# Basics of Machine Learning and its Application in Neutrino Physics

## Lecture 1: Basics of ML

Igor Ostrovskiy, IHEP

2nd JUNO Neutrino Summer School
Hangzhou, August 2025

# Introduction

- Two 1.5 hr lectures covering basic concepts in ML (specifically, neural networks, or NNs) and how they are / can be applied in neutrino physics
- Expected audience level: familiarity with (neutrino) physics and its main experiments; no specific experience with ML
- 1.5 hr is a lot! Split time between lecture slides and hands-on activities:
  - Occasional quiz questions/polls during the lecture
  - Guided tutorials at the end of today and tomorrow
- 1.5 hr is not a whole lot!
  - Only select topics covered and at an intro level. The goal is to introduce main concepts and provide simple hands-on exercise thus lowering the threshold of entry into the field. Highlight two algorithms that became (and will remain) workhorses of physics analysis

# Introduction

- Topics covered in lectures:
  - Lecture 1. Basics of ML/NNs
    - What is ML? Difference from usual data “fitting”
      - What is a neural network, specifically?
    - Supervised vs. Unsupervised, Classification vs. Regression
    - Supervised algorithms: Basic concepts
    - Supervised algorithms: Workhorses you will use (CNN, MLP)
  - Lecture 2. Applications in neutrino physics
    - Case of EXO-200: Successful examples of MLP, CNN, GAN, and data-based training
    - Examples of much larger detectors: NOvA, LAr TPC

# Introduction

- To answer the occasional quiz questions, will try to use the WeChat's poll app

  First question:

  1) Does this quiz setup work?
     - Yes
     - No

ML lecture quiz Q1

诚邀您填写本问卷，扫码即可！

# Introduction

- For the tutorials/activity, I will use Jupyter notebooks and realistic data from a 0ν and another experiment
  - Password: cUaw (please don’t share further)
- Confirmed to work with:
  - python 3.7.1, torch 1.13.1, cpu
  - python 3.11.9, torch 2.5.1, gpu (cuda 12.5, Tesla T4)
  - python 3.12.11, torch 2.7.1, gpu (cuda 12.5, A100)
- You don’t have to run the scripts, can just follow the tutorial on the screen
  - Or feel free to join forces with others to be able to play with the scripts yourselves

# What is ML?

- Fundamentally, ML is
  - A type of algorithm that enables solving data analysis problems without explicit programming
  - That is, an algorithm to ***learn*** how to solve, not an algorithm to solve

credit: XKCD

# What is ML?

- Technically*, ML is:
  - A software algorithm that
    - Takes a set of inputs (***data***)
    - Applies a parameter-dependent transformation on the inputs to compute an output (***forward propagation***)
    - Compares the output to a given expectation (***loss***)
    - Changes the parameters to minimize the difference between output and expectation (***backward propagation***)

*at least the NNs that we will be mostly discussing today

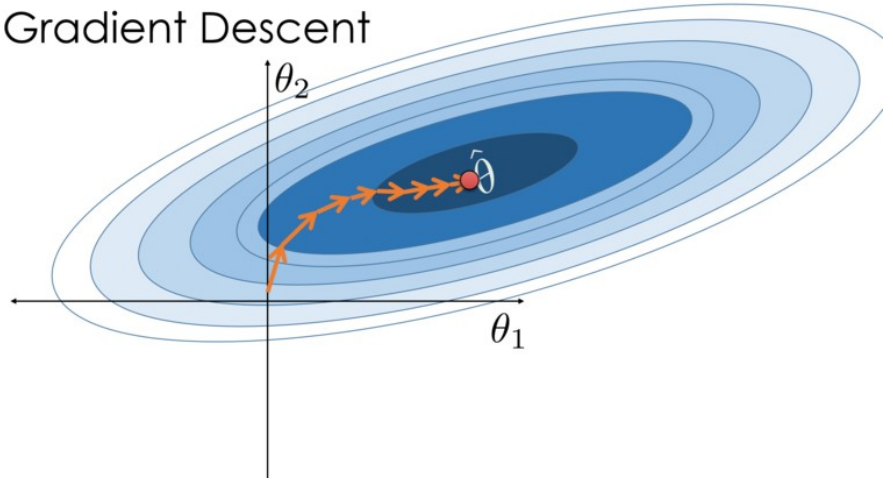
# What is ML?

- Does not this sound just like the usual ***fitting***?!
  - It does, but the number of the inputs and parameters is so vast (often >$10^6$), that special methods and considerations need to be applied, making ML a separate subject in software/data analysis


credit: https://statustown.com

# fitting vs ML

- If you do a simple fit of a histogram with a function in ROOT, it invokes the MINUIT's MIGRAD algorithm
- What it does behind the curtain is finds a minimum of some function ($\chi^2$ or *-logL*) by following the path of steepest descent. Mathematically, this is done by computing the Gradient (~derivatives of the function wrt parameters) using all data points



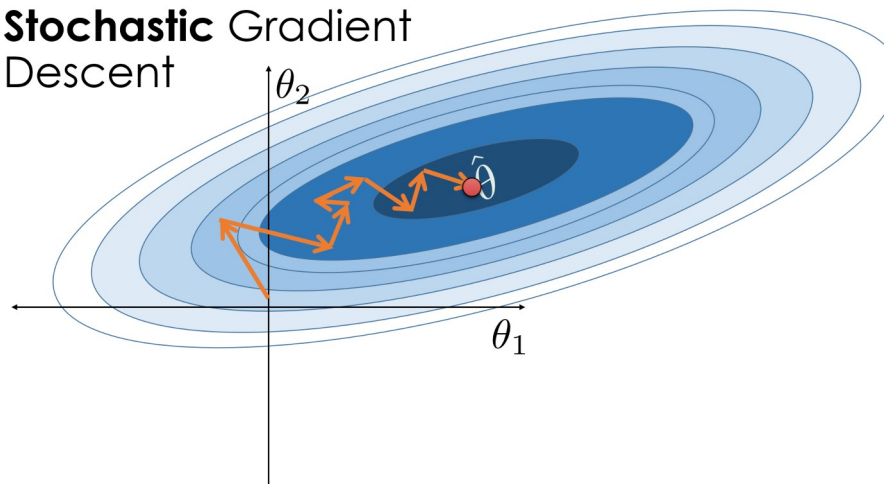

credit: https://optimization.cbe.cornell.edu

# fitting vs ML

- However, if the dataset and number of parameters are very large, this becomes computationally prohibitive
- ML's special method: Stochastic Gradient Descent (SGD)
- SGD computes gradient only for a "batch", a subset of data
  - "Noisy" but tractable
  - Can "jump" out of shallow local minima by luck

$$\theta_{n+1} = \theta_n - \epsilon \nabla_\theta L(\theta; X)$$

*L* - loss function, ***ϵ* - "learning rate"**, *θ* - parameter set, *X* - subset of data ("batch")



