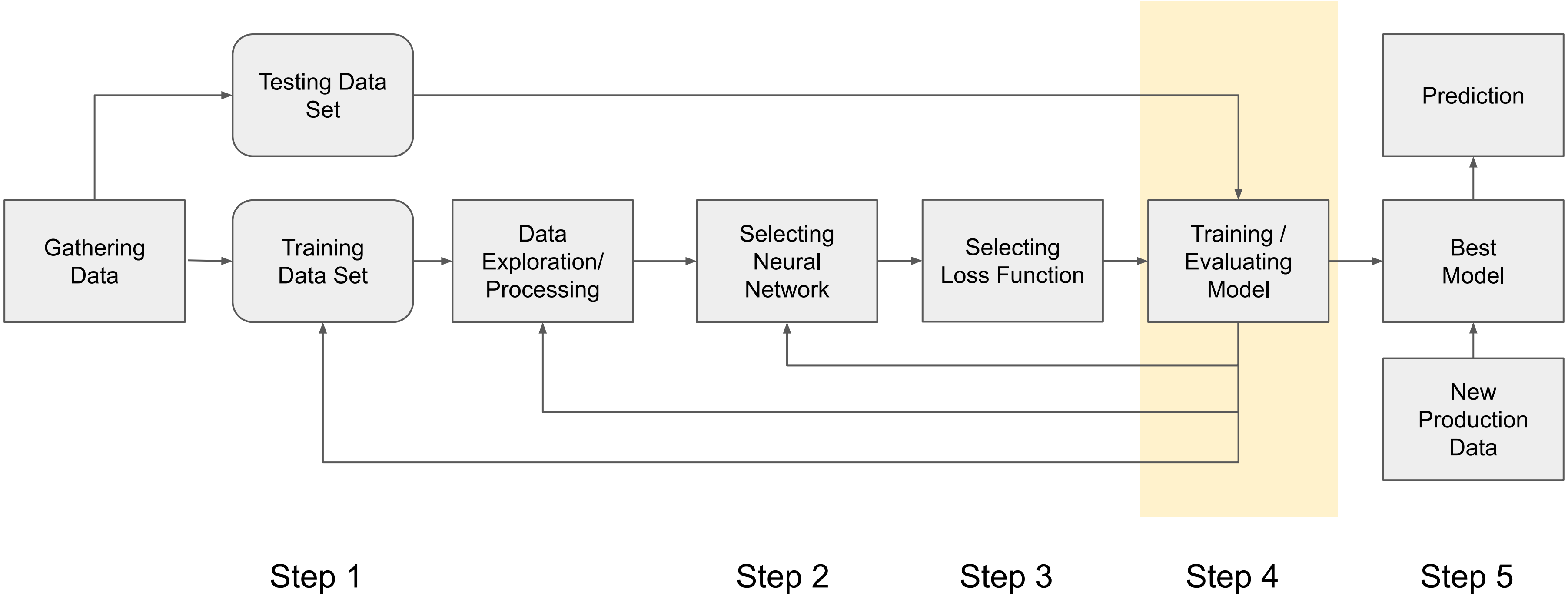
$$L_{CE} = -\sum_{i}^{C} t_i \log(s_i)$$

- Contrastive:

