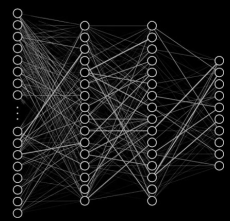
Learning physics by *learning* physics

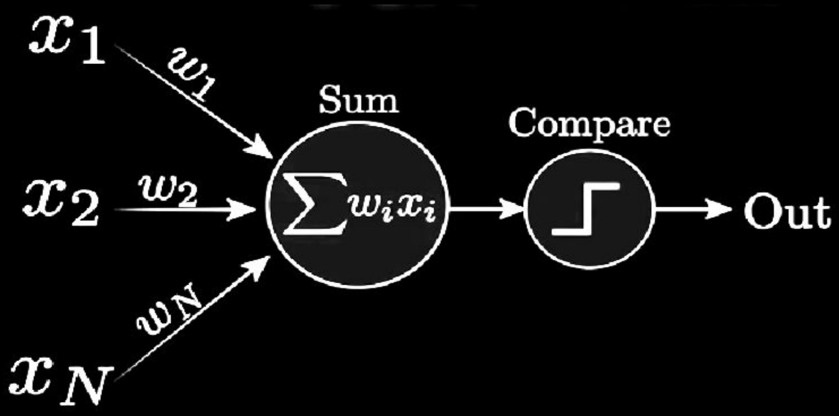


Lyle Kenneth Geraldez

GitHub repo: https://github.com/schwarzschlyle/numerical-relativity

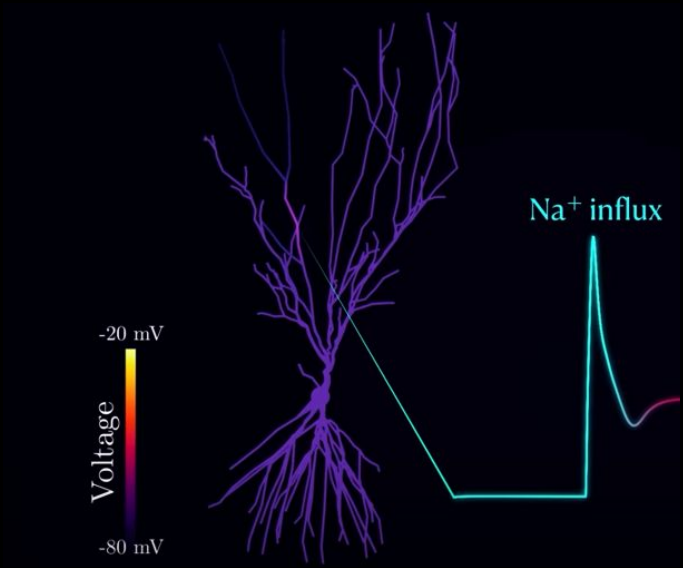
What is a neural network?

Perceptron



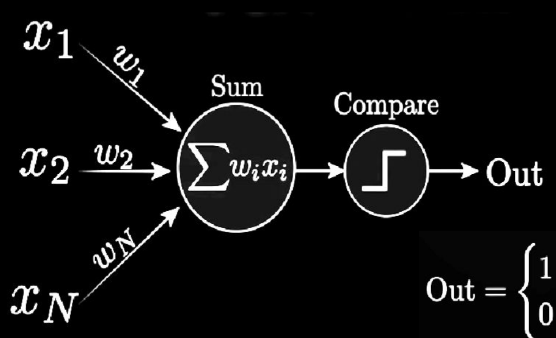
$$\Theta = x_1 w_1 + x_2 w_2 + \dots + x_N w_N$$

$$\text{Out} = \begin{cases} 1 \text{ if } \Theta \geq \text{threshold} \\ 0 \text{ if } \Theta < \text{threshold} \end{cases}$$



source: https://www.youtube.com/watch?v=hmtQPrH-gC4 (Dendrites: Why Biological Neurons Are Deep Neural Networks)

Neuron: Generalized Perceptron



$$\text{Out} = \begin{cases} 1 \text{ if } \Theta \geq \text{threshold} \\ 0 \text{ if } \Theta < \text{threshold} \end{cases}$$

$$\Theta = x_1 w_1 + x_2 w_2 + ... + x_N w_N$$

Activation of a Neuron

$$\sigma(\mathbf{W}^\ell \mathcal{N}^{\ell-1}(\mathbf{x}) + \mathbf{b}^\ell) \in \mathbb{R}^{N_\ell}$$

