Introduction to Deep Learning and Differentiable Physics

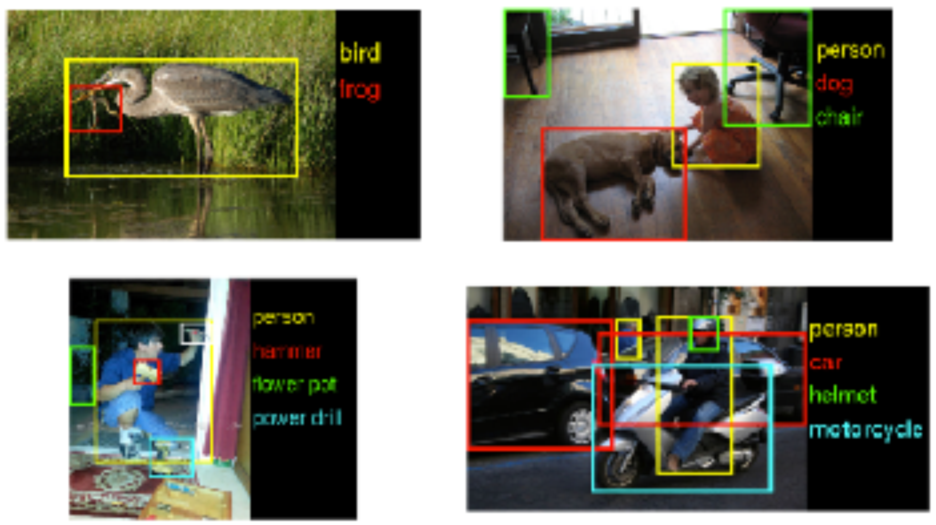
François Rousseau

Institut Mines-Télécom

Why Deep Learning?

- Why is deep learning so popular?
  - Very good performance
  - End-to-end approach
- Why now?
  - Hardware: GPU
  - Datasets: ImageNet, Kaggle, etc.
  - User friendly libraries: PyTorch or Keras (and previously Caffe & Theano)
  - Algorithmics advances

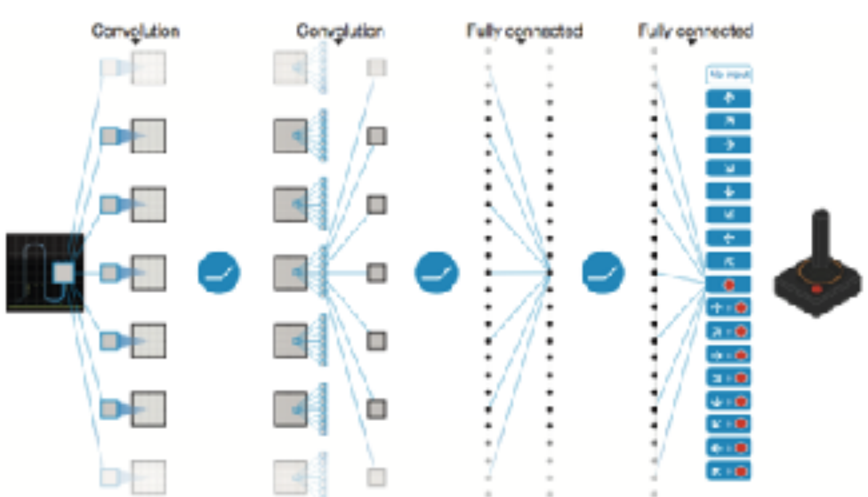
Large Scale Visual Recognition Challenge



