

Physics based learning for forward and inverse problems

Amuthan Ramabathiran

Dept. of Aerospace Engineering, IIT Bombay.

E-mail: amuthan@aero.iitb.ac.in

Code: <https://github.com/Amuthan/AICTEworkshop2021>

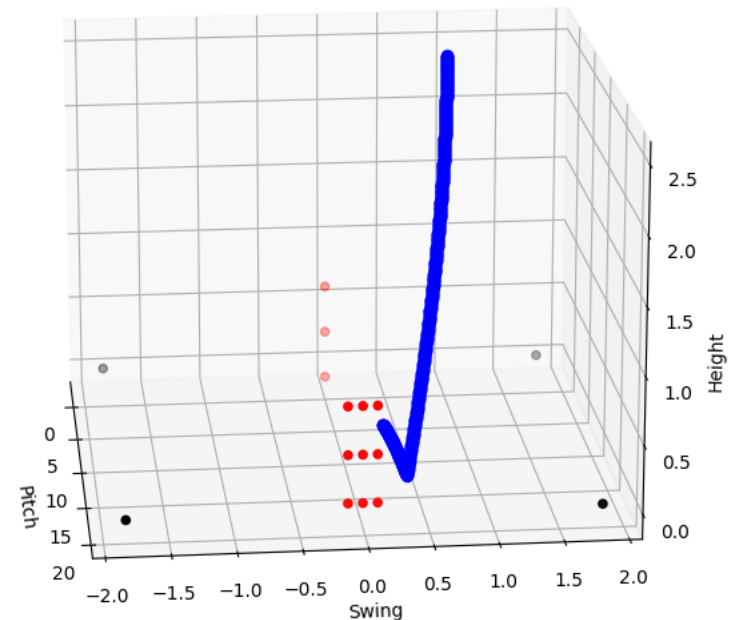
ATAL-AICTE Online Workshop (Dec 6-10, 2021)

Multi-Disciplinary Design Optimization in Engineering: Basics



Outline

- Differentiable physics
- Review of Optimization
- Automatic Differentiation
- Deep Neural Networks
- Application: ODEs
- **Hands-on session:**
 - Simple examples with PyTorch
 - *Competition:* Who is the best bowler?



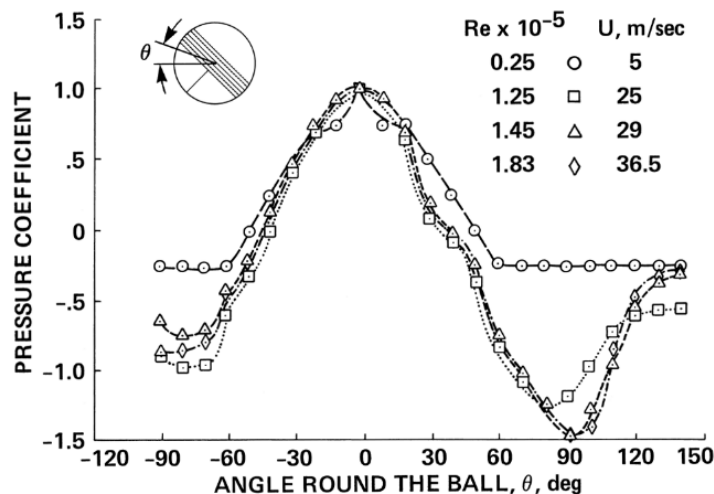
Working knowledge
of Python required!

Data based vs physics based modeling

How do we predict the swing of a cricket ball?

Data based approach:

Use data from experiments or numerical simulations directly



Physics based approach:

Use known physics about fluid flow around ball

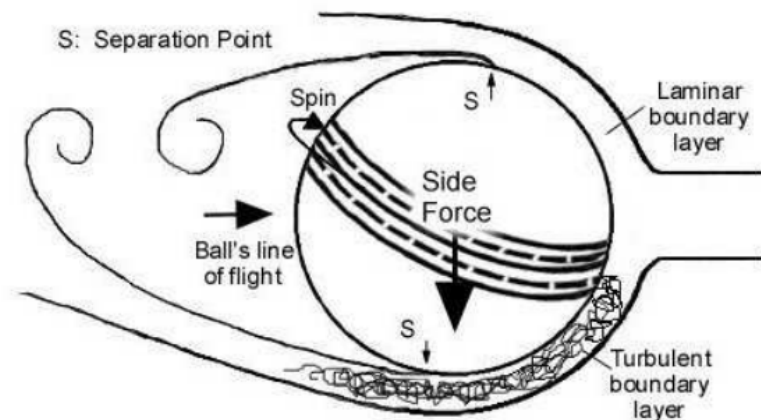
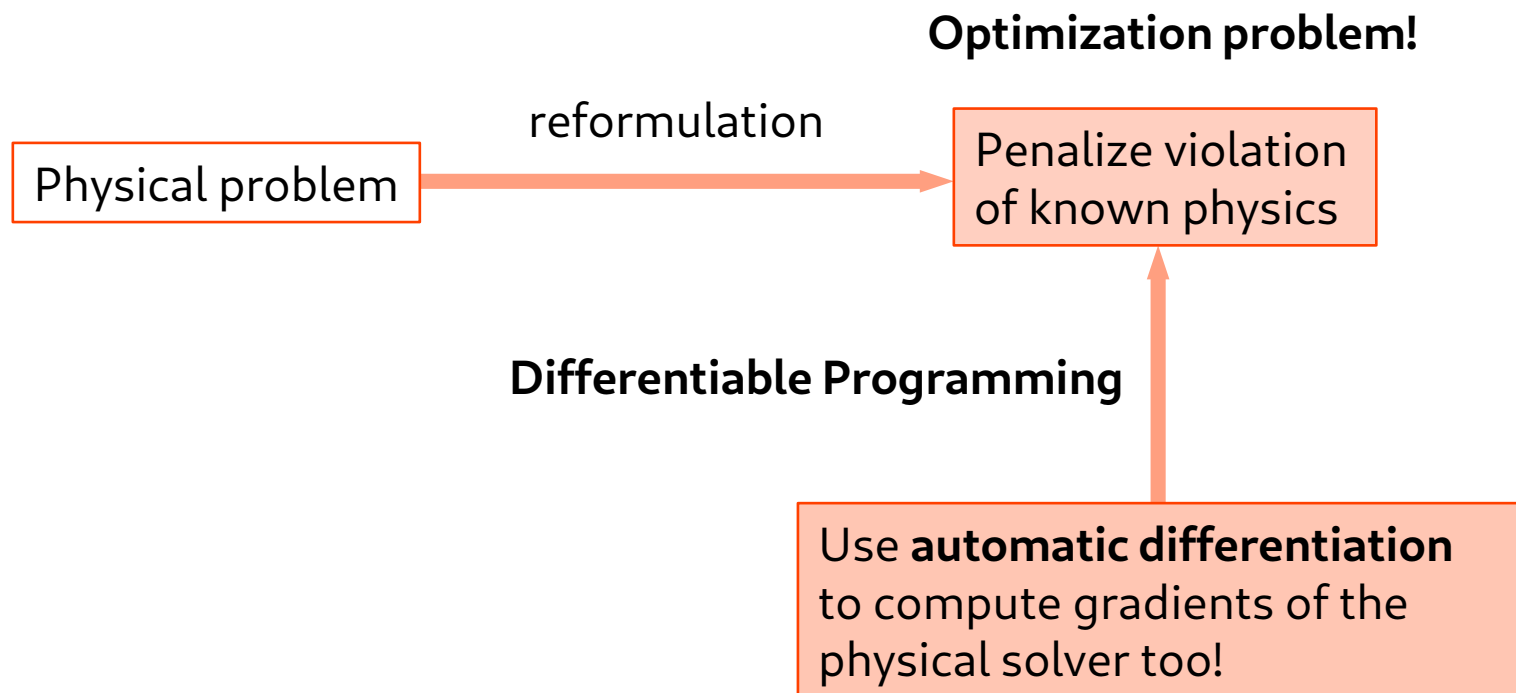


