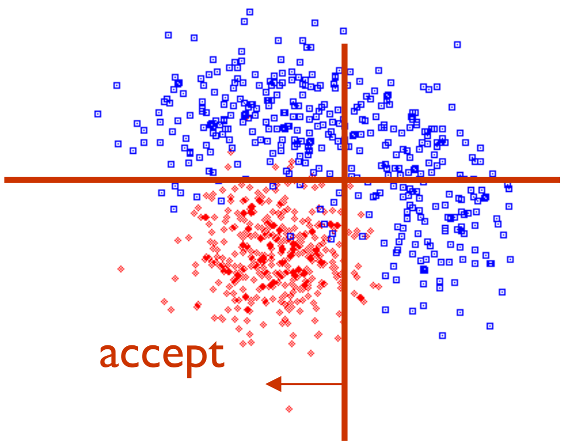
Introduction to deep Learning for high energy physics

Tae Jeong Kim (Hanyang University)

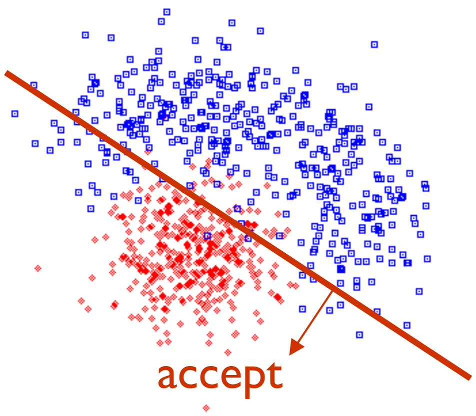
For 2nd MadAnalysis 5 workshop at KIAS

Feb. 18 in 2020

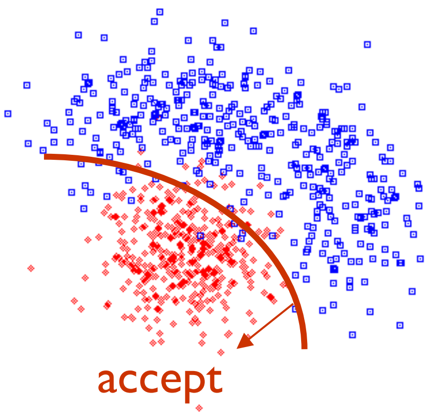
Event selection



traditional way



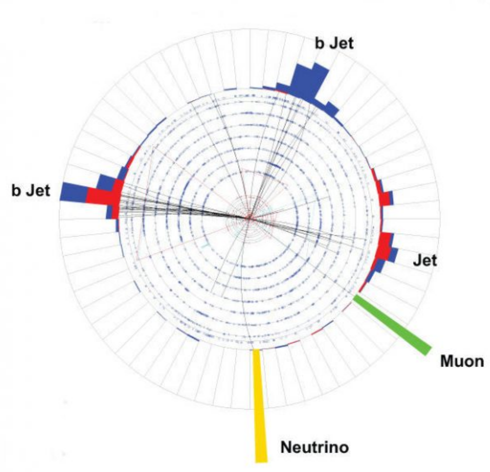
linear

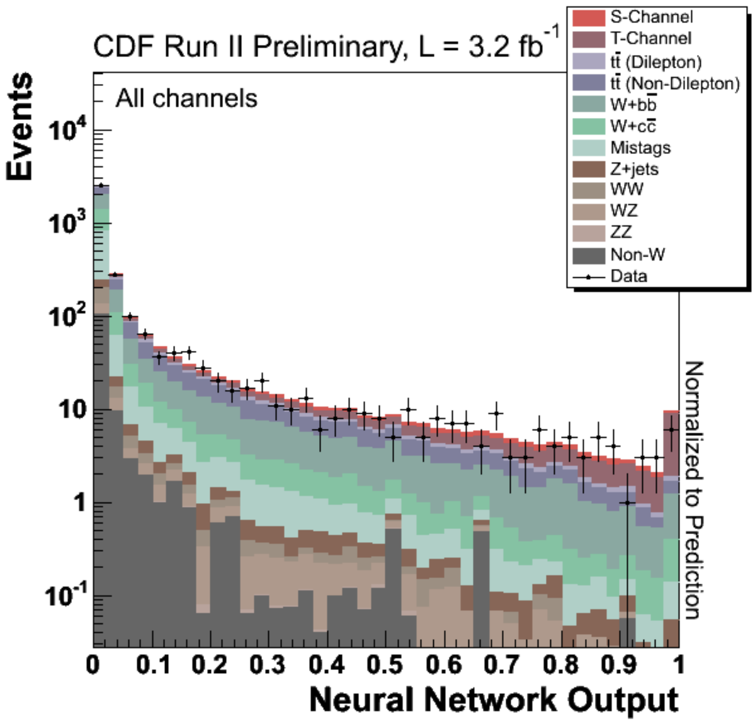


nonlinear

Single top quark discovery with ML

- The small cross section → simple cut and count does not work
- Single top quark was discovered with a help of machine learning technique in 2009