Real Time Recognition

Image captioning

Algorithm playing ATARI



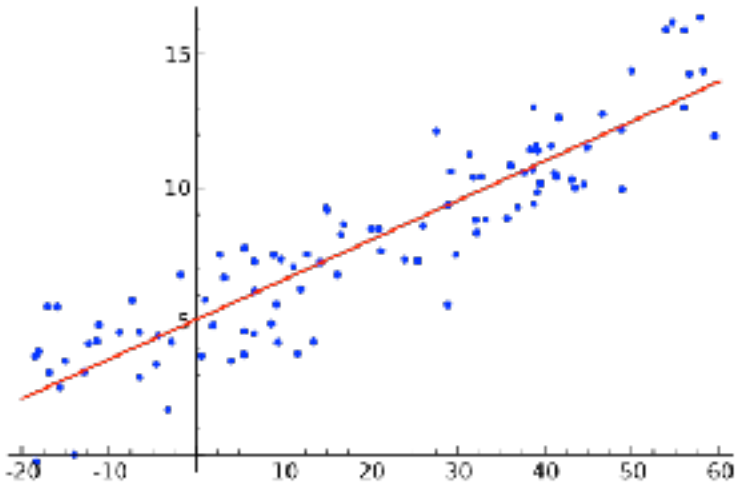
Real Time Translation

Text generator

Overview

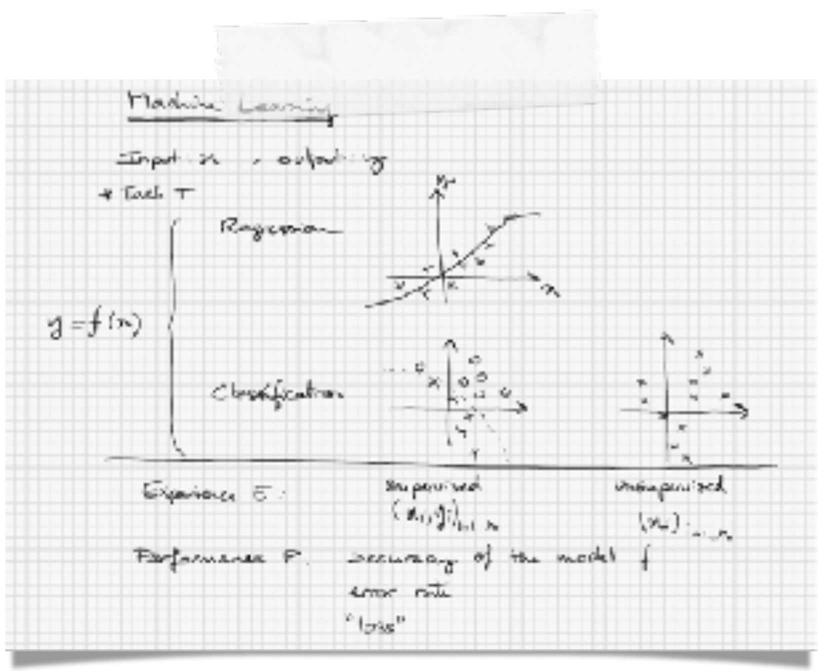
Machine
Learning

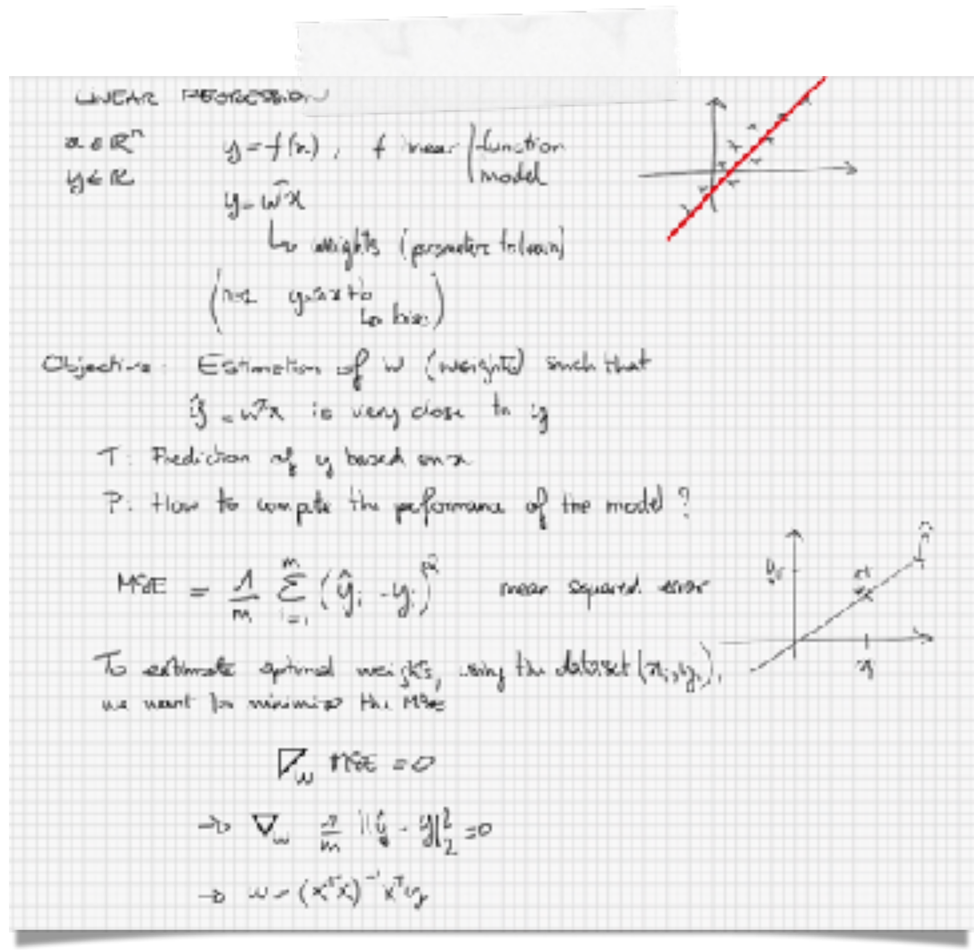
Machine learning vocabulary

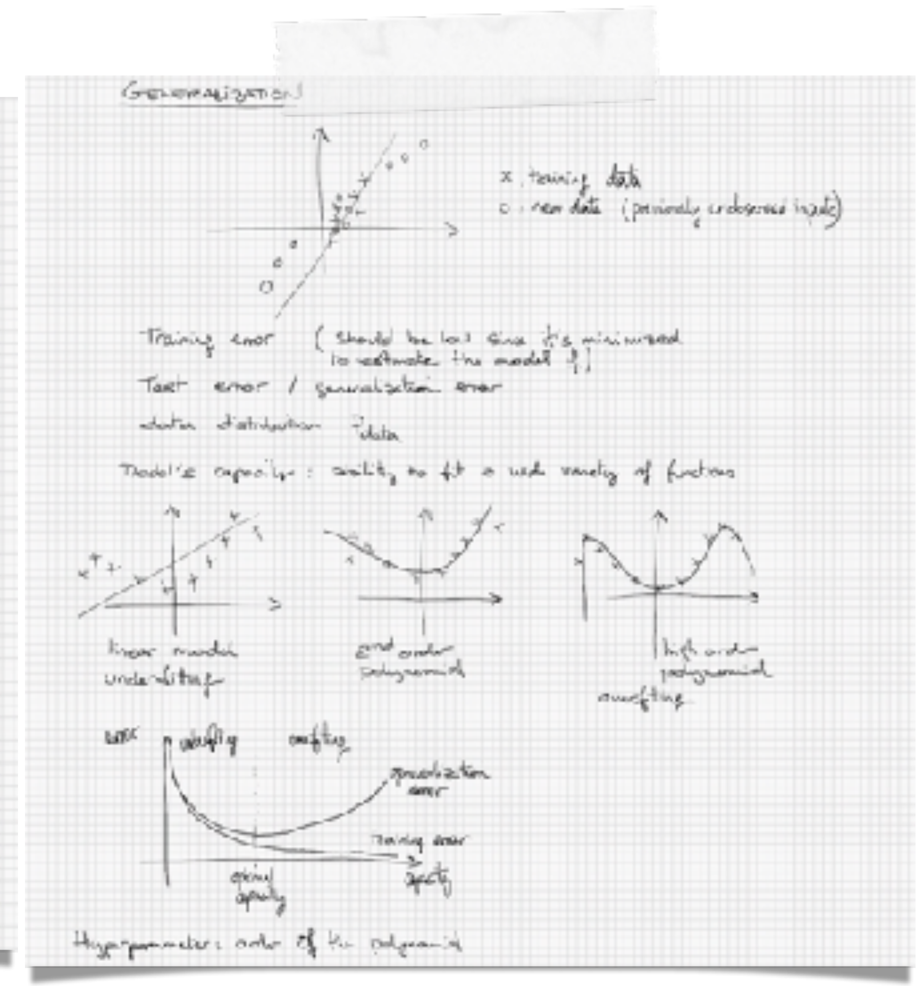


- Data: input x and output y
- Task: regression, classification
- Experience: supervised (x_i, y_i) or unsupervised (x_i)
- Performance measure: accuracy of the model or error rate

Basics of Machine Learning







Overview

Machine Learning

Neural Networks

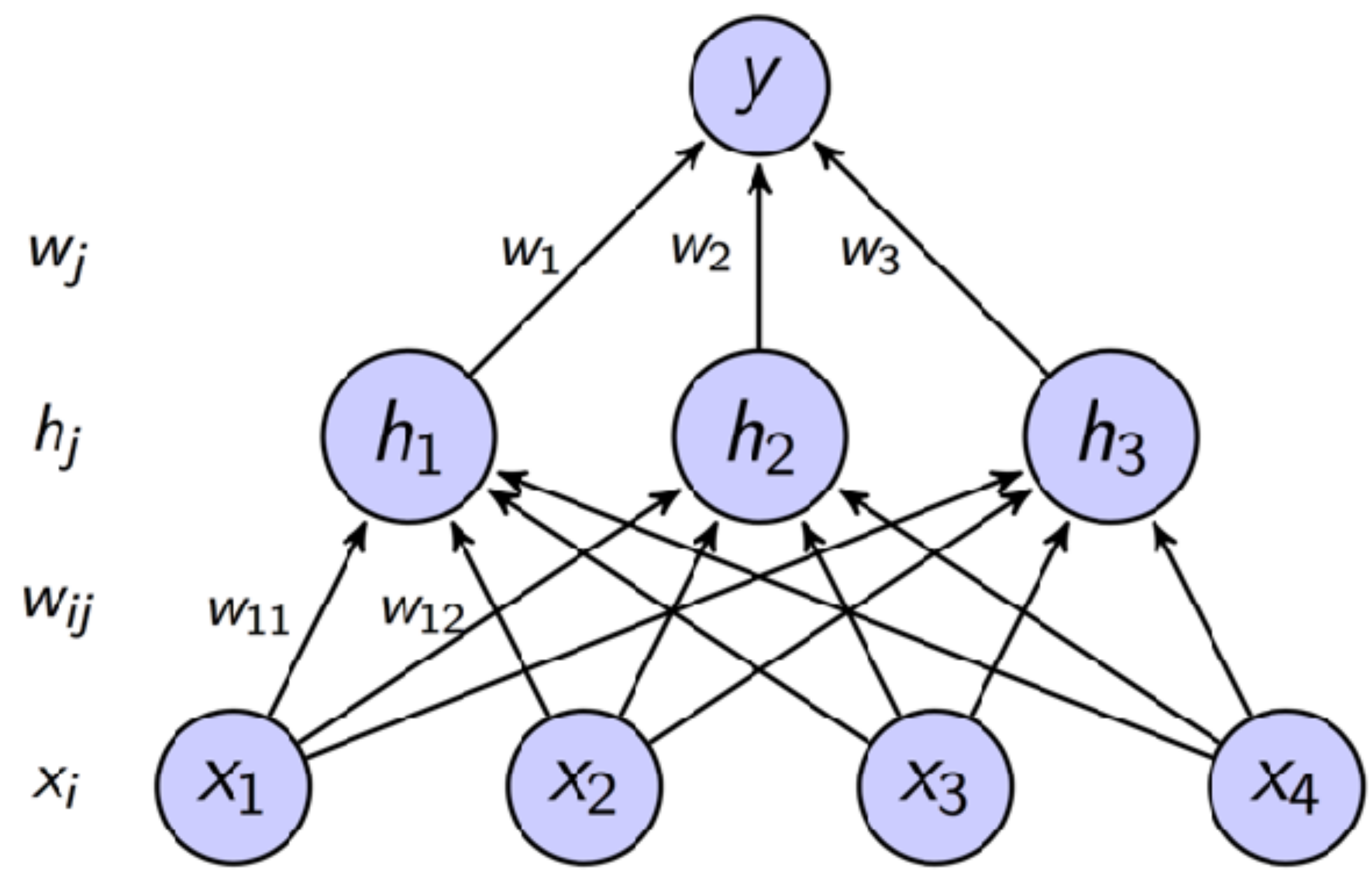
Neural networks

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$$

Parameters to learn

- Objective: approximate a function f^*
- NN: composition of functions $f = f_n \circ \ldots \circ f_2 \circ f_1$
- How: estimate parameters θ of the neural network via a minimization problem: $\arg\min_\theta \sum_i (f(x_i; \theta) - y_i)^2$