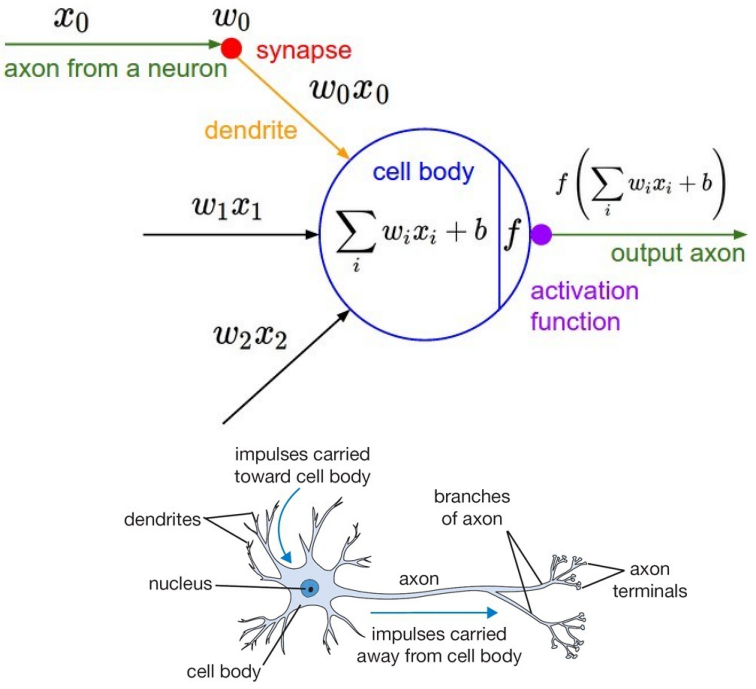

credit: https://optimization.cbe.cornell.edu

# fitting vs ML

- When fitting, models are often analytical, providing a clear physical meaning of the parameters and a clear explanation of how changing inputs would affect outputs
- With ML, the model is often a “black box”

# What is a neural net, specifically?

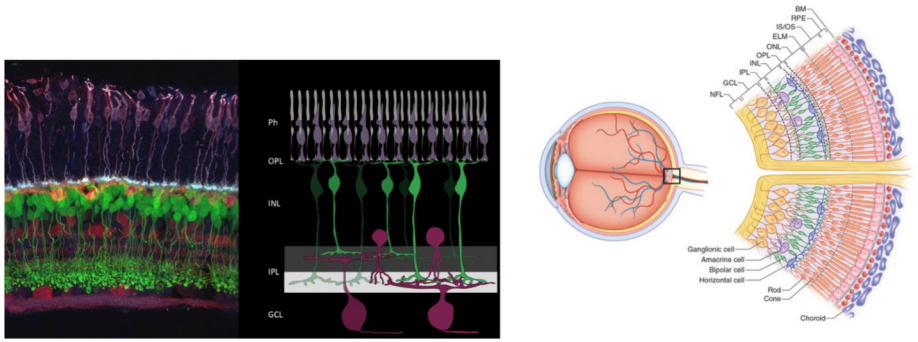
- A function made up of several identical computational elements (***neuron***s)
- Each neuron:
  1) Takes several input values and forms their ***weight***ed sum, with an overall offset (called ***bias***) added
  2) Computes the output by applying the ***activation function***



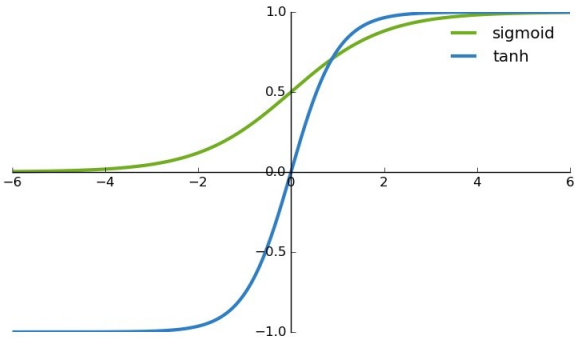

credit: Stanford CS231n (cs231n.github.io)

# What is a neural net, specifically?

- Activation function originally inspired by biology
- A simple decision-making computation. The simplest - if input is above a threshold, neuron fires
  - Mathematically, a common way to model realistic threshold behavior is with ***TanH*** or ***Sigmoid***

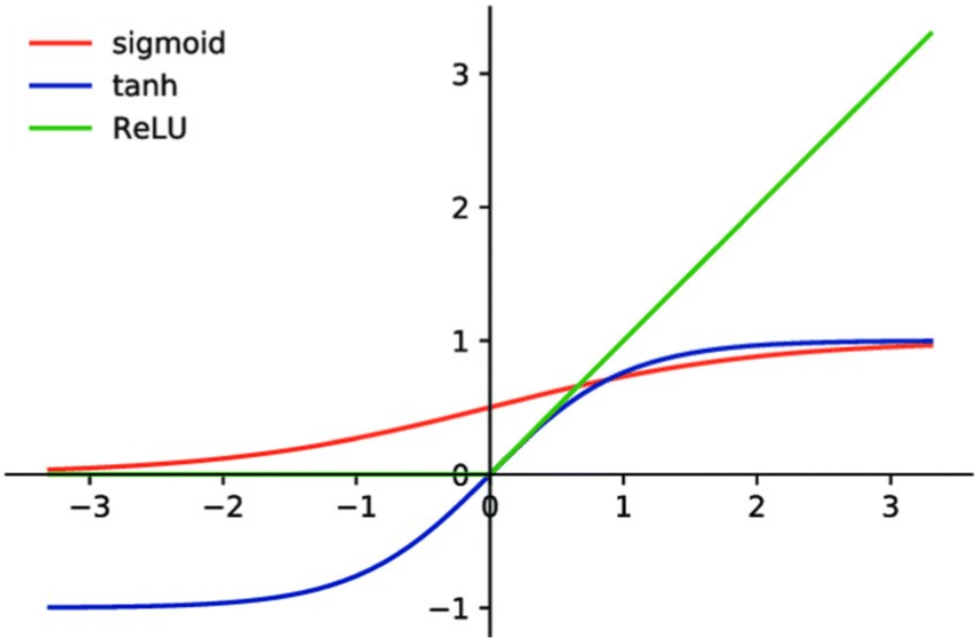



Bleckert A, Schwartz GW, Turner MH, Rieke F, Wong RO. Visual space is represented by nonmatching topographies of distinct mouse retinal ganglion cell types. Curr Biol. 2014 Feb 3;24(3):310-5.



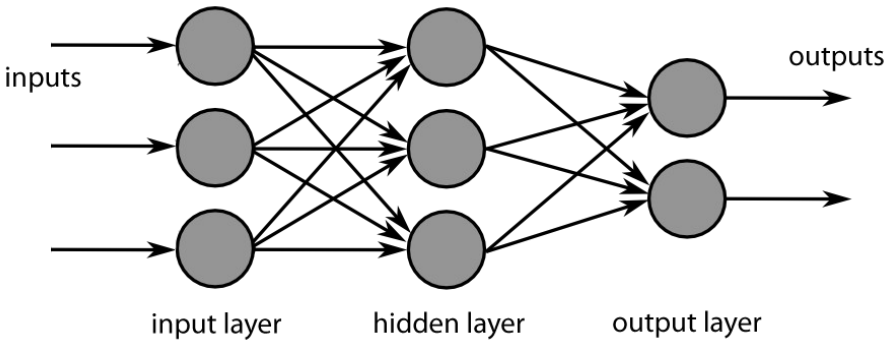
# What is a neural net, specifically?

