Review references

- The Deep Learning book by Ian Goodfellow, et. al. (https://www.deeplearningbook.org/ [https://www.deeplearningbook.org/])
- Neural Networks and Learning Machine, Simon Haykin (a bit historical and solid intro)
- HEP ML living review (https://iml-wg.github.io/HEPML-LivingReview/ [https://iml-wg.github.io/HEPML-LivingReview/]), check various topic specific reviews
- The Principles of Deep Learning Theory (Robert, Yaida, Hanin): 2106.10165 [https://arxiv.org/abs/2106.10165] (https://deeplearningtheory.com/ [https://deeplearningtheory.com/])

Introduction Myself

Sung Hak Lim

- Postdoc at Rutgers University
- have been working on ML applications to collider physics / astrophysics

My ML experience

- Classification
  - Developing ML based jet tagger with physics inspirations
    - 1807.03312 [https://arxiv.org/abs/1807.03312], 1904.02092 [https://arxiv.org/abs/1904.02092], 2003.11787 [https://arxiv.org/abs/2003.11787], 2010.13469 [https://arxiv.org/abs/2010.13469]
- Density Estimation and Generative Models
  - measuring gravitational potential and dark matter in the Milky way and other galaxies without assumptions
    - 2205.01129 [https://arxiv.org/abs/2205.01129], 2305.13358 [https://arxiv.org/abs/2305.13358]
  - Upsampling N-body simualted galaxy
    - 2211.11765 [https://arxiv.org/abs/2211.11765]

Current Applications of ML in HEP

HEP is in a very ideal setting for applying ML for solving problems

- Data is easily available
  - High quality simulations is available (MadGraph5, Pythia, Herwig, Sherpa, Delphes, GEANT …)
  - Huge data is available from various experiments (LHC open datasets (complicated to use but you may try :))
- data is also clean (underlying principle is mostly understood) and less uncertain (systematic uncertainty and bias). medical, finance, economy datasets are often very noisy, lack of statistics…

Because of

- one very early example in 1990: Finding gluon jets with a neural trigger(https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.65.01321 [https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.65.01321])
- Use of boosted decision tree (BDT) is getting popular around Higgs discovery
- Use of neural networks are also getting popular after the success of image classfication (ex. jet image: 1511.05190 [https://arxiv.org/abs/1511.05190])
  - I jumped into this field around this time.

Currently ML is applied in various field in HEP!

- classification problems:
  - optimizing event selection criterion in LHC to maximize signal-to-background ratio
  - object tagging (jet tagging…)
- anomaly detection
  - can we find out anomalous signature (non-SM signals) quickly in the huge stockpile of data from the LHC?
  - Early examples based on autoencoders 1808.08979 [https://arxiv.org/abs/1808.08979] 1808.08992 [https://arxiv.org/abs/1808.08992]
- regression:
  - NNPDF: the parton distribution function modeled by neural networks (https://nnpdf.mi.infn.it/ [https://nnpdf.mi.infn.it/])
- sample generation:
  - fast detector simulation: GEANT full detector simulation is quite slow… can we speed up using ML? (https://calochallenge.github.io/homepage/ [https://calochallenge.github.io/homepage/])
- density estimation
  - dark matter detection (my recent works 2205.01129 [https://arxiv.org/abs/2205.01129] 2305.13358 [https://arxiv.org/abs/2305.13358] :)
  - parameter inferences (posterior estimation)

See living review of ML for particle physics in order to check vast amount of applications! (https://iml-wg.github.io/HEPML-LivingReview/ [https://iml-

wg.github.io/HEPML-LivingReview/])

Note: ML is not a tool only for experimentalists. Even theorists (including myself) are using ML for physics study! Don't be scared of applying ML to your problem :)

Basics ML

Statistical Learning Theory

A glorified *curve fitting* using *data*

- Prepare data in some space (observations?)
  - supervised: learning using reference labels (may be truth soln)
    - classifying objects: (data and truth labels)
      - binary classification: $(\vec{x}, y) \in R^d \times \{0, 1\}$
    - regression: (data and truth values)
      - 1D function regression: $(\vec{x}, y) \in R^d \times R$
  - unsupervised: learning without reference solution
    - $\vec{x} \in R^d$
    - anomaly detection
      - autoencoders: physics application 1808.08992 [https://arxiv.org/abs/1808.08992] 1808.08979 [https://arxiv.org/abs/1808.08979]
    - density estimations
- Design a function model doing your job (candidates explaining observations)
  - classifier score: if higher, sample is more likely to be a signal not background
  - anomaly score: if higher, the sample is more anomalous
  - density estimation: estimate probability of training dataset
  - generator: generate synthetic data using random numbers
  - and so on!
- Train your model
- make predictions using the trained model

ML is using statistics heavily, so learning the language of statistics is also important

See also: statistical learning theory [https://en.wikipedia.org/wiki/statistical learning theory]

Maximum likelihood estimation (MLE)

If you're using parametric function to do ML, maximum likelihood estimation is one standard technique for optimizing your parametric function.