Machine learning

Machine in Industry

Autonomous driving

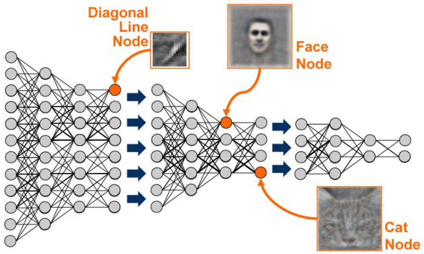
Face recognition

Customers pattern

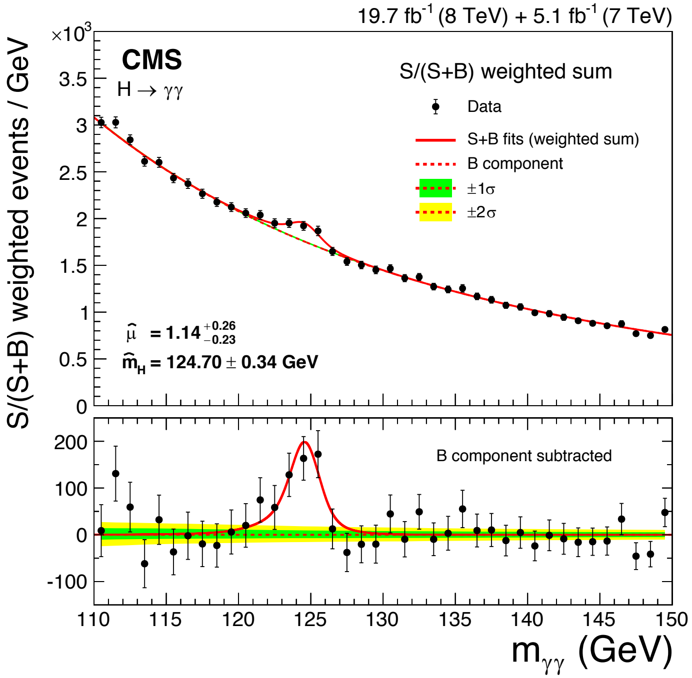
Voice recognition

Success of Machine learning

Big data + GPU



Machine learning in Higgs discovery

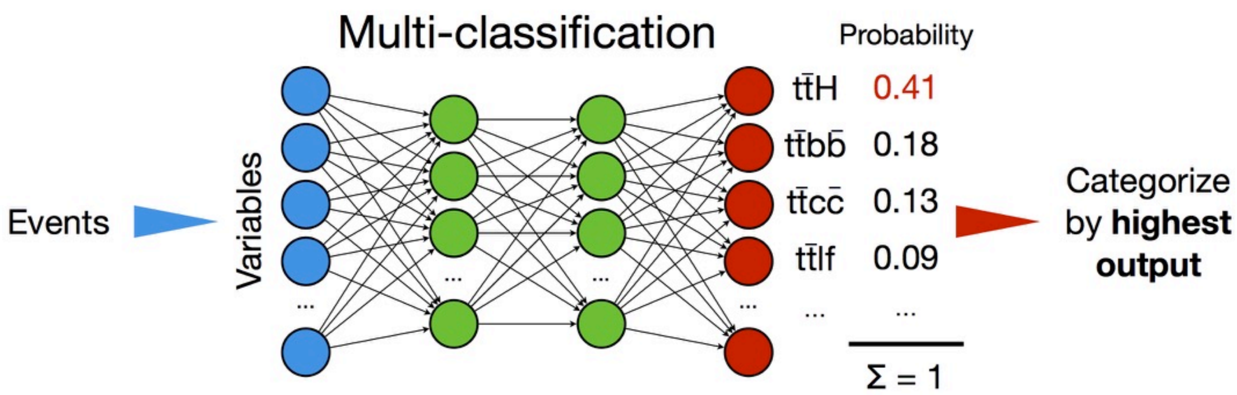


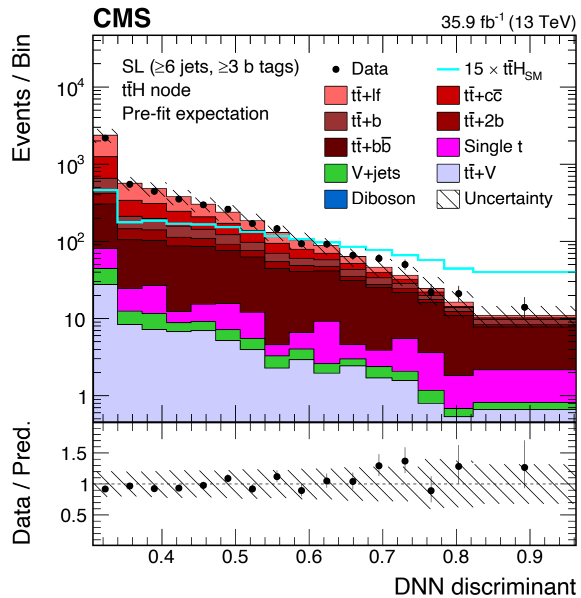
Machine learning

- Photon energy by regression
- Photon ID by Boosted Decision Tree (BDT)
- Multivariate Data Analysis for event classification

Event categorization with Deep Neural Network

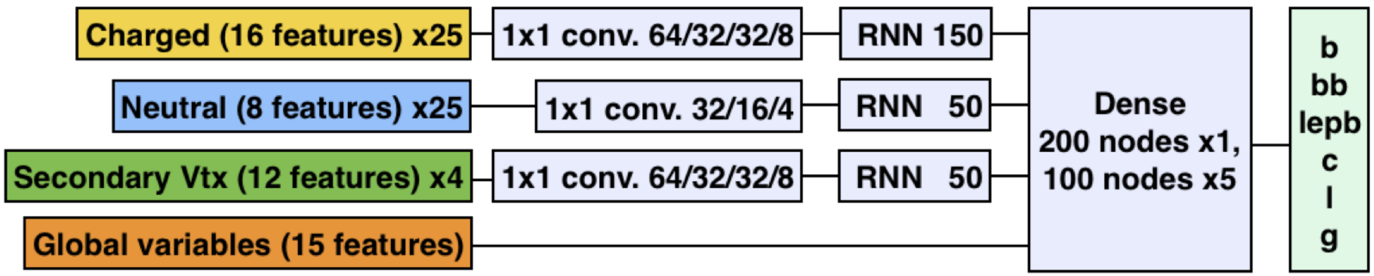
- Precision of categorization scheme using jets & b-tags is difficult with high b-tag multiplicity
- Use DNNs to categorize using most probable process and jets