- By now, TanH and Sigmoid are less common than another, also simple, activation function: ***Rectified Linear Unit*** (***ReLU***)
- It offers fewer vanishing gradient problems and is computationally efficient



# What is a neural net, specifically?

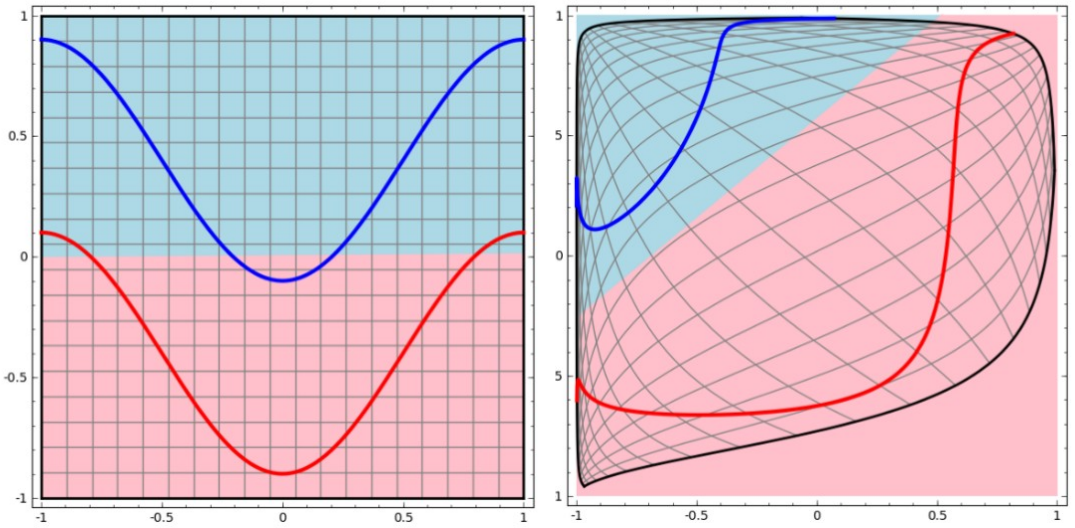
- By itself, a neuron does not do much, but the magic happens when one combines (many) neurons into a ***network***
- ***The Universal Approximation Theorem*** ~ a neural network with a single hidden layer can approximate ***any*** continuous function
  - as with all mathematics, need to mind the fine-print if want a rigorous statement




credit: Wikipedia

# What is a neural net, specifically?

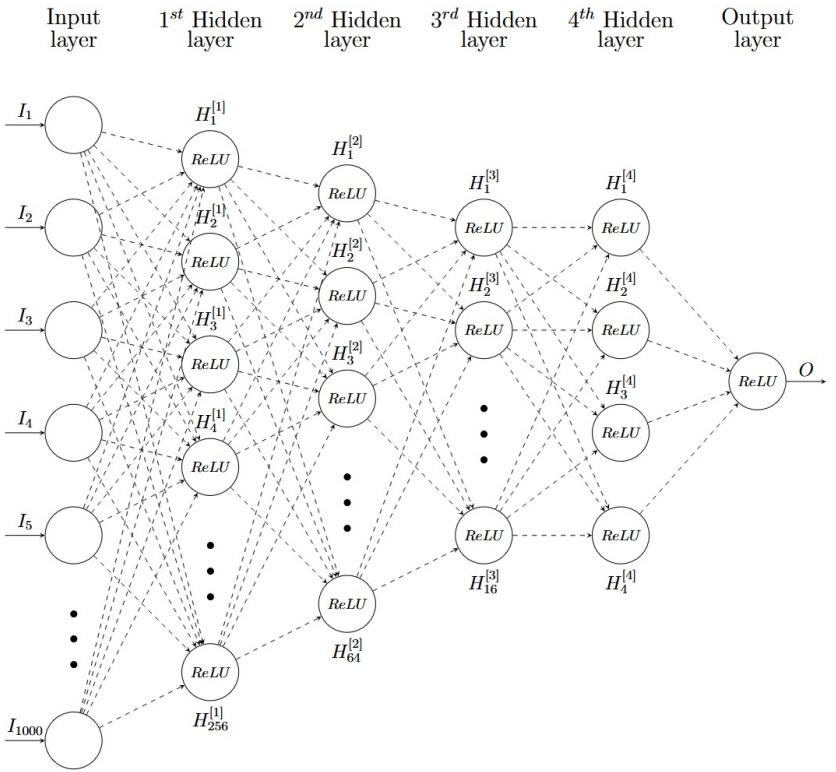
- Note also how this connection between the input layer (vector) to the hidden one is similar to a coordinate transformation, i.e., going from one set of basis vectors to another
- Where can this come in handy?
- Perhaps, this could be used to find a basis in which the originally complex and nonlinear separation boundary between signal and background events becomes trivial?




credit: Javier Duarte

# What is a neural net, specifically?

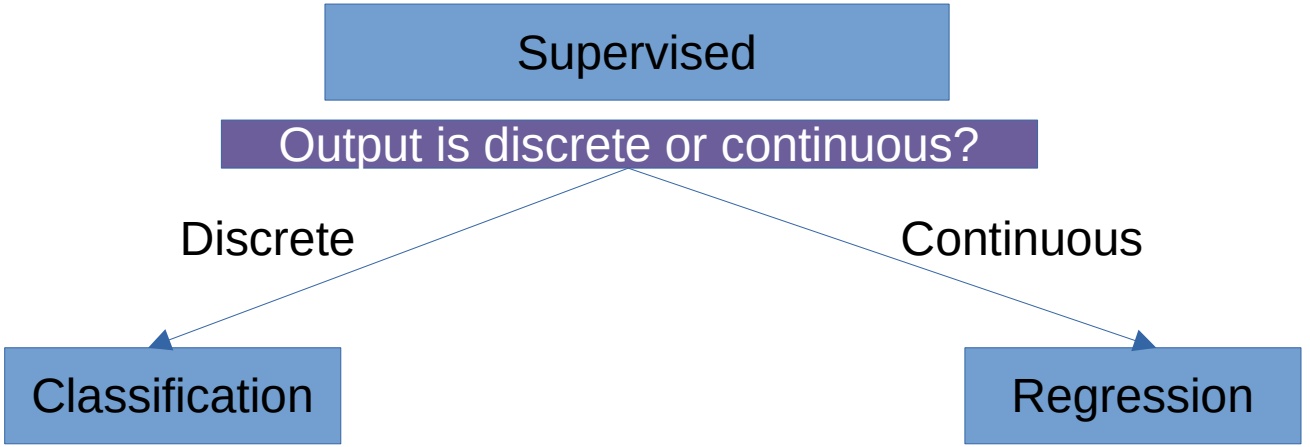
- Why ***Deep Learning***?
- The Universal Approximation Theorem does not say how many neurons this single neural layer must have to approximate a target function
- In practice, going “deep”, i.e., adding several layers with a reasonable number of neurons is computationally better than having one very wide layer
- A simple example of a deep network is a ***Multi Layer Perceptron (MLP)***. An architecture where neurons are ***fully connected (FC)***, i.e., each neuron in a layer “sees” all neurons in preceding layer