* MLE: select model that makes your observed data is most probable.

A parameter estimation method by maximizing given likelihood function.

$$L(\text{param}|\text{data}) = p(\text{data}|\text{param})$$

we have multiple data, i.e., $\text{data} = (\text{data}_1, \text{data}_2, \cdots, \text{data}_N)$

$$L(\text{param}|\text{data}) = p(\text{data}_1, \text{data}_2, \cdots, \text{data}_N|\text{param})$$

If data is independently and identically distributed (IID)

$$L(\text{param}|\text{data}) = p(\text{data}_1|\text{param}) \times p(\text{data}_2|\text{param}) \times \cdots \times p(\text{data}_N|\text{param})$$

Conventionally we minimize sum of negative log likelihood (NLL). solution is the same.

$$-\log L(\text{param}|\text{data}) = -\sum_{i=1}^{N} \log p(\text{data}_i|\text{param})$$

If the parameter is continous, NLL is differentiable, we minimize NLL by analytically solving $\nabla_{\text{param}} NLL = 0$ or numerically solving it by using gradient descent methods.

Example: 1D Gaussian

$$-\log L(\mu, \sigma^2|x_1, \cdots, x_N) = \frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i - \mu)^2$$

The maximum likelihood estimator

$$-\frac{d}{d\mu}\log L(\mu,\sigma^2|x_1,\cdots,x_N)=0 \quad \rightarrow \quad \hat{\mu}=\frac{1}{N}\sum_{i=1}^{N}x_i$$

$$-\frac{d}{d\sigma^2}\log L(\mu,\sigma^2|x_1,\cdots,x_N)=0 \quad \rightarrow \quad \hat{\sigma}^2=\frac{1}{N}\sum_{i=1}^{N}(x_i-\hat{\mu})^2$$

Example: regression

$\mathrm{data}=((x_1,y_1),\cdots(x_1,y_N))$

$$\hat{y}=f(x;\vec{\theta})=\theta_1 x+\theta_2 : \text{if linear regression}$$

Statistical model: the deviation is Gaussian distributed with unknown mean $f(x;\theta)$ and variance σ^2

Likelihood model:

$$-\log L(\vec{\theta},\sigma^2|x_1,\cdots,x_N)=\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-f(x_i;\vec{\theta}))^2+\mathrm{const.}$$

minimizing this with respect to $\vec{\theta}$ is equivalent to the mean square error (MSE) minimization

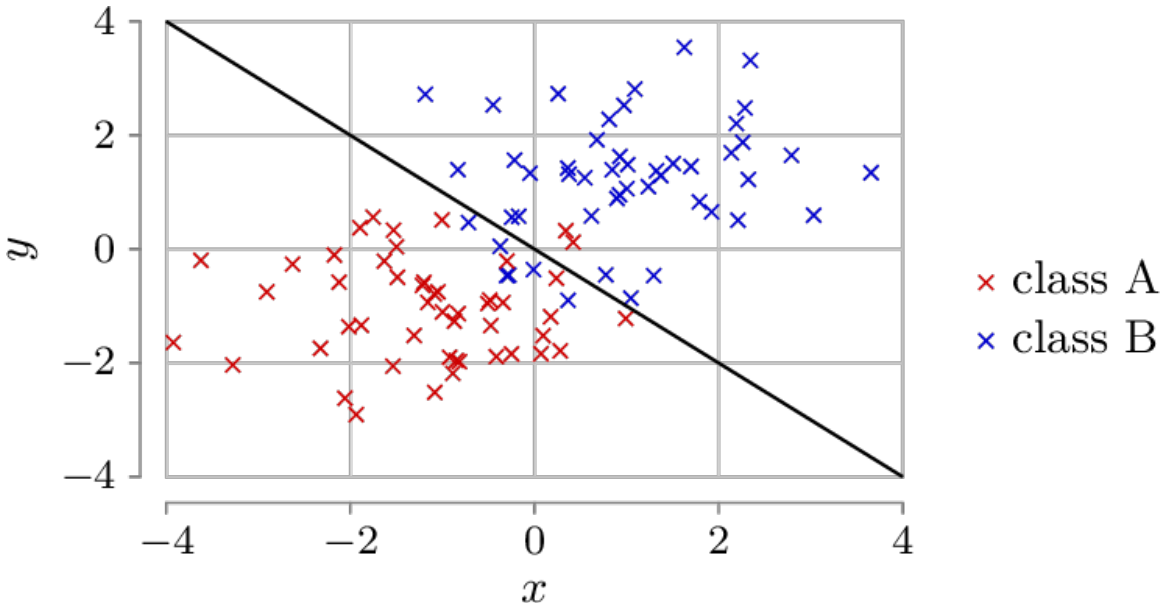
$$\mathrm{MSE}(\vec{\theta})=\frac{1}{N}\sum_{i=1}^{N}(x_i-f(x_i;\vec{\theta}))^2$$