Image source: Mehta (2014): <https://people.eng.unimelb.edu.au/imarusic/proceedings/19/10.pdf> (+picture on title slide)

Differentiable physics

Differentiable physics enhances data based modeling with known physics prior – **best of both worlds!**



Review of finite dimensional optimization

Unconstrained minimization problem: given **objective function** $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find

$$x^* \in \operatorname{argmin}_{x \in \mathbb{R}^n} f(x).$$

First order necessary condition for optimality:

$$x^* \text{ is optimal} \Rightarrow \nabla f(x^*) = 0.$$

Second order sufficient condition for minimizer:

$$\nabla^2 f(x^*) \succ 0 \Rightarrow x^* \text{ is a local minimizer of } f.$$

Gradient descent algorithm

Iterative algorithms construct a **minimizing sequence** (x_1, x_2, \dots) that converge to the true minimizer:

$$\lim_{n \rightarrow \infty} x_n = x^* \in \operatorname{argmin}_{x \in \mathbb{R}^n} f(x).$$

Gradient descent algorithm creates a minimizing sequence by stepping in the direction of the negative gradient of the objective function:

$$x_{n+1} = x_n - \alpha_n \nabla f(x_n).$$

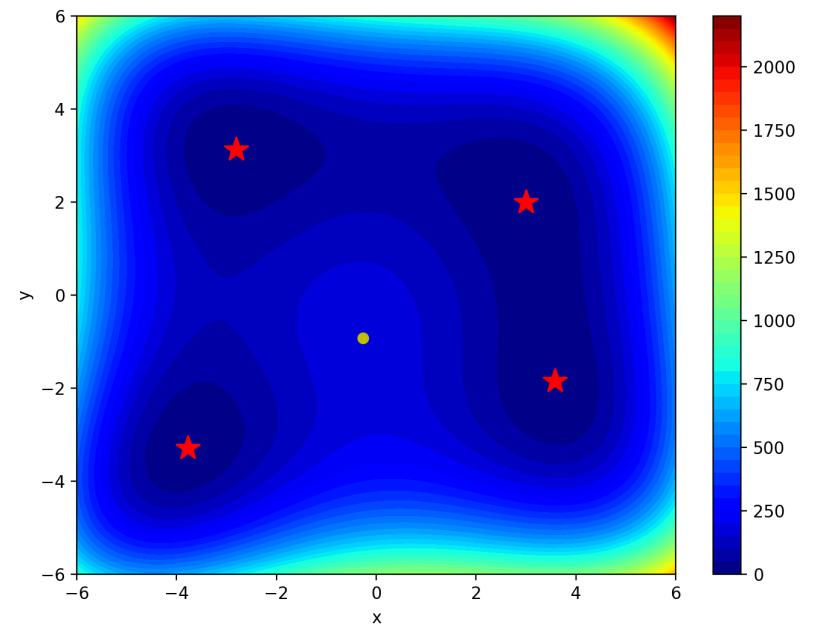
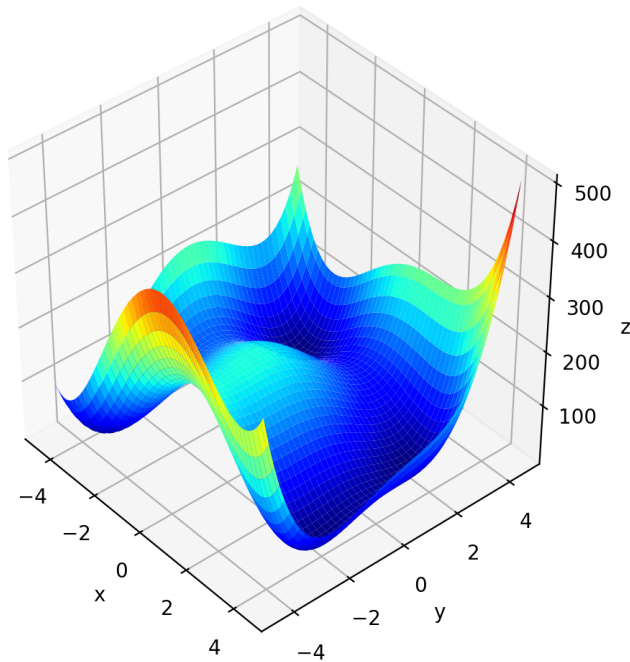
α_n : step size / **learning rate**

Gradient descent algorithm is **guaranteed to converge** to a local minimizer with appropriate choice of step size!