# Unsupervised vs. Supervised

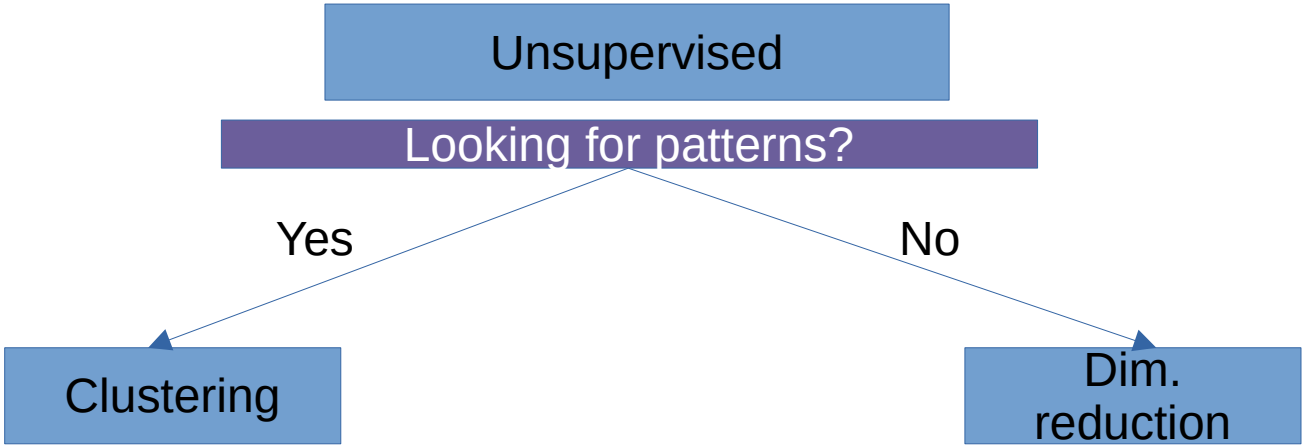

- ***Labeled*** data - desired answer is known for each event
  - Calibration datasets; Monte Carlo datasets with “truth” information

# Classification vs. Regression




- ***Discrete*** - event classes (signal, background)
- ***Continuous*** - event energy, position vector

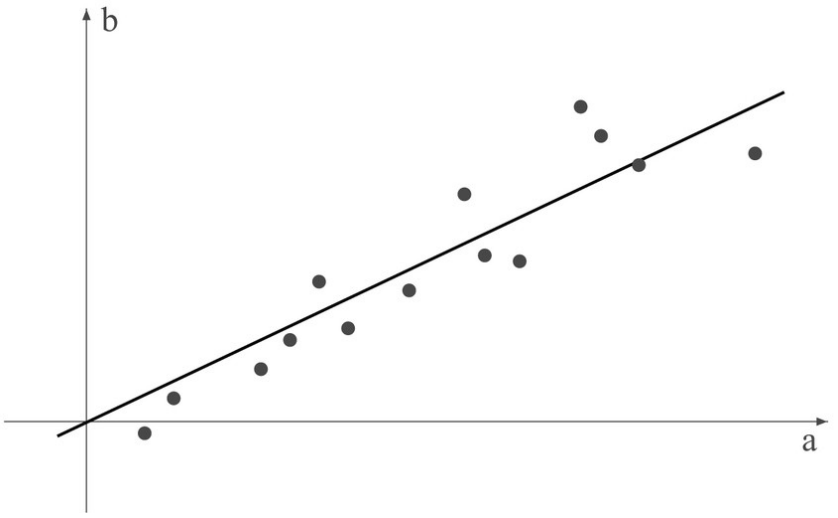
# Clustering vs. Reduction




- ***Clustering*** - find good “distance” metric and group events based on “closeness”
- ***Dimensionality reduction*** - summarize data using small number salient features

# Quiz

2) Fitting a straight line to a set of points is an example of which type of algorithm:
    1) Supervised, regression
    2) Supervised, classification
    3) Unsupervised, clustering
    4) Unsupervised, dim. reduction



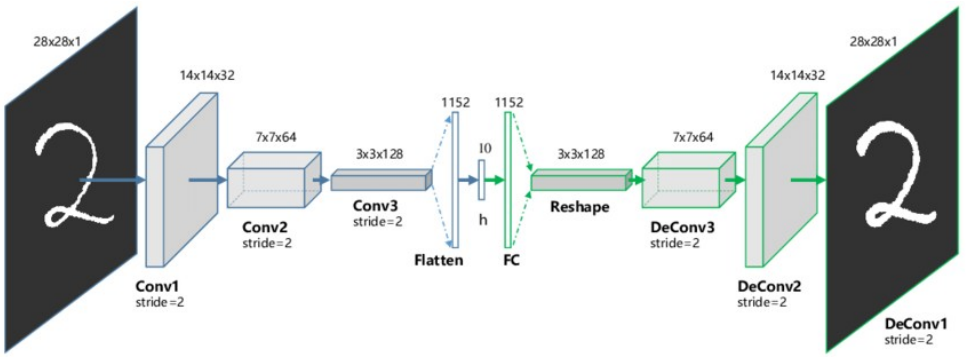

ML lecture quiz Q2

诚邀您填写本问卷，扫码即可!

# Quiz

***3)*** ***Autoencoder*** is a neural network with the same input and output shapes. It tries to reproduce input data. What kind of algorithm is this?

1) Supervised, regression
2) Supervised, classification
3) Unsupervised, clustering
4) Unsupervised, dim. reduction



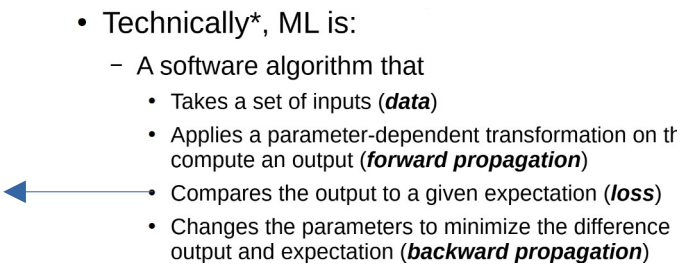

credit: https://cs.toronto.edu

ML lecture quiz Q3

诚邀您填写本问卷，扫码即可！

# Supervised algos: Loss

- When the true answer is known for each event, constructing regression loss function is easy
- For regression, most common is to use ***Mean Squared Error (MSE)***
  - Penalizes large errors more than small errors (makes it sensitive to outliers)
  - Related to ***L2 loss*** (same, just not averaged over samples)
- Less common is ***Mean Absolute Error (MAE)***, related to ***L1 loss***
  - Less sensitive to outliers
  - Non-differentiable at zero. Requires special optimization approaches