These functions to be optimized for finding out best fit parameters are often called as loss functions.

summary of modern ML architecture

- data
  - supervised: if data contains true values to learn (regression, labeled classification)
  - unsupervised: if data does not contains true values
- function model
  - use a sufficiently expressive parametric function
  - linear model, neural networks!
- loss function
  - function describing your statistical goal.
  - This function is encoding almost all the things you're going to do with ML, so it is VERY IMPORTANT!
- optimizer
  - how to optimize your loss function?
  - typically use improved version of gradient descent (ADAM, 1412.6980 [https://arxiv.org/abs/1412.6980]), (other algorithms are also applicable)

Classifier



Early days: we design a feature and do event selection with simple optimized cut on a few number of features)

- example:) if we want to select Higgs jets: jet mass ~ 125 GeV, two-prong jet (2-subjettiness/1-subjettiness < threshold)
- pros: reproducible,
- cons: classification performance is not the best.

BSM searches in experiment is becoming more and more like finding needle in a garbage yard. Need to squeeze out the signal selection performance!

Binary Classification

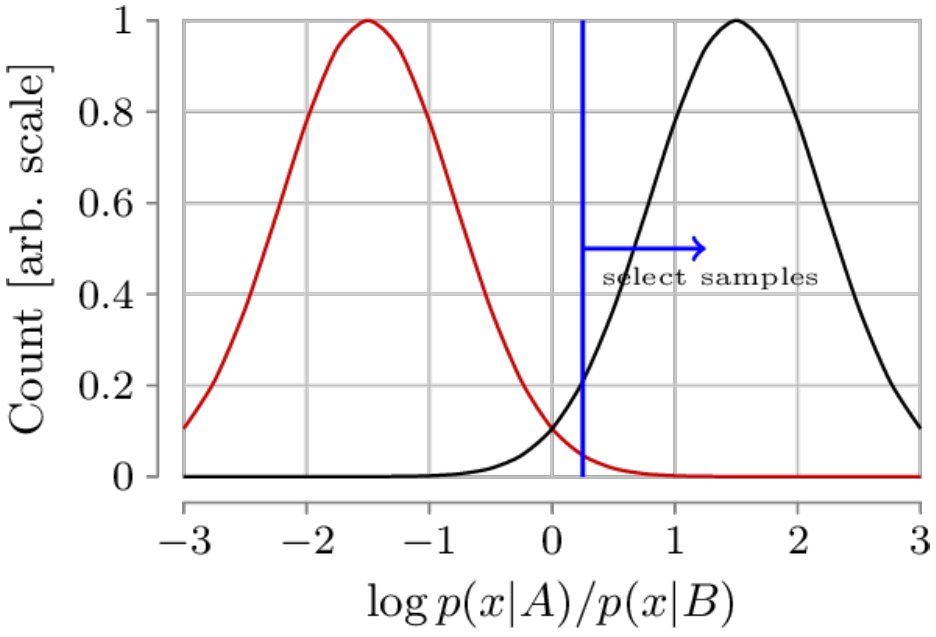
Q: For given two data distributions

- class A (label $y = 1$): $\{x_1, \cdots, x_{N_A}\} \sim p(x|A)$
- class B (label $y = 0$): $\{x_{N_A+1}, \cdots, x_{N_A+N_B}\} \sim p(x|B)$

What is the best selection boundary in order for classifying samples from A or B?

Answer: Neyman-Pearson lemma [https://en.wikipedia.org/wiki/Neyman-Pearson lemma]

- Likelihood ratio (LR) $\frac{p(x|A)}{p(x|B)}$ is the most powerful test statistics



But we only have data, so we have to estimate either

- LR directly: $\frac{p(x|A)}{p(x|B)}$
- some function monotonically related to LR
  - posterior: $p(A|x) = \frac{p(x|A)p(A)}{p(x|A)p(A)+p(x|B)p(B)}$:

See also Binary classification [https://en.wikipedia.org/wiki/Binary classification]

Cross Entropy Loss Function

Derivation from Bernoulli Trial

For given samples in y=1, the probability $p(1|x), p(0|x) = 1 - p(1|x)$

The likelihood of class 1 is then

$$L(y=1) = p(1|x)$$

The likelihood of class 0 is then

$$L(y=0) = p(0|x) = 1 - p(1|x)$$

These two expression can be combined as follows

$$L(y) = p(1|x)^y (1 - p(1|x))^{1-y}$$

this is simple Bernoulli distribtuion.