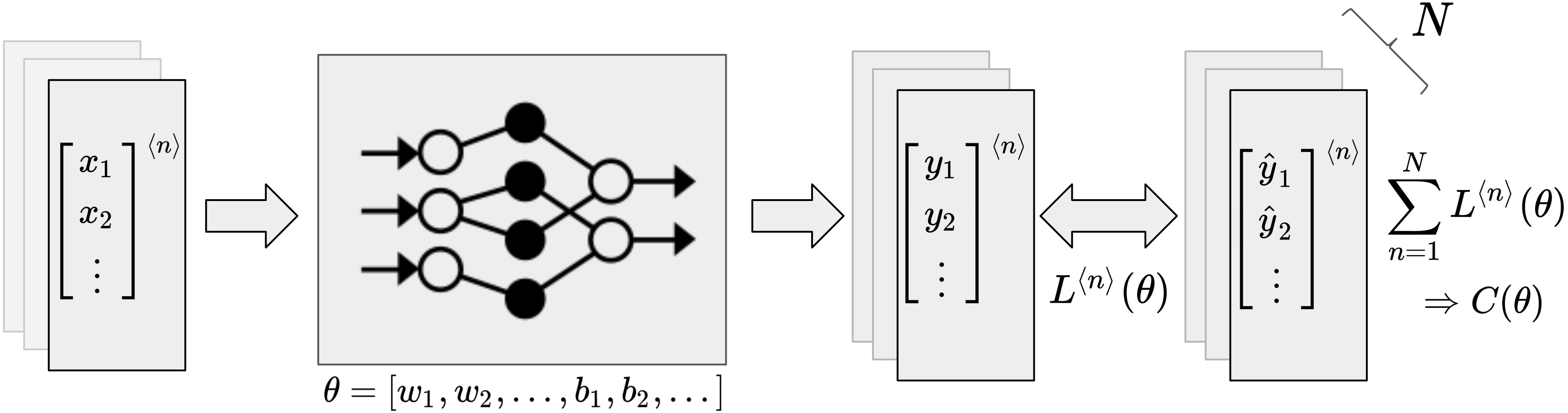


  - Ranking Loss/Margin Loss/Contrastive Loss/Triplet Loss

Workflow for a deep learning project



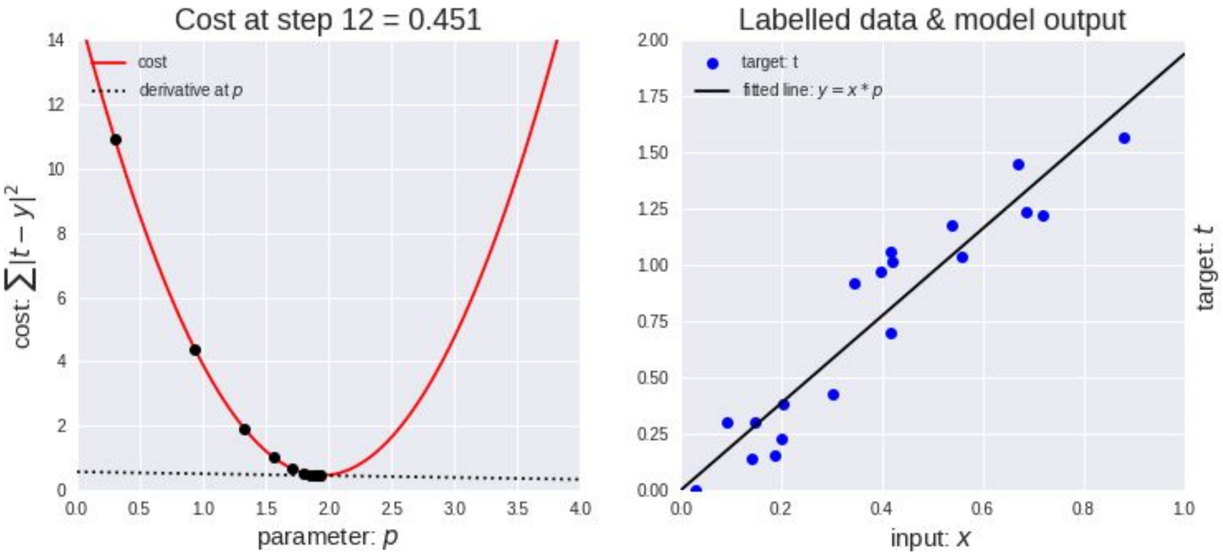
Training a DNN is an optimization problem



- We know how to compute *C(θ)*, analytically or numerically.
- Start from an arbitrary initialization of θ_0, and get an initial $C_0(\theta)$
- Apply optimization algorithm to minimize *C(θ)*

Neural Network's Optimization

- Gradient Descent (a 1st-order approach) $\theta \longleftarrow \theta - \eta \nabla L(\theta)$
  - Most popular algorithm
    - Pros: simple and fast
    - Cons: sometimes hard to tune



[Source Link]

Gradient-Descent Optimizers

- Stochastic GD / Mini-Batch GD
- Adding momentum:
    - Classical Momentum (CM)
    - Nesterov's Accelerated Gradient (NAG)
- Adaptive learning rate:
    - AdaGrad, AdaDelta, ...
    - RMSprop
- Combining the two
    - **ADAM** (as **default** in many libs)
- Beyond Adam:
    - Lookahead (2019), RAdam (2019)
    - AdaBound/AmsBound (ICLR 2019)
    - Range (2019)
    - AdaBelief (NeurIPS 2020 Spotlight)