$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2$$

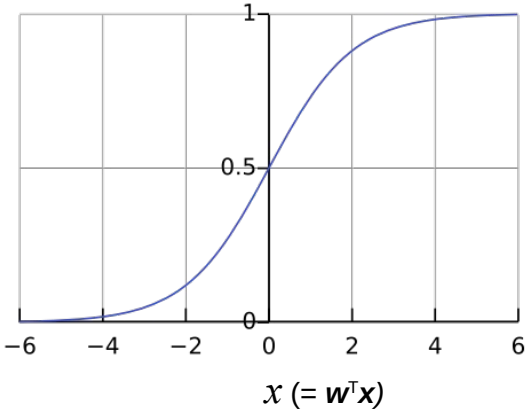
$$L1 = \sum |y_{\text{true}} - y_{\text{pred}}|$$

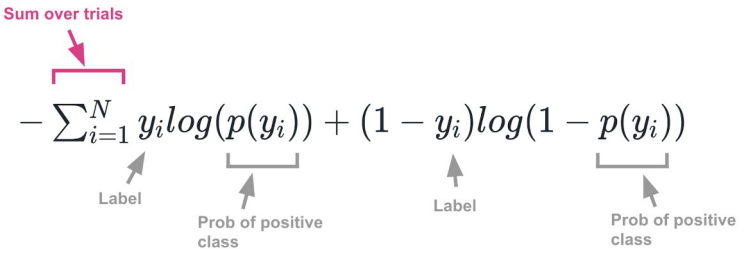
$$L2 = \sum (y_{\text{true}} - y_{\text{pred}})^2$$

# Supervised algos: Loss

- For binary classification (S vs B), the last layer should have one neuron with a sigmoid activation function
  - Corresponds to probability ($\sigma \rightarrow P \in [0,1]$) that input (***x***) belongs to S
- The network is then trained (adjusts the weights, ***w***) to minimize the neg-log-likelihood of the data given the model. The corresponding loss function is often called "(binary) ***cross-entropy***"

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



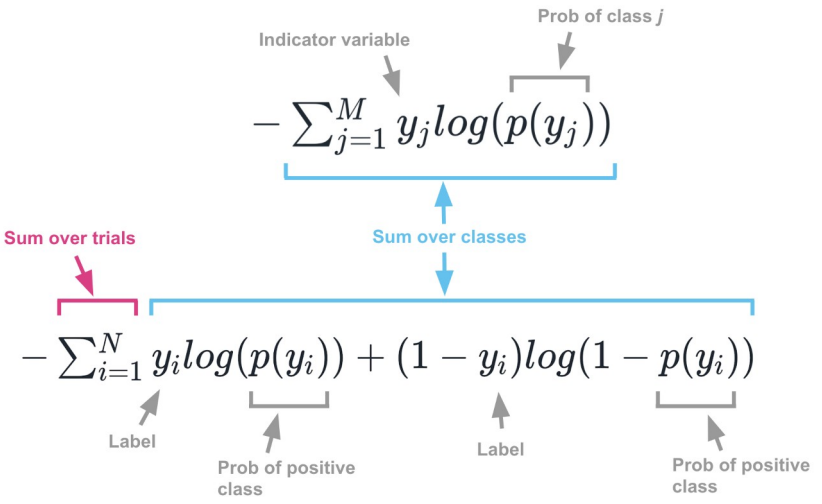




credit: blog.csdn.net

# Supervised algos: Loss

- For multi-class classification tasks, the normalized exponential function ("***softmax***") is used as the activation function in the last layer of the network
  - Corresponds to the probability (*P ϵ[0,1]*) that an input (***x***) belongs to a given class (*y = j*)
  - Generalization of the sigmoid to multiple classes
  - The number of softmax neurons in the last layer = number of classes
- The network is then trained (adjusts the weights, ***w***) to minimize the neg-log-likelihood of the data given the model. The corresponding loss function is often called "***cross-entropy***"

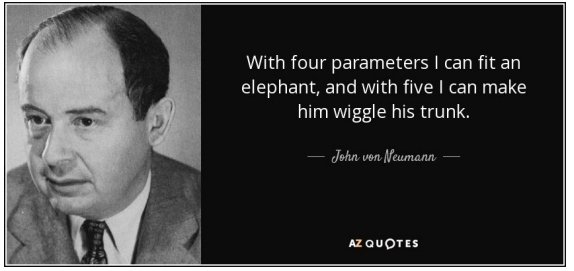
$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

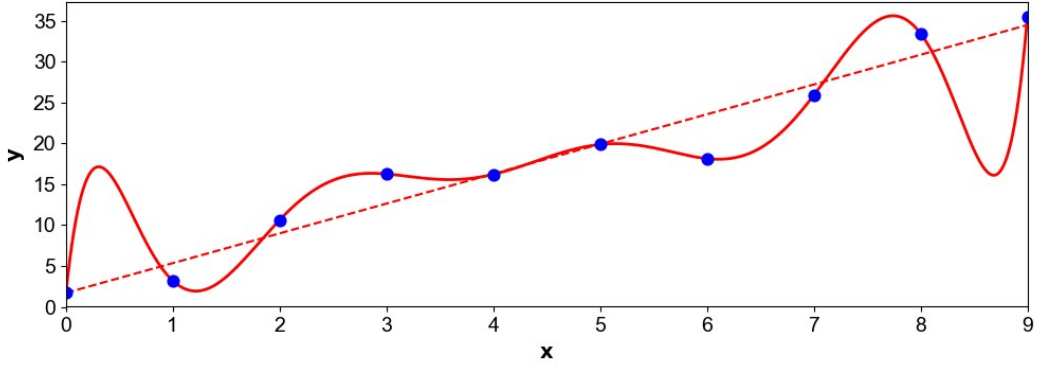



credit: blog.csdn.net

# Supervised algos: Regularization

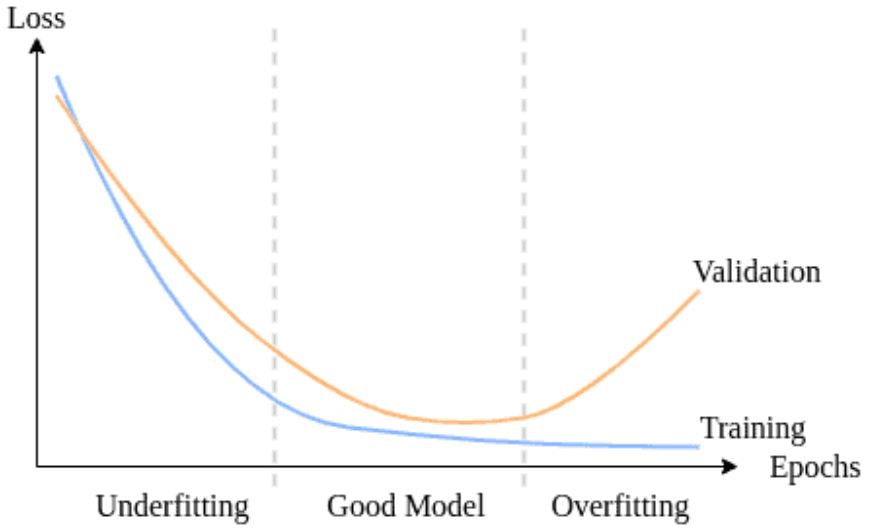
- NNs can have millions of parameters
- This can lead to the NN achieving small loss on the given data, but large loss on unseen data of the same type
- This is a well-known problem for functions that have too many parameters wrt data: they ***overfit*** the given points, failing to ***generalize*** to the unseen ones