The maximum likelihood estimation on $p(1|x)$

$$L(y) = \prod_{i=1}^{N} p(1|x^{(i)})^{y^{(i)}} (1 - p(1|x^{(i)}))^{1-y^{(i)}}$$

So if your dataset x, y is drawn from the joint distribution $p(x, y)$, we can estimate the probability $p(y|x)$ by minimizing the following cross entropy loss

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \left[y^{(i)} \log p(1|x^{(i)}) + (1 - y^{(i)}) \log (1 - p(1|x^{(i)})) \right]$$

This loss is also sometimes called logistic loss (see also logistic regression)

derivation from joint distribution

Your data $(\vec{x}, y)$ is sampled from $p(\vec{x}, y)$

$p(y)$: percentage of samples in class y of your training dataset.

We want to estimate $p(y|\vec{x})$

$$\hat{p}(\vec{x}, y; \theta) = \hat{p}(y|\vec{x}; \theta) p(\vec{x})$$

KL divergence: zero if two distributions are the same.

$$KL(P||Q) = \int dX p(X) \log \frac{p(X)}{q(X)}$$

$$KL(P||Q) = \int dx dy p(\vec{x}, y) \log \frac{p(\vec{x}, y)}{\hat{p}(y|\vec{x}; \theta) p(\vec{x})}$$

$$KL(P||Q) = -\sum_y p(y) \int dx p(\vec{x}|y) \log \hat{p}(y|\vec{x}; \theta) + \text{const.}$$

Equivalent to cross entropy $H(P, Q)$ between $P \sim p(x, y)$ and $Q \sim \hat{p}(y|\vec{x}; \theta) p(\vec{x})$.

$$KL(P||Q) = -\sum_y \frac{p(y)}{N_y} \sum_{\vec{x}^{(i)} \in \text{samples in class } y} \log \hat{p}(y|\vec{x}^{(i)}; \theta) + \text{const.}$$

$$CE = -\sum_y \frac{p(y)}{N_y} \sum_{\vec{x}^{(i)} \in \text{samples in class } y} \log \hat{p}(y|\vec{x}^{(i)};\theta)$$

If $y = 0$ or 1 This KL divergence reproduce the binary cross-entropy loss function.

Overfitting

Neural networks are sufficiently expressive, so that it could memorize the data, especially when the network is over-parametrized (number of parameters > number of training sample)

Empirical distribution:

$$p(x,y) = \frac{1}{N}\sum_{i=1}^{N} \delta_{yy^{(i)}}\delta(x - x^{(i)})$$

If we use this as Q,

$$\hat{p}(y|\vec{x}) = \begin{cases} 1 & \vec{x} \in \text{class y dataset} \\ 0 & \vec{x} \notin \text{class y dataset} \end{cases}$$

This is global minimum since KL divergence will be exactly zero.

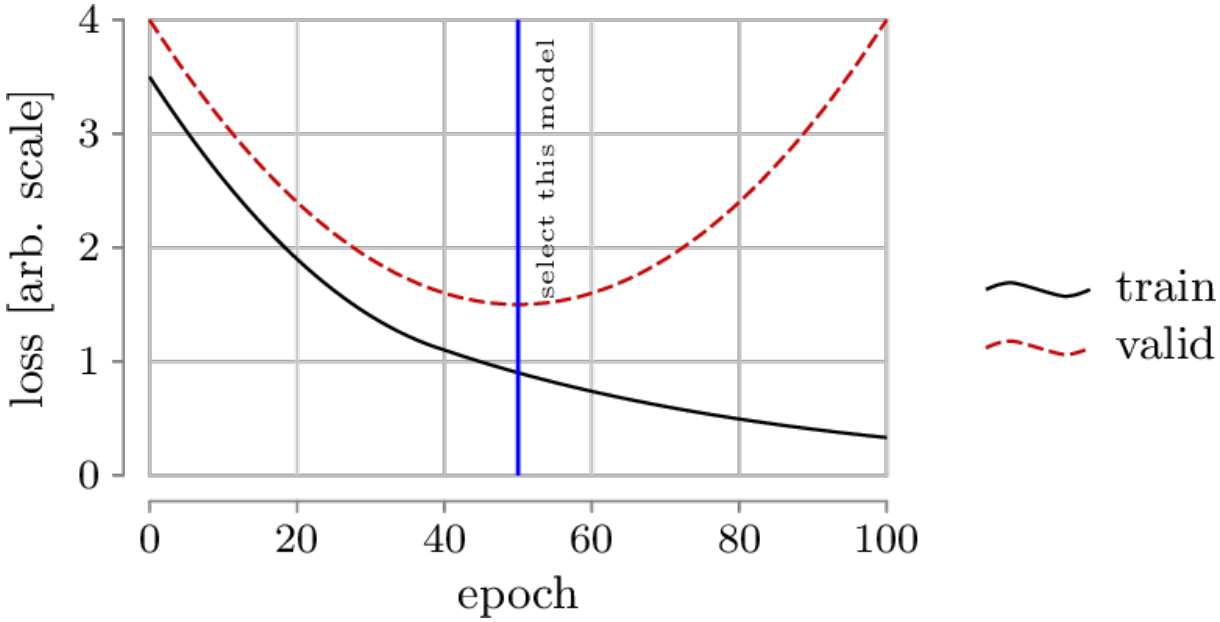
Solution?