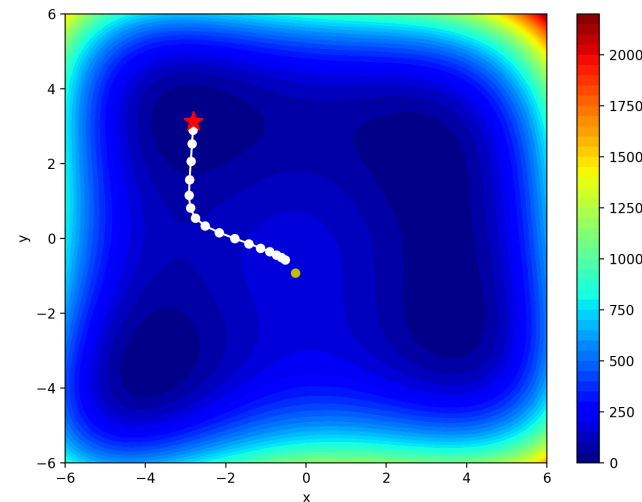
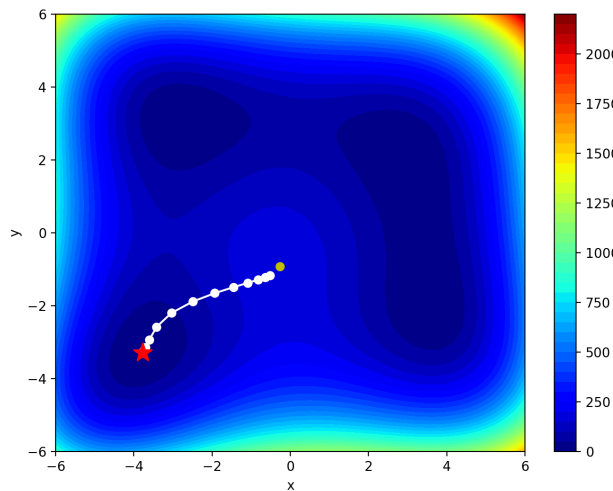
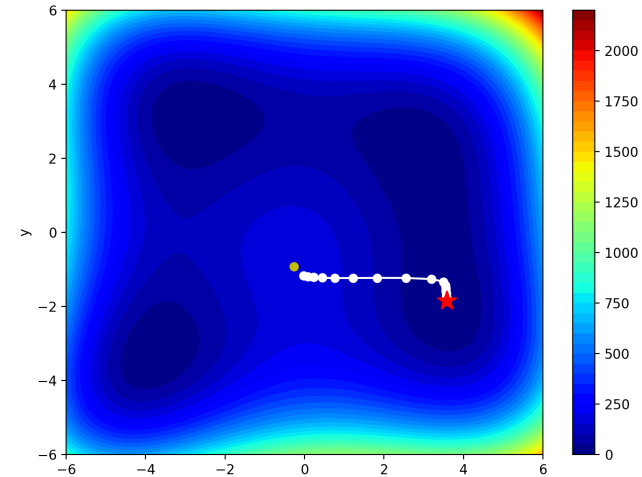
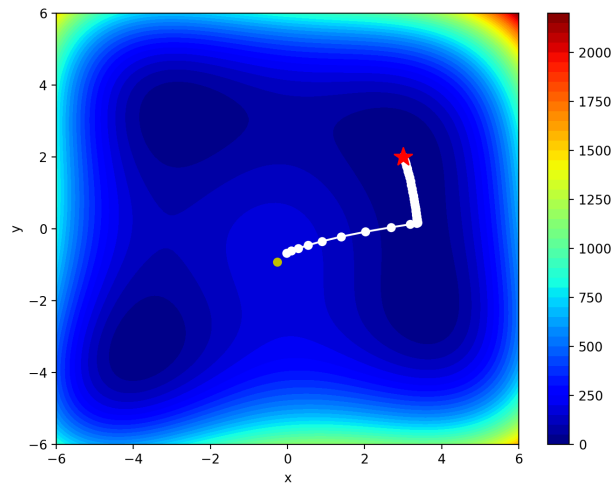
Example: Himmelblau function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

(One local maximum and four global minima)



Gradient descent for Himmelblau function



Equality constrained optimization: Lagrange multipliers

Given objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and functions $g_k : \mathbb{R}^n \rightarrow \mathbb{R}$, $1 \leq k \leq m (< n)$, find

$$x^* \in \operatorname{argmin}_{x \in K} f(x),$$
$$K = \{x \in \mathbb{R}^n \mid g_k(x) = 0, 1 \leq k \leq m\}.$$

Construct the **Lagrangian** $L : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$, with **Lagrange multipliers** $\lambda = (\lambda_1, \dots, \lambda_m)$

$$L(x, \lambda) = f(x) + \sum_{k=1}^m \lambda_k g_k(x).$$

If $x^* \in \mathbb{R}^n$ is a local minimizer, then there exists $\lambda^* \in \mathbb{R}^m$ such that

$$\nabla_x L(x^*, \lambda^*) = 0, \quad \nabla_\lambda L(x^*, \lambda^*) = 0.$$

Equality constrained optimization: Penalty method

A simpler approach is to penalize deviations from the constraints and adding it to the objective function:

$$x_\epsilon^\star \in \operatorname{argmin}_{x \in \mathbb{R}^n} f(x) + \frac{1}{\epsilon} \|g(x)\|^2.$$

Here $\epsilon > 0$ is a small parameter. In the limit $\epsilon \rightarrow 0$ we get the true solution:

$$\lim_{\epsilon \rightarrow 0} x_\epsilon^\star = x^\star \in \operatorname{argmin}_{x \in K} f(x).$$

Easy to implement, but no simple rationale to choose ϵ !