Gradient descent vs Momentum vs AdaGrad vs RMSProp vs Adam

(Source)

Higher Order Optimization Algorithms

- Newton-like methods (2nd-order methods)

$$\theta \longleftarrow \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$$

    - Prons: **fewer** iterations, fewer hyperparameters
    - Cons: much more **costly** in each iteration, more storing
  - DFP/Broyden/BFGS/L-BFGS: a quasi-newton one
    - Good for low dimensional models
  - Conjugate gradient (CG): between GD and Newton
    - moderately high dimensional models

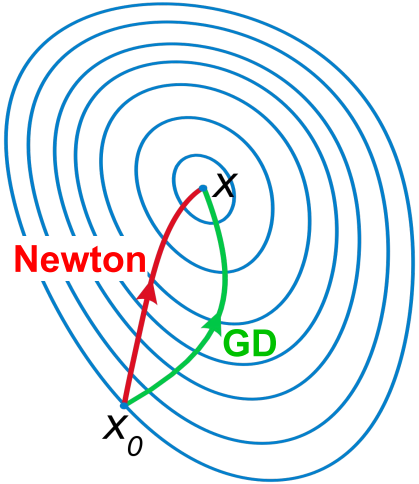
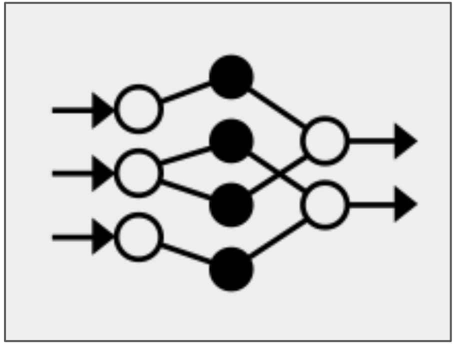


Figure from Wikipedia

- Natural gradient descent methods $\nabla_\theta L(\theta) = F^{-1} \nabla_\theta L(\theta)$
  - K-FAC (Martens and Grosse, 2015)
  - Shampoo (Gupta, et al., 2018)
  - K-BFGS (Goldfarb, et al., NeurIPS 2020)

Using Gradient Descent to train DNN

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}^{\langle n \rangle}$$



$$\theta = \underbrace{[w_1, w_2, \ldots, b_1, b_2, \ldots]}_{\text{Millions of parameters!}}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix}^{\langle n \rangle} \quad L^{\langle n \rangle}(\theta) \quad \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \end{bmatrix}^{\langle n \rangle} \quad N$$

$$\sum_{n=1}^{N} L^{\langle n \rangle}(\theta)$$

$$\Rightarrow C(\theta)$$

$$\theta_0 \to \nabla C(\theta_0) \to \theta_1 \to \nabla C(\theta_1) \to \theta_2 \to \ldots$$

$$\theta_1 = \theta_0 - \eta \nabla C(\theta_0)$$

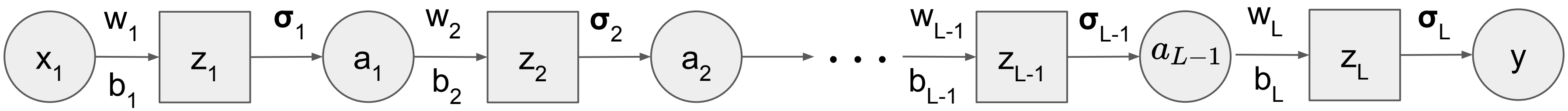
$$\theta_2 = \theta_1 - \eta \nabla C(\theta_1)$$

$$\vdots$$

$$\nabla C(\theta) = \sum_{n=1}^{N} \begin{bmatrix} \frac{\partial L^{\langle n \rangle}(\theta)}{\partial w_1} \\ \frac{\partial L^{\langle n \rangle}(\theta)}{\partial w_2} \\ \vdots \\ \frac{\partial L^{\langle n \rangle}(\theta)}{\partial b_1} \\ \vdots \end{bmatrix}$$

