CMG

PYTORCH FOR DEEP LEARNING RESEARCH

THONG NGUYEN

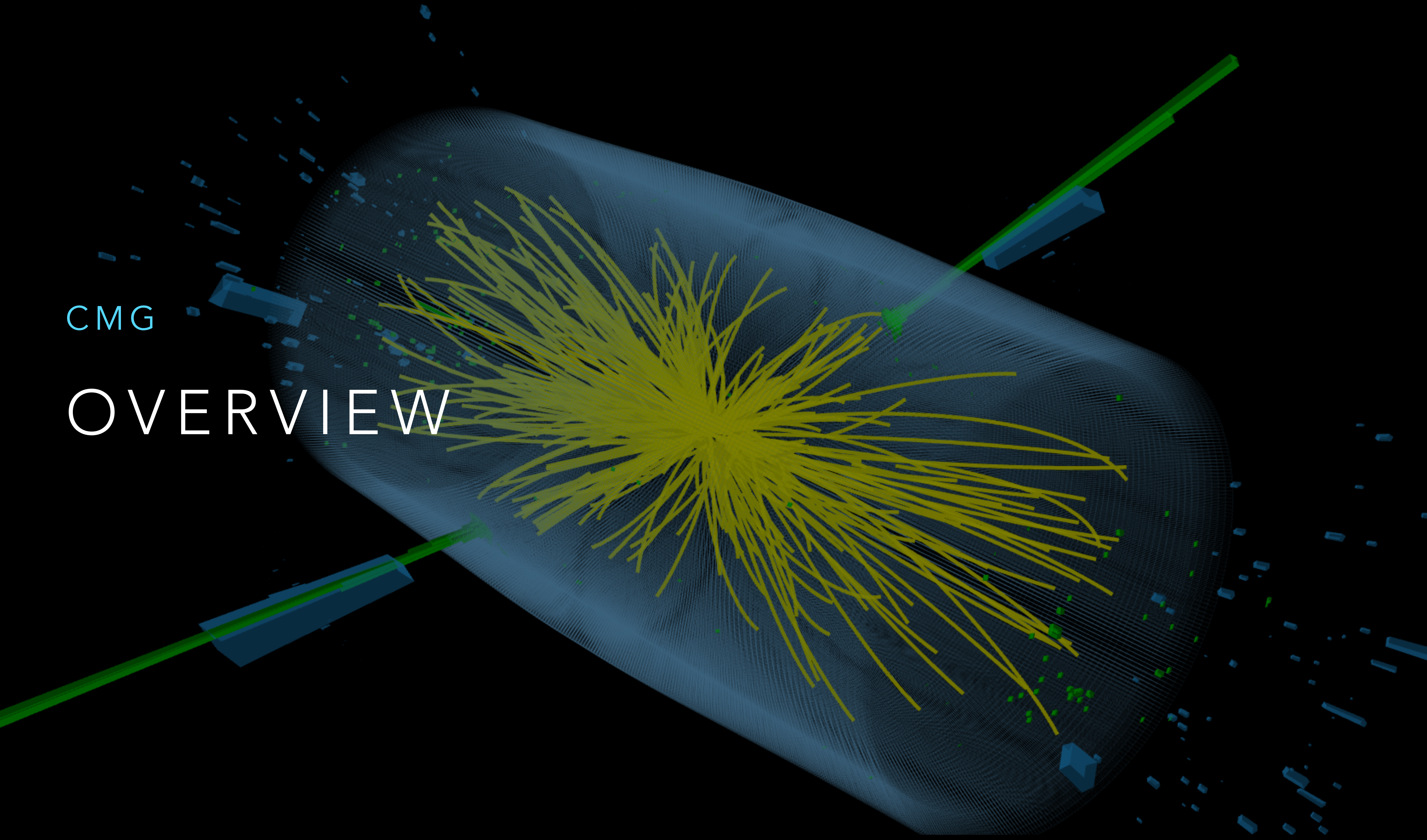
Caltech

OUTLINE

- Overview: ML and its applications
- Introduction to Artificial Neural Networks
  - Supervised learning
  - Neural networks
  - (Stochastic) gradient descent
  - Backpropagation (chain rule)
  - Practicalities: overfitting, hyperparameter optimization
- Tools
  - ML: Keras/TensorFlow, **PyTorch**
  - CMS/HEP: rootpy, root_numpy
- Exercises

CMG

OVERVIEW



Caltech

"All the impressive achievements of deep learning amount to just curve fitting."

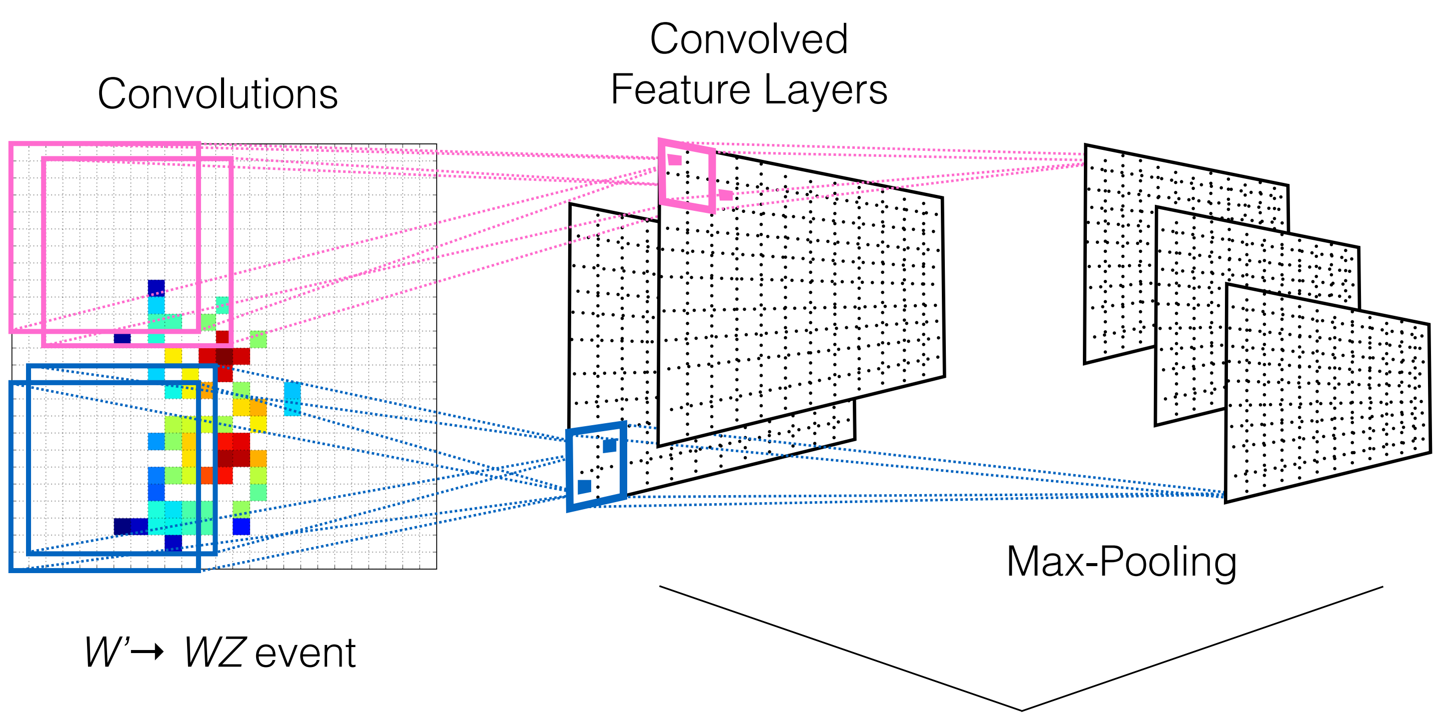
-Judea Pearl

WHAT IS MACHINE LEARNING?

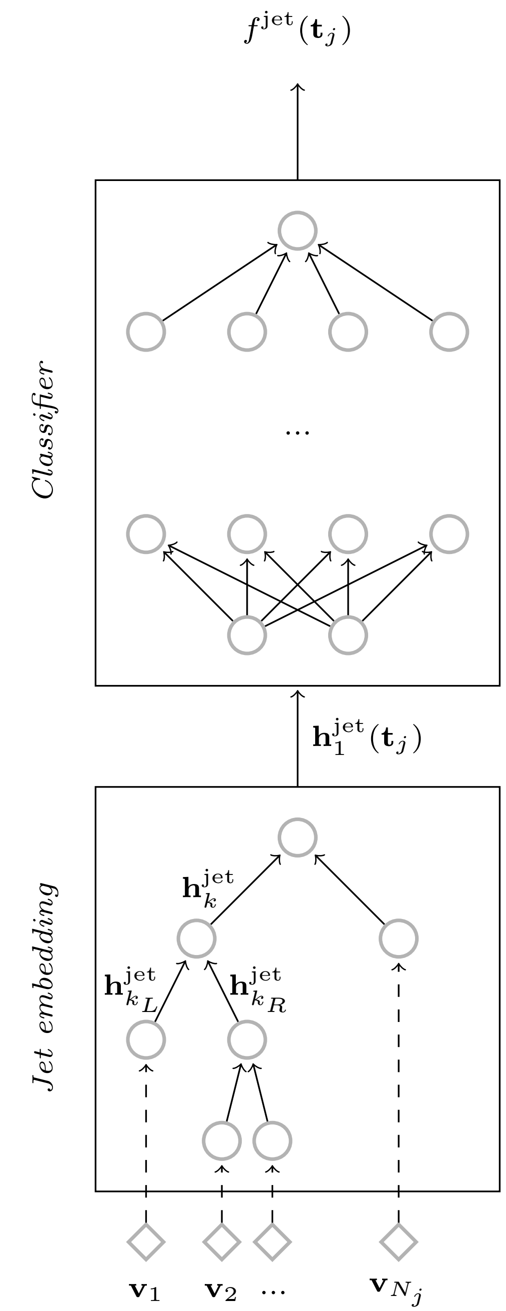
- Learning **mathematical models** from **data** that
  - characterize the **patterns**, regularities, and relationships amongst **variables** in the system
- Three key components:
  - **Model:** chosen mathematical model (depends on the task / available data)
  - **Learning**: estimate statistical model from data
  - **Prediction and Inference**: using statistical model to make predictions on new data points and infer properties of system(s)