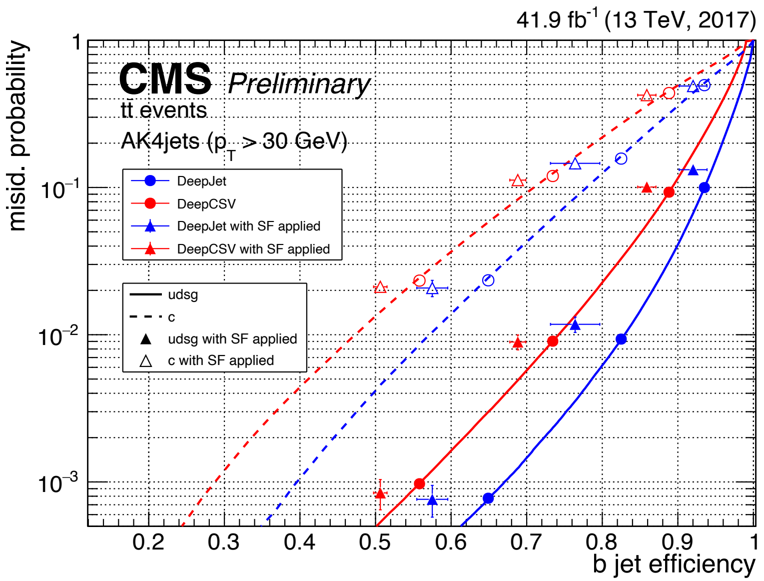


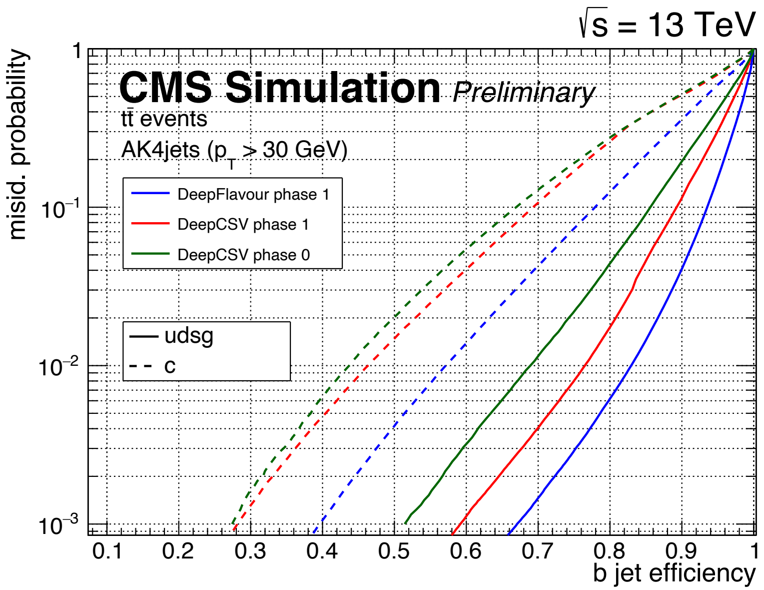


JHEP 03 (2019) 026

Performance with DNN for b-tagging





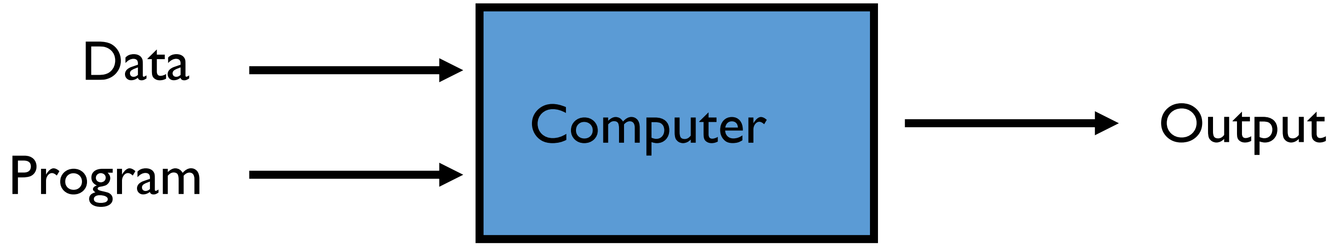


CMS DP-2018/058

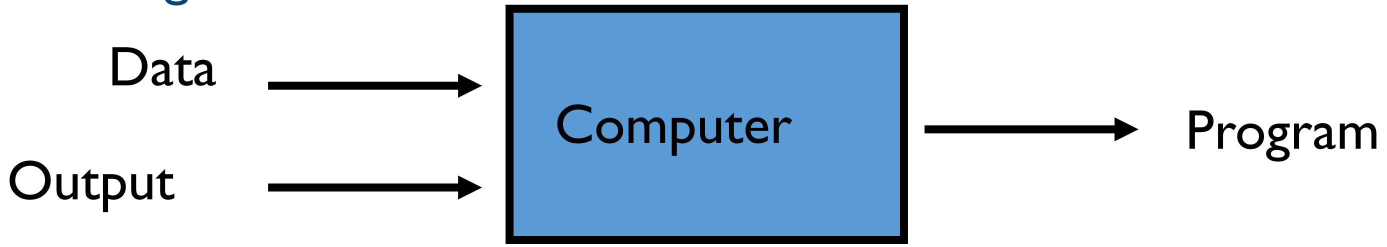
CMS DP-2018/033

What is Machine learning?

Traditional Programming



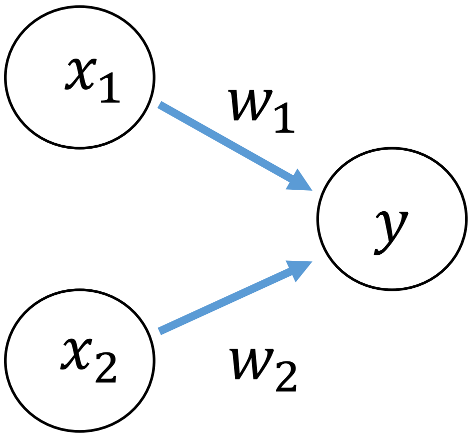
Machine Learning



- Examples : spam filter
  - Traditionally you would write a detection algorithm for each of the pattern from spam → need to add rules forever
  - Machine learning learns automatically which words and phrases are good predictors of spam → short, easier to maintain and accurate

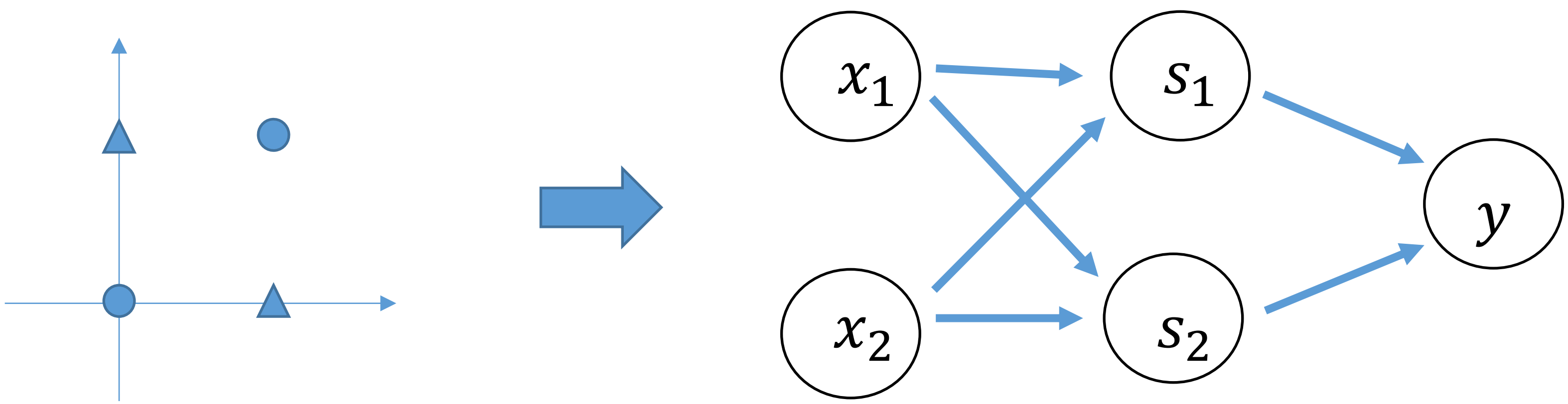
Perceptron

- The idea of perceptron was created by Frank Rosenblatt in 1957



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

- Multi-layer perceptron



Activation function

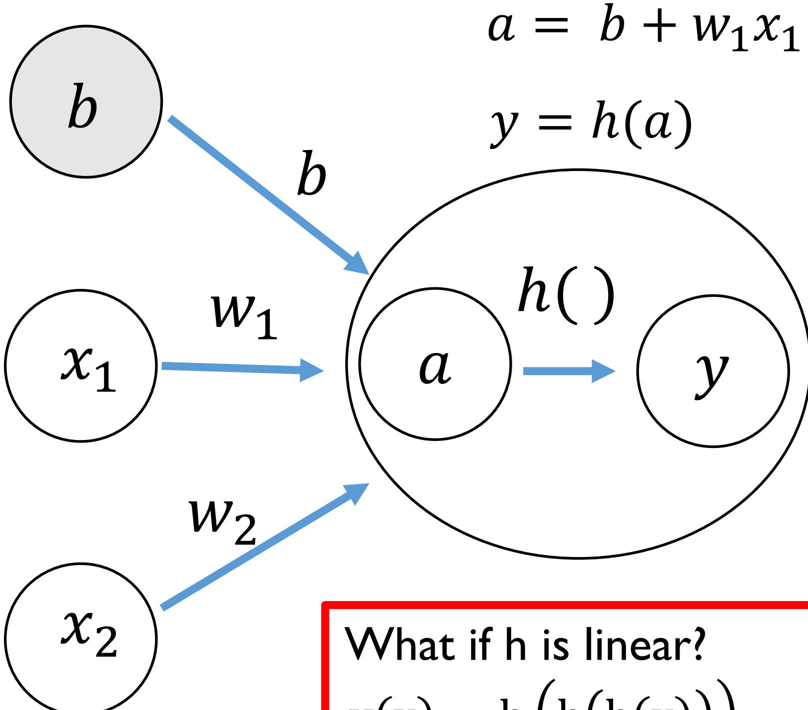
non-linear is essential for deep neural network!