Step	Sigmoid	Tanh	ReLU	Leaky ReLU	Softmax
					$f(x_j) = \frac{e^{x_j}}{\sum_j e^{x}}$
Linear	Non -Linear	Non-Linear	Non -Linear	Non-Linear	Non-Linear
Non-Differentiable	Differentiable	Differentiable	Differentiable	Differentiable	Differentiable
-	Supports Backpropagation	Supports Backpropagation	Supports Backpropagation	Supports Backpropagation	Supports Backpropagation
-	Vanishing Gradient Problem	Vanishing Gradient Problem	Dying Neuron Problem	-	-
Not used in Deep Neural Networks	Suitable in Output Layer for Binary Classification	Not much popular now	Suitable in Hidden layers	Suitable in Hidden layers	Suitable in Output layer for Multiclass Classification

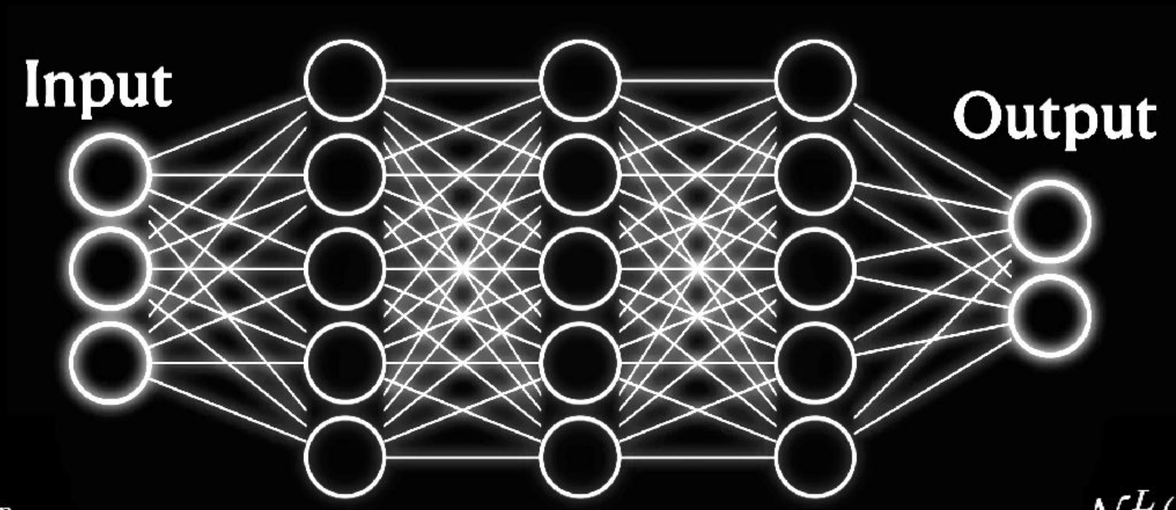
source: https://www.youtube.com/watch?v=hmtQPrH-gC4 (Dendrites: Why Biological Neurons Are Deep Neural Networks)

Neural Network

$\mathcal{N}^L(x;\theta(W,b))$

$\mathcal{N}^L(\mathbf{x}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ $\quad f(\vec{x}) = f_{L-1}(f_{\dots}(f_2(f_1(\vec{x}))))$

$$\mathcal{N}^\ell(\mathbf{x}) = \sigma(\boldsymbol{W}^\ell \mathcal{N}^{\ell-1}(\mathbf{x}) + \boldsymbol{b}^\ell) \in \mathbb{R}^{N_\ell} \quad \text{for} \quad 1 \leq \ell \leq L-1$$