Solution: Train / Validation split

Instead of using full dataset for the training we divide

- training dataset: samples used for training.
- validation dataset: samples independently check the loss value

If your training is going toward overfitted solution, training loss will decrease, but validation loss is not. P: validation , Q: training, since $P \neq Q$, KL divergence will be finite



Pick up the parameters with minimum validation loss.

Neural Network

Artificial Neural Network

- highly flexible parametric function

$$y^j = W^{ij}x^i + B^j$$

Activation Function

Sigmoids

- Unit step

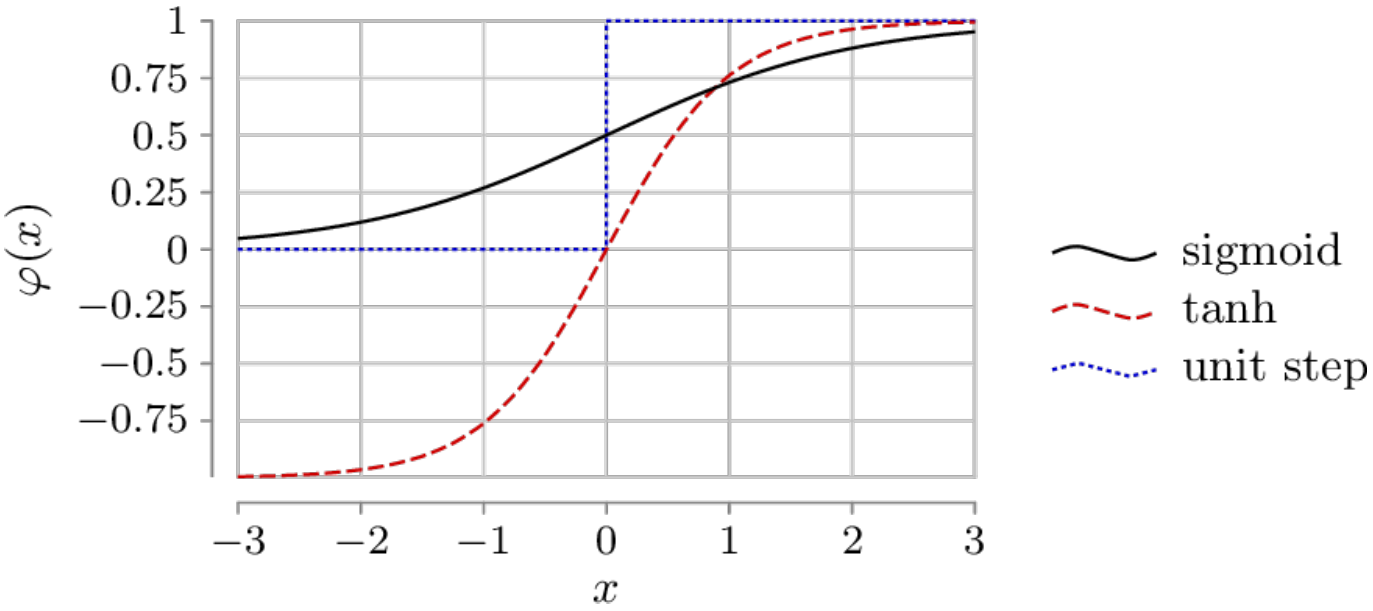
$$\varphi(x) = \begin{cases} 1 & x >= 0 \\ 0 & x < 0 \end{cases}$$