- h(x) is the activation function which determine whether or not we activate the sum of the input

Adding bias to perceptron

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

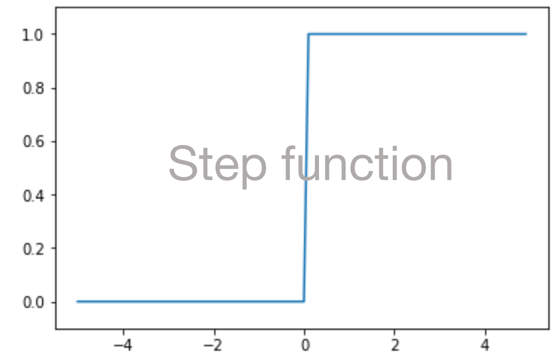
$$y = h(a)$$

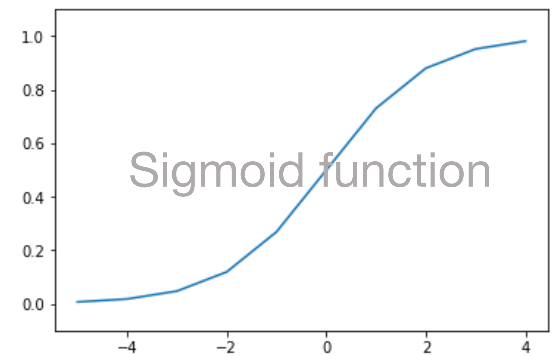


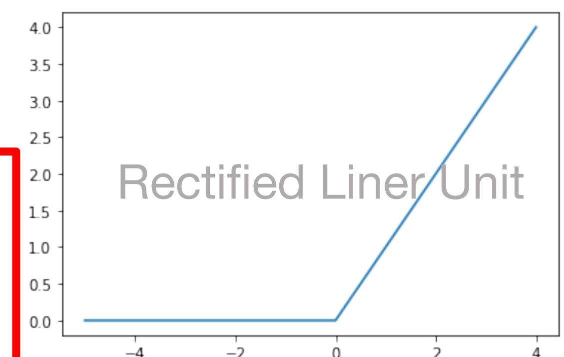
What if h is linear?

$$y(x) = h\Big(h\big(h(x)\big)\Big) = c * c * c * x = ax$$

No reason to have multi-layers

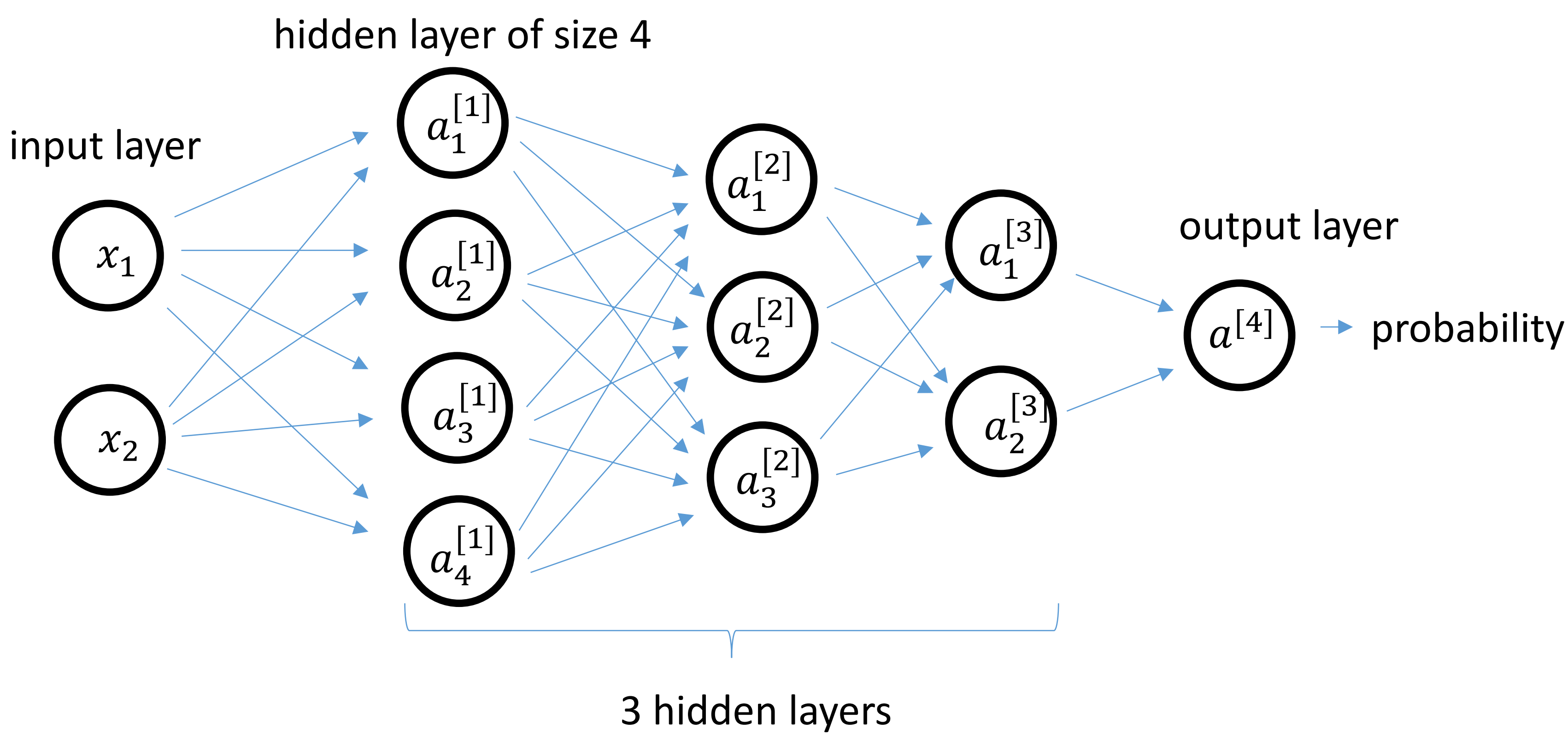






Deep neural network

- Weight (w) and bias (b) have to be determined manually by human
- In neural network, we will let computer to determine the weight (w) and bias (b)



Output layer

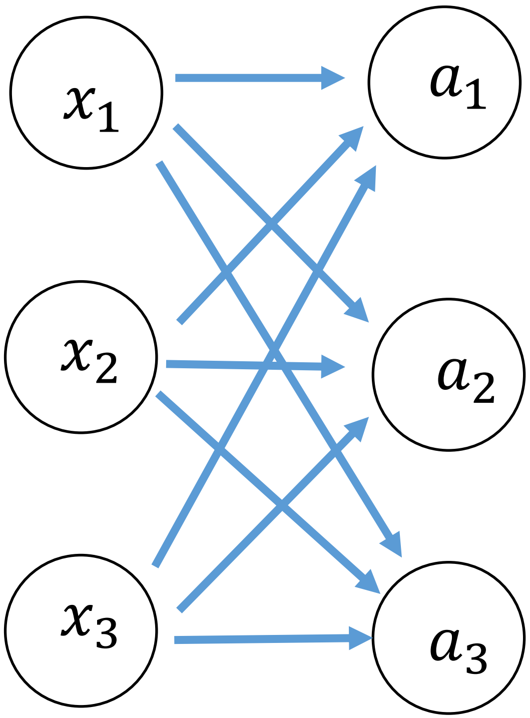
- Regression : parameter determination
- Classification
  - Binary classification : sigmoid function
  - Multi-classification : softmax function