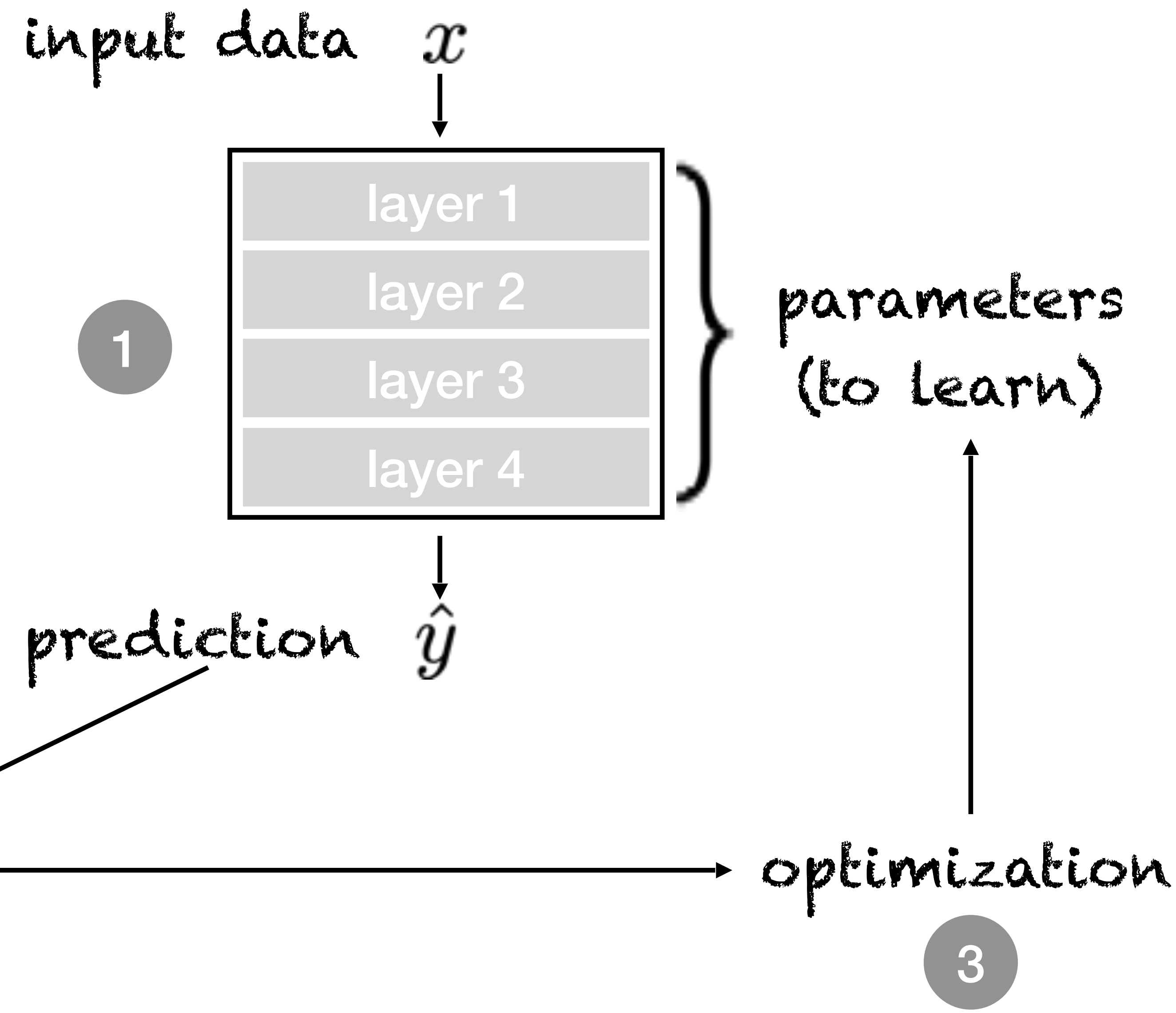
Neural networks



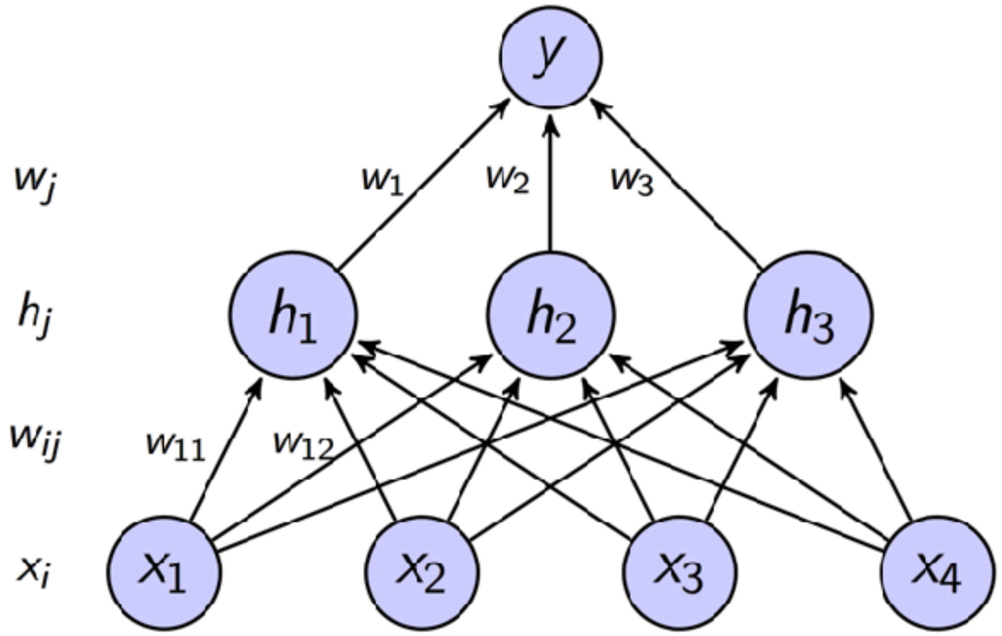
$$f(x) = \sigma(\sum_j w_j \cdot h_j) = \sigma(\sum_j w_j \cdot \sigma(\sum_i w_{ij} x_i))$$

Overview of NN

1. **Prediction**
2. **Loss**
3. **Update parameters**



In practice




```python
import torch
import torch.nn as nn
```

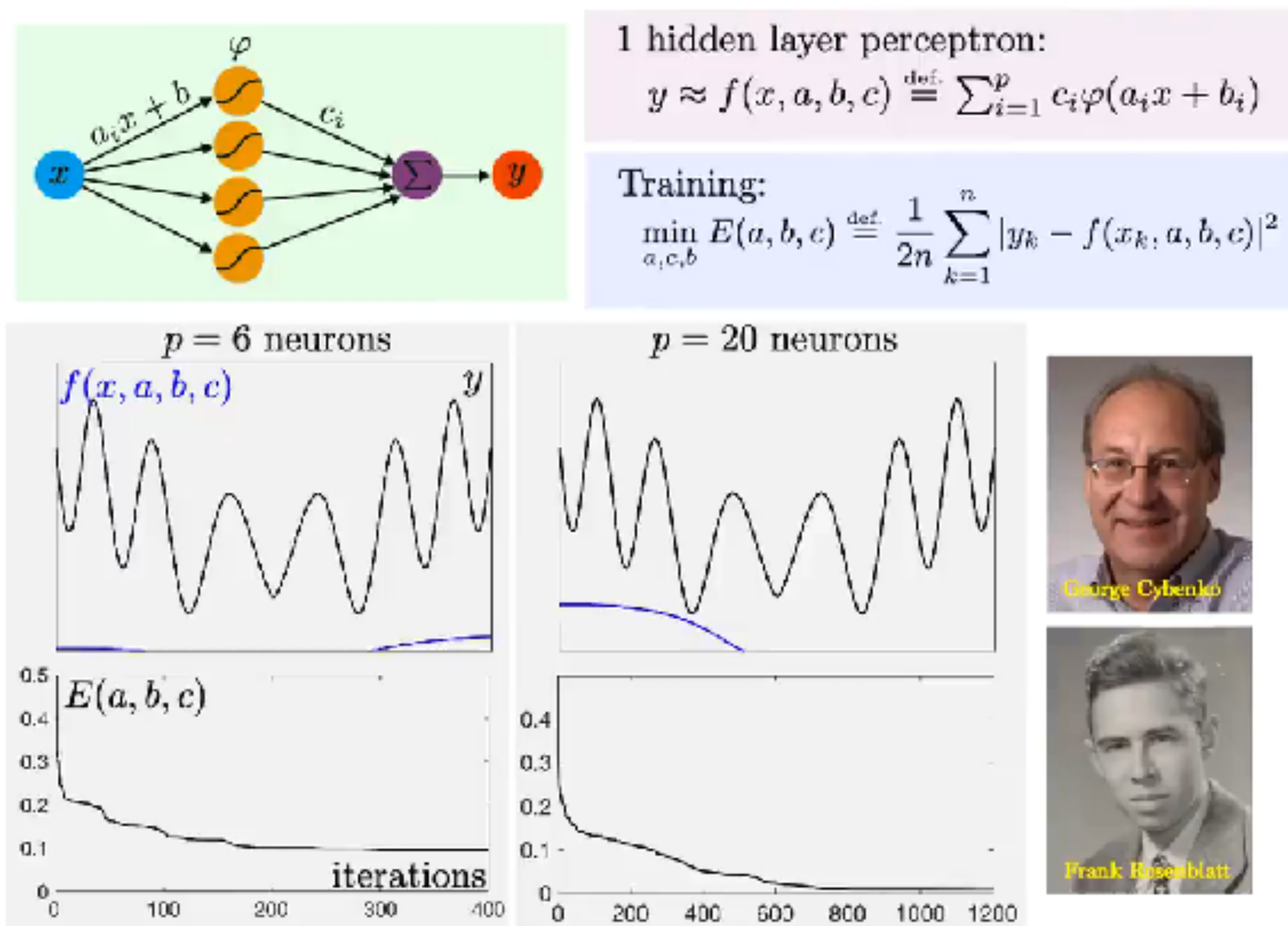
```python
class Net(nn.Module):

  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = nn.Linear(4,3)
    self.fc2 = nn.Linear(3,1)

  def forward(self, x):
    x = self.fc1(x)
    x = F.relu(x)
    x = self.fc2(x)
    x = F.softmax(x)
    return x

net = Net()
```

```python
criterion = nn.MSELoss()
```

```python
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
```

```python
for i in range(epochs):       # training loop
  y_pred = net(x)             # compute the prediction using current parameters
  loss = criterion(y_pred,y)  # compute loss
  optimizer.zero_grad()       # initialize the gradient to zero
  loss.backward()             # back propagation
  optimizer.step()            # update the parameters values using the gradient
```

Universal Approximation Theorem



[From Gabriel Peyré]

Gradient-based approach for parameter estimation

- Gradient:

$$\frac{\partial J(\theta)}{\partial \theta} = \lim_{\delta\theta \to 0} \frac{J(\theta + \delta\theta) - J(\theta)}{\delta\theta}$$

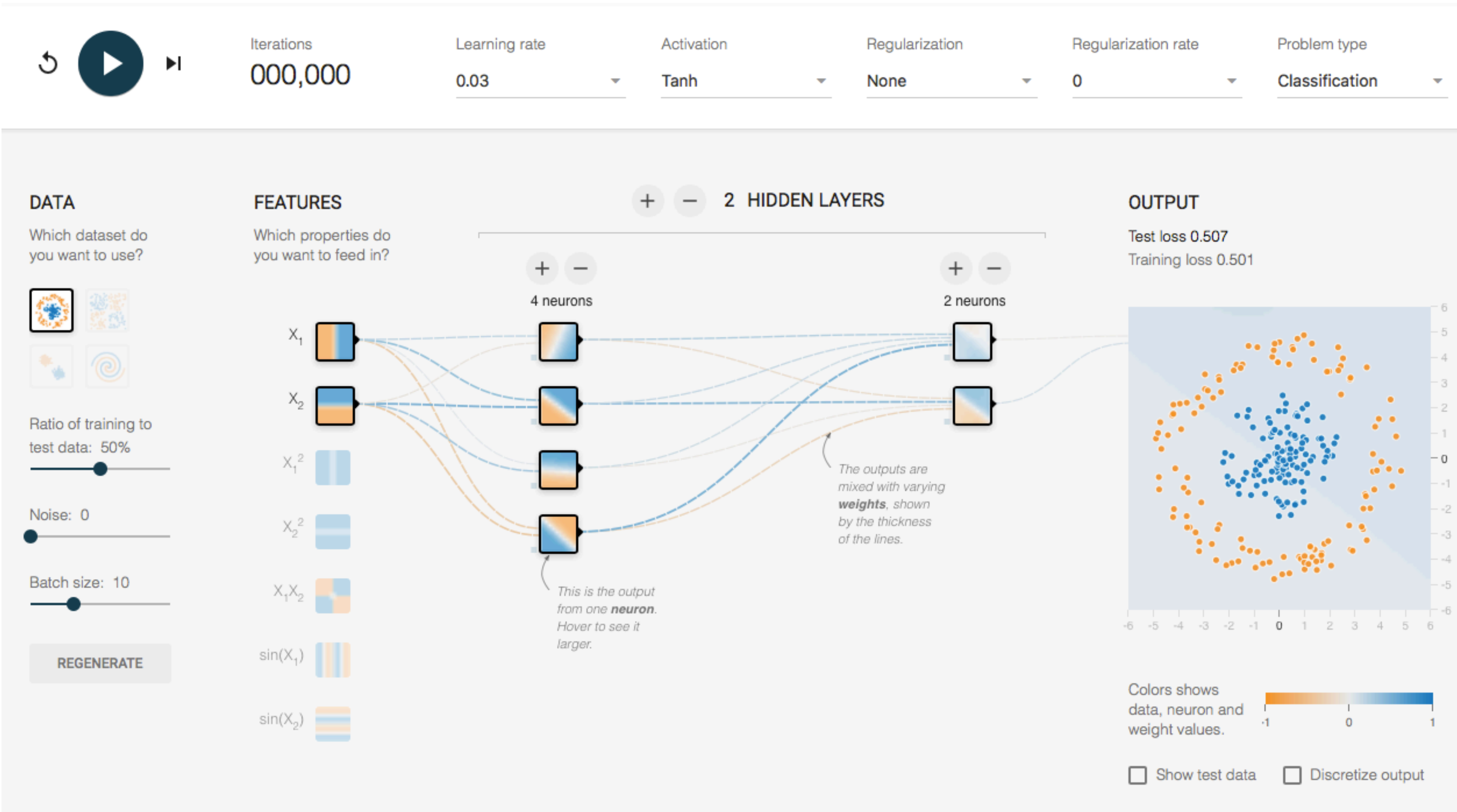
- Gradient descent:

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

Learning rate

Practicing Deep Learning



Introduction_regression.ipynb

Overview

Machine Learning

Neural Networks

Automatic Differentiation

Automatic differentiation

- How to compute the derivative of a function f ?

$$f'(a) = \frac{df}{dx}(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

- Unlike finite difference method, AD does not incur truncation errors.