$l = 0$

$l = L$

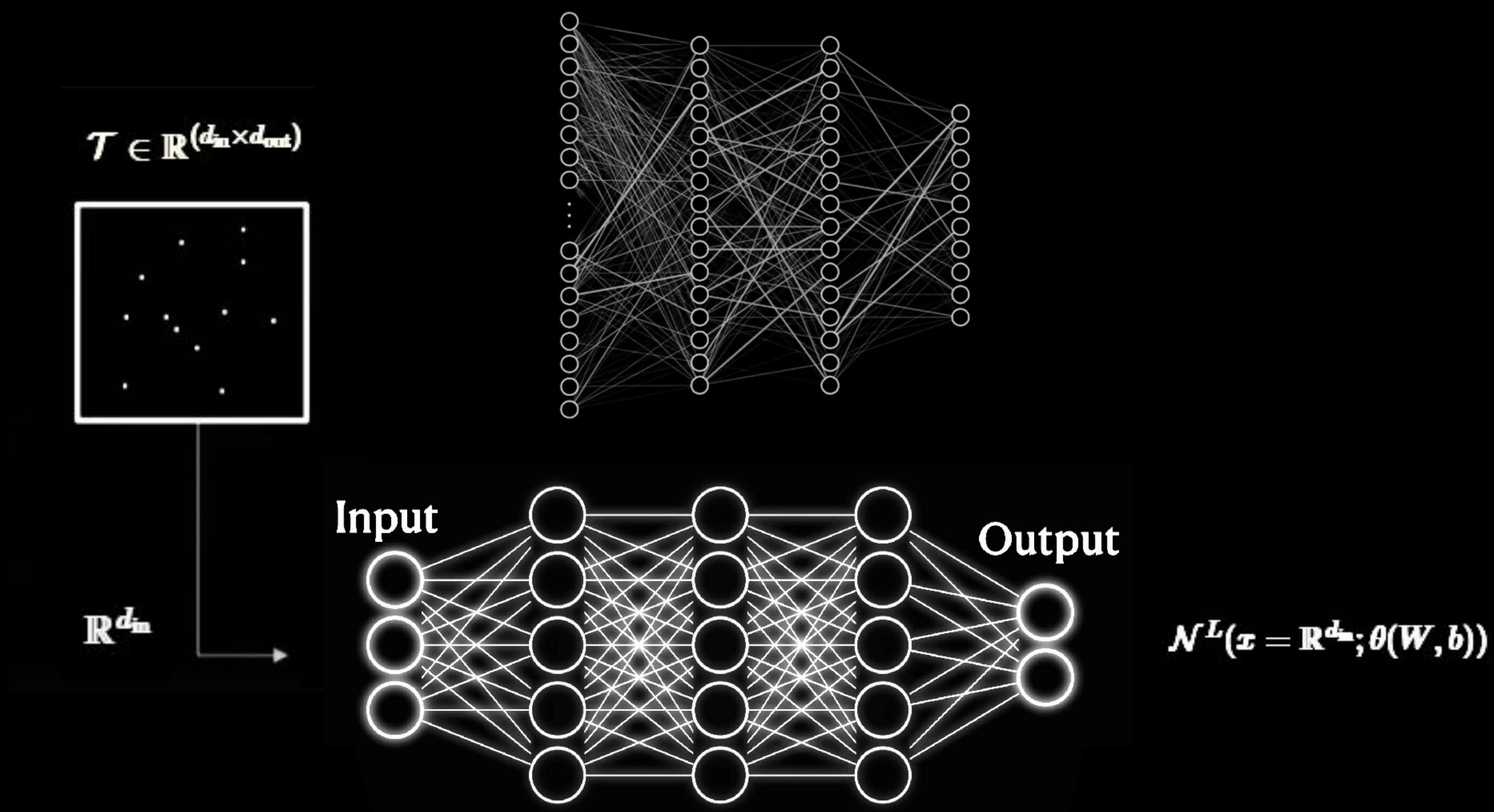
$\mathcal{N}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$

$$\mathcal{N}^L(\mathbf{x}) = \boldsymbol{W}^L \mathcal{N}^{L-1}(\mathbf{x}) + \boldsymbol{b}^L \in \mathbb{R}^{d_{\text{out}}}$$