MACHINE LEARNING APPS

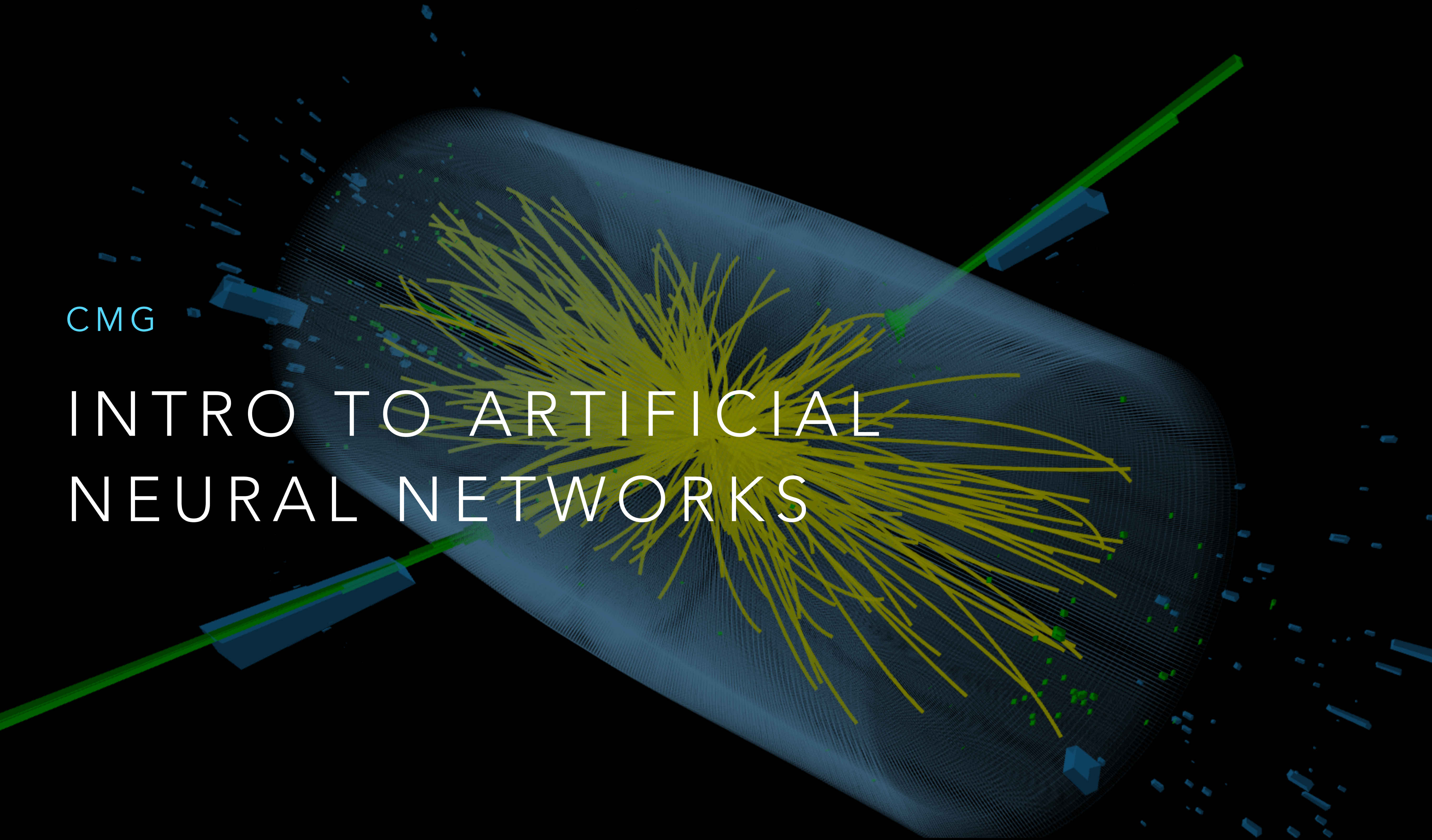
- Many applications in HEP:
  - Convolutional neural networks using an analogy between calorimeters and images
  - Recursive neural networks built upon an analogy between QCD and natural languages
  - …



[L. de Oliveira, et al. arXiv:1511.05190]



[G. Louppe, et al. arXiv:1702.00748]



CMG

INTRO TO ARTIFICIAL NEURAL NETWORKS

Caltech

TYPES OF LEARNING

- Unsupervised Learning
  - Clustering
  - Dimensional reduction
  - …
- **Supervised Learning**
  - Classification
  - Regression

SUPERVISED LEARNING

- Given N examples with features $\{x_i \in \mathcal{X}\}$ and **targets** $\{y_i \in \mathcal{Y}\}$, learn function mapping **h(x)=y**
  - **Classification**: $\mathcal{Y}$ is a finite set of **labels** (i.e. classes)

$\mathcal{Y} = \{0, 1\}$ for **binary classification**,
encoding classes, e.g. Higgs vs Background

$\mathcal{Y} = \{c_1, c_2, \ldots c_n\}$ for **multi-class classification**

represent with "**one-hot-vector**"

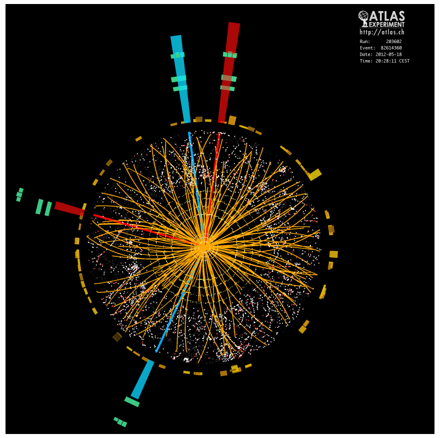
$\rightarrow y_i = (0, 0, \ldots, 1, \ldots 0)$

were k^{th} element is 1 and all others zero for class c_k

SUPERVISED LEARNING

- Given N examples with features $\{\mathbf{x}_i \in \mathcal{X}\}$ and **targets** $\{y_i \in \mathcal{Y}\}$, learn function mapping **h(x)=y**
  - **Classification:** $\mathcal{Y}$ is a finite set of **labels** (i.e. classes)
  - **Regression:** $\mathcal{Y}$ = Real Numbers
    - Example: jet mass, b-tag score

SUPERVISED LEARNING



h(**x**; **w**)
Function with adjustable parameters

Loss Function

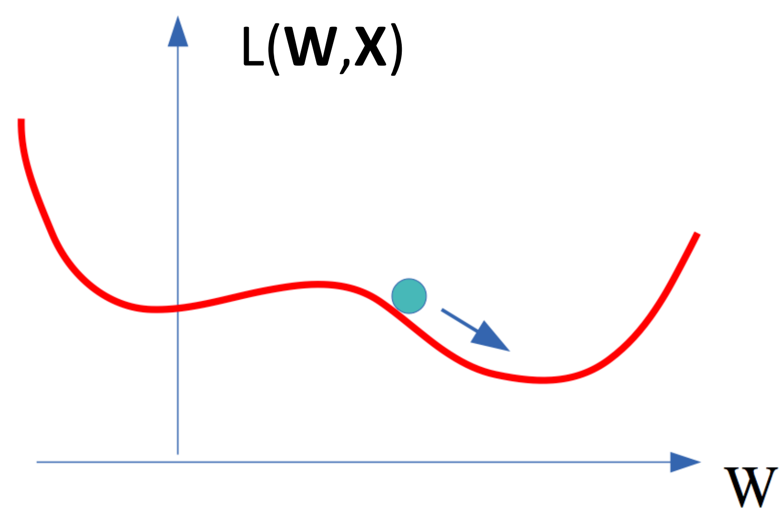
Compare prediction with true label

Loss

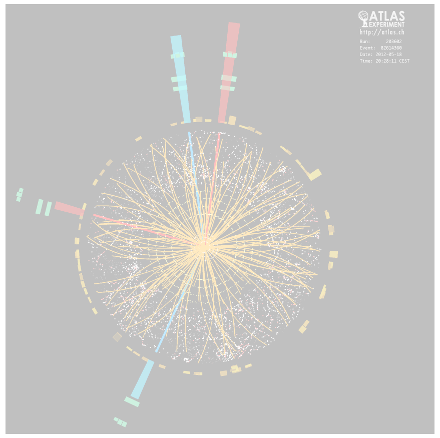
True labels:
Higgs = 1
Bkg = 0

- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize loss
  - Use a labeled *training-set* to compute loss
  - Adjust parameters to reduce loss function
  - Repeat until parameters stabilize
- Estimate final performance on *test-set*

Y. Le Cun



SUPERVISED LEARNING



h(**x**; **w**)
Function with adjustable parameters

Loss Function

Compare prediction with true label

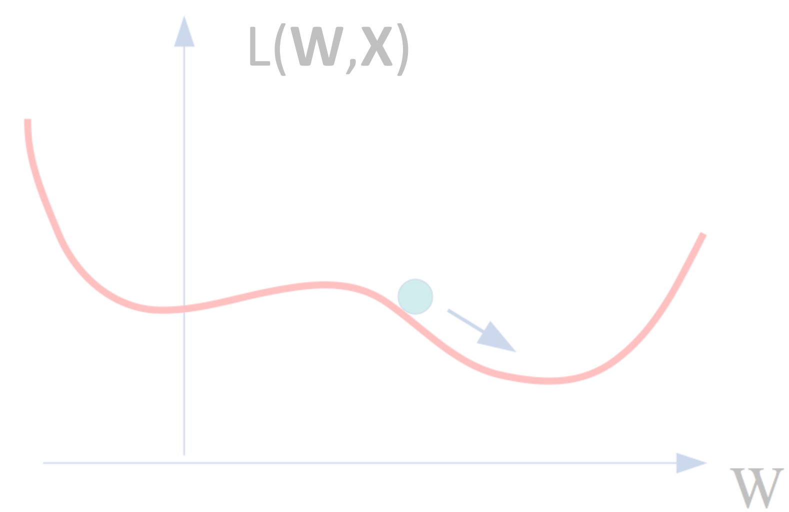
Loss

True labels:
Higgs = 1
Bkg = 0

- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize loss
  - Use a labeled *training-set* to compute loss
  - Adjust parameters to reduce loss function
  - Repeat until parameters stabilize
- Estimate final performance on *test-set*