How to compute the gradient vector with millions of elements **efficiently?**

Backpropagation: a game of chain rule



$$y = \sigma_L\Big(w_L \cdot \sigma_{L-1}\big(\cdots w_2 \cdot \underbrace{\sigma_1(\underbrace{w_1 \cdot x + b_1}_{z_1})}_{a_1} + b_2\big) + b_L\Big)$$

$$\frac{\partial C\big(y(w)-\hat{y}\big)}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial C}{\partial z} = \frac{\partial z}{\partial w}\left[\frac{\partial a}{\partial z}\frac{\partial C}{\partial a}\right] = \frac{\partial z}{\partial w}\left[\sigma' \cdot \left(\frac{\partial z_{(+1)}}{\partial a}\frac{\partial C}{\partial z_{(+1)}}\right)\right]$$

① Forward Pass

$$\frac{\partial z_1}{\partial w_1} = x_1 \longrightarrow \frac{\partial z_2}{\partial w_2} = a_1 \longrightarrow \cdots \longrightarrow \frac{\partial z_{L-1}}{\partial w_{L-1}} = a_{L-2} \longrightarrow \frac{\partial z_L}{\partial w_L} = a_{L-1}$$

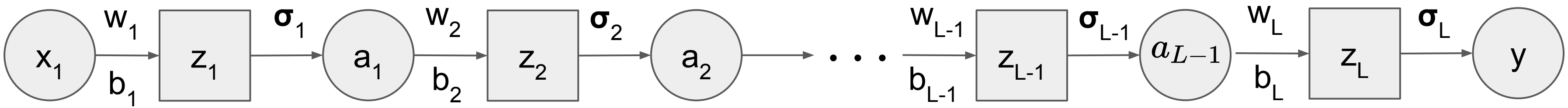
② Backward Pass

$$\frac{\partial C}{\partial z_1} = \sigma_1'\left[w_2\frac{\partial C}{\partial z_2}\right] \longleftarrow \cdots \longleftarrow \frac{\partial C}{\partial z_{L-1}} = \sigma_{L-1}'\left[w_L\frac{\partial C}{\partial z_L}\right] \longleftarrow \frac{\partial C}{\partial z_L} = \sigma_L'\frac{\partial C}{\partial y} \longleftarrow \frac{\partial C}{\partial y}$$

Qiyang Hu

Gradient vanishing/exploding in DL training

- Causes



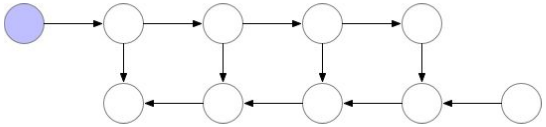
$$y = \sigma_L\Big(w_L \cdot \sigma_{L-1}\big(\cdots w_2 \cdot \sigma_1(w_1 \cdot x + b_1) + b_2\big) + b_L\Big)$$

  - Gradients in initial layers = Multiplication of Gradients at prior layers
  - Small variation around 1 results in vanishing/exploding

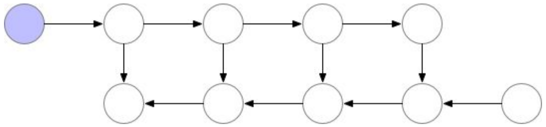
- Techniques to resolve:
  - General: adjusting learning rate, dropout, batch normalization, layer normalization
  - For gradient exploding: gradient clipping, weight regularization
  - For gradient vanishing: activation function, proper initialization parameters, LSTM, skip connections

Backprop beyond the traditional neural networks

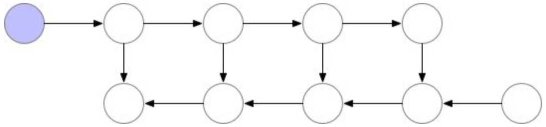
- Gradient checkpointing (source)