$l = 1$ $\quad l = 2$ $\quad l = L-1$

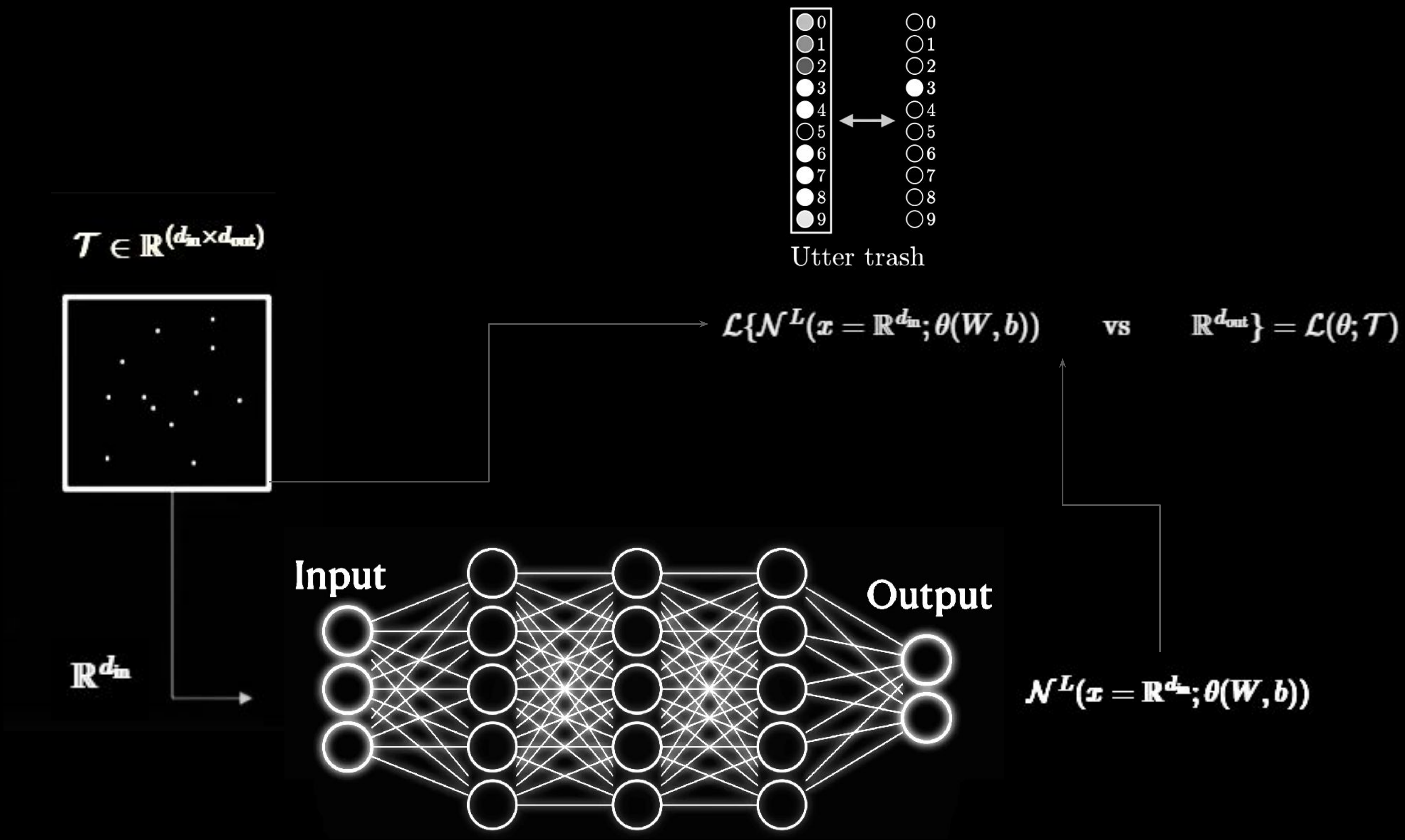
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Learning Status: Feeding...