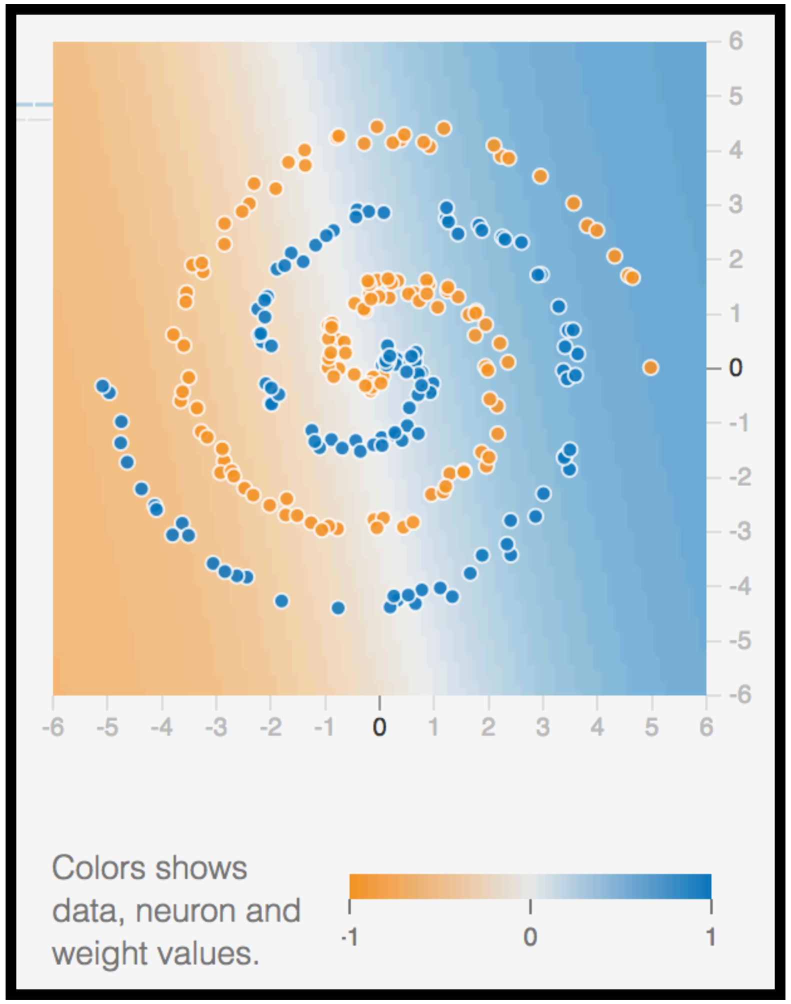
Y. Le Cun

A neural network!



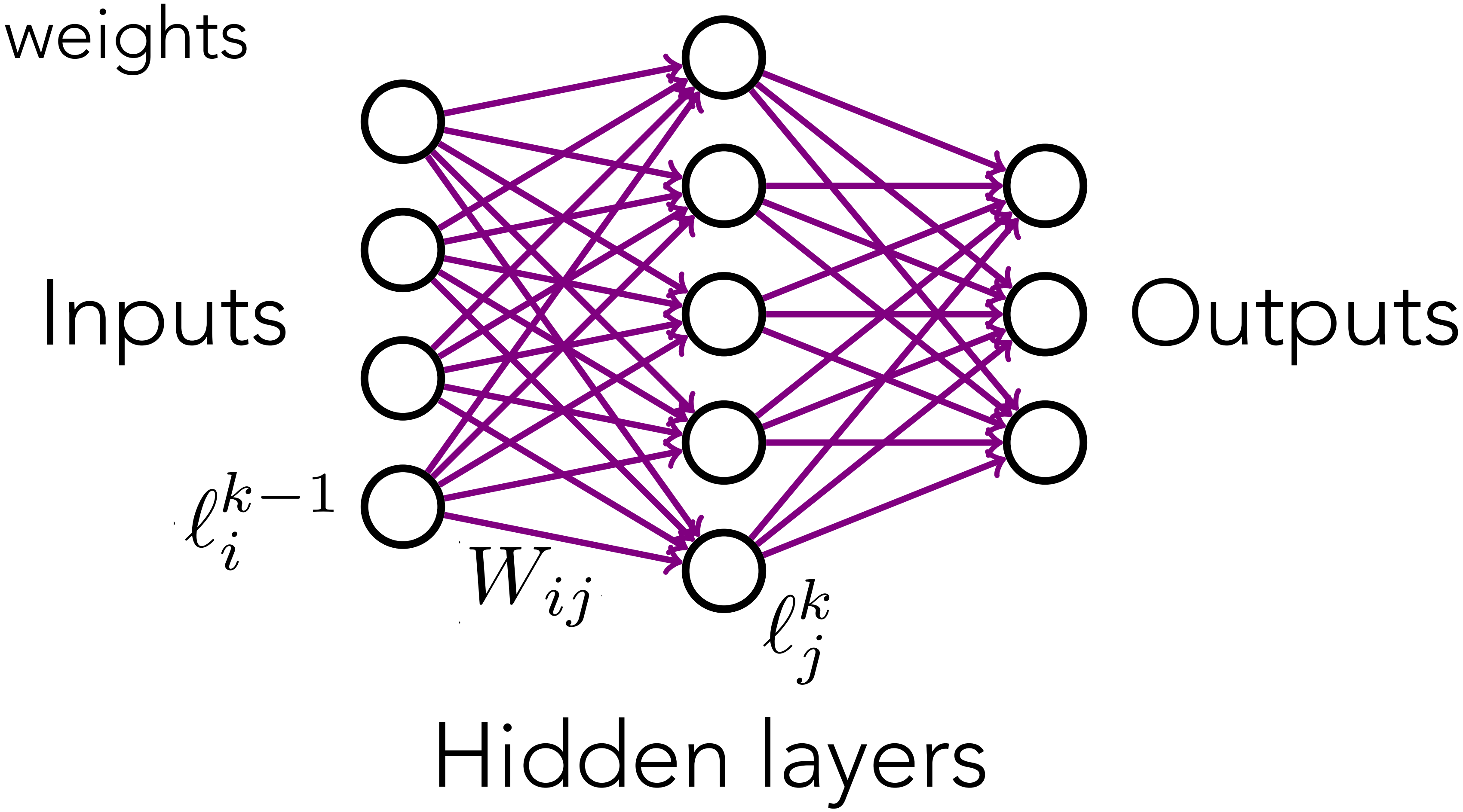
NEURAL NETWORK

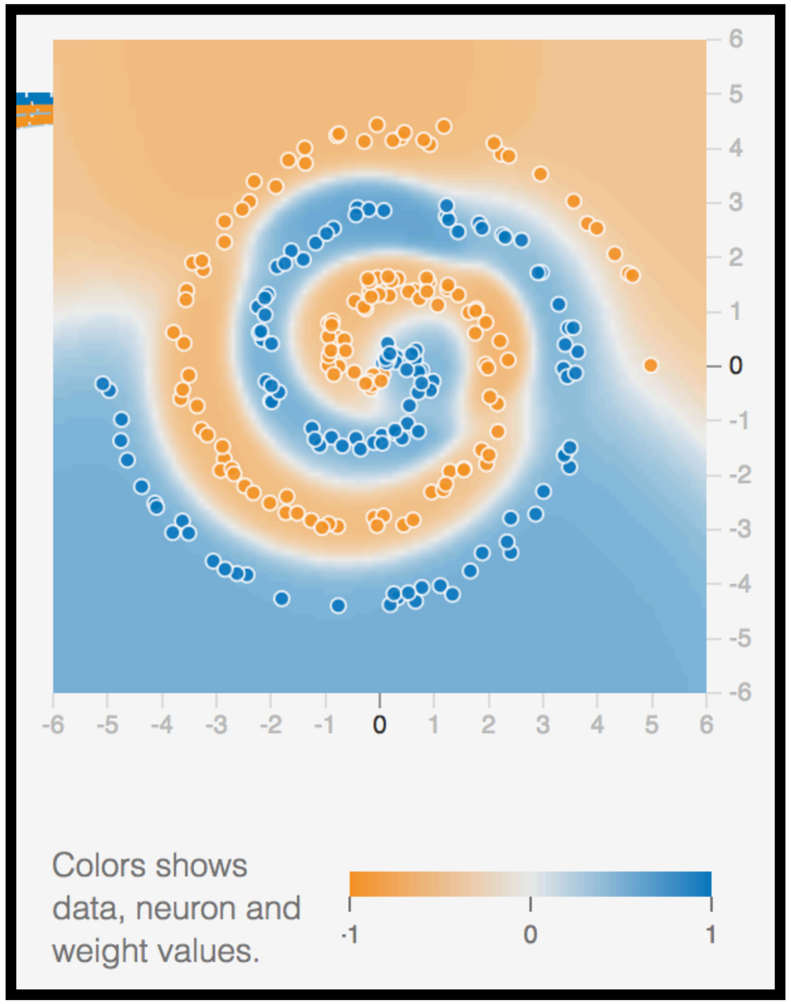
- Universal approximation theorem:
    - Simple neural networks can represent a wide variety of complicated functions.
    - Neural network layer: an MxN matrix taking an input vector of length M outputs a vector of length N.