# Supervised algos: Regularization

- Check for the presence of the problem
  - Always monitor the loss as the training progresses and split the data into the ***training*** and ***validation sets***; the latter not used in backprop, only to compare the losses
- Several ways to keep it under control:
  - More data
  - ***Early stopping***
  - ***Regularization***



# Supervised algos: Regularization

- Common regularization methods:
  - Add extra term to the loss to penalize large absolute (*L1*) or squared (*L2*) sum of the weights

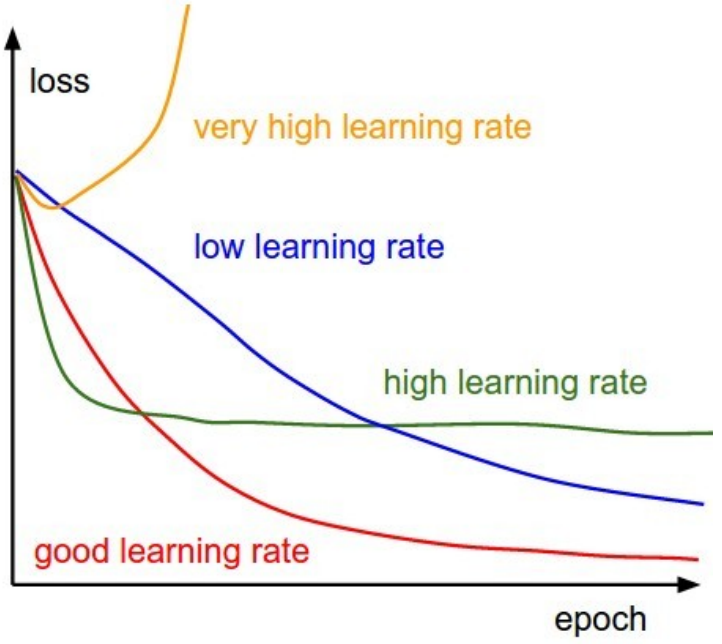
    $L = C + \lambda \cdot R$,

    where $R$ is the sum of squared/absolute weights and $\lambda$ is tweak parameter. *L2* is more common (in PyTorch implemented as a part of Adam optimizer as ***weight decay***)
  - Use ***Dropout***

    Randomly “kills” a certain fraction of neurons during training

# Supervised algos: Training

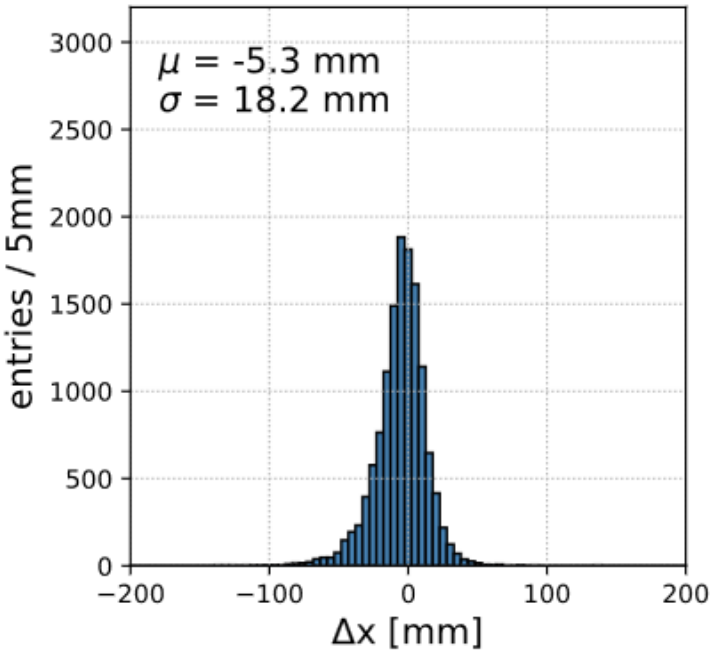
- Tips for successful training:
  - Consider data preprocessing
    - ***Zero-centering*** is common, may help with faster learning
    - Outlier removal (if physically motivated)
  - Initialize the weights properly
    - ***Xavier Glorot*** for TanH
    - ***He Kaiming*** for ReLU and Leaky ReLU (+small initial bias)
    - Including ***batch-normalization layer***s before nonlinearity may help reduce poor init. choice
  - Babysit the training
    - Try different learning rates for small number of epochs to choose an optimal (ideally, train in steps - start with higher rate, then decrease)
    - Likewise, start from small, if any, regularization




credit: Stanford CS231n (cs231n.github.io)

# Supervised algos: Evaluation

- For regression networks, the loss itself (minus any reg. terms) is often what you were after, as reconstruction accuracy in physics is commonly measured with MSE and related metrics
- For classification, there are two main tools to be aware of: ***confusion matrix*** and ***ROC curve***




2018 JINST 13 P08023

# Supervised algos: Evaluation

- Confusion matrix conveniently summarizes all main aspects of a classifier:
  - Correct classification rates (***True Positive*** and ***True Negative***) on the diagonal
  - ***Type-1*** errors (what physicists call background contamination) and ***Type-2*** errors (efficiency losses) off-diagonal
- Can be used to compute related metrics important in physics:
  - Signal efficiency (aka Recall, aka Sensitivity)
  - Background rejection (aka Specificity)

| | Predicted S | Predicted B |
|---|---|---|
| Actual S | **TP** | **FP** |
| Actual B | **FN** | **TN** |

***Signal Efficiency*** = TP/(TP+FN)

***Background Rejection*** = TN/(TN+FP)

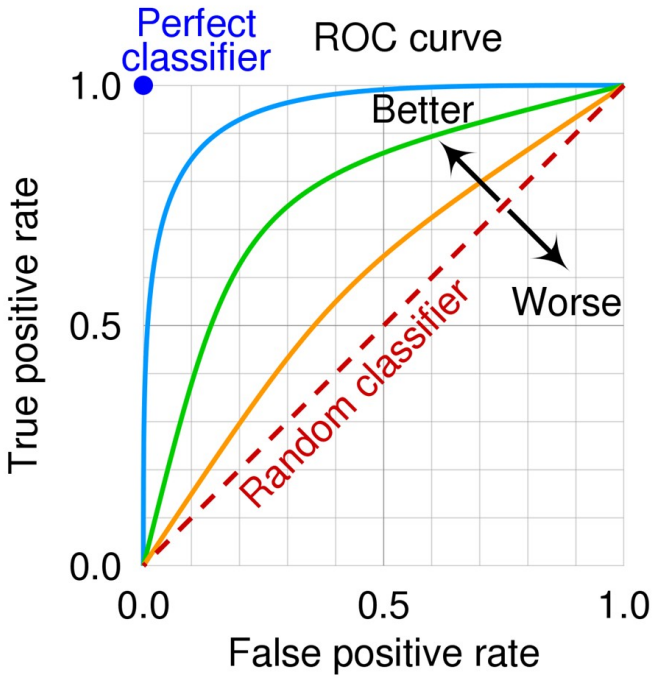
***Accuracy*** = (TP+TN)/(TP+TN+FP+FN)