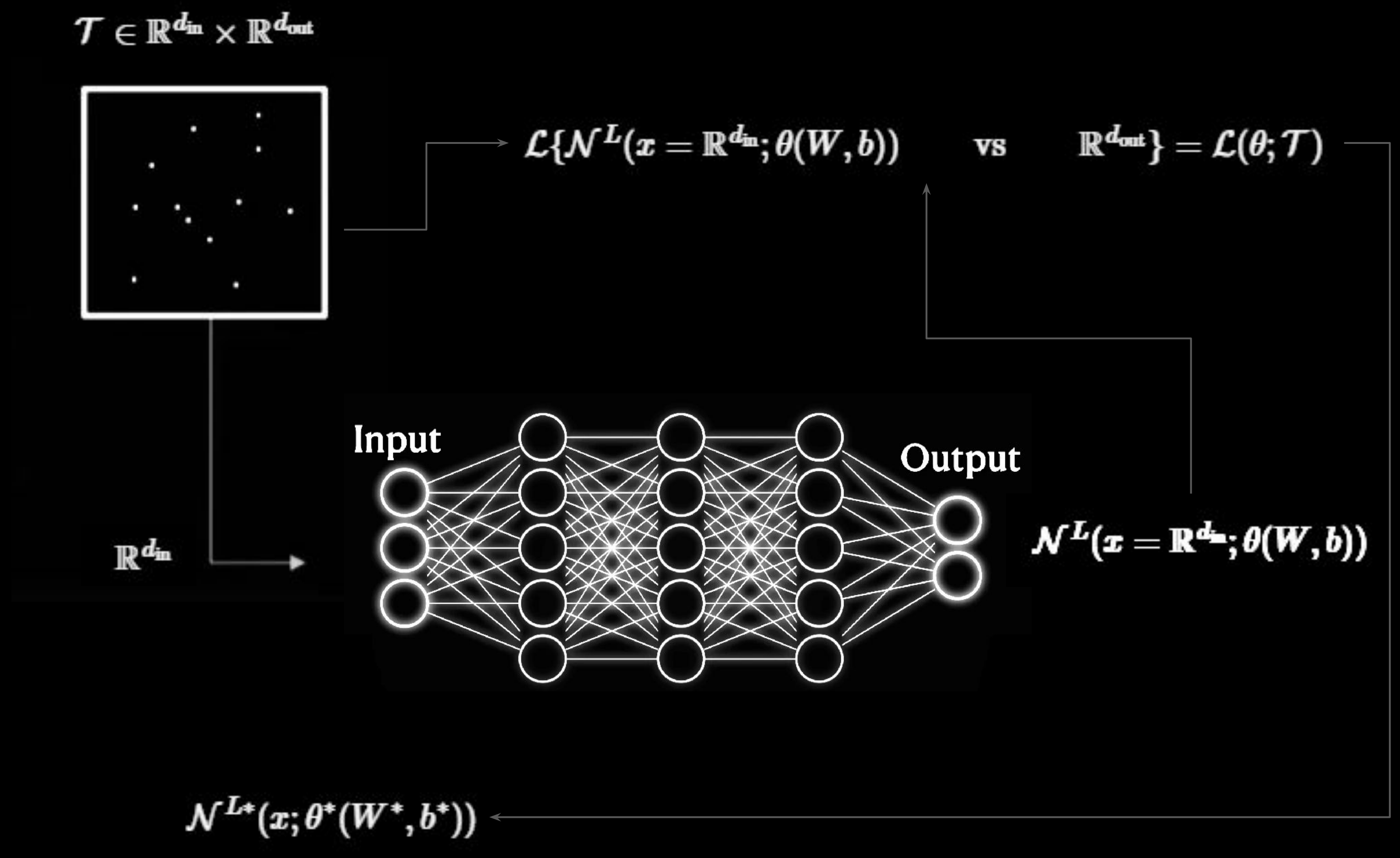
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Learning Status: Assessing...



Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Learning Status: Correcting...




```
Iteration 1: loss 4.4160
Iteration 2: loss 3.3112
Iteration 3: loss 2.8802
```

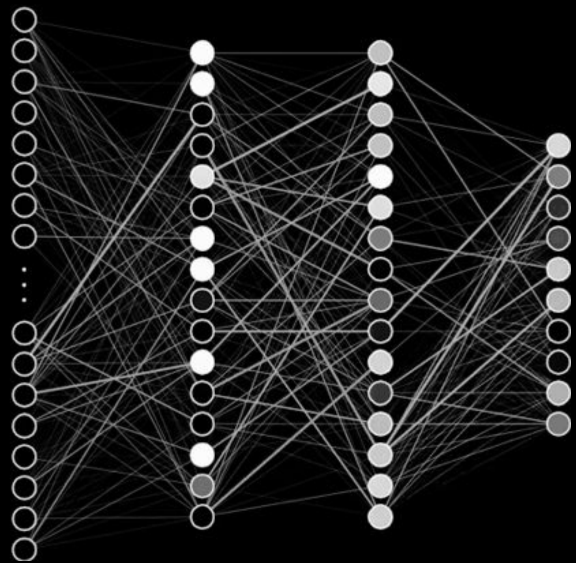
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Learning Status: Correcting...