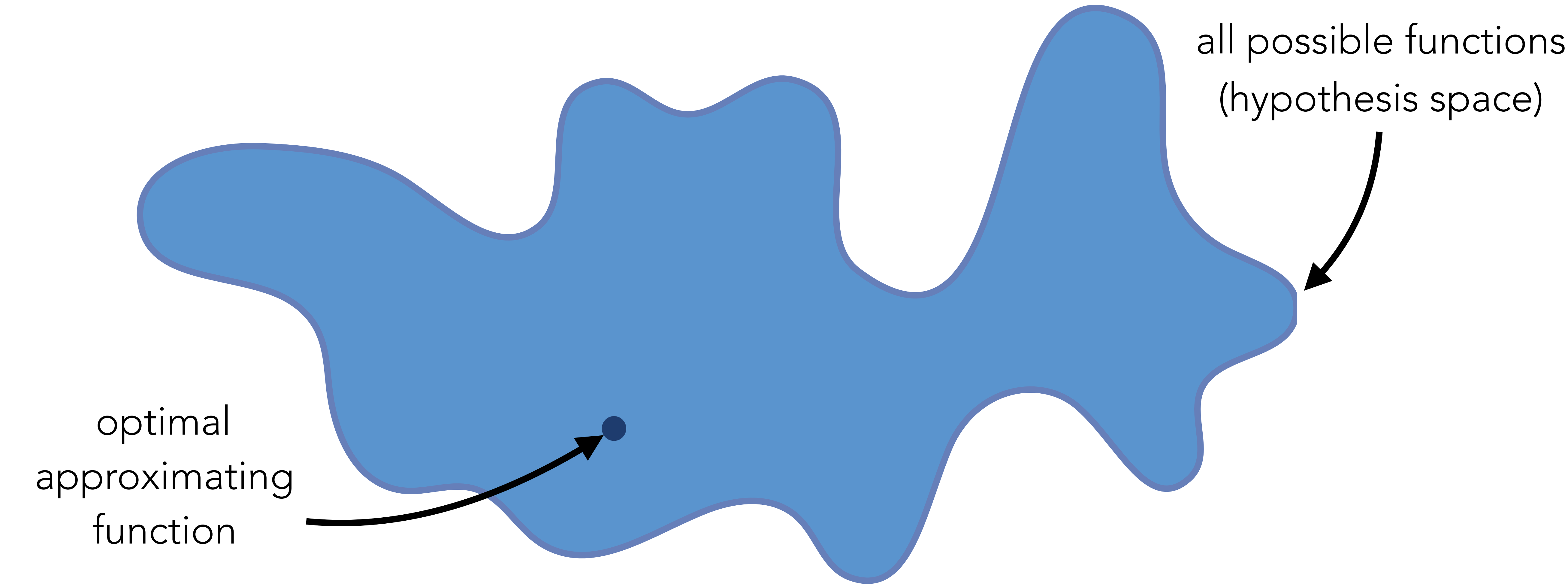
NEURAL NETWORK

- **Multiple layers**: output of previous layer is ***fed forward*** to next layer after applying ***non-linear*** activation function $\boxed{\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)}$
- **Fully connected**: many independent weights
- **Learning**: Use analytic derivatives and stochastic gradient descent to find optimal weights





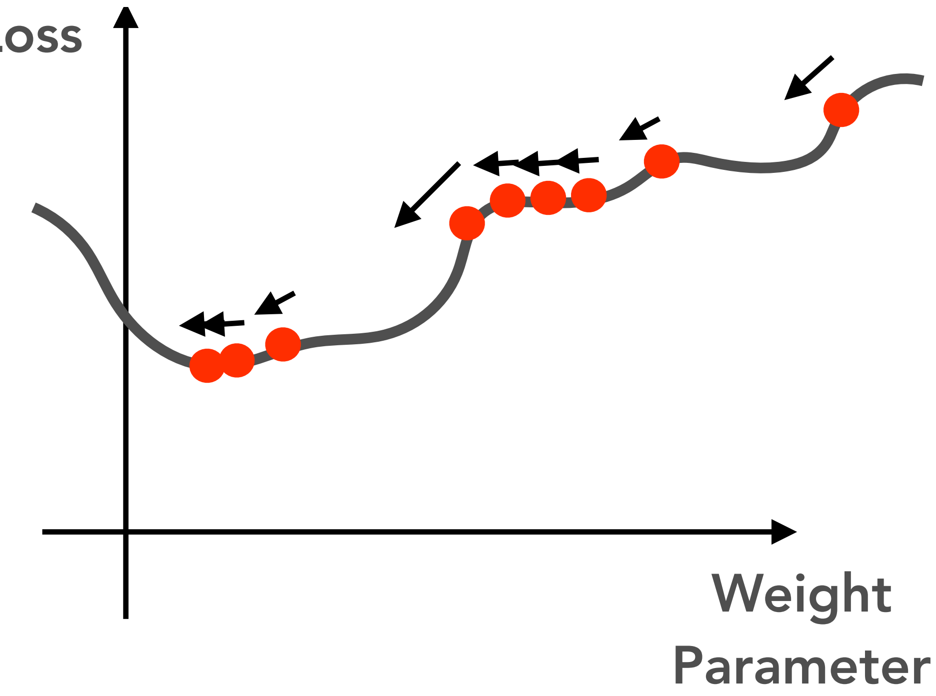
neural networks are universal function approximators,
but we still must find an optimal approximating function



we do so by <u>adjusting the weights</u>

LEARNING = OPTIMIZATION

learning as *optimization*



to learn the weights, we need the **derivative** of the loss w.r.t. the weight

i.e. *"how should the weight be updated to decrease the loss?"*

$$w = w - \alpha \frac{\partial \mathcal{L}}{\partial w}$$

with multiple weights, we need the **gradient** of the loss w.r.t. the weights

$$\mathbf{w} = \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}$$

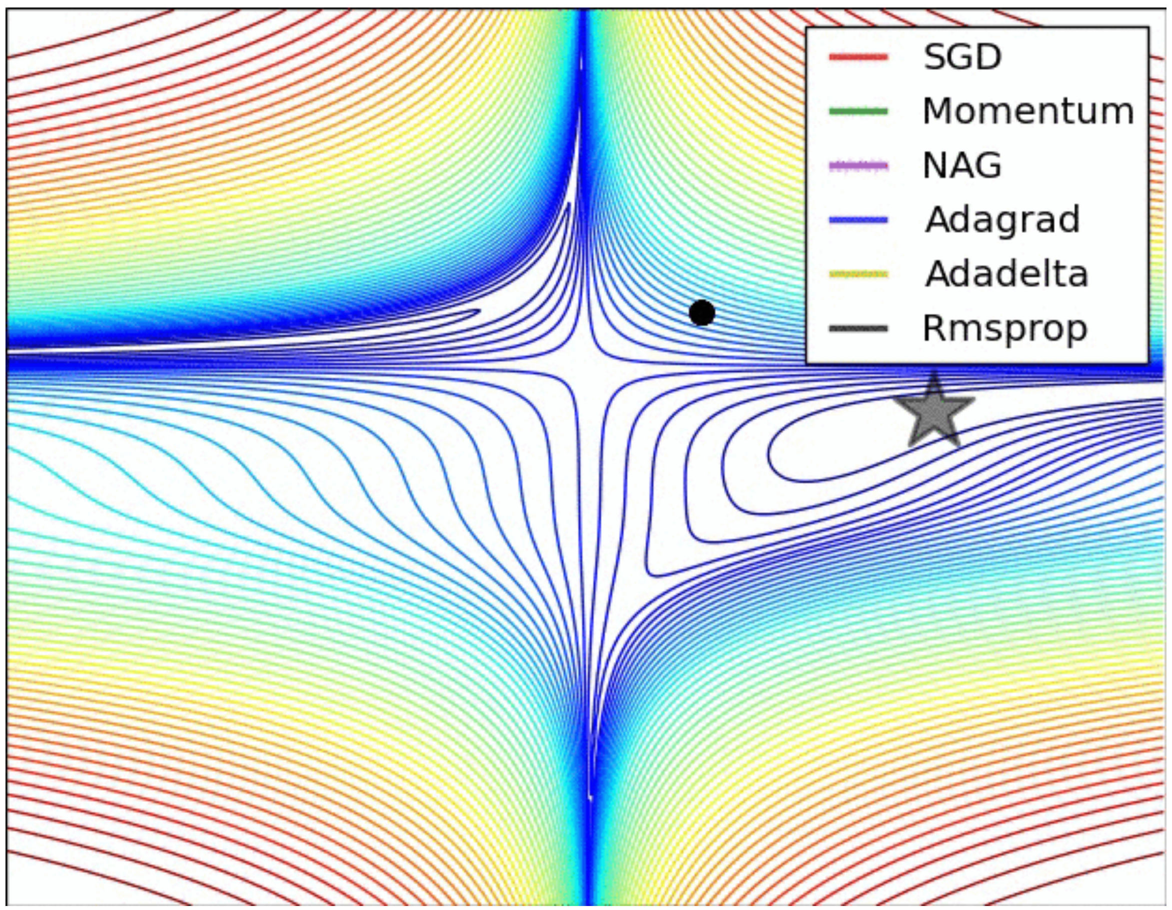
STOCHASTIC GRADIENT DESCENT

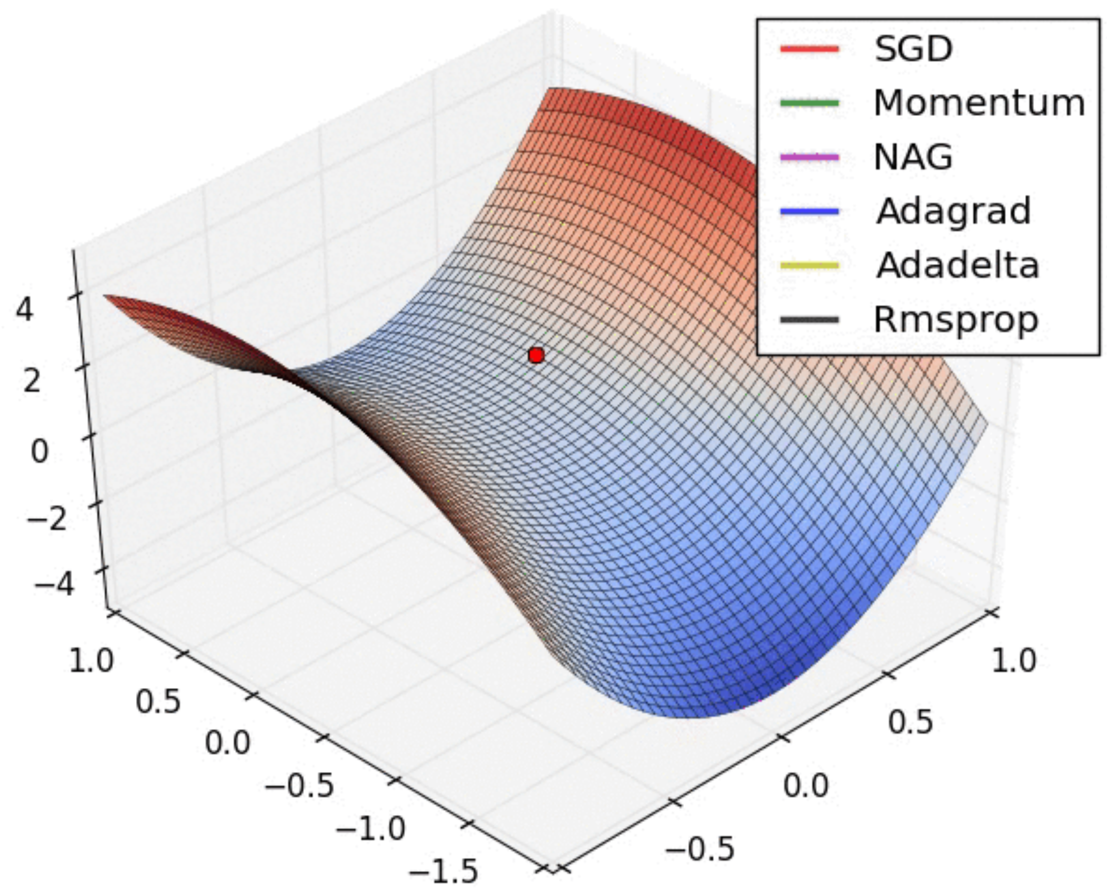
See animated gifs: http://ruder.io/optimizing-gradient-descent/