***Precision*** = TP/(TP+FP)

***F-1 score*** = 2 * (Precision*Recall)/(Precision+Recall)

# Supervised algos: Evaluation

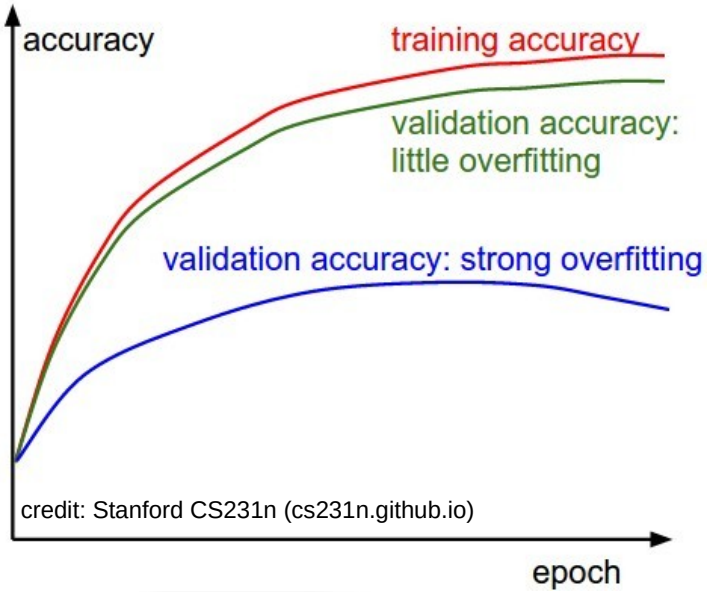
- The rates depend on the ***threshold***
  - 0.5 is used by default
- You may prefer a different value
  - E.g., if not limited by statistics, may prefer cleaner sample (less Type-1) at the cost of some efficiency loss (more Type-2)
- ***Receiver Operating Characteristic*** (ROC) curve visualizes this nicely
- ***Area Under the Curve*** (AUC) is often used for comparing different models, but some consider this to be oversimplification




credit: nomidl.com

# Quiz

4) What are some of the valid ways to control overfitting?
    1) Add L2 regularization term to the loss function
    2) Randomly drop a fraction of neurons from the network each training step
    3) Significantly increase amount of training data
    4) Stop training when the validation loss starts diverging from the training loss
    5) All of the above

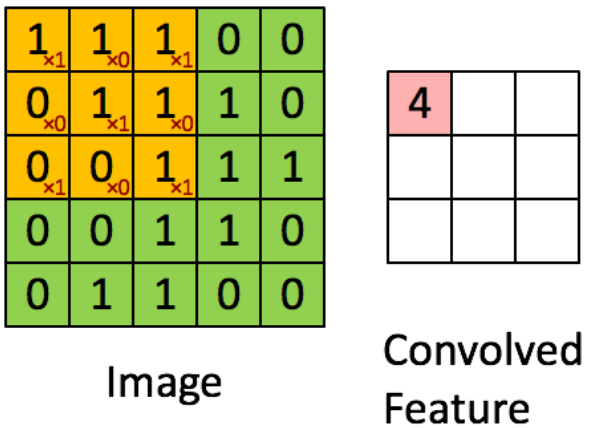



ML lecture quiz Q4

诚邀您填写本问卷，扫码即可！

# Convolutional Neural Nets

- MLP are and will remain useful for neutrino/particle physics
  - Can extract high-level features almost directly from raw data, w/o “traditional” reconstruction steps
  - Will see an example of this in today’s tutorial
- Another workhorse added recently to our toolbox - CNNs
  - Designed for images, and our data are often similar
- Main trick of a CNN, compared to MLP - ***locality***. Don’t couple the whole input vector to the layer of neurons at once. Instead, have a small-ish set of neurons (***filter***s, aka ***kernel***s) connect to a small-ish local region (***receptive field, F***) at a time and “scan” over the input step by step
  - Scales much better with input size!
- They are also touted as ***translation-invariant***
  - Only they really are not by default (arXiv:2110.05861)




credit: http://deeplearning.stanford.edu

# Convolutional Neural Nets

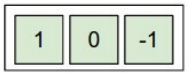
- ***Depth (D)***: number of filters we would like to use, each learning to look for something different in the input
- ***Stride (S)***: Filter sliding step. If 1 (common) then filters move one "pixel" at a time
- ***Zero-padding (P)***: Adding zeros to the input image around the borders. Commonly used to ensure input and output sizes are the same

**Output volume: (*W*−*F*+2*P*)/*S*+1 (xD)**

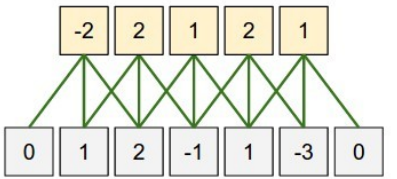
<u>1D example</u>

Input (*W*=5, *P*=1):

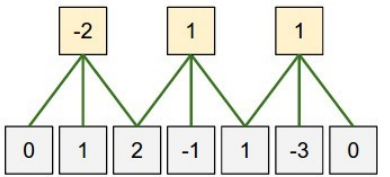

Filter (*F*=3):
*D*=1




Result (*S*=1)
OV=5




Result (*S*=2)
OV=3




credit: Stanford CS231n (cs231n.github.io)

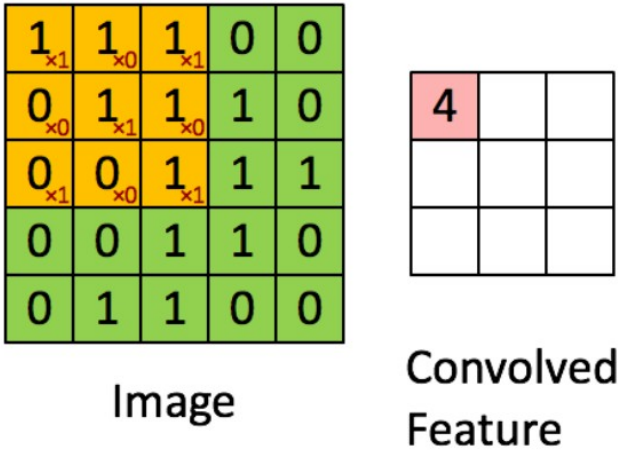
# Convolutional Neural Nets

- In this example, the ***convolutional layer (CL)*** is represented by one neuron and contributes only nine weights and a bias (shared for the whole image), so a total of 10 parameters