Gradient Descent Optimization

$$w^l_{jk} \rightarrow w^l_{jk} - \frac{\eta}{n} \sum_x \frac{\partial C_x}{\partial w^l_{jk}}$$

$$b^l_j \rightarrow b^l_j - \frac{\eta}{n} \sum_x \frac{\partial C_x}{\partial b^l_j}$$

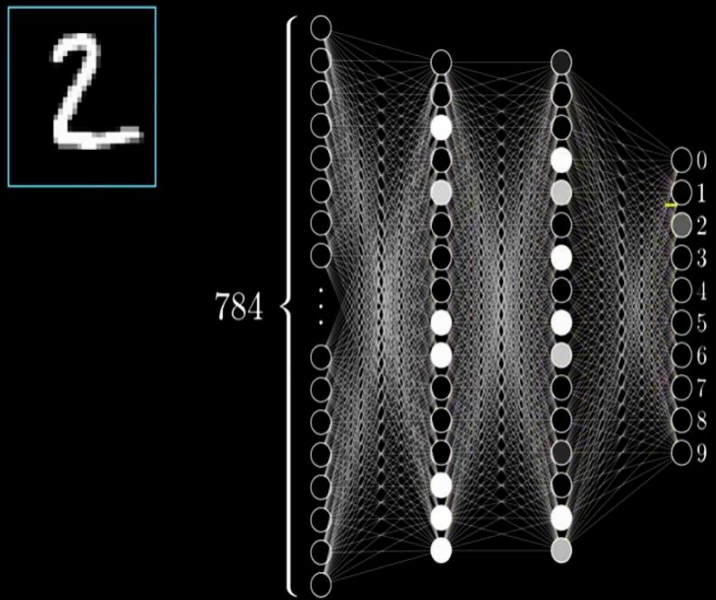


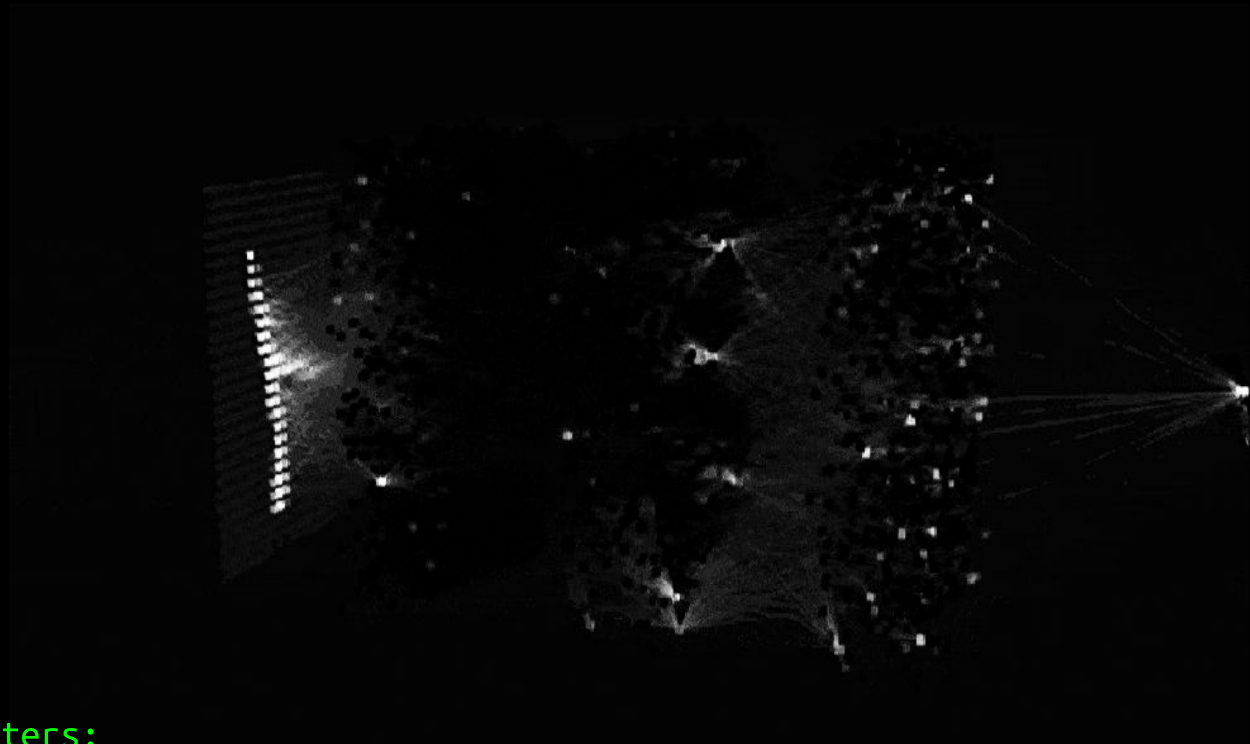
Backpropagation

$$\begin{aligned}
\delta^L_j &= \frac{\partial C_x}{\partial a^L_j} \sigma'(z^L_j), \\
\delta^\ell_j &= \sum_k w^{\ell+1}_{kj} \delta^{\ell+1}_k \sigma'(z^\ell_j), \\
\frac{\partial C_x}{\partial b^\ell_j} &= \delta^\ell_j, \\
\frac{\partial C_x}{\partial w^\ell_{jk}} &= \sum_k a^{\ell-1}_k \delta^\ell_j.
\end{aligned}$$

Source: https://www.youtube.com/watch?v=IHZwWFHWa-w&t=507s (Gradient descent, how neural networks learn | Chapter 2, Deep learning

The “Hello World” of NN





Parameters:

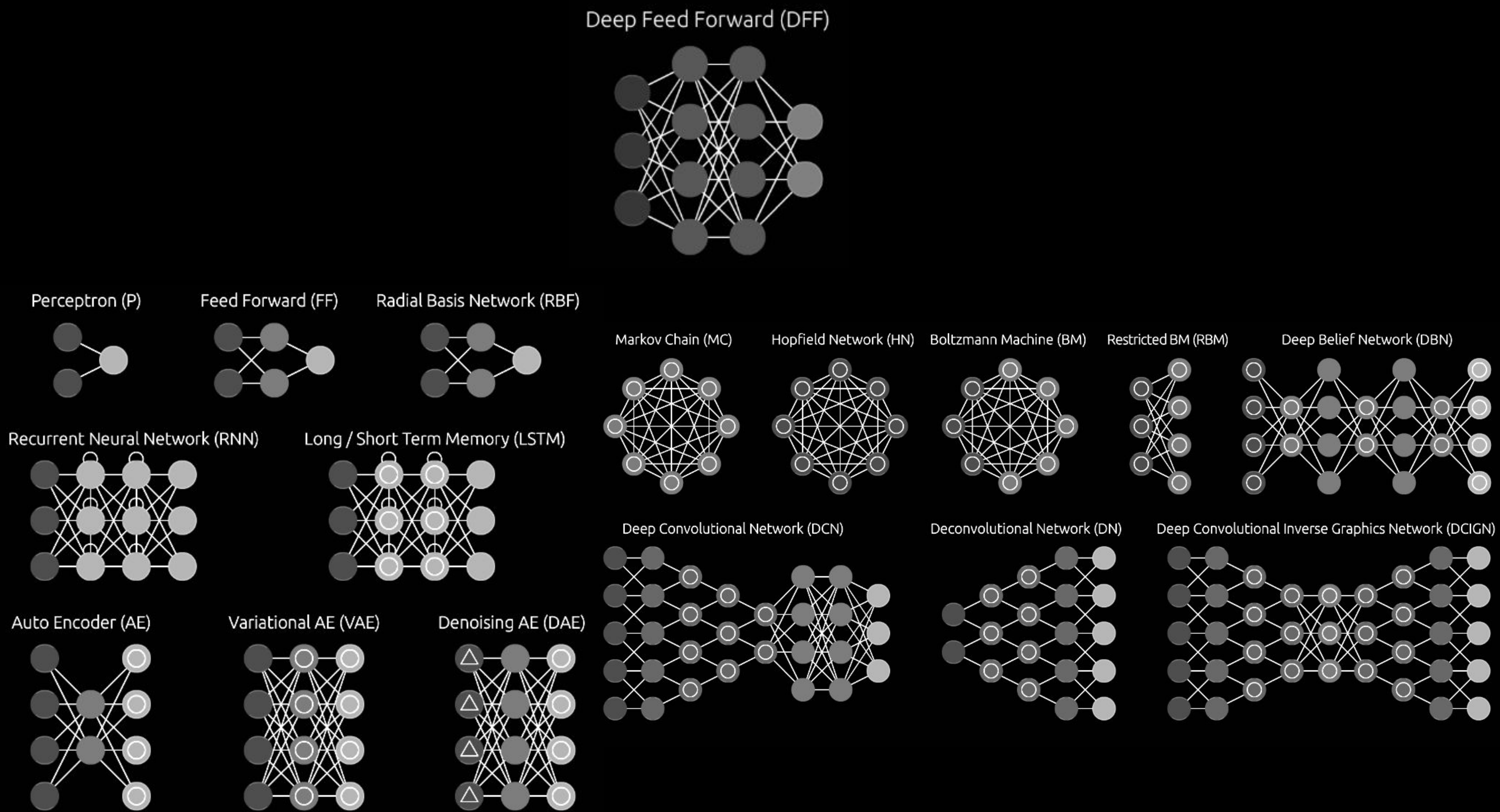
1. Weights
2. Biases

Hyperparameters:

1. Hidden layers
2. Neurons per layer
3. Learning rate
4. Activation function
5. Epochs
6. Optimizer
7. Initializer
8. Regularization parameters
9. Dropout rate

Source: https://www.youtube.com/watch?v=IHZwWFHWa-w&t=507s (Gradient descent, how neural networks learn | Chapter 2, Deep learning

Neural Network Architectures



Source: https://www.asimovinstitute.org/neural-network-zoo/ (Neural Network Zoo)

Universal Function Approximation Theorem

THEOREM 2.1. *Let* $\mathbf{m}^i \in \mathbb{Z}_+^d$, $i = 1, \ldots, s$, *and set* $m = \max_{i=1,\ldots,s} |\mathbf{m}^i|$. *Assume* $\sigma \in C^m(\mathbb{R})$ *and that* σ *is not a polynomial. Then the space of single hidden layer neural nets*

$$\mathcal{M}(\sigma) := \operatorname{span}\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in

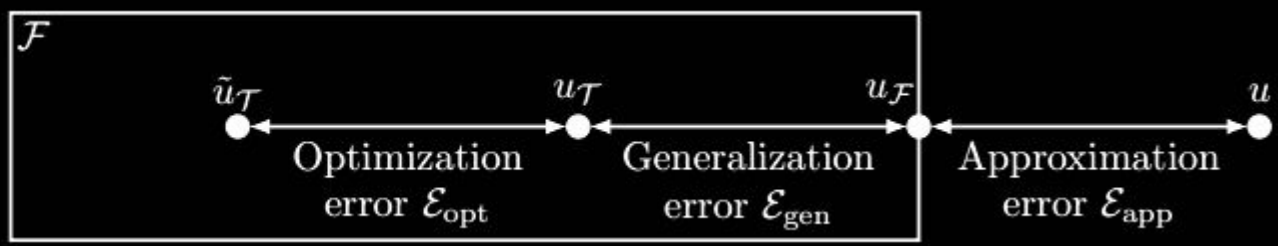
$$C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^s C^{\mathbf{m}^i}(\mathbb{R}^d),$$