stochastic gradient descent (SGD): $w = w - \alpha \tilde{\nabla}_w \mathcal{L}$

use *stochastic gradient* estimate to *descend* the surface of the loss function

recent variants use additional terms to maintain "memory" of previous gradient information and scale gradients per parameter





local minima and saddle points are largely not an issue

in many dimensions, can move in exponentially more directions

BACKPROPAGATION

a neural network defines a function of composed operations

$$f_L(\mathbf{w}_L, f_{L-1}(\mathbf{w}_{L-1}, \ldots f_1(\mathbf{w}_1, \mathbf{x}) \ldots))$$

and the loss $\mathcal{L}$ is a function of the network output

⟶ use chain rule to calculate gradients

chain rule example

$$y = w_2 e^{w_1 x}$$

input x output y parameters w_1, w_2

evaluate parameter derivatives: $\frac{\partial y}{\partial w_1}, \frac{\partial y}{\partial w_2}$

define

$v \equiv e^{w_1 x} \longrightarrow y = w_2 v$

$u \equiv w_1 x \longrightarrow v = e^u$

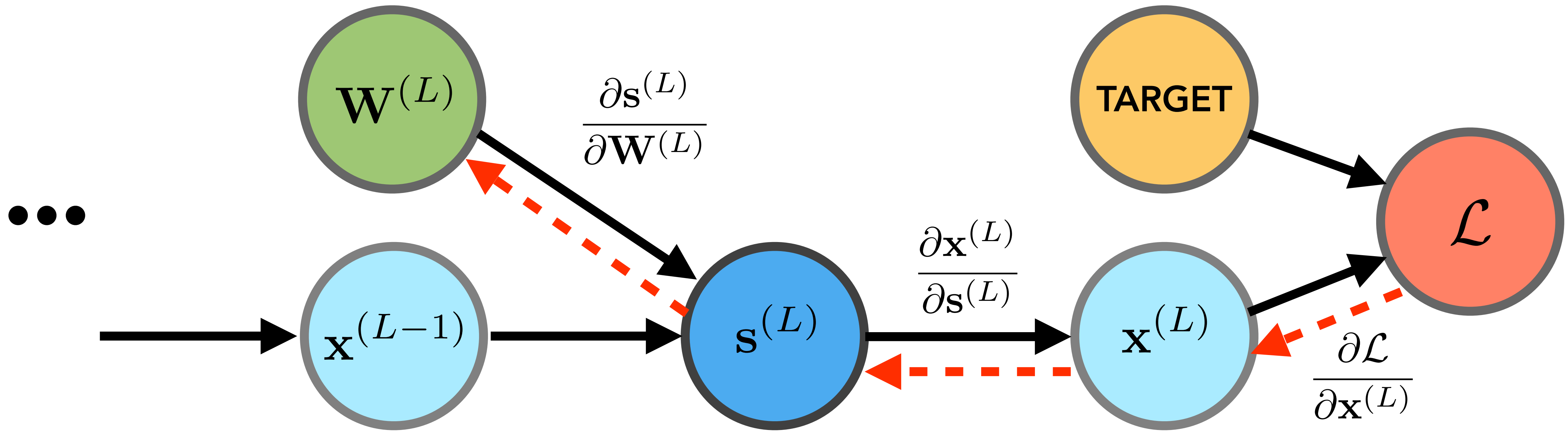
then

$$\frac{\partial y}{\partial w_2} = v = e^{w_1 x}$$

chain rule

$$\frac{\partial y}{\partial w_1} = \boxed{\frac{\partial y}{\partial v}\frac{\partial v}{\partial u}\frac{\partial u}{\partial w_1}} = w_2 \cdot e^{w_1 x} \cdot x$$

BACKPROPAGATION



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

depends on the form of the loss

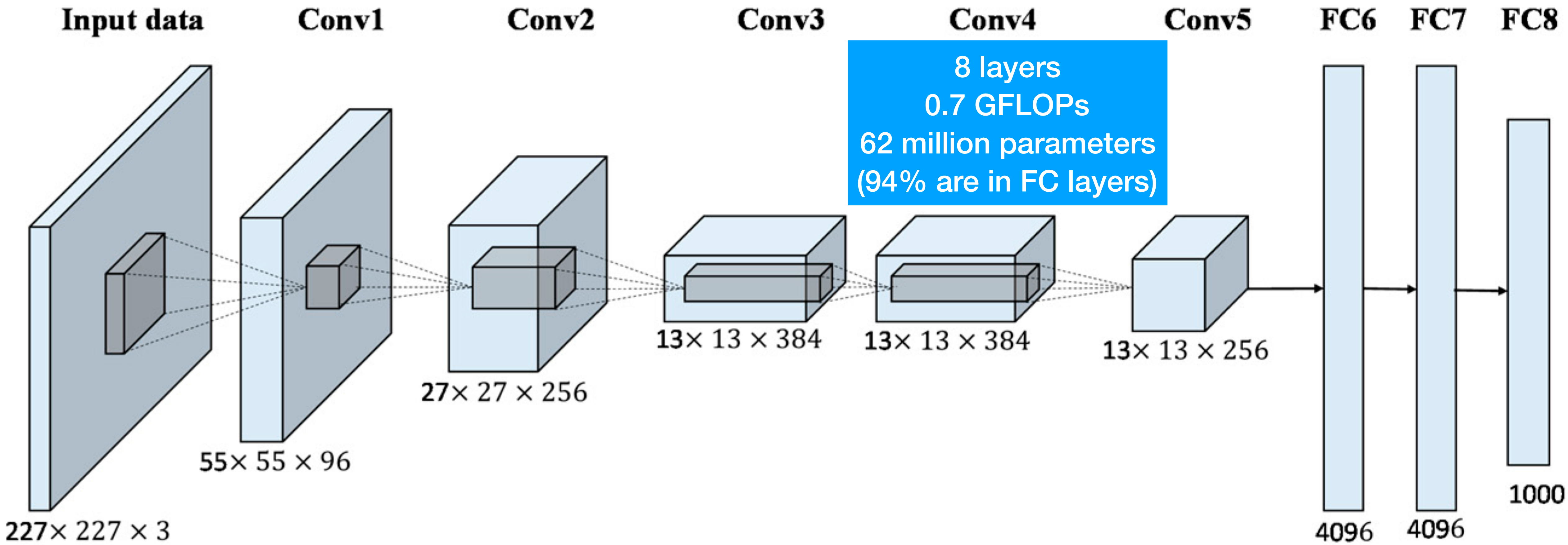
derivative of the non-linearity

$$\frac{\partial}{\partial \mathbf{W}^{(L)}} (\mathbf{W}^{(L)\intercal} \mathbf{x}^{(L-1)})$$
$$= \mathbf{x}^{(L-1)\intercal}$$

CONVOLUTIONAL NETWORKS

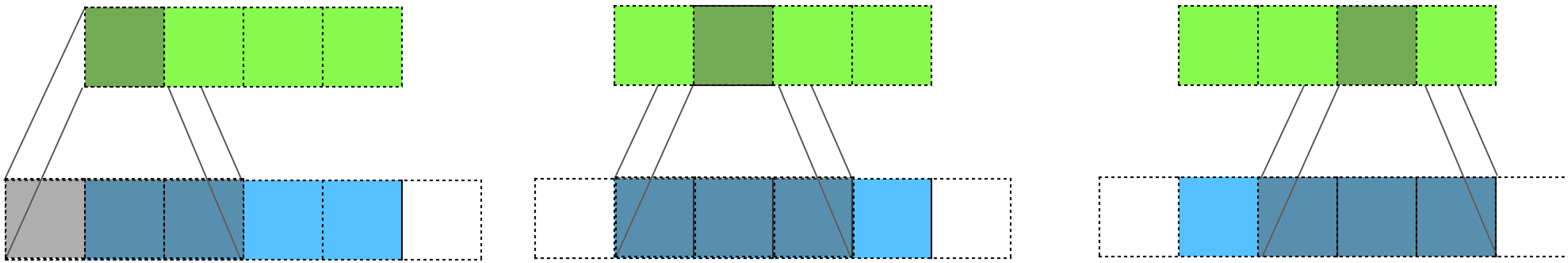
- Main task is computer vision/image recognition
- Control the number of parameters by baking in assumptions like locality and translation invariance to share weights within a layer