**Output volume (*W*x*H*x*D*):**

**(*W*−*F*+2*P*)/*S*+1 x (*H*−*F*+2*P*)/*S*+1 x *D***

2D example




*W* = 5, H = 5 (5x5x1)
Filter: *F* = 3
*S* = 1, *P* = 0
OV = 3x3x1 (*D*=1, one filter)

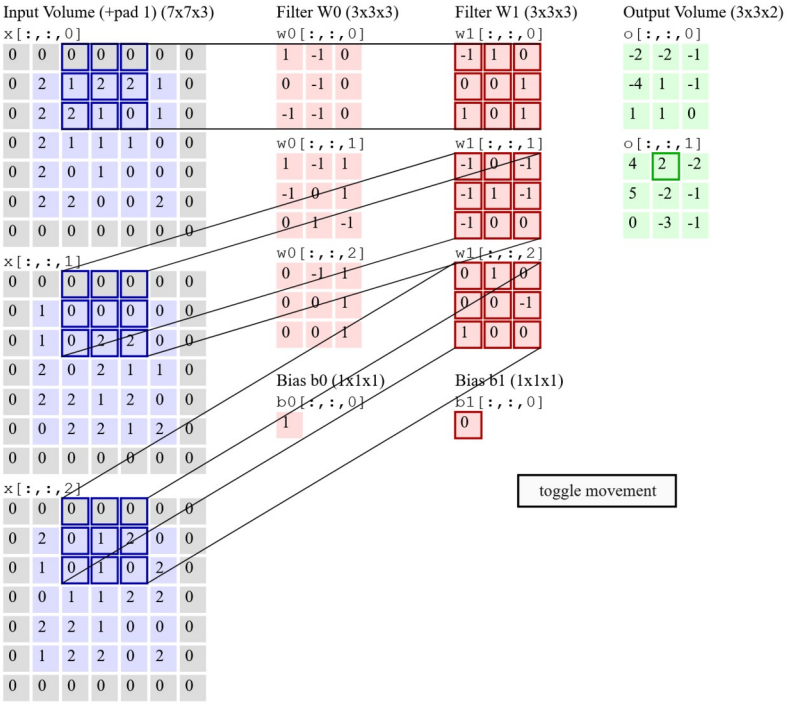
# Convolutional Neural Nets

- Input: 5x5x3, *P*=1
- Two filters with *F*=3, *S*=2
- OV = 3 x 3 x 2
- 2 neurons,  56 parameters
  - 27 weights and 1 bias per kernel

**Output volume (*W*x*H*x*D*):**

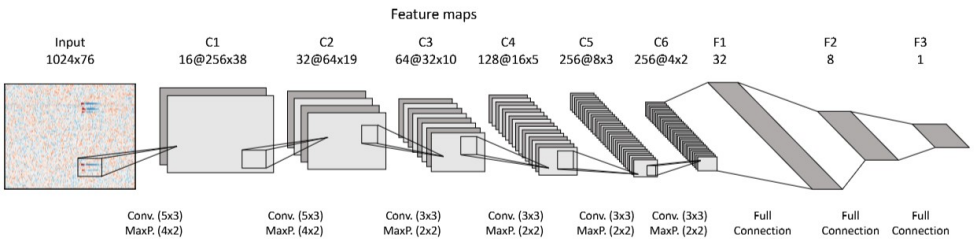
**(*W*−*F*+2*P*)/*S*+1 x (*H*−*F*+2*P*)/*S*+1 x *D***

“3D” example (2D with 3 colors)



# Convolutional Neural Nets

- After a CL, a nonlinearity is applied as usual
- In a deep NN, several CLs are used, with different number of filters and *F*
- To further reduce number of parameters and control overfitting, a ***pooling layer (PL)*** is commonly applied between CLs
  - 4x2 ***max*** PL here reduces 1024x76 data frame to 256x38, taking maximum value in each 4x2 element of the original frame
  - PL does reduce level of detail in the data, so don't overdo it. https://arxiv.org/abs/1412.6806
- In the end, the usual FC layers are used to reduce output to the desired outcome



# Other designs

- In the past few years, several extensions potentially more applicable to specific types of inputs
  - Spherical CNN (JUNO's detectors are on a sphere, not planar)
  - Hexagonal CNN (Some Cerenkov cosmic ray/nu detectors)
  - Sparse CNN (Some long-baseline nu detectors often have just a few % of non-zero pixels)
- "But as a universal function approximator with 1M+ parameters, can't a vanilla CNN simply learn by itself how to map a plane to a sphere? Why an extra complication with new designs?"
  - It can, but - like with MLP cf. CNN - will be less efficient (more parameters, longer/finicky training) than a design that naturally includes the relevant geometric prior
  - Still, it's up to you to evaluate critically whether some new design has a clear-enough advantage for your specific goal before investing time in it

**Scaling Spherical CNNs**

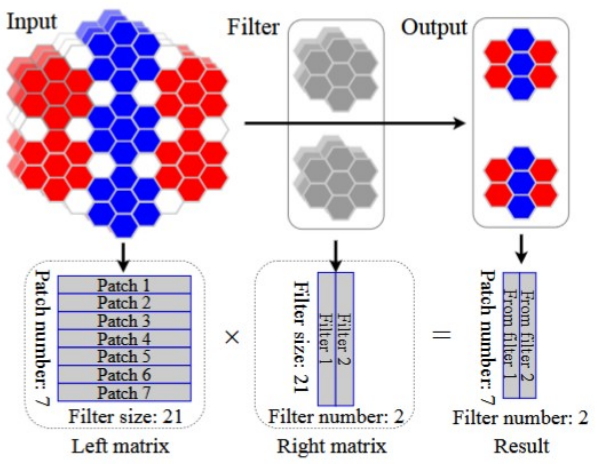
Carlos Esteves, Jean-Jacques Slotine, Ameesh Makadia

**4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks**

Christopher Choy, JunYoung Gwak, Silvio Savarese

**HexCNN: A Framework for Native Hexagonal Convolutional Neural Networks**

Yunxiang Zhao, Qiuhong Ke, Flip Korn, Jianzhong Qi, Rui Zhang



# Other designs

- ***Relational Inductive Bias*** - a set of built-in assumptions on how entities interact or relate to one another
  - Guides learning by constraining the space of possible functions to those that respect the underlying relational structure of the data
  - Choose design with an appropriate rel. inductive bias

| Component | Entities | Relations | Rel. inductive bias | Invariance |
|---|---|---|---|---|
| Fully connected | Units | All-to-all | Weak | - |
| Convolutional | Grid elements | Local | Locality | Spatial translation |
| Recurrent | Timesteps | Sequential | Sequentiality | Time translation |
| Graph network | Nodes | Edges | Arbitrary | Node, edge permutations |

**Relational inductive biases, deep learning, and graph networks**

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, Razvan Pascanu

# Now, let’s start Tutorial

- Jupyter notebook and some waveform data
- Password: cUaw (please don’t share further)