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)} = \frac{C exp(a_k)}{C \sum_{i=1}^{n} exp(a_i)}$$

$$= \frac{exp(a_k + logC)}{\sum_{i=1}^{n} exp(a_i + logC)}$$

$$= \frac{exp(a_k + C')}{\sum_{i=1}^{n} exp(a_i + C')}$$

C' to prevent from being $\frac{\infty}{\infty}$



Training

- From the training dataset, determine the weights automatically
- Will use loss function to find the weights in a way to minimize the loss function

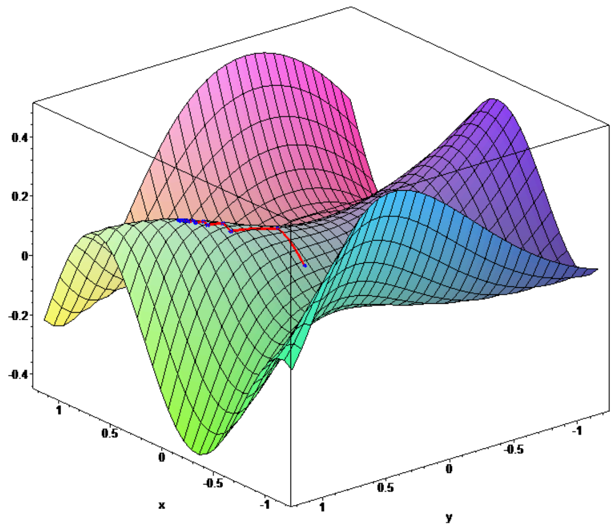
training sample

$$\underbrace{(x_1, d_1), (x_2, d_2), \ldots, (x_n, d_n)}_{\text{training data}}$$

Adjust w_{ij} and b_i so that output y_n is close to d_n

Gradient decent

- Find minimum of the loss function



$$L = \frac{1}{2}\sum_k (y_k - t_k)^2$$ Mean Squared Error

$$L = -\sum_k t_k log y_k$$ Cross entropy

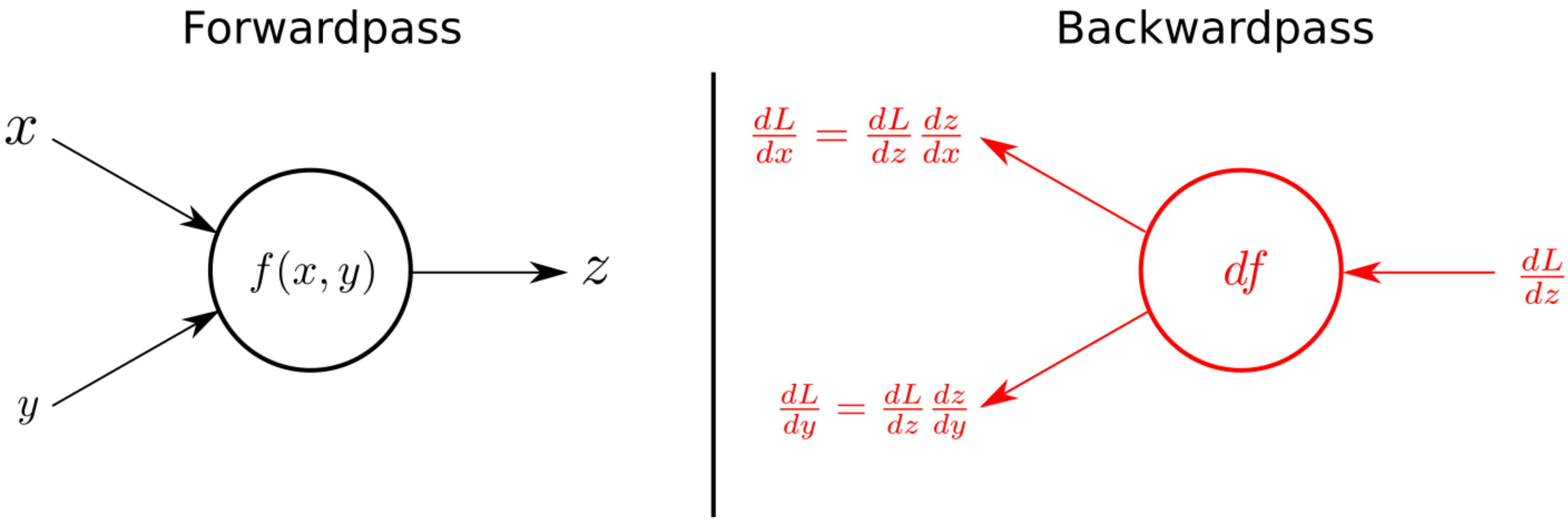
$$w = w - \eta \frac{\partial L}{\partial w}$$

η = leaning rate (hyperparameter)

- Mini-batch
  - If training data is large, it is not feasible to calculate the loss over the whole data
  - Randomly choose fraction of data and calculate the loss approximately
  - The fraction of data (N samples) is mini-batch

$$L = -\frac{1}{N}\sum_n \sum_k t_k log y_k$$

Forward and Backward

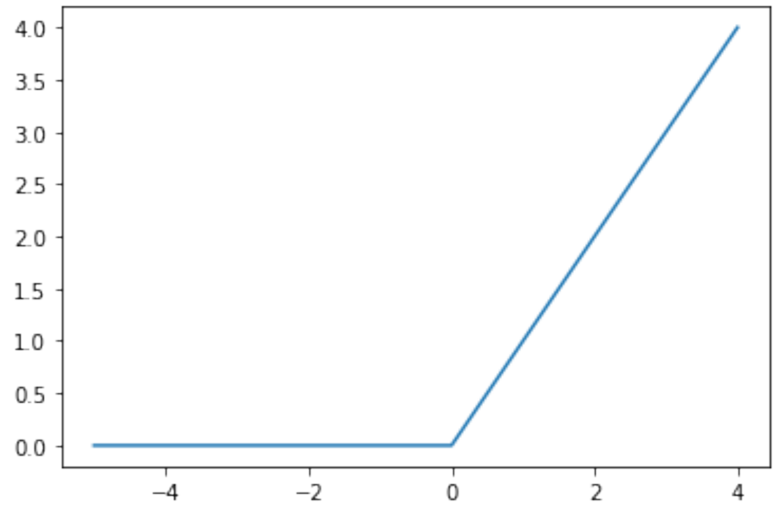


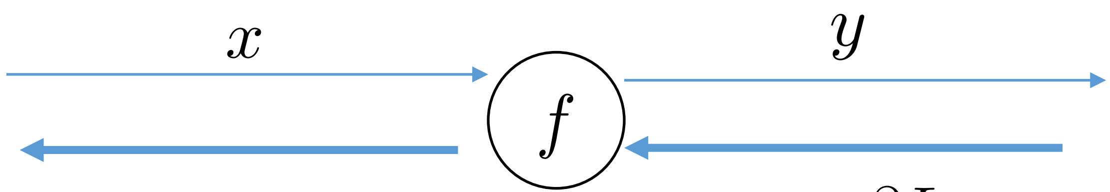
- We need to know how much x or y is changed when loss is changed
- Can rely on the chain rules in this case to calculate the derivatives analytically