AD example

Given this: $y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$

How to compute the value of y for $x_1 = 1.5$ and $x_2 = 0.5$?

$$\begin{array}{lllll} v_{-1} & = & x_1 & = & 1.5000 \\ v_0 & = & x_2 & = & 0.5000 \end{array}$$

AD example

Given this: $y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$

How to compute the value of y for $x_1 = 1.5$ and $x_2 = 0.5$?

$$
\begin{array}{lclclcl}
v_{-1} & = & x_1 & = & 1.5000 & & \\
v_0 & = & x_2 & = & 0.5000 & & \\
\hline
v_1 & = & v_{-1}/v_0 & = & 1.5000/0.5000 & = & 3.0000
\end{array}
$$

AD example

Given this: $y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$

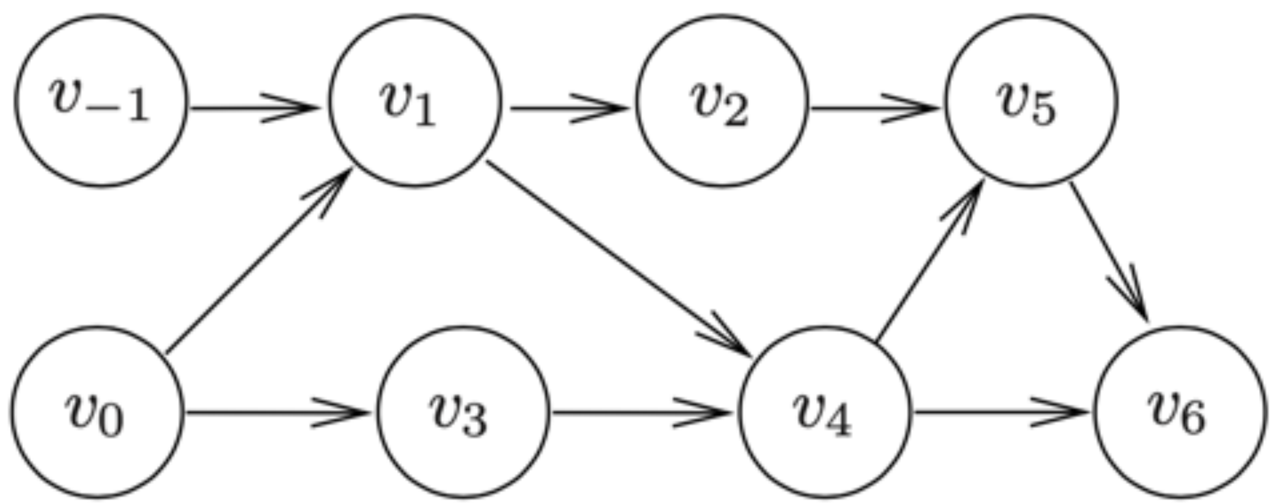
How to compute the value of y for $x_1 = 1.5$ and $x_2 = 0.5$?

$$
\begin{array}{lllllll}
v_{-1} & = & x_1 & = & 1.5000 & & \\
v_0 & = & x_2 & = & 0.5000 & & \\
\hline
v_1 & = & v_{-1}/v_0 & = & 1.5000/0.5000 & = & 3.0000 \\
v_2 & = & \sin(v_1) & = & \sin(3.0000) & = & 0.1411 \\
v_3 & = & \exp(v_0) & = & \exp(0.5000) & = & 1.6487 \\
v_4 & = & v_1 - v_3 & = & 3.0000 - 1.6487 & = & 1.3513 \\
v_5 & = & v_2 + v_4 & = & 0.1411 + 1.3513 & = & 1.4924 \\
v_6 & = & v_5 * v_4 & = & 1.4924 * 1.3513 & = & 2.0167 \\
\hline
y & = & v_6 & = & 2.0167 & &
\end{array}
$$

Evaluation trace

AD example

Given this: $y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$



Computational graph

Forward mode

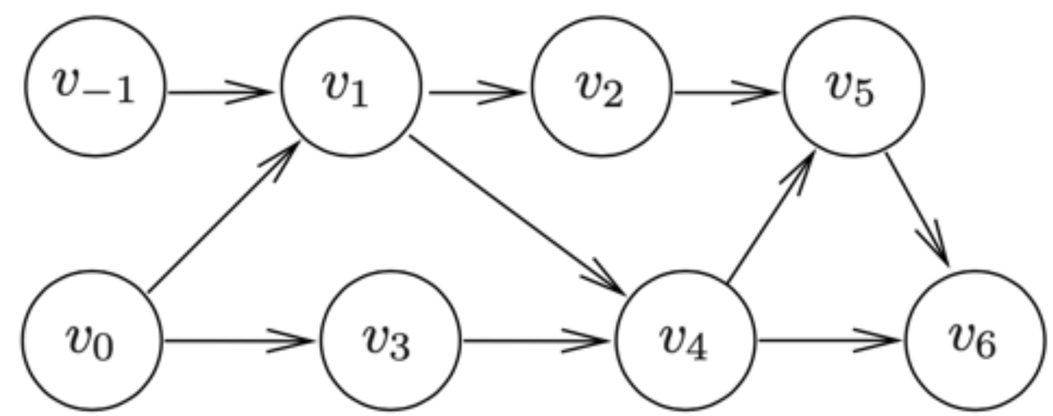
We want to differentiate the output variable y with respect to x_1.

$$\dot{v}_i = \frac{\partial v_i}{\partial x_1}$$

$$\dot{v}_1 = \frac{\partial v_1}{\partial x_1} = \frac{\partial v_1}{\partial v_{-1}}\frac{\partial v_{-1}}{\partial x_1} + \frac{\partial v_1}{\partial v_0}\frac{\partial v_0}{\partial x_1}$$

$$\dot{v}_1 = \frac{1}{v_0}\dot{v}_{-1} + v_{-1}\left(-\frac{\dot{v}_0}{v_0^2}\right)$$

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$

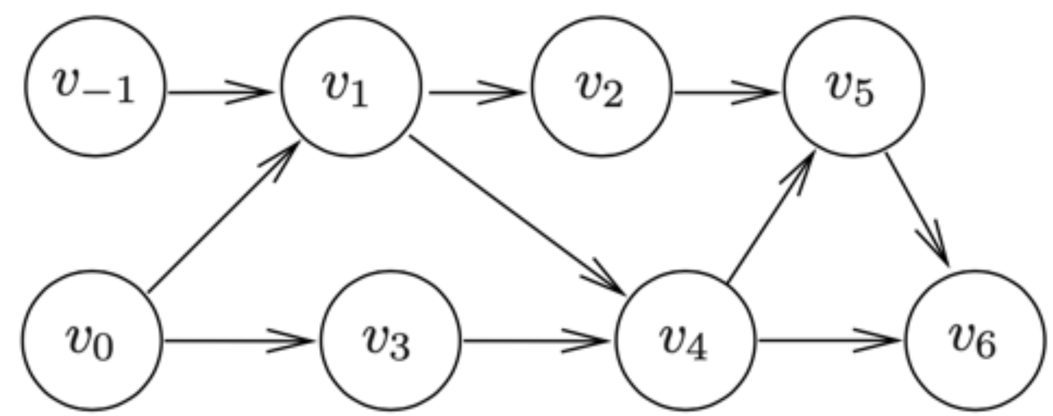


$$\begin{array}{lcl} v_{-1} & = & x_1 \\ v_0 & = & x_2 \\ \hline v_1 & = & v_{-1}/v_0 \end{array}$$

Forward mode

$v_{-1} = x_1$		$= 1.5000$		
$\dot{v}_{-1} = \dot{x}_1$		$= 1.0000$		
v_0	$= x_2$	$= 0.5000$		
$\dot{v}_0$	$= \dot{x}_2$	$= 0.0000$		
v_1	$= v_{-1}/v_0$	$= 1.5000/0.5000$	$=$	3.0000
$\dot{v}_1$	$= (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0$	$= 1.0000/0.5000$	$=$	2.0000

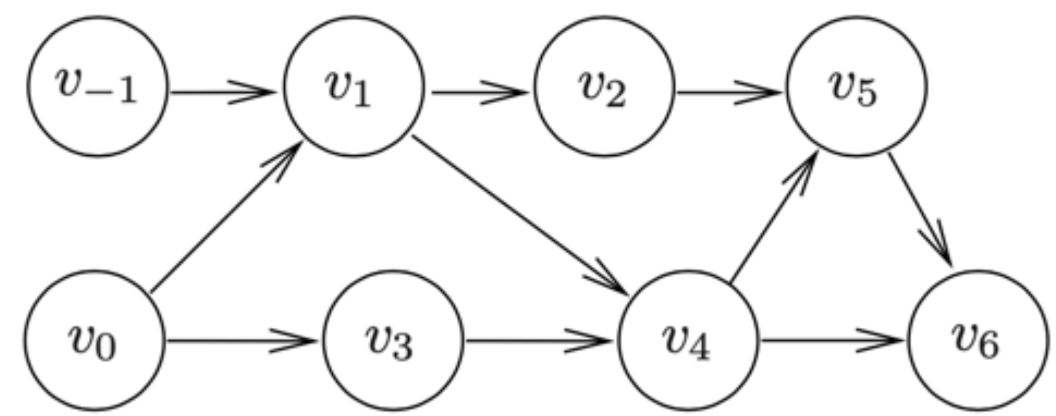
$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



Forward mode

$$\begin{array}{lll}
v_{-1} = x_1 & = 1.5000 & \\
\dot{v}_{-1} = \dot{x}_1 & = 1.0000 & \\
v_0 = x_2 & = 0.5000 & \\
\dot{v}_0 = \dot{x}_2 & = 0.0000 & \\
\hline
v_1 = v_{-1}/v_0 & = 1.5000/0.5000 & = 3.0000 \\
\dot{v}_1 = (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0 & = 1.0000/0.5000 & = 2.0000 \\
v_2 = \sin(v_1) & = \sin(3.0000) & = 0.1411 \\
\dot{v}_2 = \cos(v_1) * \dot{v}_1 & = -0.9900 * 2.0000 & = -1.9800
\end{array}$$