- sigmoid

$$\varphi(x) = \frac{1}{1+e^{-x}}$$

- tanh (mathematically equivalent to sigmoid)

$$\varphi(x) = \tanh(x)$$



Universal approximation theorem: an NN with single hidden layer (with sufficiently large width) can approximate arbitrary functions

$$F(x) = U^i \varphi(W^i x + B^i)$$

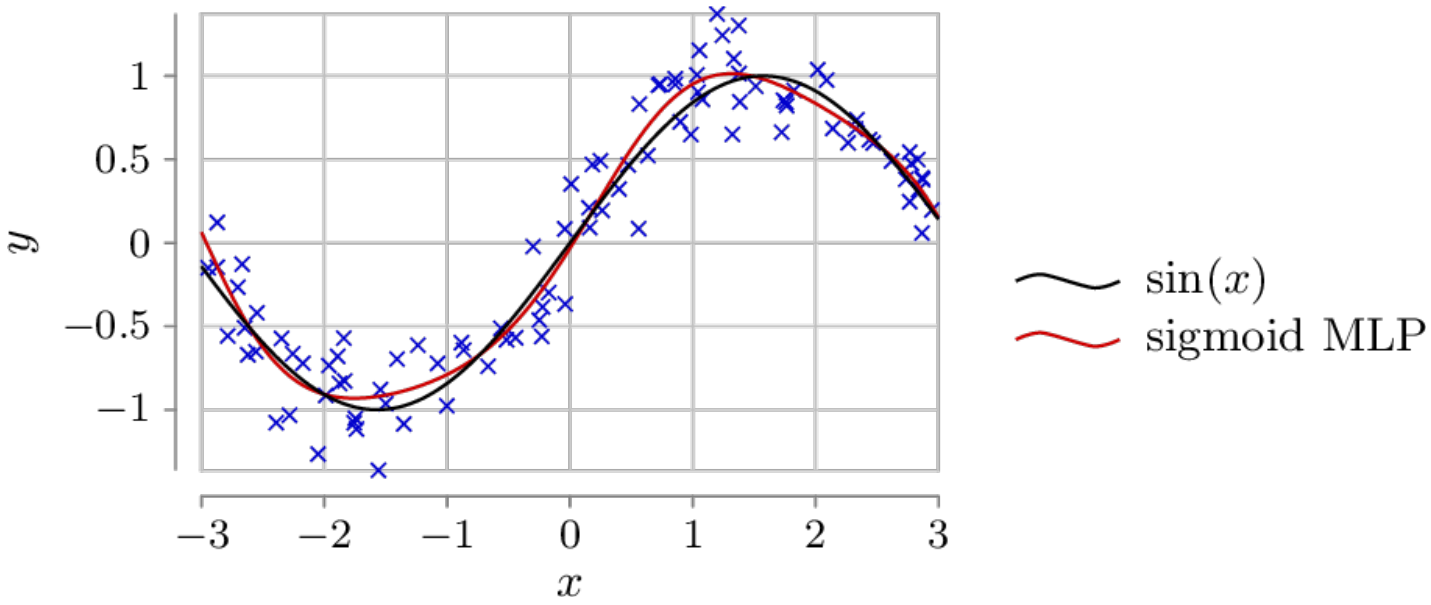
see a visual demonstration: http://neuralnetworksanddeeplearning.com/chap4.html [http://neuralnetworksanddeeplearning.com/chap4.html]

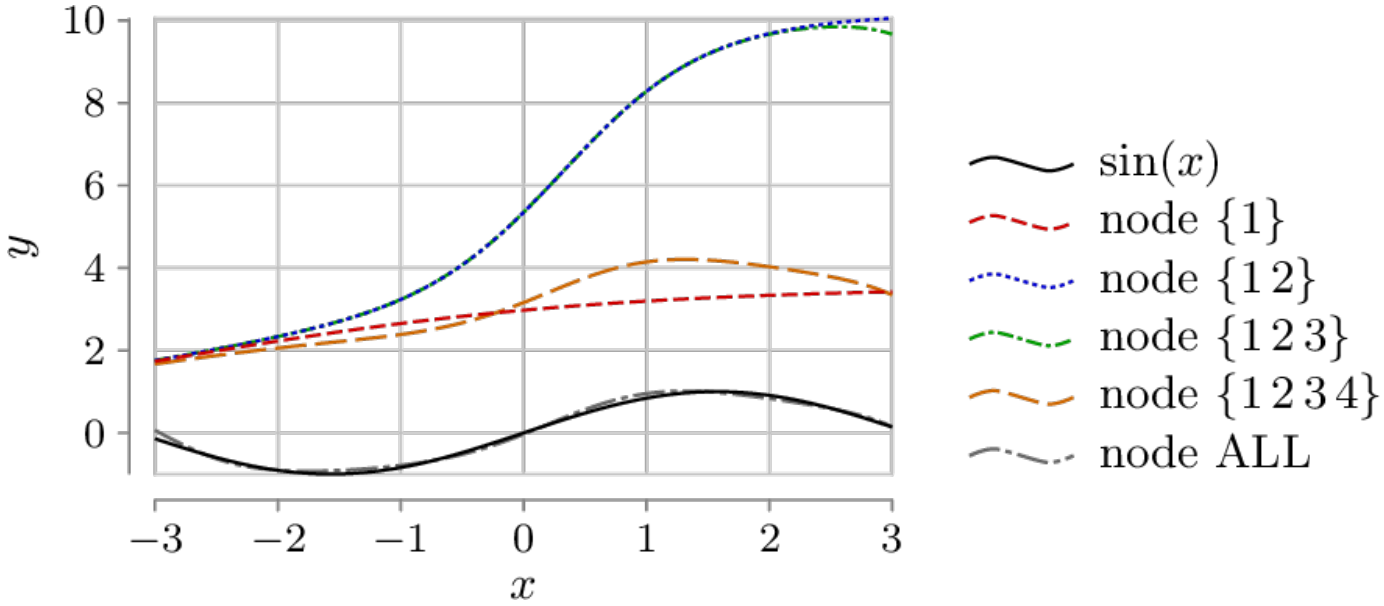
sigmoid MLP: data fitting with binning?