Backward propagation with ReLU function

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \le 0) \end{cases}$$

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \le 0) \end{cases}$$



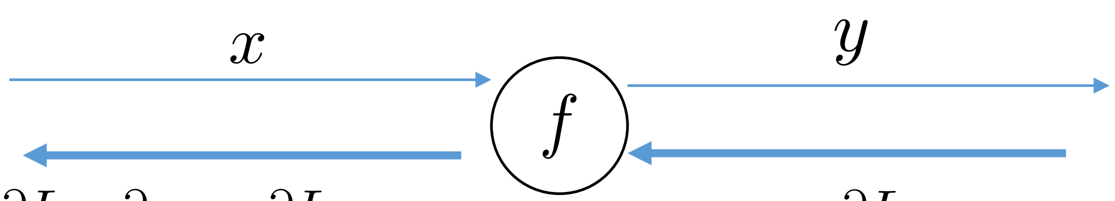


$x > 0$ $\quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{\partial L}{\partial y} \quad\quad \frac{\partial L}{\partial y}$

$x \le 0$ $\quad 0 \quad\quad \frac{\partial L}{\partial y}$

Backward propagation with Sigmoid function

$$y = \frac{1}{1+e^{-x}} \qquad \frac{\partial y}{\partial x} = y^2 e^{-x}$$



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{\partial L}{\partial y} \cdot y^2 e^{-x} \qquad \frac{\partial L}{\partial y}$$

can be simplified as follows:

$$\begin{aligned}
\frac{\partial L}{\partial y} y^2 e^{-x} &= \frac{\partial L}{\partial y} \frac{1}{(1+e^{-x})^2} e^{-x} \\
&= \frac{\partial L}{\partial y} \frac{1}{1+e^{-x}} \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{\partial L}{\partial y} y(1-y)
\end{aligned}$$

Optimization

- Sometimes training a very large deep neural network is painfully slow
- We can speed up the training using a faster optimizer instead of using the regular Gradient descent optimizer

$$w = w - \eta \frac{\partial L}{\partial w}$$

Momentum

$$v = \alpha v - \eta \frac{\partial}{\partial w}$$

$$w = w + v$$

Adaptive gradient

$$h = h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W = W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

Adam

$$m = \beta_1 m + (1 - \beta_1) \frac{\partial L}{\partial W}$$

$$v = \beta_2 v + (1 - \beta_2) \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$\hat{m} = \frac{m}{1 - \beta_1}$$

$$\hat{v} = \frac{v}{1 - \beta_2}$$

$$w = w - \frac{\eta}{\sqrt{\hat{v} + \epsilon}} \hat{m}$$

Momentum

- Imagine a bowling ball rolling down a gentle slop on a smooth surface
  - It will start out slowly but it will quickly pick up momentum until it eventually reaches terminal velocity
  - v is a new variable corresponding to velocity
- In contrast, gradient descent will simply take small regular steps down the slope
  - It takes much more time to reach the bottom

$$v = \alpha v - \eta \frac{\partial}{\partial w}$$

$$w = w + v$$

AdaGrad (adaptive gradient)

$$h = h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$W = W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

- Gradient is scaled down by a factor of $\sqrt{h}$
- Low learning rates for frequently occurring features and high learning rates for infrequent features
- No need to tune the learning rate
- Often stops too early before reaching the global optimum
- Should not use it to train deep neural network
- Might be efficient for simple tasks (Linear regression)

Adam

- Adam stands for adaptive moment estimation
- Combination of Momentum and RMSProp (AdaGrad)

$$m = \beta_1 m + (1 - \beta_1)\frac{\partial L}{\partial W}$$

$$v = \beta_2 v + (1 - \beta_2)\frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$

$$\hat{m} = \frac{m}{1 - \beta_1}$$

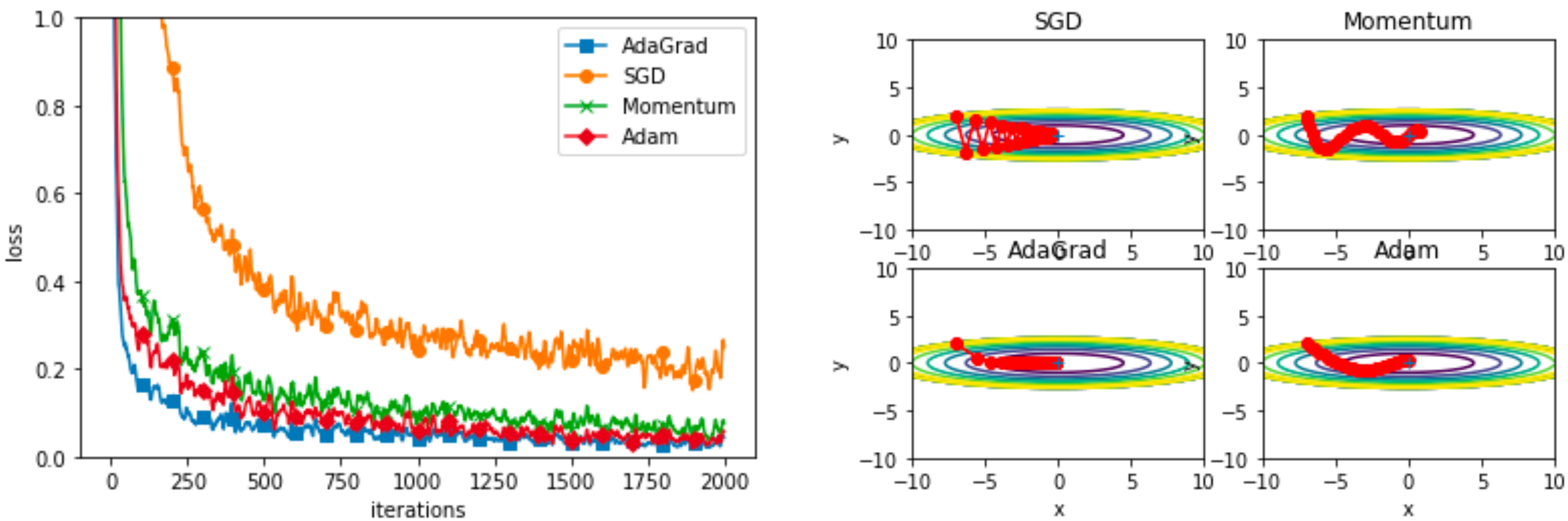
$$\hat{v} = \frac{v}{1 - \beta_2}$$

$$w = w - \frac{\eta}{\sqrt{\hat{v} + \epsilon}}\hat{m}$$

- m and v are initialized at 0, they will be biased toward 0 at the beginning of training
- These two steps will help boost m and v at the beginning of training

$$\beta_1 = 0.9 \quad \beta_2 = 0.999 \quad \eta = 0.001$$

Comparisons



- Gradient decent would not be the best way to optimize
- Other method such as Adam should be considered for fast optimization

Overtraining

- Overfitting in statistics is production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably
- Can happen when…
  - many weight parameters
  - training data is small
- Possible solutions:
  - Select one with fewer parameters
  - Gather more training data
  - Reduce the noise in the training data (fix data errors and remove outliers)
- Early stopping can also be one of the options to avoid overtraining
- But we can usually get much higher performance when we combine it with other regularization techniques (see next slide)

Weight decay (L2 regularization)