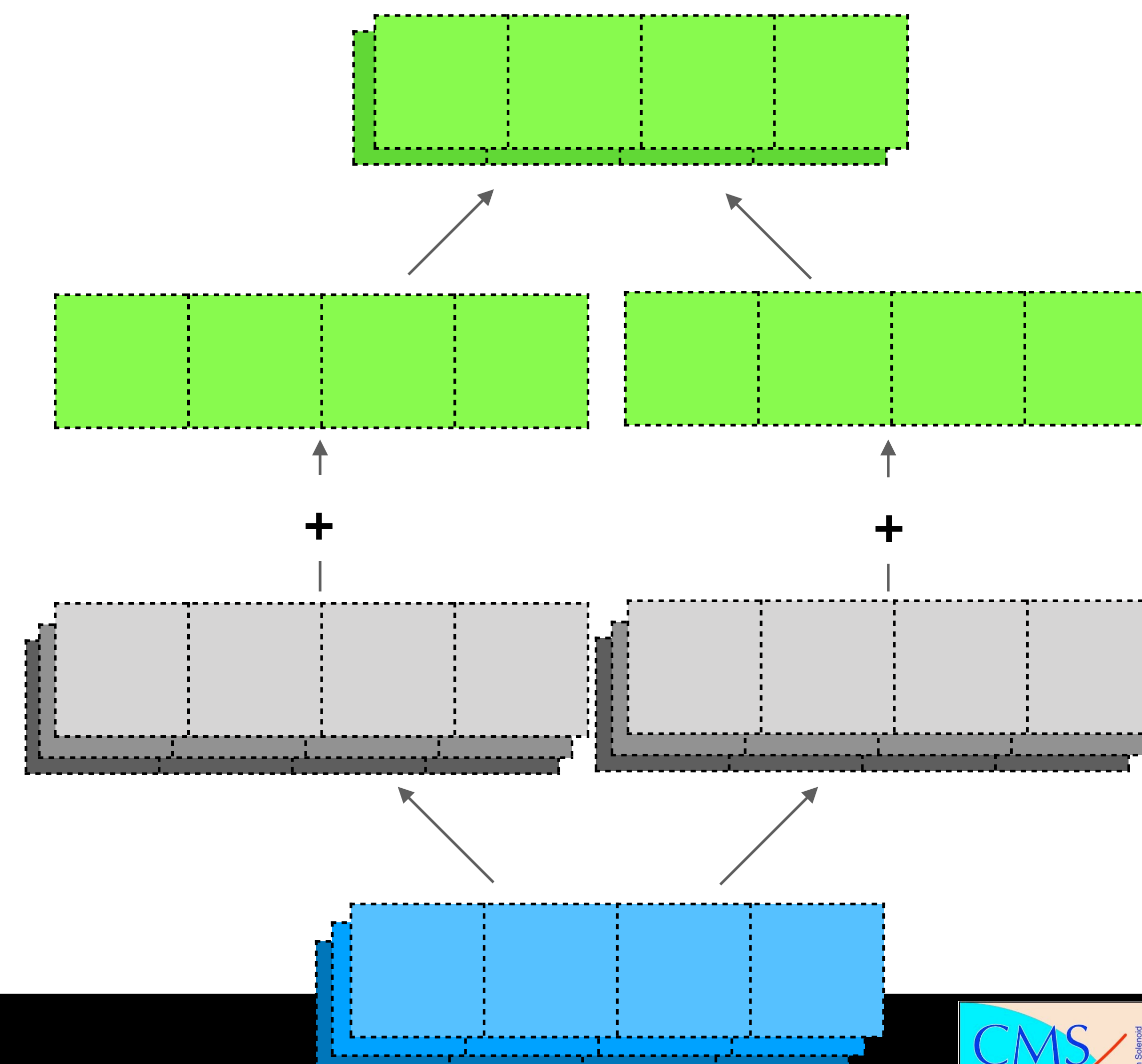
Krizhevsky, et al. [NIPS 4824]



1D CONVOLUTIONAL LAYER

Input
Filter
Output

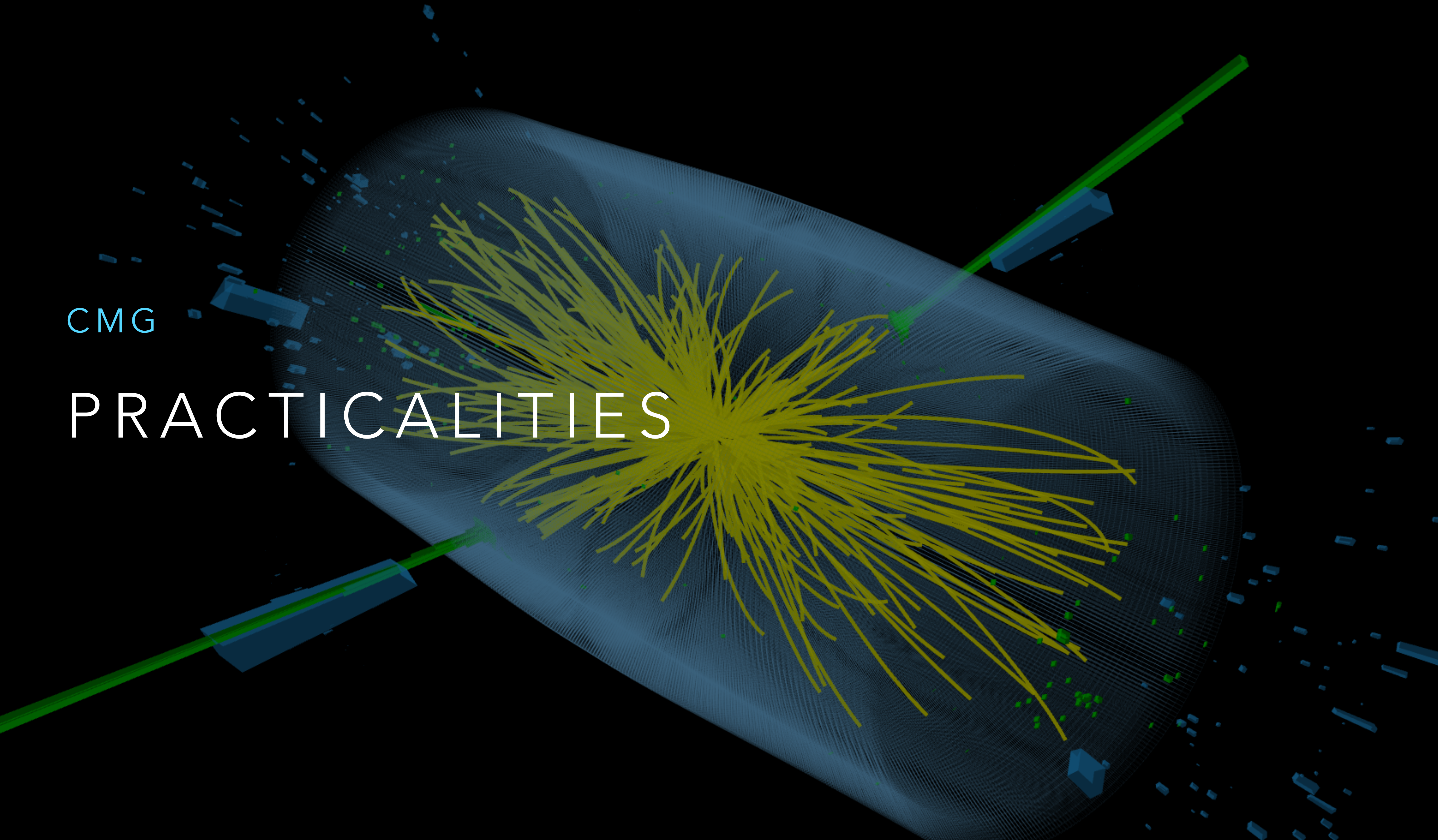




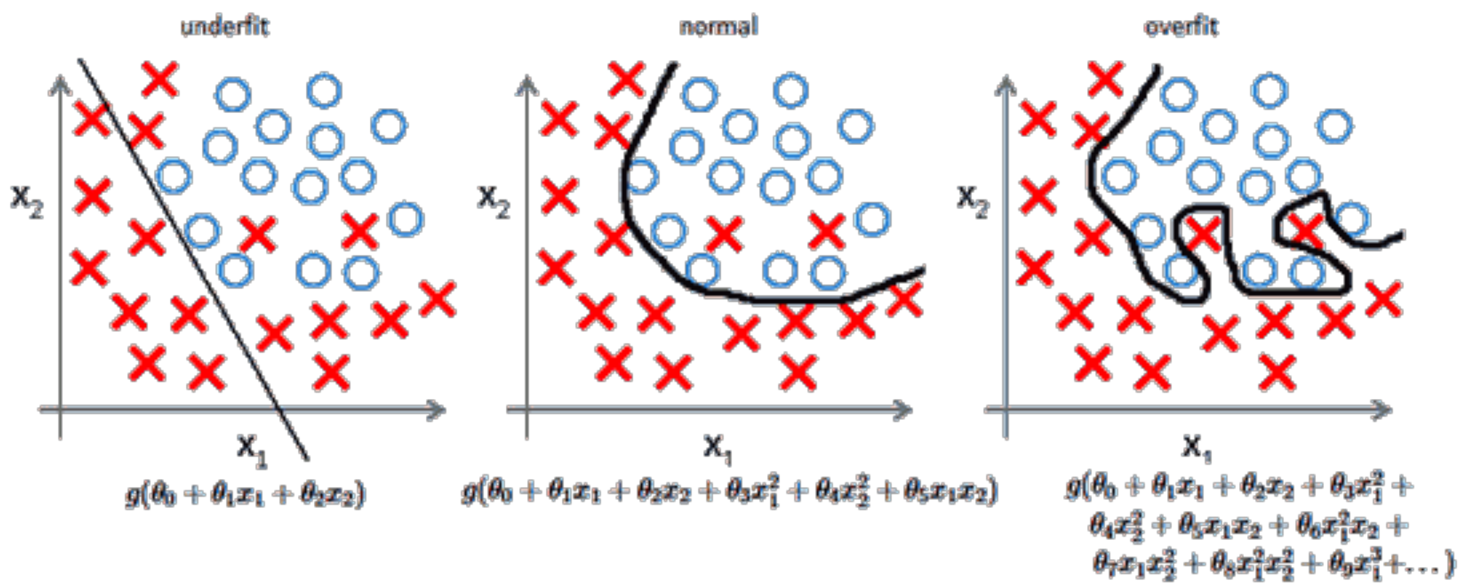
- Filter moves across input dimension
  - $c_0 = f_0{*}i_{-1} + f_1{*}i_0 + f_2{*}i_1$
- Example hyper-parameter settings:
  - Input size = 4
  - Number of channels = 3
  - Filter size = 3
  - “Same” / “Half” zero padding
  - Number of filters = 2
  - Output size = 4