Automatic differentiation

A key component of gradient based algorithms is **computing the derivatives** of the objective function – this **can be automated!**

Common approaches to computing derivatives:

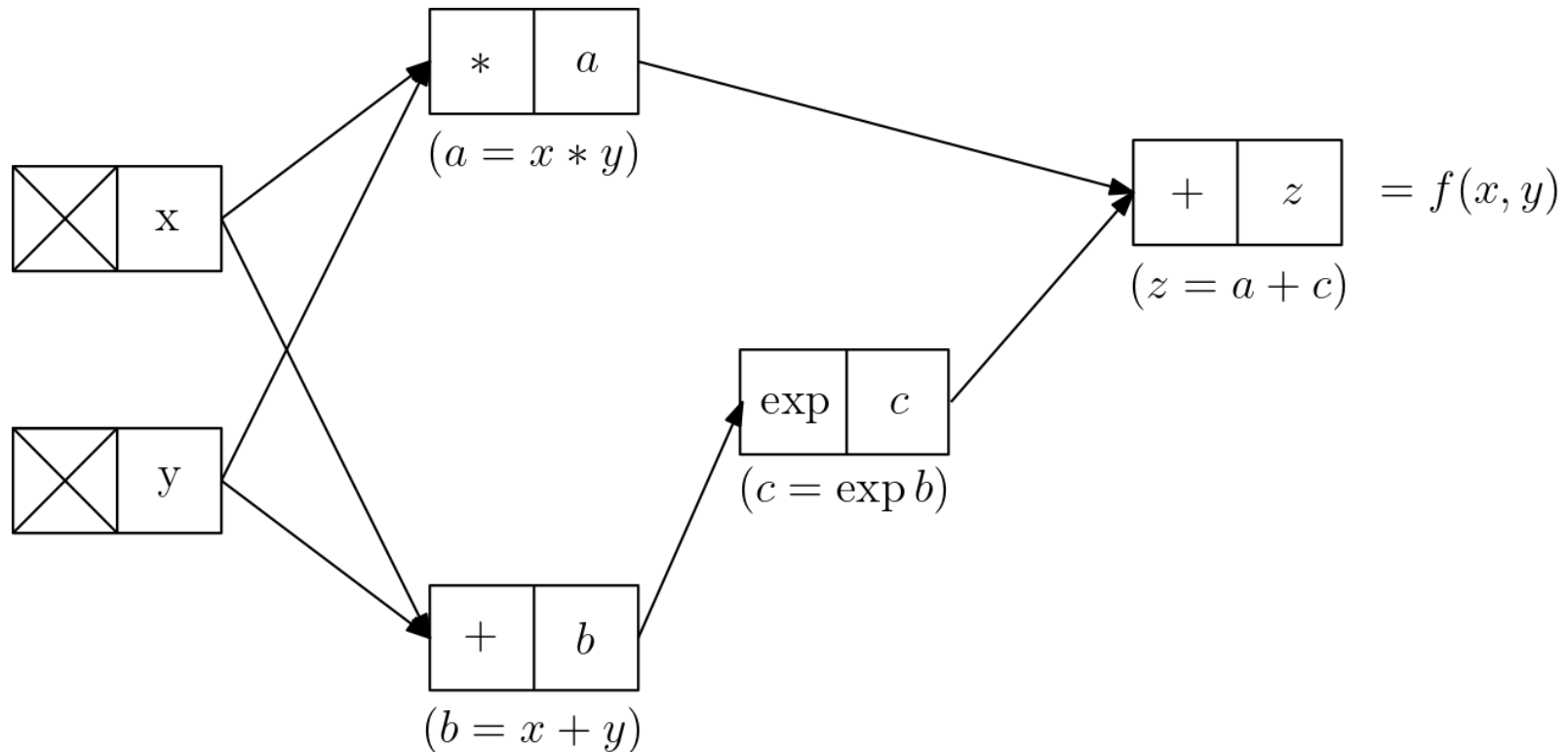
- Numerical differentiation – costly and propagates errors
- Symbolic differentiation – impractical
- **Automatic differentiation (AD)**

AD computes exact derivatives for given inputs. Two major variants:

- Forward mode AD
- Reverse mode AD

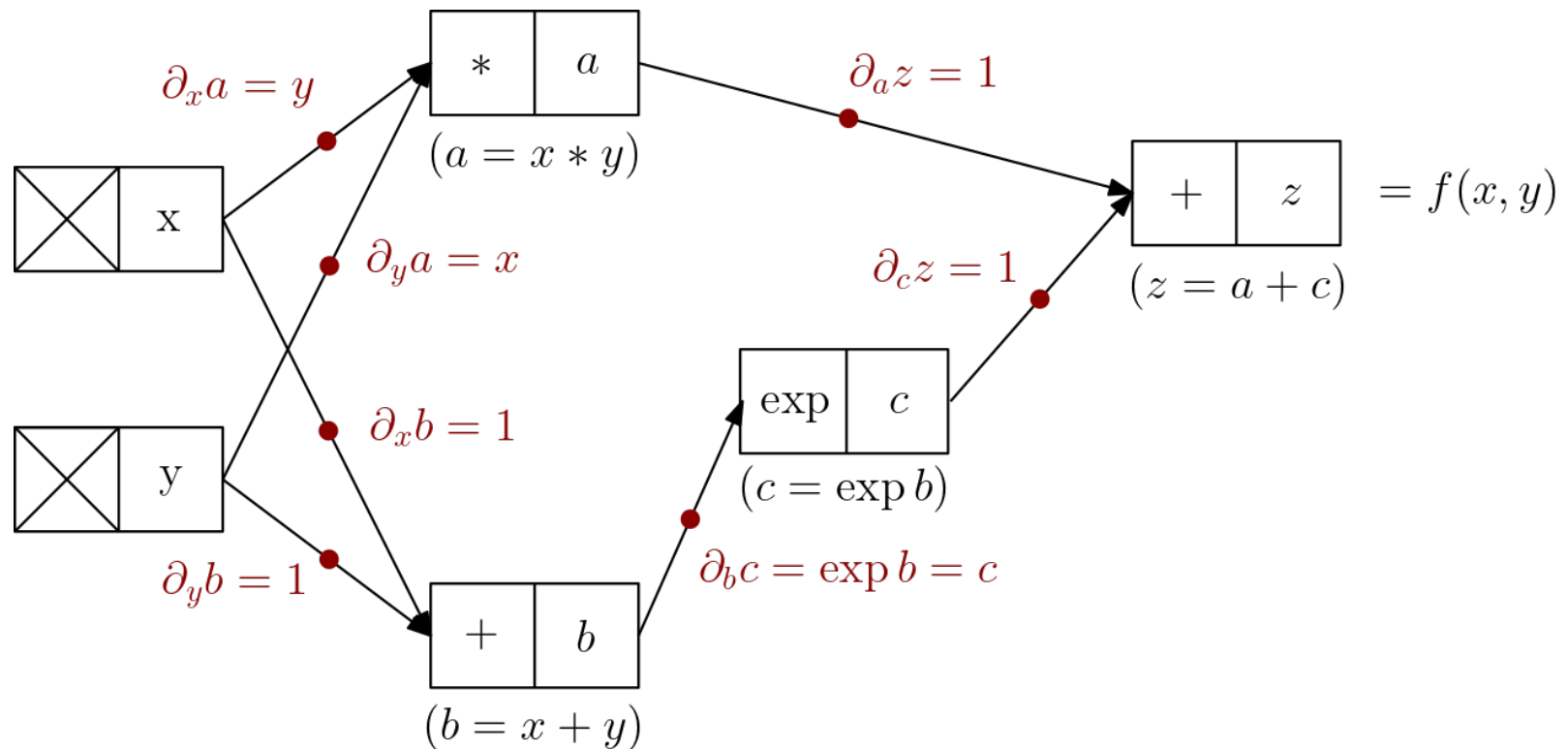
Functions as computational graphs

$$f(x, y) = xy + \exp(x + y)$$



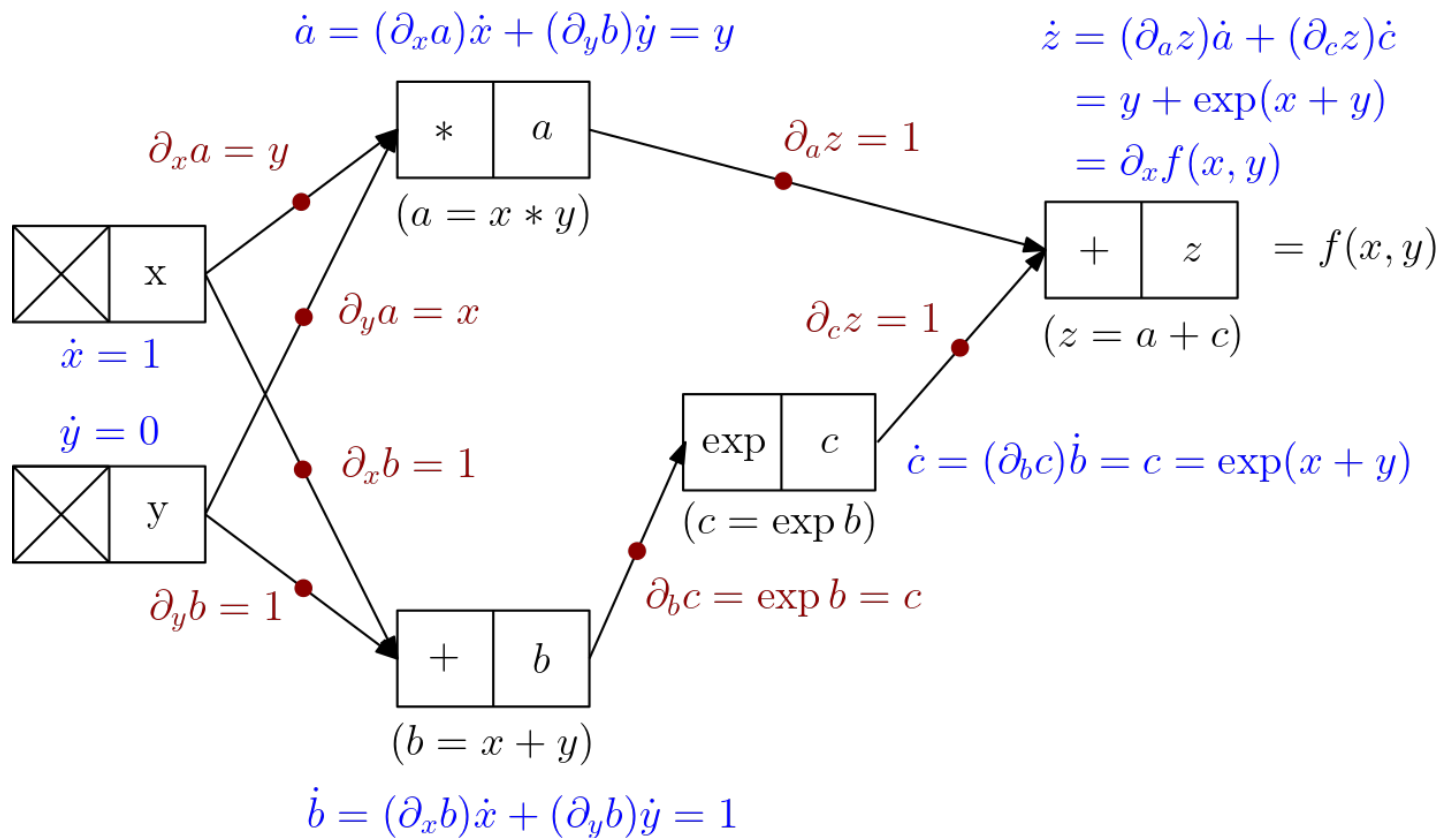
Track derivatives of elementary functions