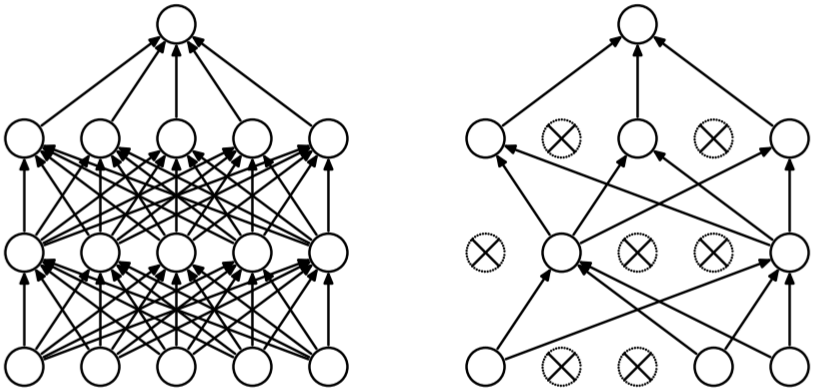
- Regularization – constraining a model to make it simpler and reduce the risk of overfitting
- Add term $\frac{1}{2}\lambda W^2$ to the loss function → $L = L + \frac{\lambda}{2}W^2$

$$W = w - \alpha\frac{\partial L}{\partial W} - \alpha\frac{\lambda}{2}\frac{\partial W^2}{\partial W}$$
$$= (1 - \alpha\lambda)W - \alpha\frac{\partial L}{\partial W}$$

New term $(1 - \alpha\lambda)W$ can constrain weights W
Prevent weights from being too high

Dropout

- At every training step, every neuron has a probability p of being temporarily "dropped out"
- It will be entirely ignored during this training step but it may be active during the next step
- Here p is called the dropout rate



(a) Standard Neural Net (b) After applying dropout.

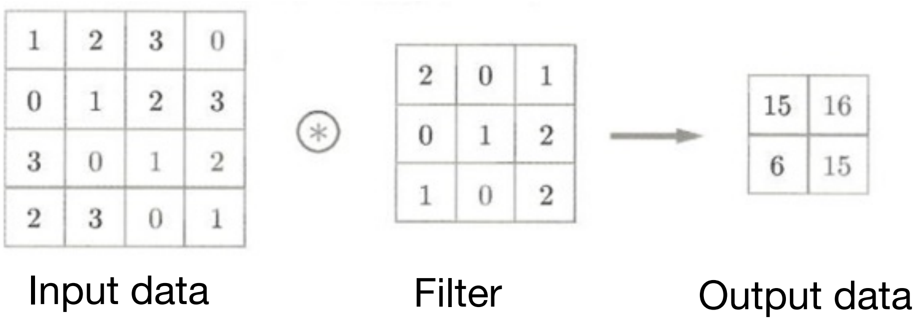
Journal of Machine Learning Research 15 (2014) 1929-1958

Why convolution neural network?

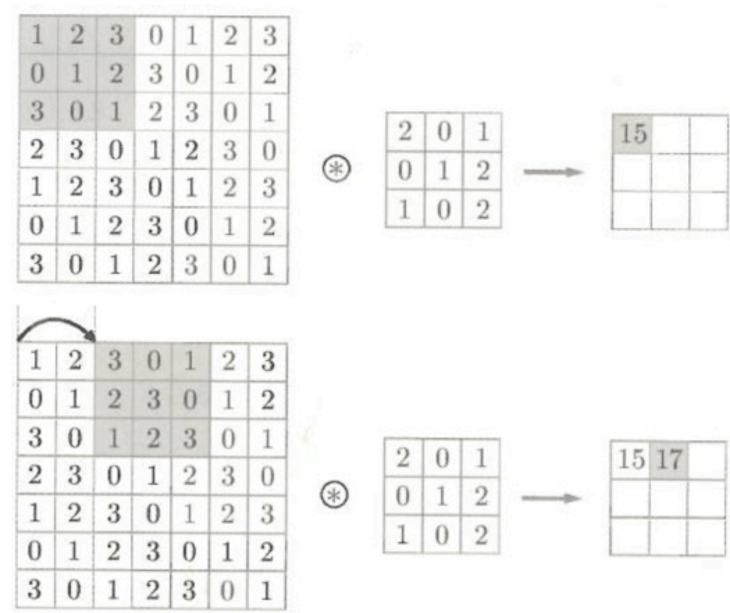
- Fully connected network has problems
  - A gray image has $28 \times 28 = 784$ weight parameters
  - For RGB color image, it has $3 \times 28 \times 28$ $(d \times h \times w) = 2352$ weight parameters
  - Ignores its spatial information
  - Has too many parameters that should be determined from training

- Convolution Neutral Network (CNN)
  - Examples : images recognition, images classification, face recognition…
  - Preserves the relationship between nearby pixels
  - Keep the spatial information throughout the layers
  - Start by collecting local information, at the end, it will represent more global, high-level and representative information