Vanilla Backprop

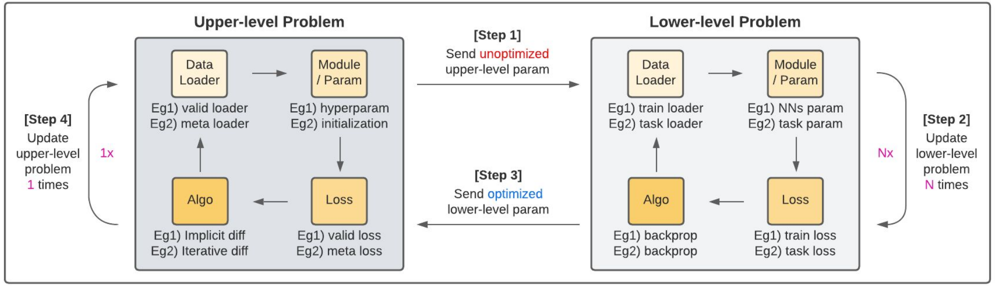


Memory-poor Backprop

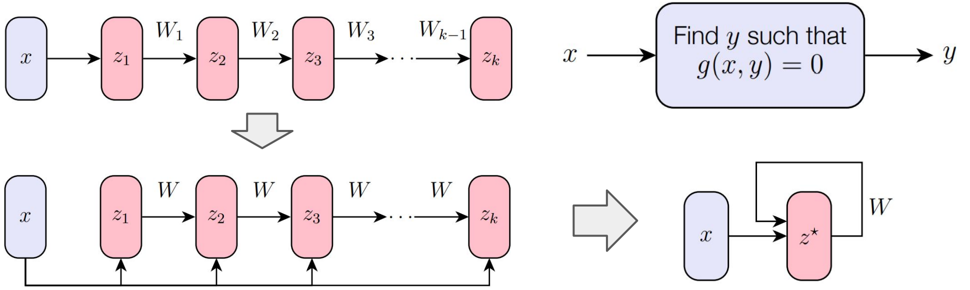


Checkpointed Backprop

- Multi-level optimization



- Deep implicit layers

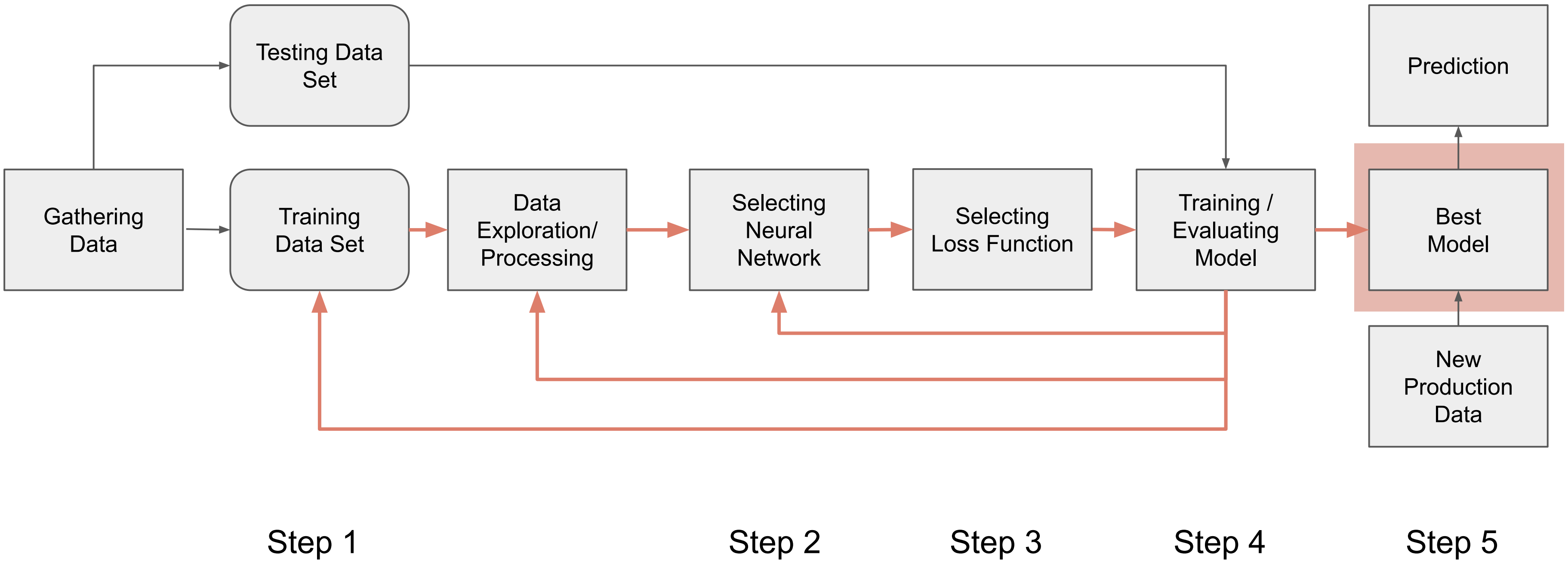


Qiyang Hu

Backprop beyond deep learning

- A different way to calculate the differentiation of iterative math expressions
  - Not approximate, unlike numerical differentiation
  - Exact, like manual or symbolic differentiation, but with constant overhead
- Automatic differentiation (algorithmic differentiation)
  - Problems constructed by differentiable directed graphs (e.g. NN)
  - General functional blocks (FF, conv, recurrent blocks, etc)