$$f(x, y) = xy + \exp(x + y)$$



Forward mode AD

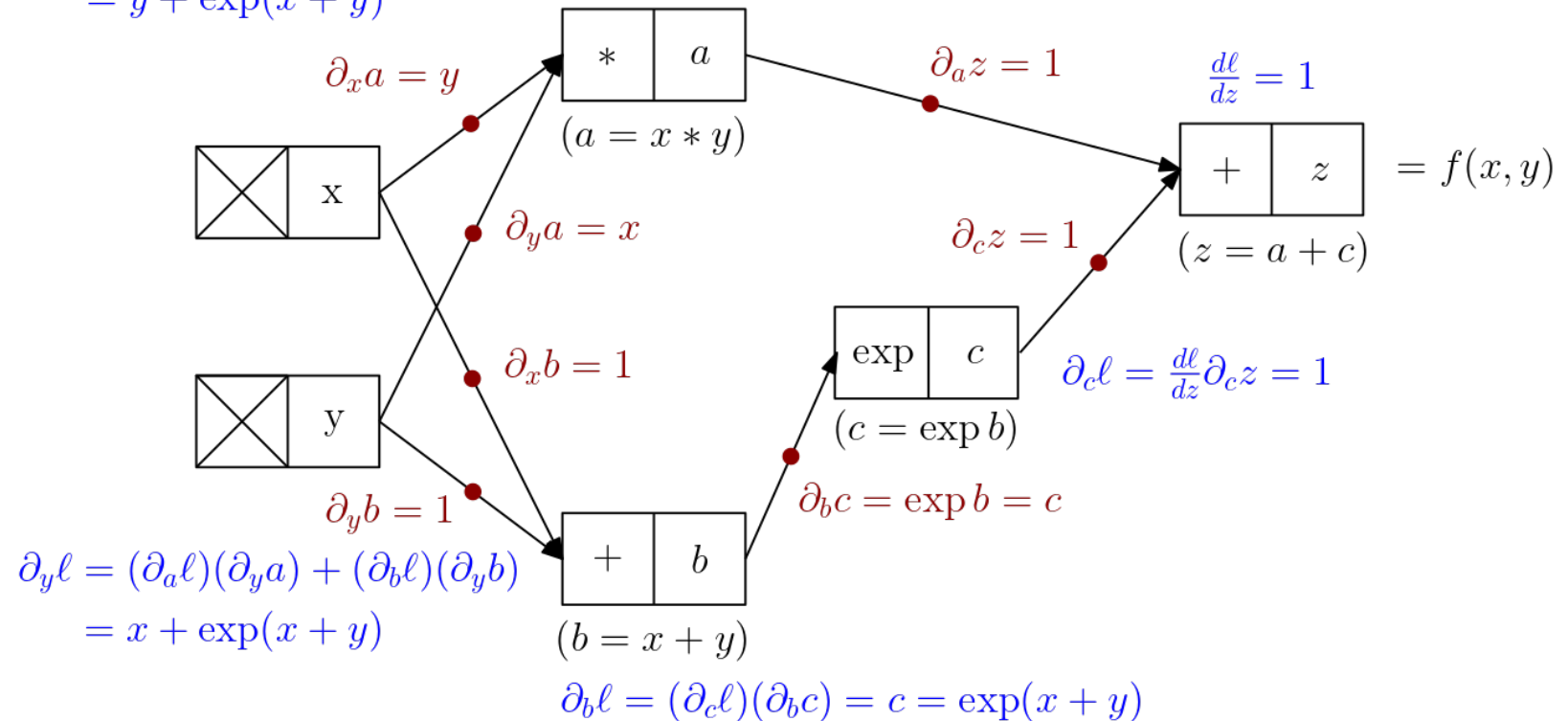
$$\partial_x f(x, y) = y + \exp(x + y)$$



Reverse mode AD

$$\partial_x f(x, y) = y + \exp(x + y) \quad \partial_y f(x, y) = x + \exp(x + y)$$

$$\begin{aligned} \partial_x \ell &= (\partial_a \ell)(\partial_x a) + (\partial_b \ell)(\partial_x b) \\ &= y + \exp(x + y) \end{aligned} \quad \partial_a \ell = \frac{d\ell}{dz} \partial_a z = 1$$



Practical aspects of AD

For computing derivatives of $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$

- Forward mode AD is faster if $m \ll n$
- Reverse mode AD is faster if $m \gg n$

Common AD implementation approaches:

- Source-to-source transformation
- Operator overloading

There are lots of high quality AD packages available now!
Demos in this workshop will use **PyTorch**.