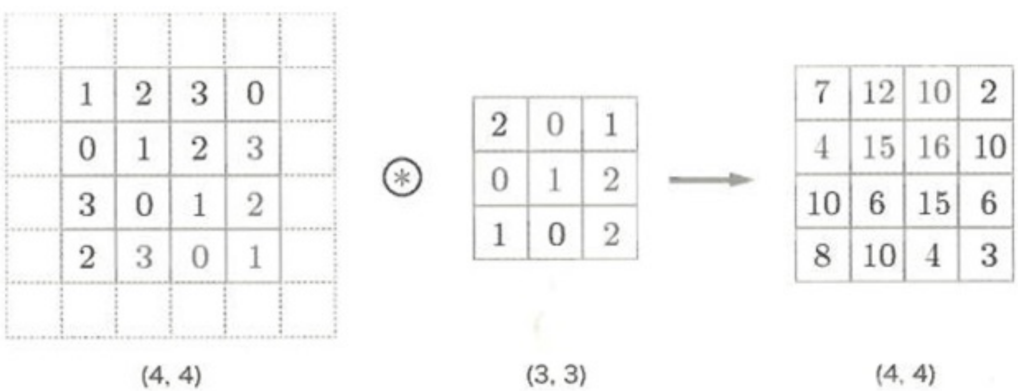
Convolution



Stride



Padding



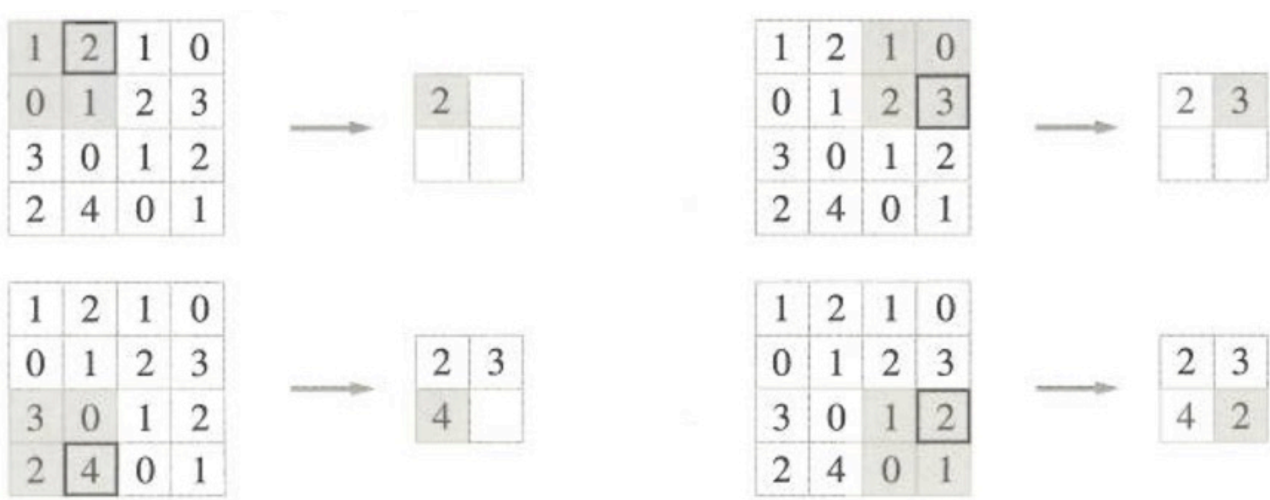
Output width and height

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

Pooling

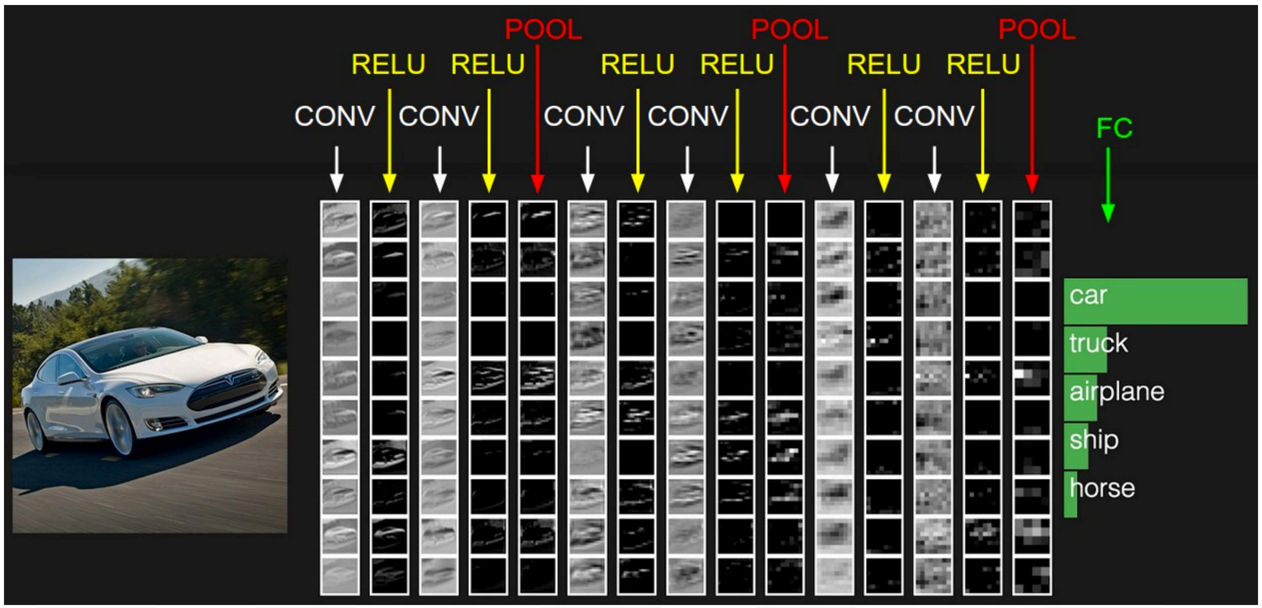
- Down-sample input representation, e.g. keeping the max value (max pooling)
- There are no parameters to be learned

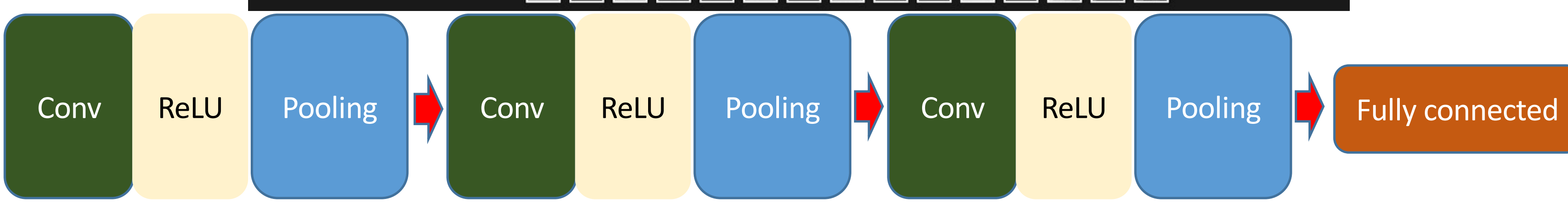


- Goal is to subsample the input image to reduce computational load, the memory usage, and the number of parameters
- Stable and solid from the variations of input data

ConvNet architecture

- Start by collecting local information, at the end it will represent more global, high-level and representative information





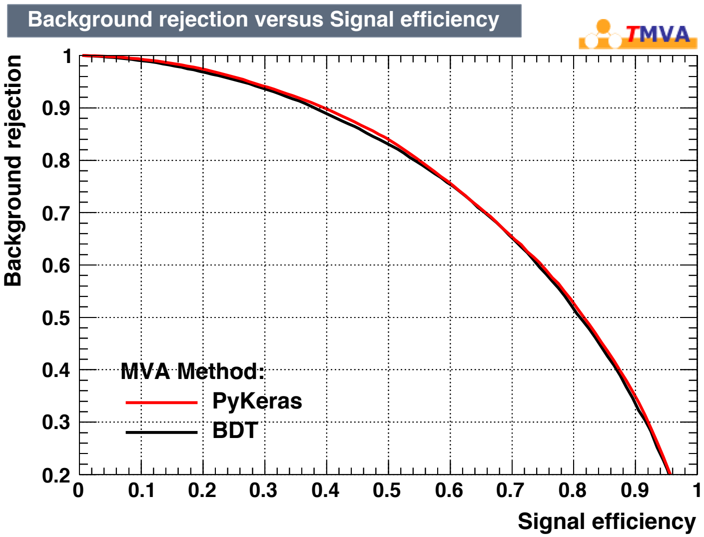
Machine Learning software and Tools

Two approaches

- Externally developed software such Tensorflow, theano, Caffe, MXNet,……
  - Too many choices guaranteed to be supported over the lifetime of particle physics experiments
  - difficulty of adaptation to HEP specific requirements
- Focus on HEP-developed ML toolkits, Toolkit for Multivariate Analysis (TMVA) in ROOT.
  - long-term support in HEP
  - Can be adapted to specific needs of HEP
  - Challenges in incorporating new algorithms and ideas

TMVA

- TMVA has been used for multivariate data analysis in High Energy Physics for two decades
- Compatible with ROOT data format
- Now deep learning framework is available in TMVA
  - PyMVA interface to scikit-learn
  - PyKeras interface to Keras
  - High-level interface to Theano, TensorFlow deep-learning library



Conclusion

- Since Higgs discovery, we have been looking for new physics
- With HL-LHC, it is getting more challenging to analyze data
- Not only better computing resource but also different approaches to big data analysis are required
  - Rare process
  - Huge pileup background
  - Unknown physics
- Machine learning would be the promising approach
- Right moment to apply machine learning in High Energy Physics