CMG

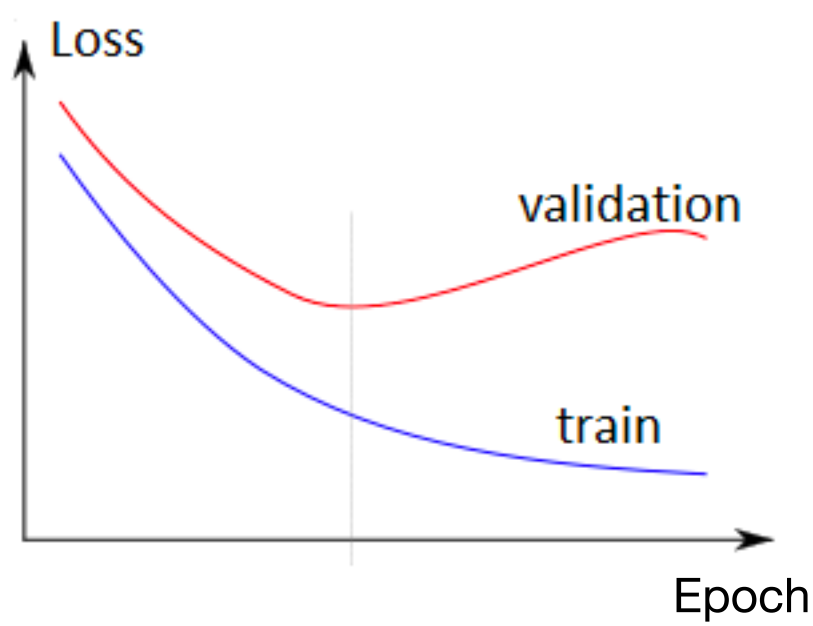
PRACTICALITIES



OVERFITTING

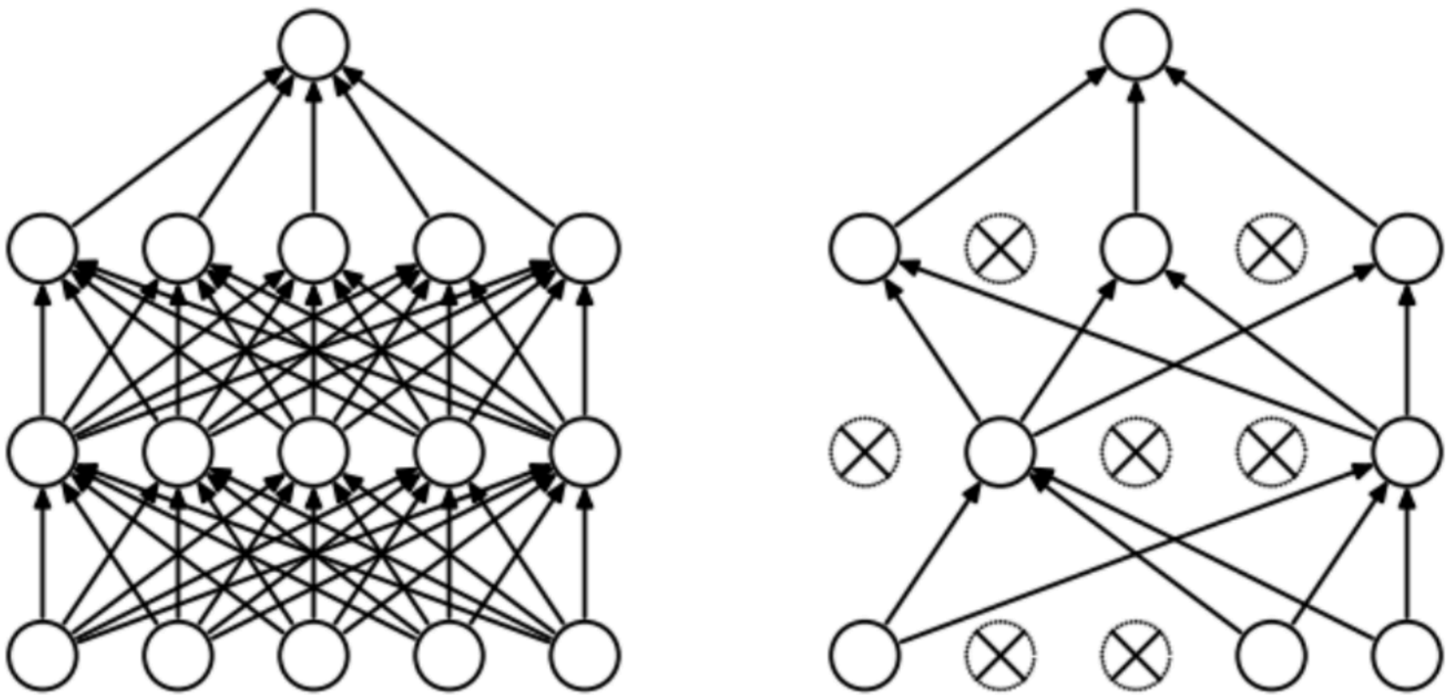


- Split data to training/validation/test sets:
  - After each *epoch* (one iteration of training on the whole dataset), validate the model on the validation set. Stop training early when overfitting appears.
  - Benchmark final model on the test set.



Caltech

DROPOUT



(a) Standard Neural Net

(b) After applying dropout.

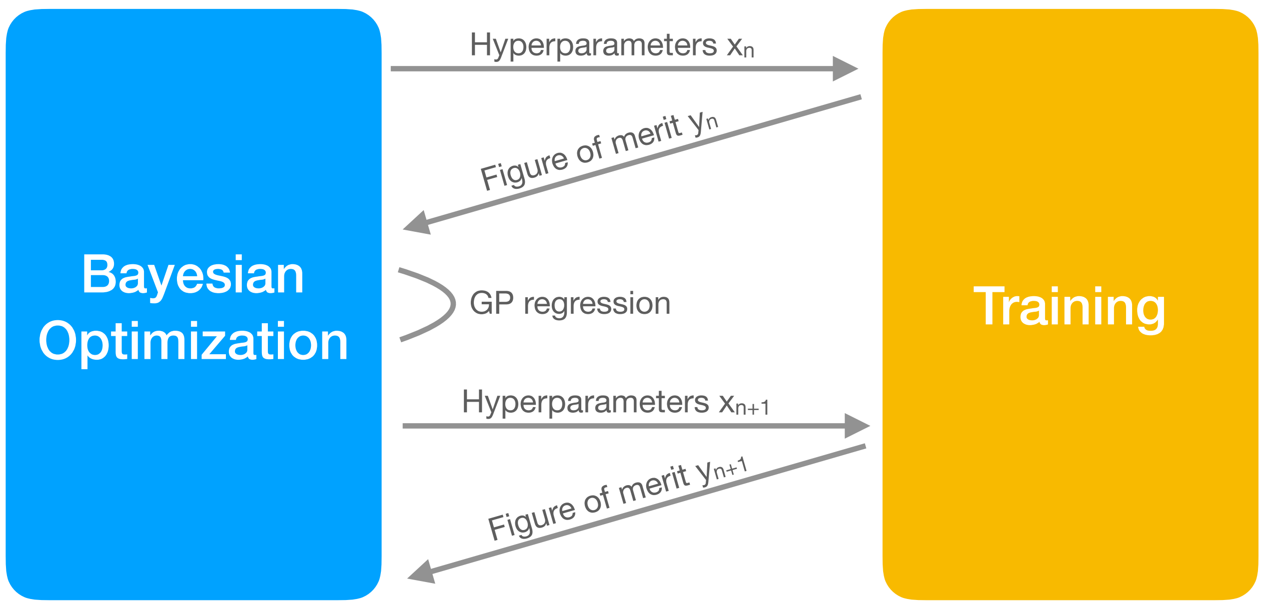
Srivastava et. al.

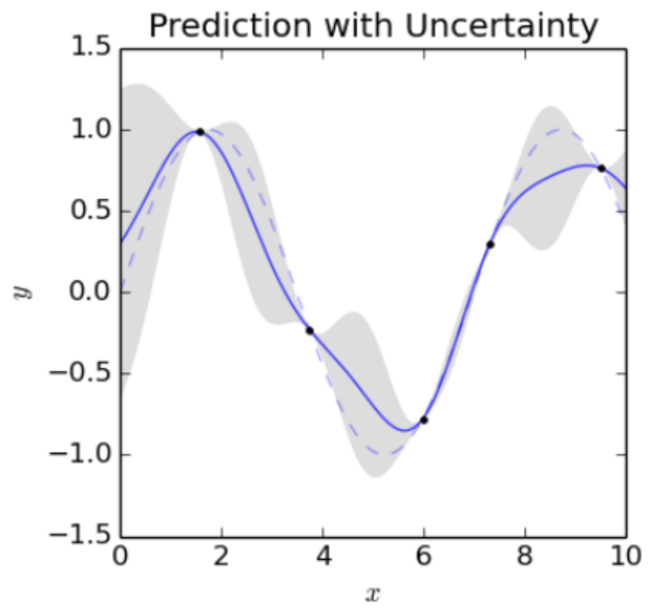
- Randomly remove connections between layers
- Effective against overfitting.

HYPERPARAMETER OPTIMIZATION

- Hyperparameters: Initial parameters to design the neural networks, not learnable via SGD.
  - Example: Number of hidden layers, number of neurons in each layer, learning rate, etc.
- Solutions: Random search, grid search, Bayesian optimization, evolutionary algorithm.
  - Minimize f(x) where x: set of hyperparameters, f(x): model performance given the set of hyperparameters.