Example: fitting sin x using sigmoids

Example: fitting $\sin(x)$ with noise

network: 1 hidden layer with 5 nodes. loss: MSE loss





Rectifiers

- ReLU

$$\mathrm{ReLU}(x) = \max(0, x)$$

- LeakyReLU

$$\varphi(x) = \max(0.1x, x)$$

- ELU 1511.07289 [https://arxiv.org/abs/1511.07289]

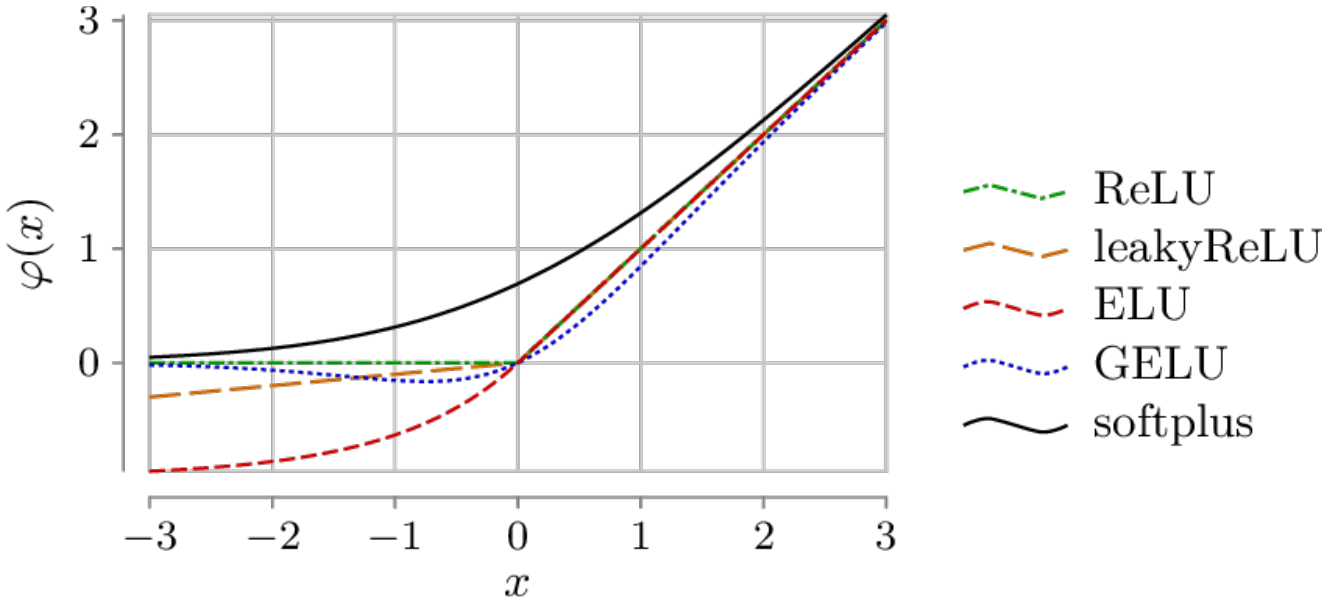
$$\varphi(x) = \begin{cases} x & x > 0 \\ \exp(x) - 1 & x \leq 0 \end{cases}$$

- GELU 1606.08415 [https://arxiv.org/abs/1606.08415] (or other smoothed rectifiers)

$$\mathrm{GELU}(x) = x\Phi(x), \quad \Phi(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