  - Modularized optimization: differentiable optimizations in layer levels
- Differentiable physics
  - Physics problems represented by a sequence of differentiable operators
  - Differentiable programming
    - Enables classical numerical algorithms
    - Beyond simple chained transformations to include more complex control structures

Workflow for a deep learning project



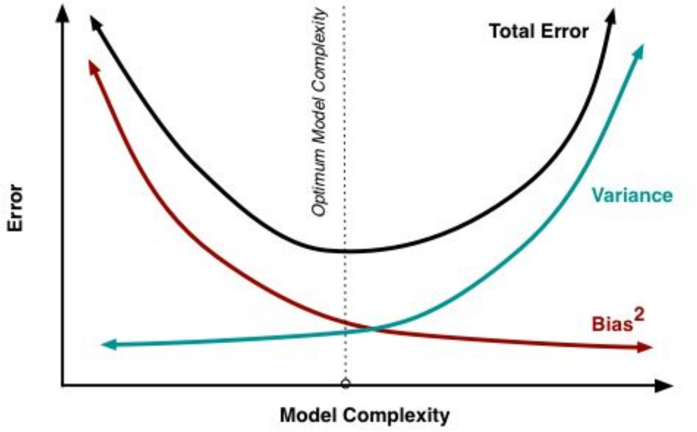
Qiyang Hu

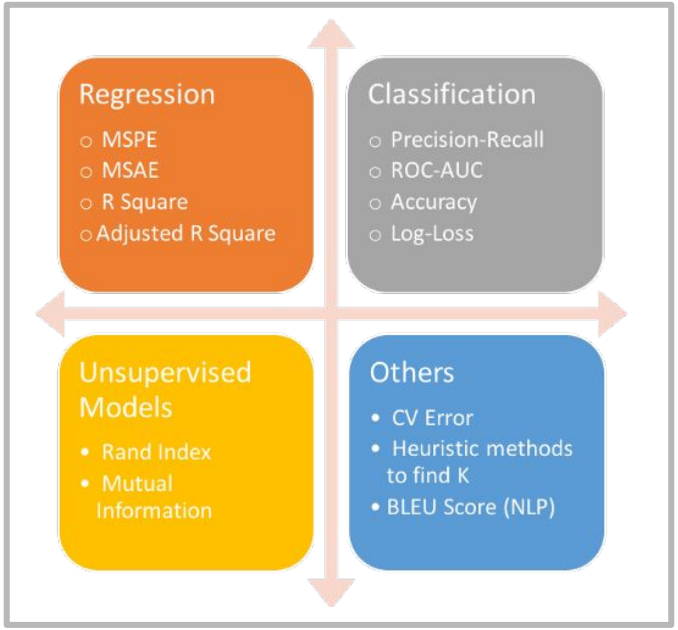
Neural Network Evaluation

- Errors in regression (e.g. MSE):
  - From Model: features, algorithm ⇒ Bias
  - From Data: insufficient observations ⇒ Variance
  - From Noise
- Bias-Variance Trade-off