BAYESIAN OPTIMIZATION

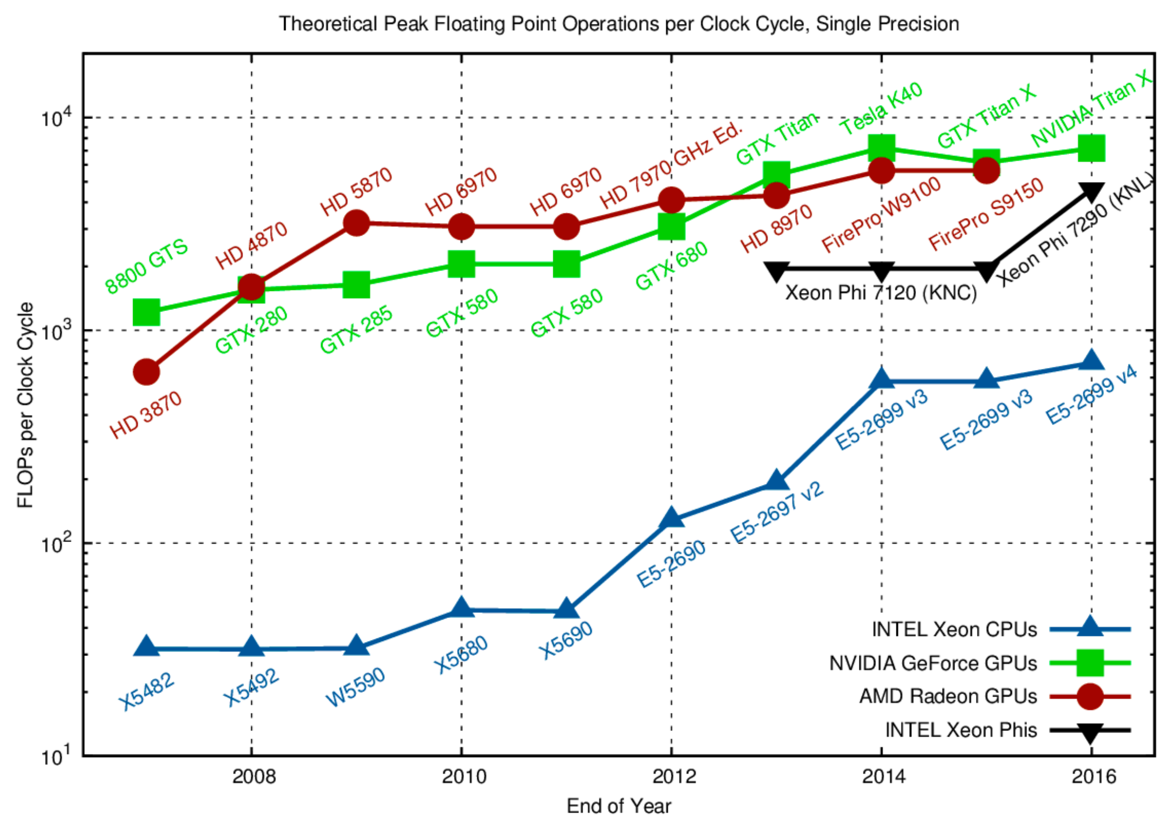




- Objective: Find the optimal point in hyperparameter space x that minimizes the objective function $y = f(x)$.
- Bayesian optimization: fit the distribution $\{y_n = f(x_n)\}_{n=1..N}$ with Gaussian process regression, predict the next value x_{N+1} that offers the best expected improvement on y.
  - x = set of hyperparameters
  - f(x) = final validation loss or negative validation accuracy of the model trained with given set of hyperparameters x.

GPU VS CPU

- GPUs: specialized hardware originally created to render games in high frame rates.
  - Graphics texturing and shading require a lot of matrix and vector operations executed in parallel.
- Deep learning also requires super fast matrix computations.



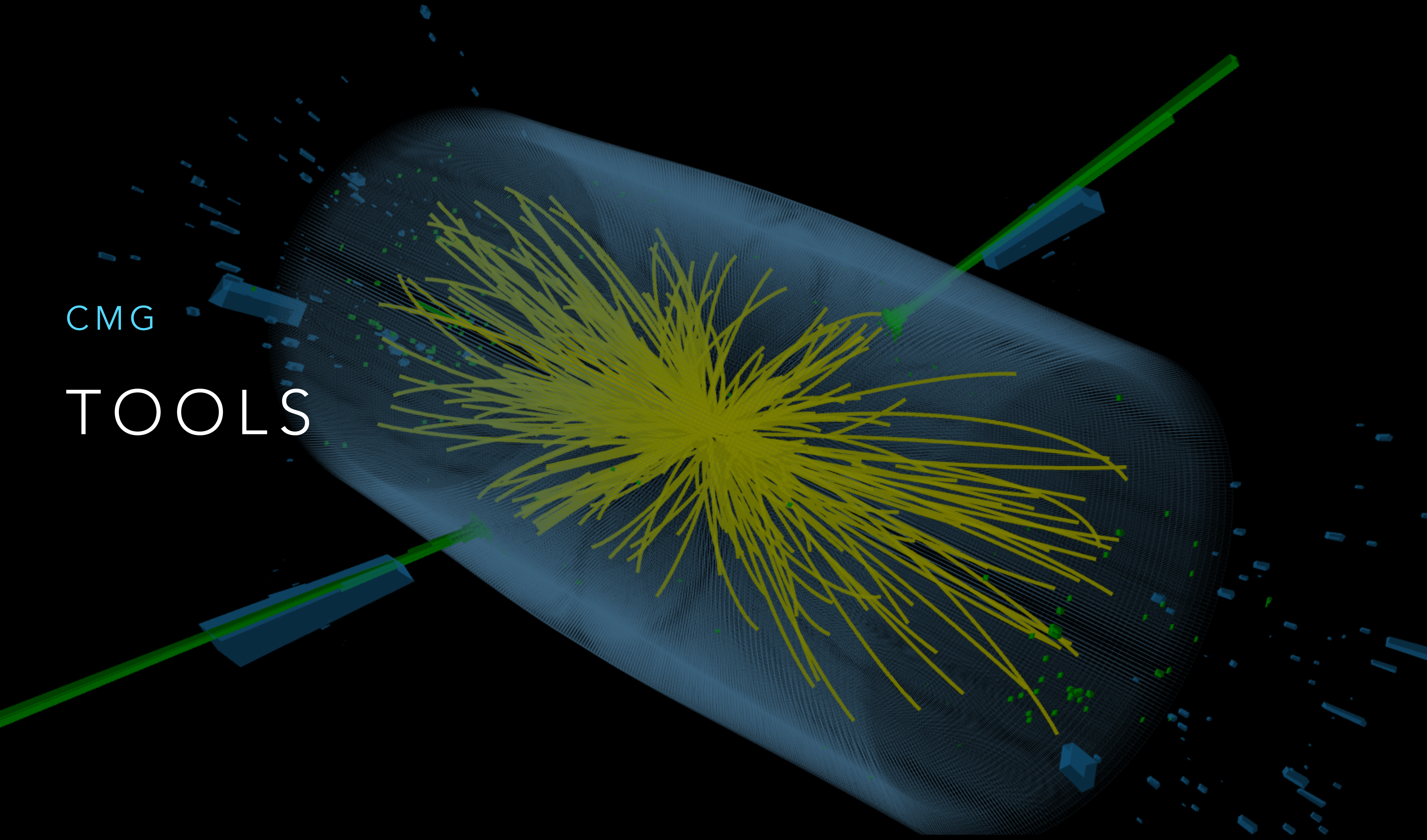
Large-scale Deep Unsupervised Learning using Graphics Processors

Rajat Raina RAJATR@CS.STANFORD.EDU
Anand Madhavan MANAND@STANFORD.EDU
Andrew Y. Ng ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford CA 94305 USA

IMCL 2009

CMG

TOOLS

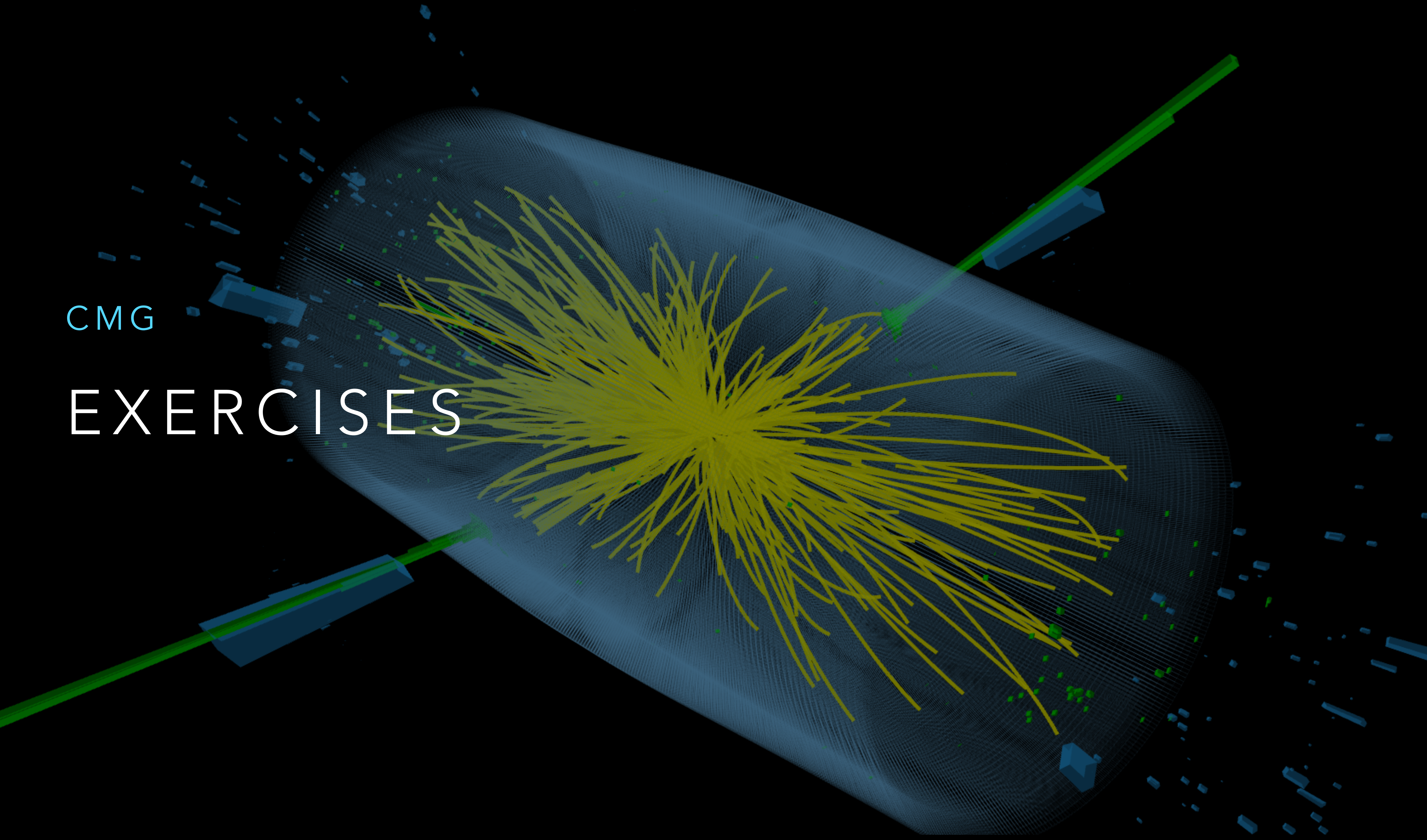


TOOLS

- Python
  - NumPy: http://www.numpy.org/
  - SciPy: https://www.scipy.org/
- Machine Learning
  - scikit-learn: http://scikit-learn.org/
  - Keras: https://keras.io/
  - PyTorch: https://pytorch.org/
- CMS/HEP
  - root_numpy: http://scikit-hep.org/root_numpy/
  - uproot: https://github.com/scikit-hep/uproot

CMG

EXERCISES



WHY PYTORCH?

- No pre-compilation, easy to print out and debug.
- Native support for multiple-GPUs.
- Maximum flexibility in prototype & implementation.

- https://github.com/thongonary/machine_learning_vbscan