 **Hands-on session!**

Supervised learning algorithms

Suppose that we are given data of the form

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \subset \mathbb{R}^d \times \mathbb{R}.$$

The goal of **supervised learning** is to find a **function** f in a specified function space \mathcal{F} that **best explains this data**.

$$\min_{f \in \mathcal{F}} \sum_{i=1}^n \ell(y_i, f(x_i)).$$

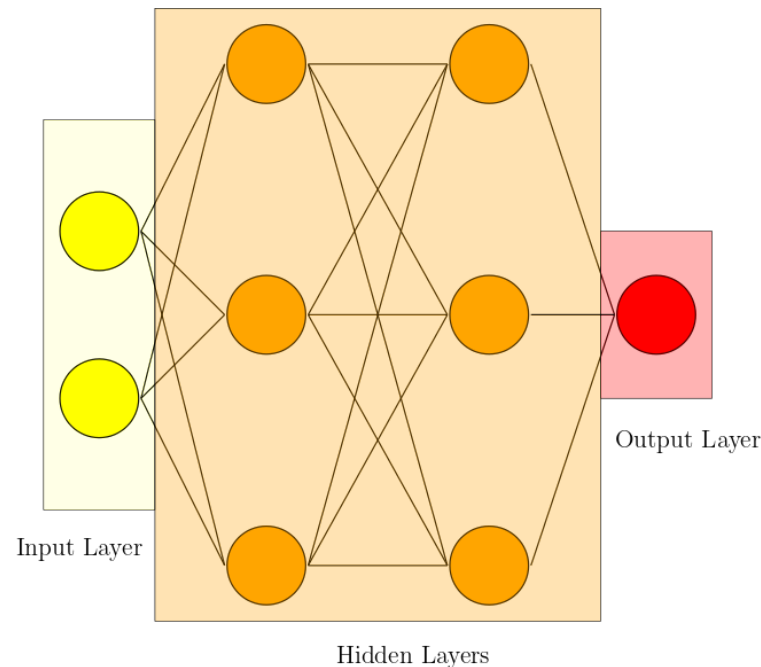
The **loss function** $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is typically chosen as an L^2 distance:

$$\ell(y, z) = \frac{1}{2} |y - z|^2.$$

Deep Neural Networks (DNNs)

A particularly interesting function space is that of **Deep Neural Networks**, which are loosely inspired by biological neural networks.

A DNN is a composition of a finite number of affine transformations and a nonlinear activation function.



Deep Neural Networks (DNNs)

Mathematically, an L layer DNN can be written as

$$DNN(x; \theta) = T_L \circ \sigma \odot T_{L-1} \circ \dots \circ \sigma \odot T_1(x).$$

Where $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ is the activation function, \odot is element-wise function application, and $\{T_l : \mathbb{R}^{n_{l-1}} \rightarrow \mathbb{R}^{n_l}\}_{l=1}^L$ are the affine functions:

$$T_l(z) = W_l z + b_l, \quad W_l \in \mathbb{R}^{n_l \times n_{l-1}}, b_l \in \mathbb{R}^{n_l}, z \in \mathbb{R}^{n_{l-1}}.$$

$$\sigma \left(\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix} \right) = \begin{bmatrix} \sigma(z_1) \\ \vdots \\ \sigma(z_m) \end{bmatrix}.$$

The parameters θ of the DNN are

$$\theta = \{(W_1, b_1), \dots, (W_L, b_L)\}.$$

Stochastic gradient descent

For loss functions that have an additive structure, the true GD update is

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \left(\sum_{i=1}^n \ell_i(\theta) \right).$$

When using DNNs, it is convenient to use the **stochastic GD / mini-batch GD** update:

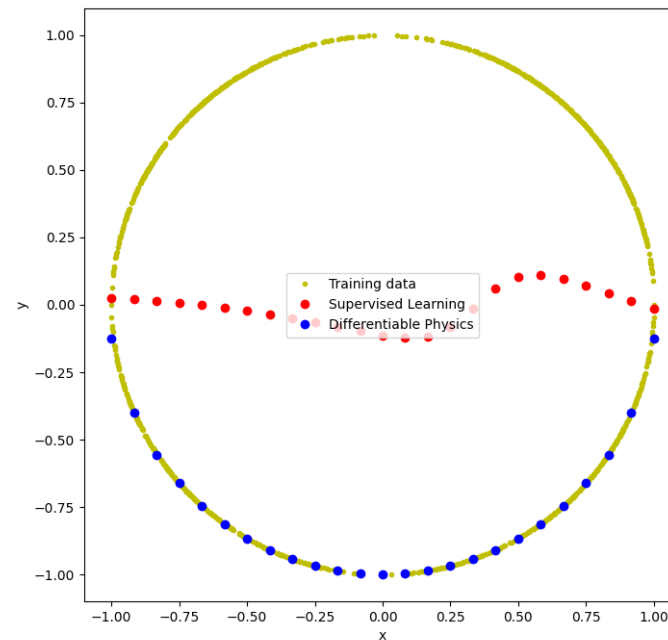
$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \ell_1(\theta), \dots, \theta \leftarrow \theta - \alpha \nabla_{\theta} \ell_n(\theta).$$

The collection of all these partial updates is called an **epoch**.

In practice, a subset of data points defined by a *batch size* are used. This is called **mini-batch SGD**.

Supervised vs physics based learning

Physics based learning is characterized by the choice of a **physically relevant loss function**.



➡ Hands-on session!

Solving ODE with DNNs

Consider the first order ordinary differential equation:

$$\frac{dx(t)}{dt} = f(x(t)), \quad x(0) = x_0.$$

To solve this over $[0, T]$, we approximate the unknown solution as a DNN $\hat{x}(t; \theta)$ and use the following loss function defined as sampling points $\{t_i\}_{i=1}^N \subset [0, T]$:

$$\ell(\theta) = \sum_{i=1}^N \left| \frac{d\hat{x}(t_i; \theta)}{dt} - f(\hat{x}(t_i; \theta)) \right|^2 + \alpha |\hat{x}(0; \theta) - x_0|^2.$$

Note: The derivatives of the DNN can also be computed using AD!

Original idea: Lagaris et al., *Artificial Neural Networks for solving Ordinary and Partial Differential Equations*, IEEE Transactions on Neural networks, Vol 9 (5), **1998**.

Modern reincarnation: Raissi et al., *Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, Vol 378 (1), **2019**.