$$\texttt{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

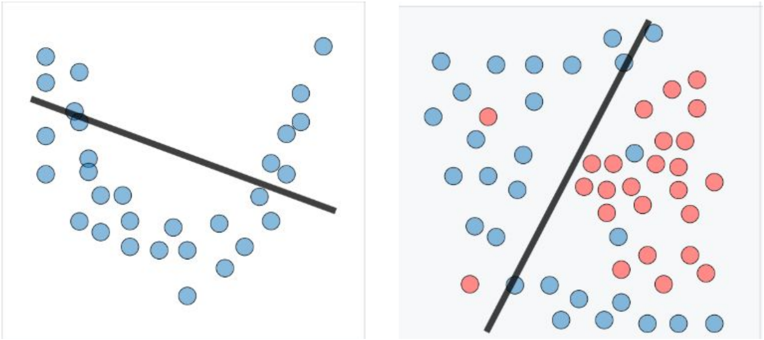
$$\mathbb{E}[(\hat{\mu} - \mu)^2] = (\mathbb{E}[\hat{\mu} - \mu])^2 + \mathrm{Var}[\hat{\mu} - \mu]$$



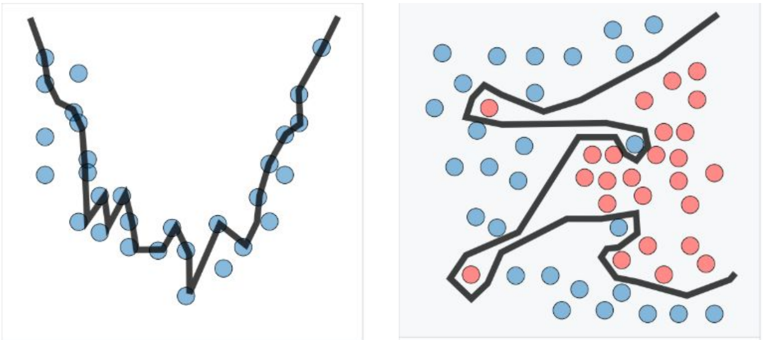


Underfitting and Overfitting

- Underfitting: model too simple:
  - Diagnose:
    - cannot even fit the training data
    - training error ~ testing error
  - Ignore the variance in training data
  - Higher prediction bias

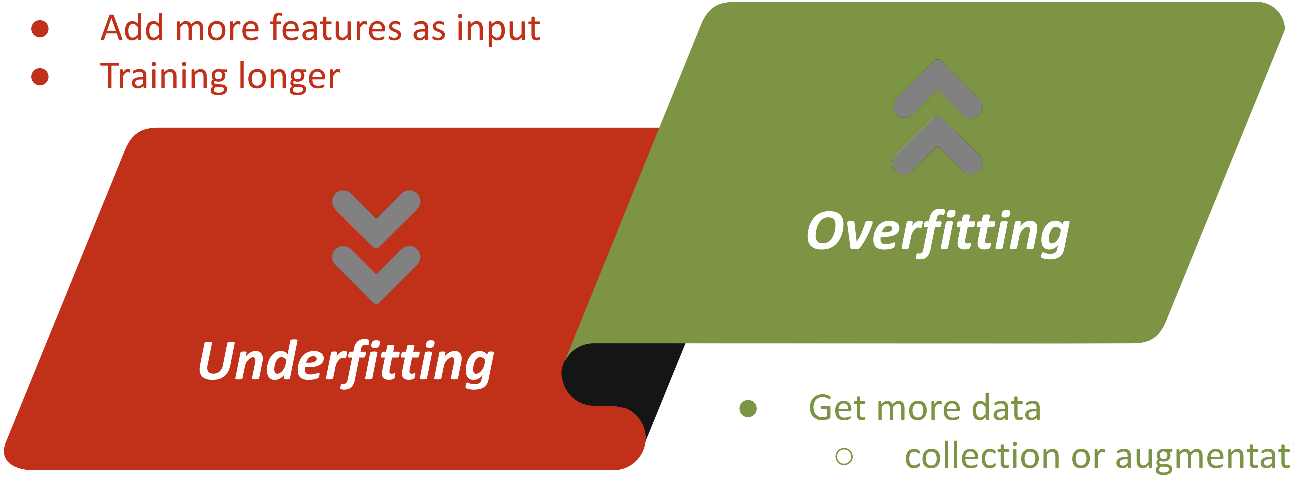


- Overfitting: model too complex
  - Diagnose:
    - well-fit for training data
    - large error for testing data
  - Over-interpret training data
  - More deviation from new data