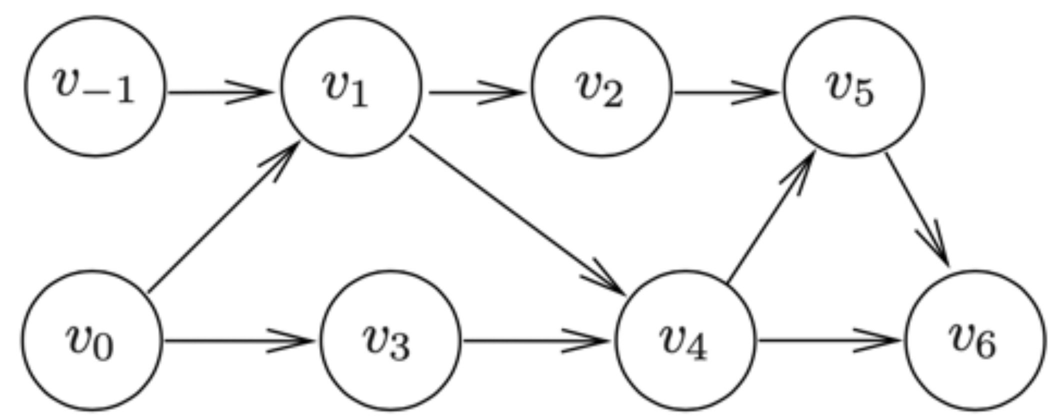
$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



Forward mode

$$
\begin{array}{llll}
v_{-1} & = x_1 & = 1.5000 & \\
\dot{v}_{-1} & = \dot{x}_1 & = 1.0000 & \\
v_0 & = x_2 & = 0.5000 & \\
\dot{v}_0 & = \dot{x}_2 & = 0.0000 & \\
\hline
v_1 & = v_{-1}/v_0 & = 1.5000/0.5000 & = 3.0000 \\
\dot{v}_1 & = (\dot{v}_{-1} - v_1 * \dot{v}_0)/v_0 & = 1.0000/0.5000 & = 2.0000 \\
v_2 & = \sin(v_1) & = \sin(3.0000) & = 0.1411 \\
\dot{v}_2 & = \cos(v_1) * \dot{v}_1 & = -0.9900 * 2.0000 & = -1.9800 \\
v_3 & = \exp(v_0) & = \exp(0.5000) & = 1.6487 \\
\dot{v}_3 & = v_3 * \dot{v}_0 & = 1.6487 * 0.0000 & = 0.0000 \\
v_4 & = v_1 - v_3 & = 3.0000 - 1.6487 & = 1.3513 \\
\dot{v}_4 & = \dot{v}_1 - \dot{v}_3 & = 2.0000 - 0.0000 & = 2.0000 \\
v_5 & = v_2 + v_4 & = 0.1411 + 1.3513 & = 1.4924 \\
\dot{v}_5 & = \dot{v}_2 + \dot{v}_4 & = -1.9800 + 2.0000 & = 0.0200 \\
v_6 & = v_5 * v_4 & = 1.4924 * 1.3513 & = 2.0167 \\
\dot{v}_6 & = \dot{v}_5 * v_4 + v_5 * \dot{v}_4 & = 0.0200 * 1.3513 + 1.4924 * 2.0000 & = 3.0118 \\
\hline
y & = v_6 & = 2.0100 & \\
\dot{y} & = \dot{v}_6 & = 3.0110 &
\end{array}
$$

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$

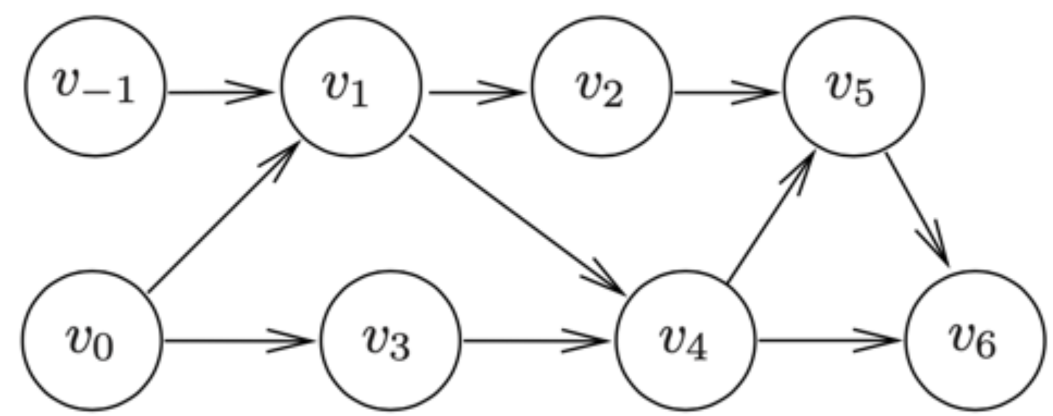


Note: need to do the same to compute $\dfrac{\partial y}{\partial x_2}$

Reverse mode

We want to calculate the sensitivity of an output variable with respect to each of the intermediate variables.

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



$$\bar{v}_i = \frac{\partial y}{\partial v_i}$$

$$\bar{v}_6 = 1$$

$$\bar{v}_5 = \frac{\partial y}{\partial v_5} = \frac{\partial y}{\partial v_6}\frac{\partial v_6}{\partial v_5} = \bar{v}_6 v_4$$

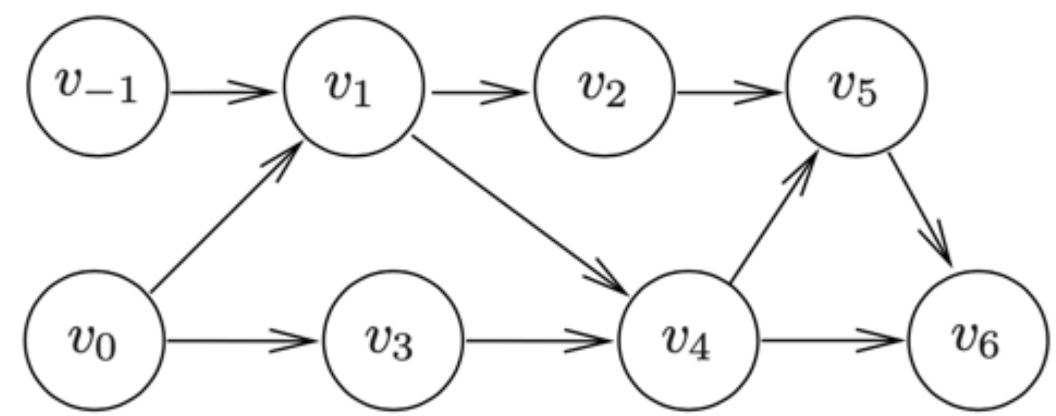
$$\bar{v}_4 = \frac{\partial y}{\partial v_4} = \frac{\partial y}{\partial v_6}\frac{\partial v_6}{\partial v_4} + \frac{\partial y}{\partial v_5}\frac{\partial v_5}{\partial v_4} = \bar{v}_6 v_5 + \bar{v}_5 \times 1$$

$$\begin{array}{lcl} v_4 & = & v_1 - v_3 \\ v_5 & = & v_2 + v_4 \\ v_6 & = & v_5 * v_4 \\ \hline y & = & v_6 \end{array}$$

Reverse mode

$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000 \\
v_0 &= x_2 = 0.5000 \\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000 \\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411 \\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487 \\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513 \\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924 \\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167 \\
y &= v_6 = 2.0167 \\
\bar{v}_6 &= \bar{y} = 1.0000
\end{aligned}
$$

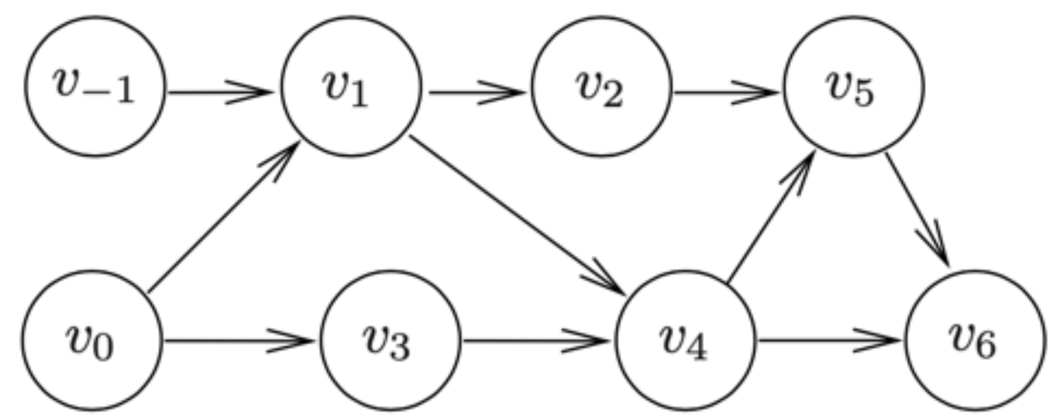
$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



Reverse mode

$$v_{-1} = x_1 = 1.5000$$
$$v_0 = x_2 = 0.5000$$
$$v_1 = v_{-1}/v_0 = 1.5000/0.5000 = 3.0000$$
$$v_2 = \sin(v_1) = \sin(3.0000) = 0.1411$$
$$v_3 = \exp(v_0) = \exp(0.5000) = 1.6487$$
$$v_4 = v_1 - v_3 = 3.0000 - 1.6487 = 1.3513$$
$$v_5 = v_2 + v_4 = 0.1411 + 1.3513 = 1.4924$$
$$v_6 = v_5 * v_4 = 1.4924 * 1.3513 = 2.0167$$
$$y = v_6 = 2.0167$$
$$\bar{v}_6 = \bar{y} = 1.0000$$
$$\bar{v}_5 = \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513$$

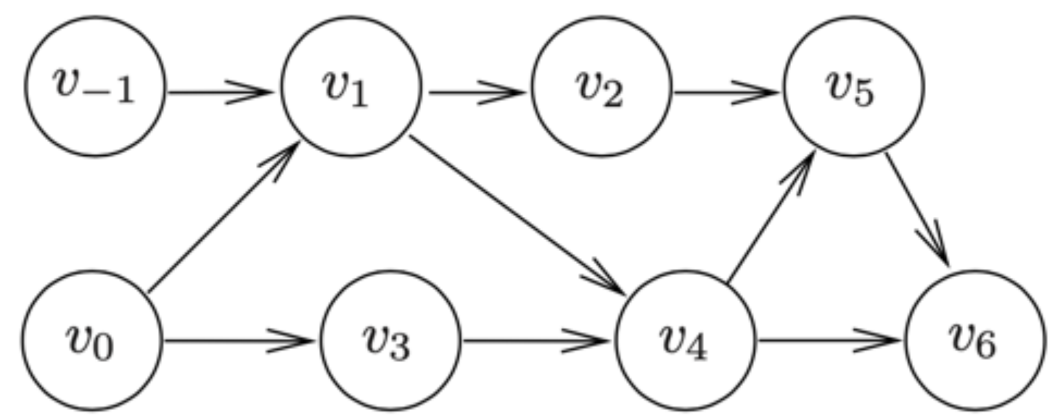
$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