Review: Forward Euler method for ODEs

The exact solution to the ODE can be written as a **flow map**:

$$\Phi_t : x_0 \mapsto x(t).$$

Here, $x(t)$ is the solution of the ODE at time t with initial condition x_0 .

In practice, we can compute only an approximation to this flow map:

$$\Phi_t^h : x_0 \mapsto x^h(t).$$

To keep things simple, we will use the **forward Euler** method for computing the approximate solution to the ODE:

$$x_{n+1} = x_n + \Delta t f(x_n).$$

 **Hands-on session!**

Inverse problem: differentiable physics

Consider the same ODE as before:

$$\frac{dx(t)}{dt} = f(x(t)), \quad x(0) = x_0.$$

Given $J : \mathbb{R} \rightarrow \mathbb{R}$, the following is an example of an inverse problem:

What should be the value of x_0 such that $J(x(T)) = J_0$?

Strategy: Learn shooting function with a DNN.

- Approximate the function $J_0 \mapsto x_0$ with a DNN.
- Use the discrepancy $\ell = |J(\Phi_T^h(x_0)) - J_0|^2$ as the loss to train the DNN.

Inverse problem: differentiable physics

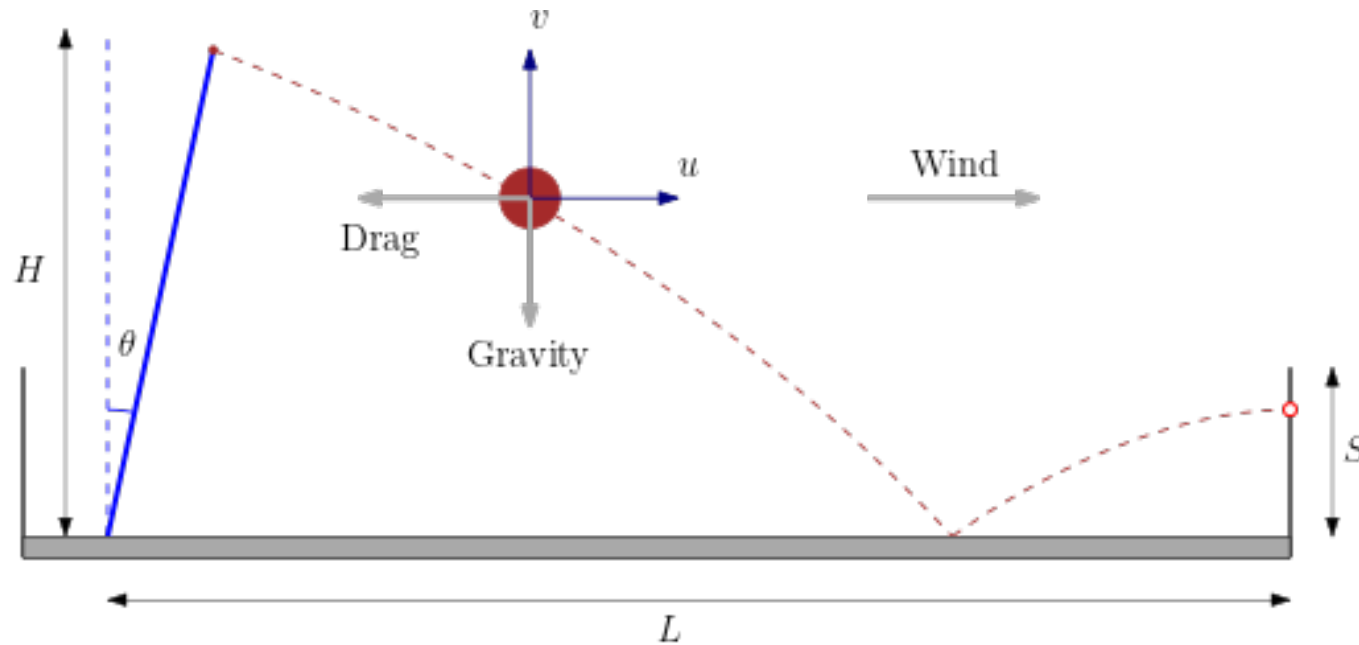
To train the DNN, we need to compute the derivative of the loss function:

$$\frac{d\ell}{d\theta} = (J(\Phi_T^h(x_0)) - J_0) \nabla_x \Phi_T^h(x_0) \nabla_\theta x_0.$$

- The first term is just an algebraic factor .
- **The second term is the derivative of the solver – differentiable physics!**
- The final term is the gradient of the DNN.

Differentiating solvers is not always easy, and often requires more work.
When developing new software, keep differentiability in mind!

Hands-on session: learning to bowl!



$$m \frac{du}{dt} = -\frac{1}{2} \rho A C_D (u - W) \sqrt{(u - W)^2 + v^2}$$
$$m \frac{dv}{dt} = -\frac{1}{2} \rho A C_D v \sqrt{(u - W)^2 + v^2} - mg$$

Simplified version of model presented in C.J. Baker, *A calculation of cricket ball trajectories*, Proc. IMechE. Vol 224 Part C: JMES, 2009.

There is more!

- Differentiable physics idea is not new – classic adjoint methods do the same! But **making solvers compatible with AD can provide very interesting new information!**
- The material discussed here can be extended to **PDEs** too. Check out: <https://www.physicsbaseddeeplearning.org/intro.html>
- **Deep Learning** is all the rage these days, but it is **not interpretable!** Check out our recent work – SPINN – for interpretable neural networks.

Thank you for your attention!
Happy Holidays and Happy New Year!

Amuthan Ramabathiran, Prabhu Ramachandran, **SPINN**: Sparse, Physics-based, and partially Interpretable Neural Networks for PDEs, Journal of Computational Physics, Vol 445 (15), 2021.
Video of my recent talk on SPINN: <https://youtu.be/dfGojwG5nU8>