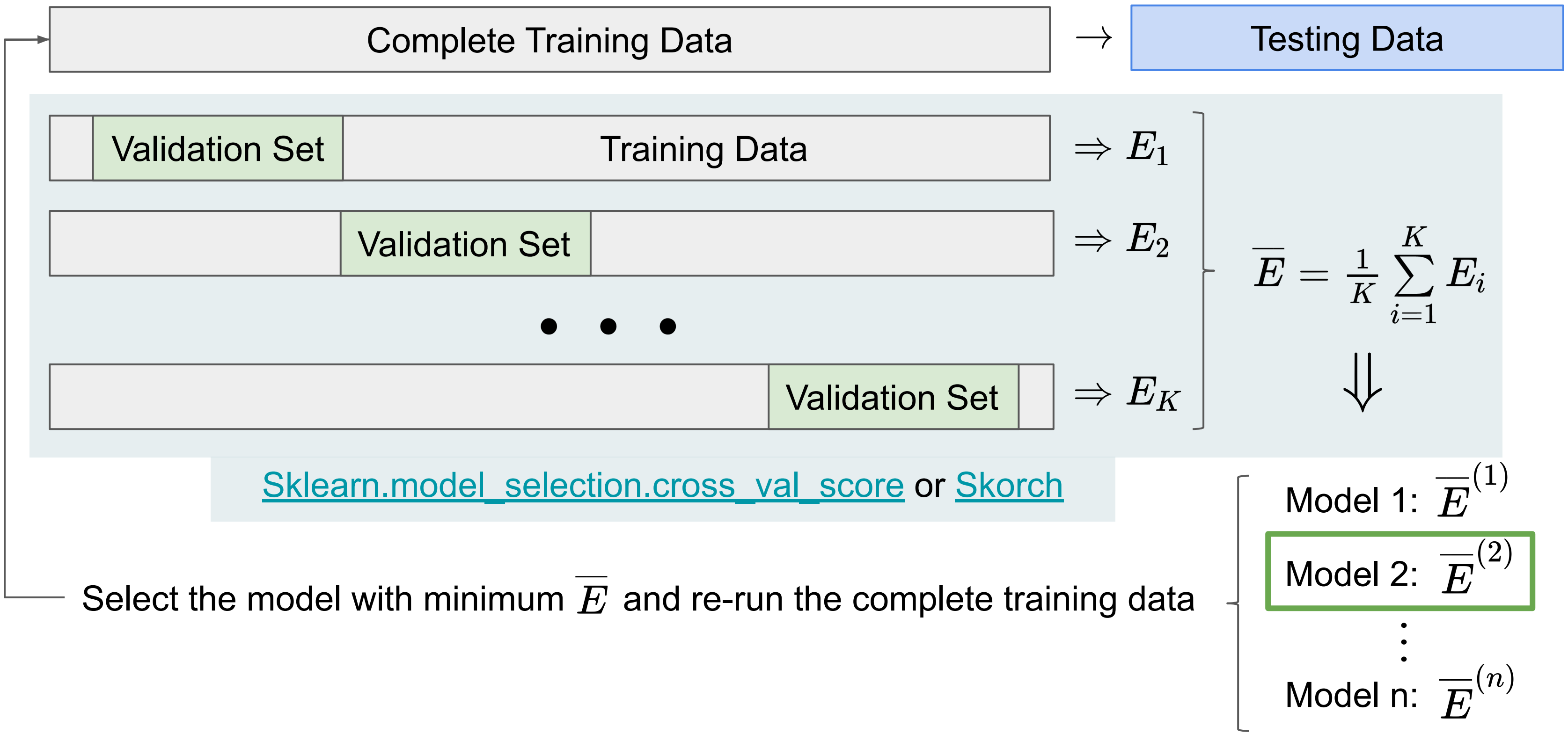
How to prevent

- Redesign the model
- Increase model’s complexity
- Add more features as input
- Training longer



- Get more data
  - collection or augmentation
- Reduce the model’s complexity
- Regularization
  - Weight Regularization
  - Early stopping

Model Selection: K-fold Cross Validation



Errors/scores in practice

Training Set	Validation Set		Public Testing Set	Private Testing Set

Error: $E^{val} < E^{Pub} < E^{Pri}$

Score: $S^{val} > S^{Pub} > S^{Pri}$

OARC Workshop Survey

http://bit.ly/3Dhp91H