Reverse mode

$$
\begin{aligned}
v_{-1} &= x_1 = 1.5000\\
v_0 &= x_2 = 0.5000\\
v_1 &= v_{-1}/v_0 = 1.5000/0.5000 = 3.0000\\
v_2 &= \sin(v_1) = \sin(3.0000) = 0.1411\\
v_3 &= \exp(v_0) = \exp(0.5000) = 1.6487\\
v_4 &= v_1 - v_3 = 3.0000 - 1.6487 = 1.3513\\
v_5 &= v_2 + v_4 = 0.1411 + 1.3513 = 1.4924\\
v_6 &= v_5 * v_4 = 1.4924 * 1.3513 = 2.0167\\
y &= v_6 = 2.0167\\
\bar{v}_6 &= \bar{y} = 1.0000\\
\bar{v}_5 &= \bar{v}_6 * v_4 = 1.0000 * 1.3513 = 1.3513\\
\bar{v}_4 &= \bar{v}_6 * v_5 = 1.0000 * 1.4924 = 1.4924\\
\bar{v}_4 &= \bar{v}_4 + \bar{v}_5 = 1.4924 + 1.3513 = 2.8437\\
\bar{v}_2 &= \bar{v}_5 = 1.3513\\
\bar{v}_3 &= -\bar{v}_4 = -2.8437\\
\bar{v}_1 &= \bar{v}_4 = 2.8437\\
\bar{v}_0 &= \bar{v}_3 * v_3 = -2.8437 * 1.6487 = -4.6884\\
\bar{v}_1 &= \bar{v}_1 + \bar{v}_2 * \cos(v_1) = 2.8437 + 1.3513 * (-0.9900) = 1.5059\\
\bar{v}_0 &= \bar{v}_0 - \bar{v}_1 * v_1/v_0 = -4.6884 - 1.5059 * 3.000/0.5000 = -13.7239\\
\bar{v}_{-1} &= \bar{v}_1/v_0 = 1.5059/0.5000 = 3.0118\\
\bar{x}_2 &= \bar{v}_0 = -13.7239\\
\bar{x}_1 &= \bar{v}_{-1} = 3.0118
\end{aligned}
$$

$$y = [\sin(x_1/x_2) + x_1/x_2 - \exp(x_2)] * [x_1/x_2 - \exp(x_2)]$$



Note: we have computed $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$ but the computational graph is required.

AD and jacobian matrix

$$F : \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \longmapsto \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{pmatrix}$$

$$J_F(M) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Forward pass: computation of a column
Reverse pass: computation of a row

In machine learning, $m = 1$ (scalar value of the loss function), so we use the reverse mode.

Back-propagation

- Back-propagation is a special case of reverse mode automatic differentiation.

- Back-propagation is an efficient way to recursively compute the gradient:

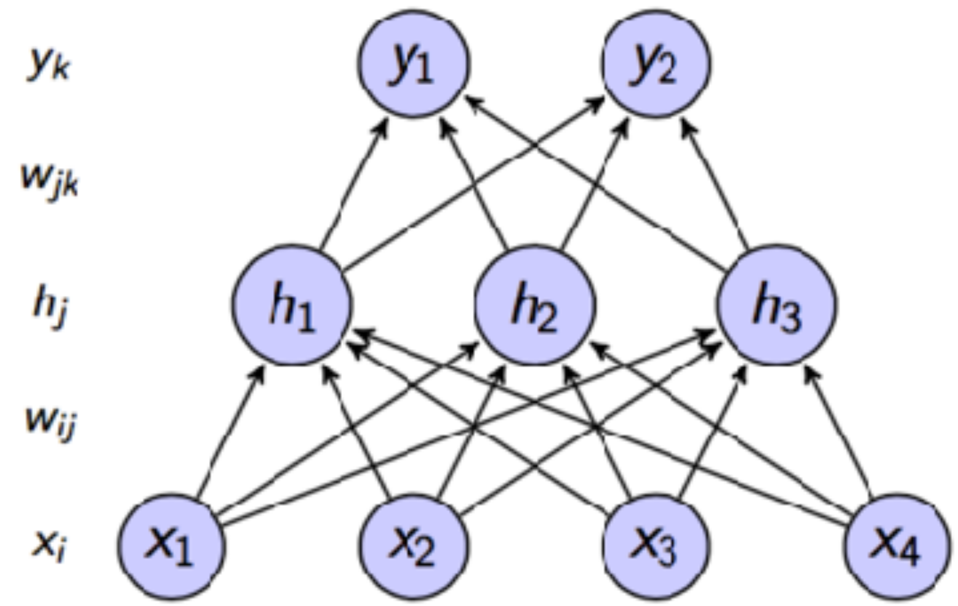
$$\frac{\partial L}{\partial u} = \sum_i \frac{\partial L}{\partial v_i} \frac{\partial v_i}{\partial u}$$

parent of node u

Back-propagation

$$Loss = \frac{1}{2}\sum_k (f(x_k) - y_k)^2$$

$$f(x_k) = \sigma(in_k)$$



$$\frac{\partial Loss}{\partial w_{jk}} = \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial w_{jk}} = \delta_k \frac{\partial\left(\sum_j w_{jk}h_j\right)}{\partial w_{jk}} = \delta_k h_j$$

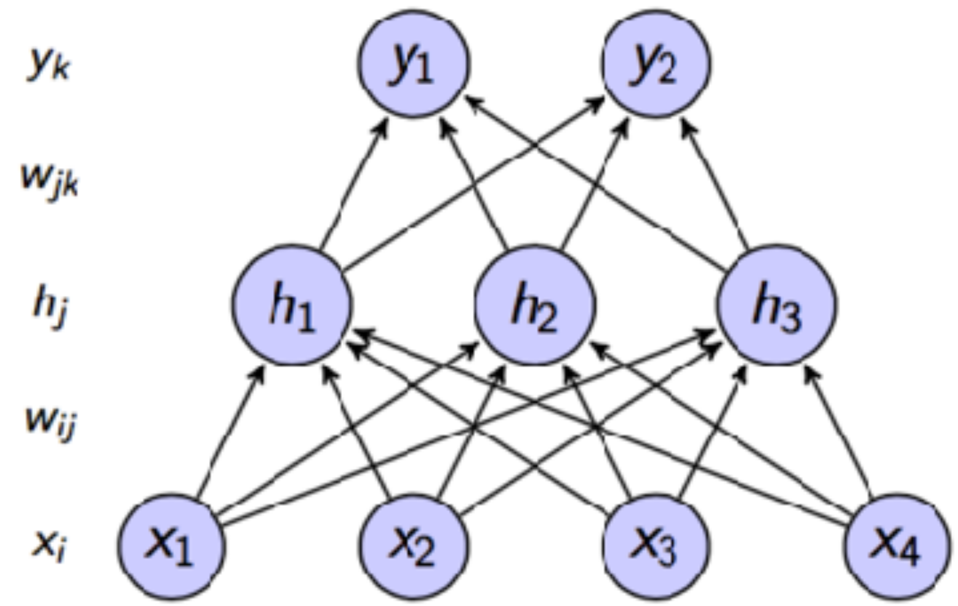
$$\frac{\partial Loss}{\partial w_{ij}} = \frac{\partial Loss}{\partial in_j}\frac{\partial in_j}{\partial w_{ij}} = \delta_j \frac{\partial\left(\sum_i w_{ij}x_i\right)}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_k = \frac{\partial}{\partial in_k}\left(\sum_k \frac{1}{2}[\sigma(in_k) - y_k]^2\right) = [\sigma(in_k) - y_k]\sigma'(in_k)$$

$$\delta_j = \sum_k \frac{\partial Loss}{\partial in_k}\frac{\partial in_k}{\partial in_j} = \sum_k \delta_k \cdot \frac{\partial}{\partial in_j}\left(\sum_j w_{jk}\sigma(in_j)\right) = [\sum_k \delta_k w_{jk}]\sigma'(in_j)$$

Back-propagation

$$Loss = \frac{1}{2}\sum_k (f(x_k) - y_k)^2$$
$$f(x_k) = \sigma(in_k)$$



Stochastic Gradient Descent:

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{\partial J(\theta^k)}{\partial \theta^k}$$

$$\frac{\partial Loss}{\partial w_{jk}} = \delta_k h_j = [\sigma(in_k) - y_k]\sigma'(in_k) h_j$$

$$\frac{\partial Loss}{\partial w_{ij}} = \delta_j x_i = [\sum_k \delta_k w_{jk}]\sigma'(in_j) x_i$$

Updates involve scaled error from output and input feature

After forward pass, compute δ_k from final layer and then δ_j for previous layer.