- softplus (have some issues, see chapter 5.3.2 of 2106.10165 [https://arxiv.org/abs/2106.10165])

$$\varphi(x) = \log(1 + e^x)$$

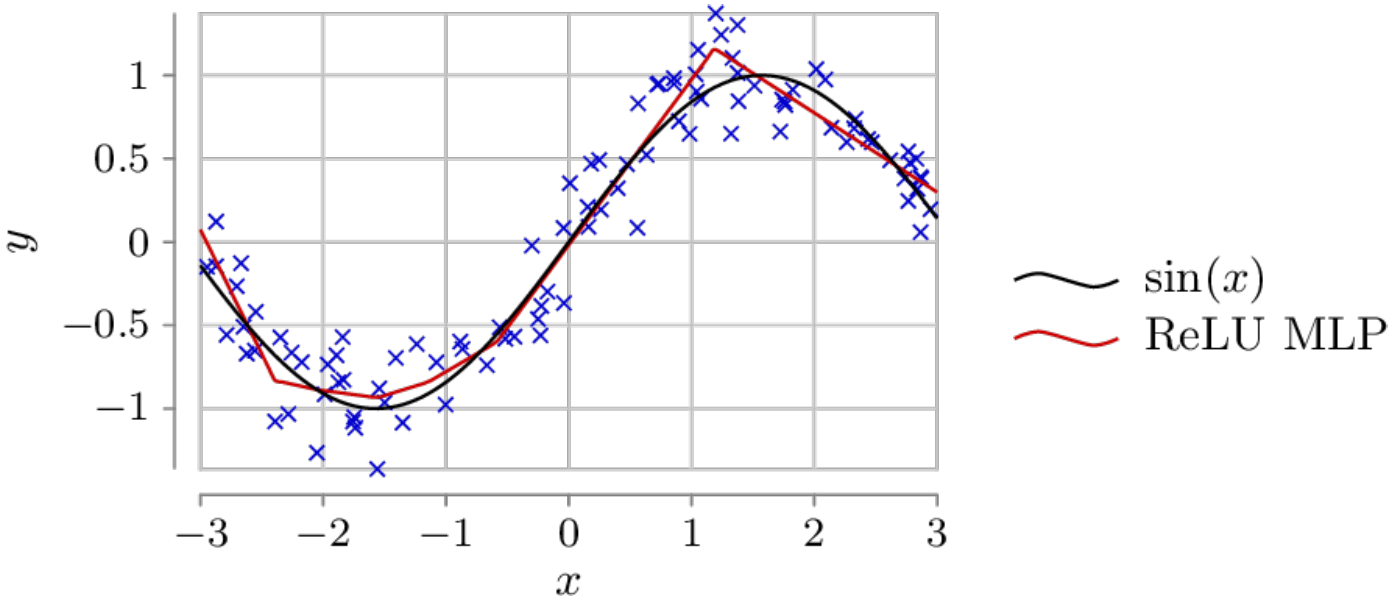


Example: fitting sin x using relu

Example: fitting $\sin(x)$ with noise

network: 1 hidden layer with 10 nodes. loss: MSE loss

ReLU MLP: piecewise linear function



specific NN architectures

Depending on data, you may use more complicated structured neural networks for modeling a function

- pixellated data (image): convolutional neural network (many references are available in internet) (see also, resnet: 1512.03385 [https://arxiv.org/abs/1512.03385],1×1 convolution: relation between CNN and MLP 1312.4400 [https://arxiv.org/abs/1312.4400])
  - early physics applications: jet image 1511.05190 [https://arxiv.org/abs/1511.05190] 1701.08784 [https://arxiv.org/abs/1701.08784] and so on…
- point distributions (point clouds): graph neural networks 1806.01261 [https://arxiv.org/abs/1806.01261]
  - early physics applications: ParticleNet 1902.08570 [https://arxiv.org/abs/1902.08570]
- sequential data (time sequence, language models?):
  - recurrent neural network and recursive neural networks
    - early physics applications: 1702.00748 [https://arxiv.org/abs/1702.00748]
  - attention layers and transformers.. (title: Attention is All You Need) 1706.03762 [https://arxiv.org/abs/1706.03762]

Please google them if you want to study about those specific type of networks. There are so many examples and tutorials :)

Comparinng Classification Performances

If time permits

- Confusion matrix
    - Tagging efficiency and mistag rate
- ROC curve [[wp>Receiver operating characteristic]

Cross Validations

Training/validations split: validation dataset is not used in gradient evaluation, can we overcome this somehow?

- K-fold Cross validations
- Monte Carlo Cross validations