Overview

Machine Learning

Neural Networks

Automatic Differentiation

Supervised Learning

Automatic_differentiation.ipynb

About supervised learning

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$$

Parameters to learn

- Supervised learning is a natural starting point for any DL project
- Supervised training is stable (and provide an upper bound)
- Garbage-in-garbage-out: know your data!
- Supervised DL is interpolation (generalization issue)
- Fully connected layers (MLPs) vs CNN

Overview

Machine Learning

Neural Networks

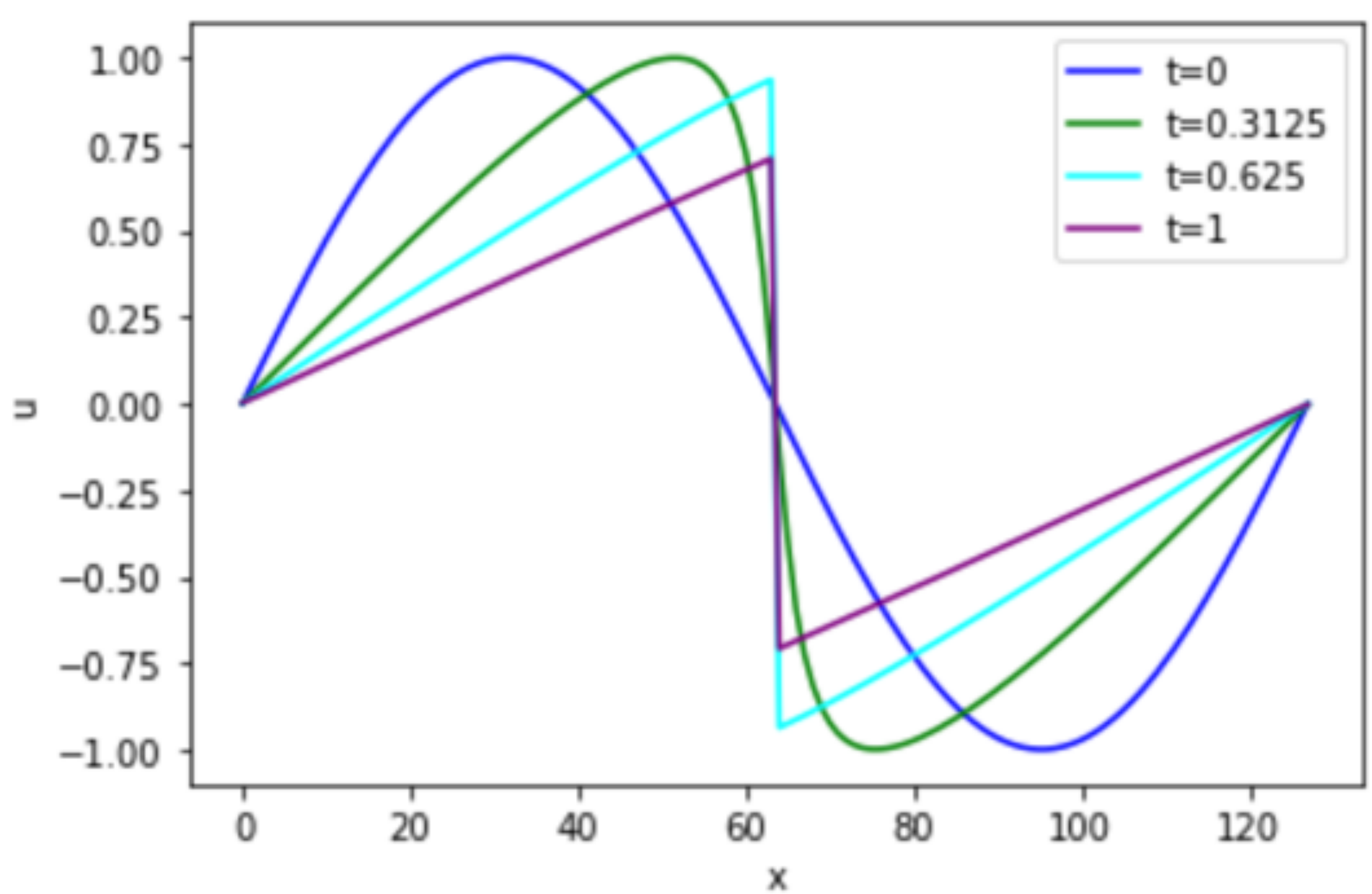
Automatic Differentiation

Supervised Learning

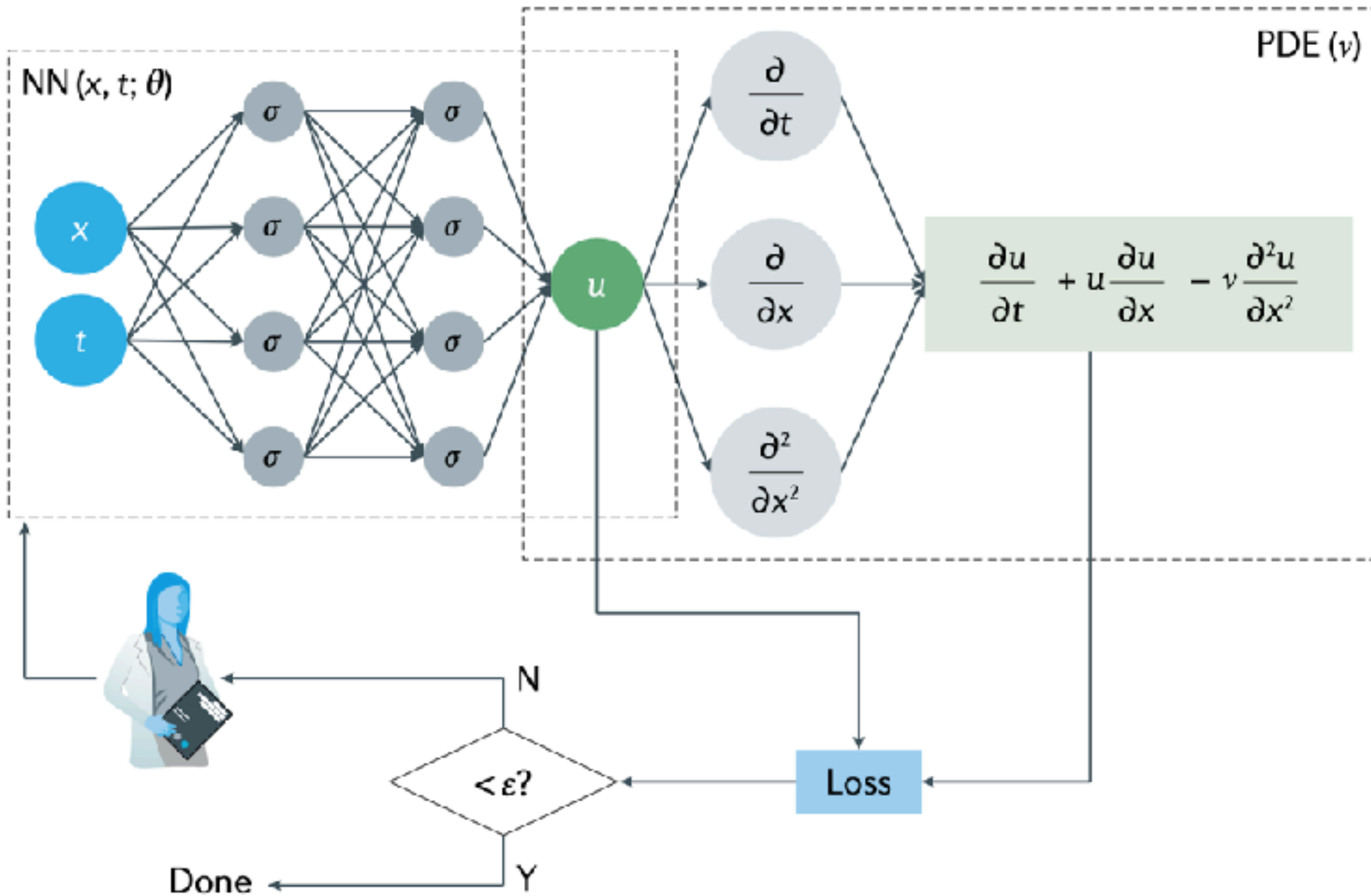
Physics-informed Learning

Burgers’ equation

$$\frac{\partial u}{\partial t} + u\nabla u = \nu \nabla \cdot \nabla u$$



Physics-informed NN



Karniadakis et al. Physics-informed machine learning. Nature Reviews Physics, 2021

Physics-informed NN

- Given a PDE for $\mathbf{u}(\mathbf{x}, t)$: $\mathbf{u}_t = \mathscr{F}(\mathbf{u}_\mathbf{x}, \mathbf{u}_{\mathbf{xx}}, \ldots)$

- To approximate $\mathbf{u}$ with a neural network, we want to minimize a residual term: $R = \mathbf{u}_t - \mathscr{F}(\mathbf{u}_\mathbf{x}, \mathbf{u}_{\mathbf{xx}}, \ldots)$

- In a supervised setting, the training objective becomes:

$$\arg\min_\theta \sum_i \alpha_0 (f(x_i; \theta) - y_i)^2 + \alpha_1 R(f(x_i; \theta))$$

Burgers_1D.ipynb

About Physics-informed NN

- Physical equations are included in the form of soft constraints
- Automatic differentiation can be use to compute high-order derivatives in PDEs
- Each derivative computation can be memory expensive (and slow)
- PINN is a inverse problem with physical regularization

Overview

Machine Learning

Neural Networks

Automatic Differentiation

Supervised Learning

Physics-informed Learning

Differentiable physics

Differentiable physics

- Consider a continuous formulation $\mathscr{P}(\mathbf{x}, \nu)$ of a physical quantity of interest $\mathbf{u}(\mathbf{x}, t) : \mathbb{R}^d \times \mathbb{R}^+ \rightarrow \mathbb{R}^d$.

- Assume that $\mathscr{P}(\mathbf{x}, \nu)$ can be decomposed as a sequence of operations $\mathscr{P}_1, \mathscr{P}_2, \ldots \mathscr{P}_m$ such that:

$$\mathbf{u}(t + \Delta t) = \mathscr{P}_1 \circ \mathscr{P}_2 \circ \ldots \circ \mathscr{P}_m(\mathbf{u}(t), \nu)$$

- Objective: find $\mathbf{u}$ that minimizes a loss given hard physical constraints (given by $\mathscr{P}$)

Differentiable physics

- Physical model $\mathscr{P}$: $$\frac{\partial d}{\partial t} + \mathbf{u} \nabla d = 0$$
- Evolution equation: $$d(t + \Delta t) = \mathscr{P}(d(t), \mathbf{u}, t + \Delta t)$$
- Objective: find a motion $\mathbf{u}$ that transform a given initial state d^0 to a target density d^{target} at final time t^e
- Minimization problem:

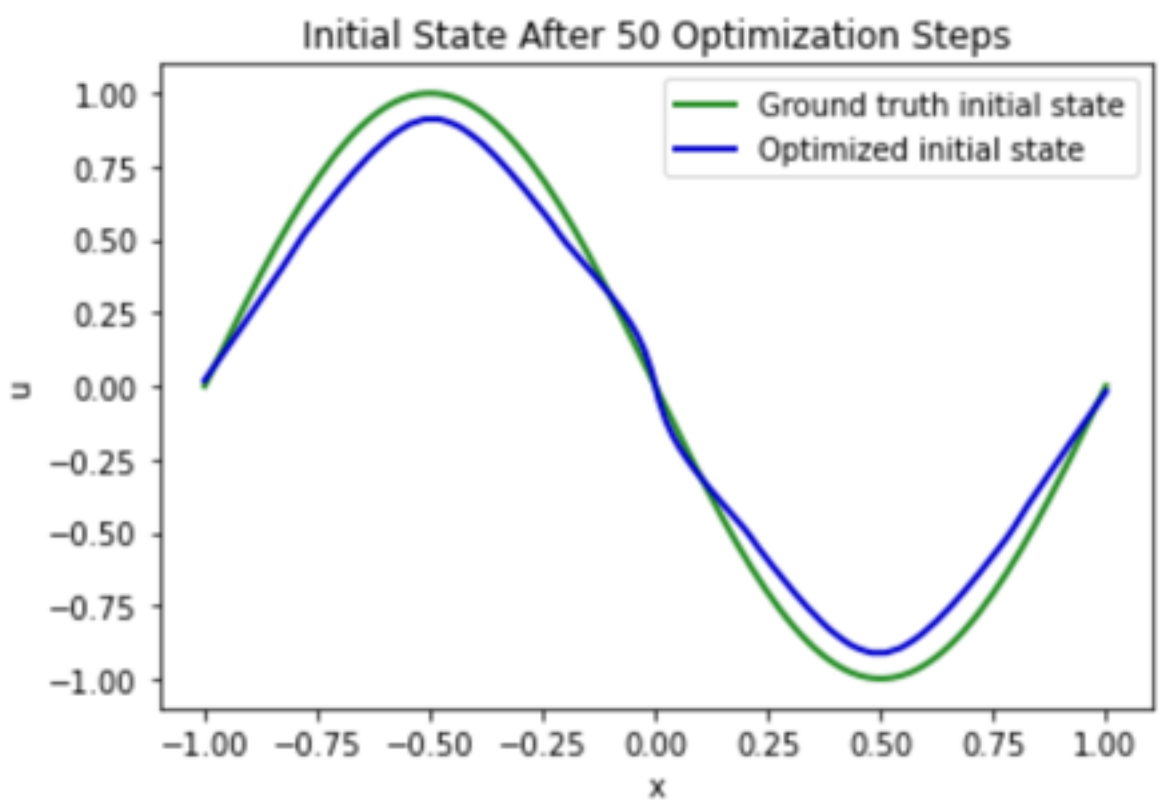
$$\arg\min_{\mathbf{u}} \|\mathscr{P}(d^0, \mathbf{u}, t^e) - d^{target}\|^2$$

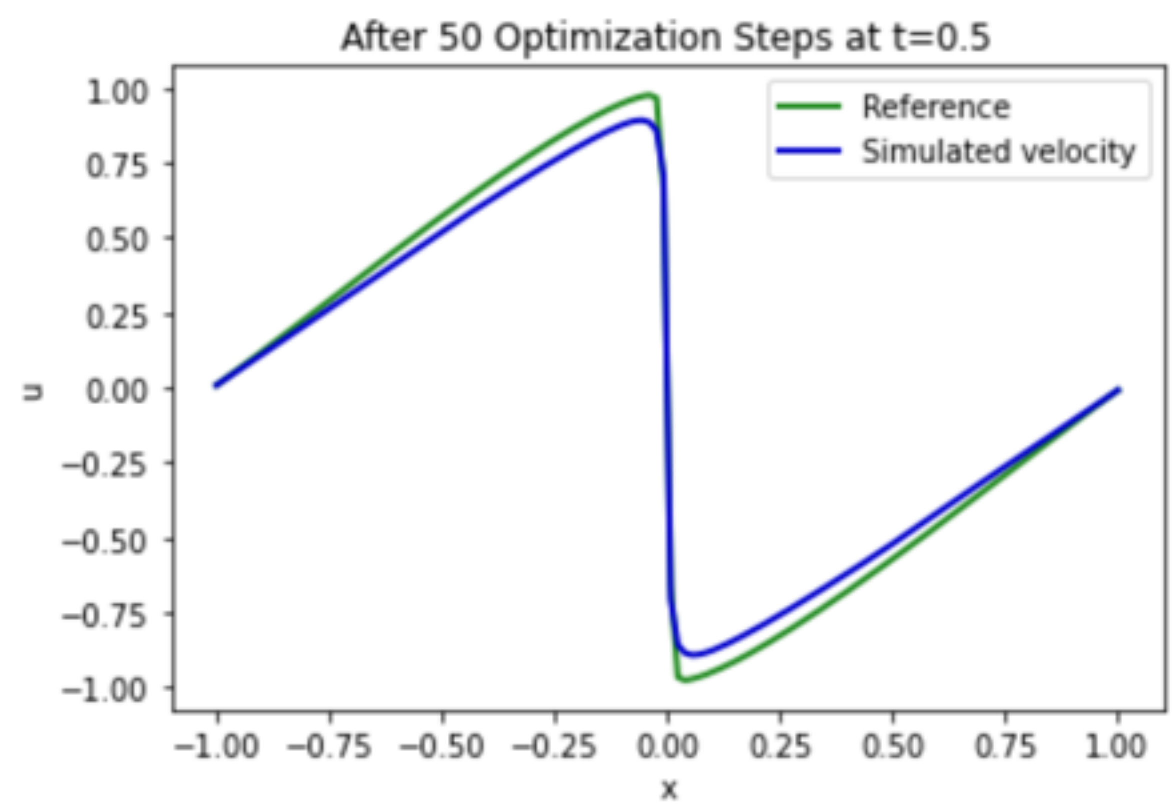
Burgers' equation and Differentiable physics

PhiFlow code

```
LR = 5.

grads=[]
for optim_step in range(5):
    velocities = [velocity]
    with math.record_gradients(velocity.values):
        for time_step in range(STEPS):
            v1 = diffuse.explicit(1.0*velocities[-1], NU, DT)
            v2 = advect.semi_lagrangian(v1, v1, DT)
            velocities.append(v2)

        loss = field.l2_loss(velocities[16] - SOLUTION_T16)*2./N # MSE
        print('Optimization step %d, loss: %f' % (optim_step,loss))

        grads.append( math.gradients(loss, velocity.values) )

    velocity = velocity - LR * grads[-1]
```

Burgers’ equation and Differentiable physics





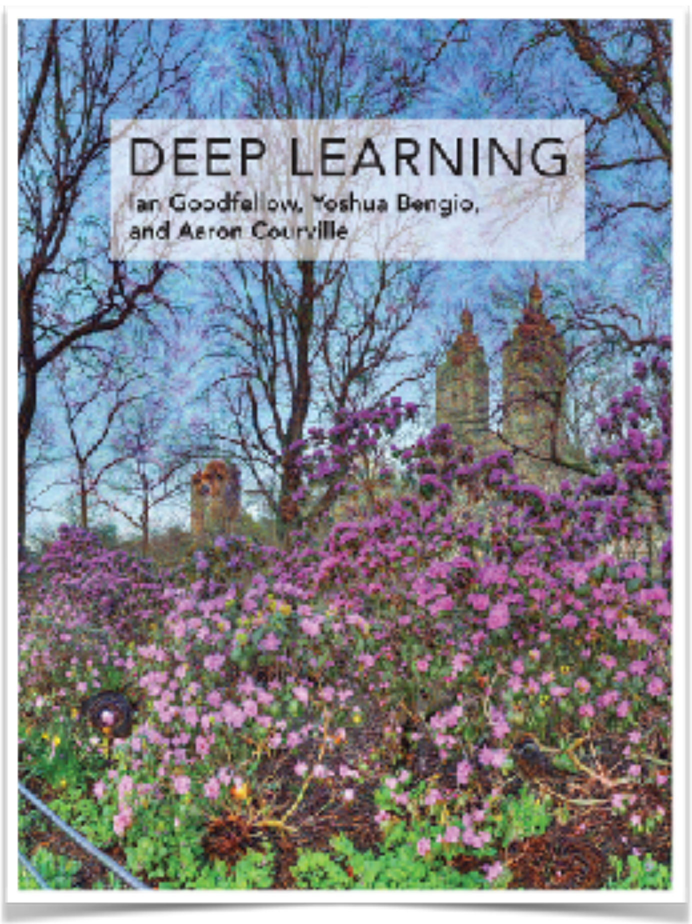
About Differentiable physics

- Makes use of physical model and numerical methods for discretization
- Efficient evaluation of simulation and derivatives
- More complicated implementation (than PINN)

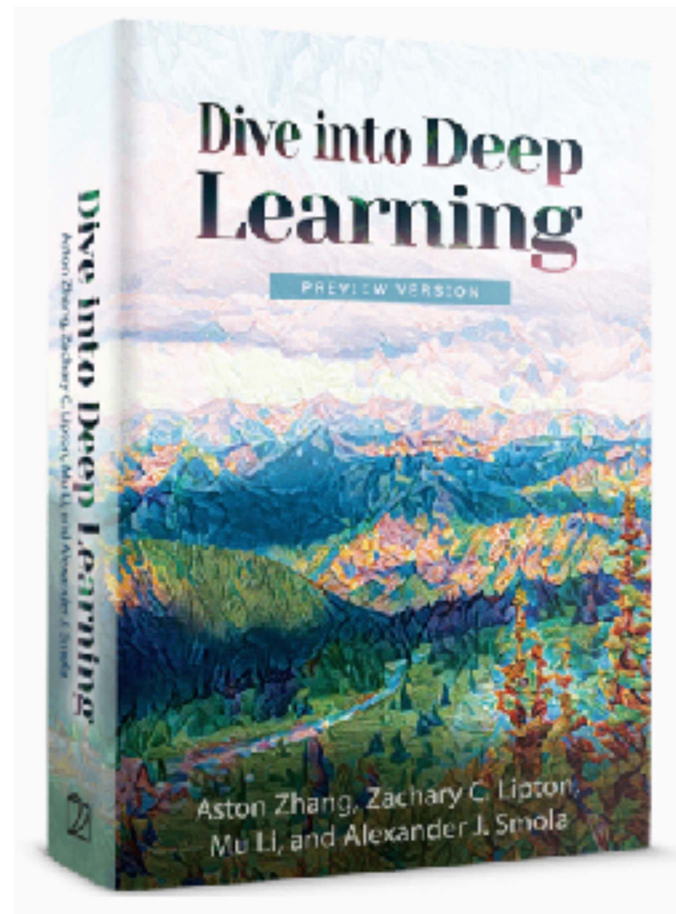
Take home messages

- Machine learning: data driven approaches
- Neural networks: composition of functions $f = f_n \circ \ldots \circ f_2 \circ f_1$
- Automatic differentiation: powerful tool for gradient computation
- Supervised learning for regression: interpolation
- PINN : add physical loss term (regularization term)
- Differentiable physics: unfold the physical equation

References



https://www.deeplearningbook.org



https://d2l.ai

https://www.physicsbased
deeplearning.org