i.e., for any $f \in C^{\mathbf{m}^1,\ldots,\mathbf{m}^s}(\mathbb{R}^d)$, *any compact* $K \subset \mathbb{R}^d$, *and any* $\varepsilon > 0$, *there exists a* $g \in \mathcal{M}(\sigma)$ *satisfying*

$$\max_{\mathbf{x} \in K} |D^{\mathbf{k}} f(\mathbf{x}) - D^{\mathbf{k}} g(\mathbf{x})| < \varepsilon$$

for all $\mathbf{k} \in \mathbb{Z}_+^d$ *for which* $\mathbf{k} \le \mathbf{m}^i$ *for some* i.

Given sufficient neurons, neural networks can approximate any function and its partial derivatives



$$\mathcal{E} := \|\tilde{u}_\mathcal{T} - u\| \le \underbrace{\|\tilde{u}_\mathcal{T} - u_\mathcal{T}\|}_{\mathcal{E}_{\text{opt}}} + \underbrace{\|u_\mathcal{T} - u_\mathcal{F}\|}_{\mathcal{E}_{\text{gen}}} + \underbrace{\|u_\mathcal{F} - u\|}_{\mathcal{E}_{\text{app}}}.$$

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Derivatives

How to differentiate? Manual

Manual differentiation

Problem: Impractical

$$f(x) = e^{2x} - x^3 \longrightarrow f'(x) = 2e^{2x} - 3x^2$$

```
def f(x):
    return np.exp(2*x) - x**3
```

```
def f_prime(x):
    return 2*np.exp(2*x) - 3*x**2
```

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

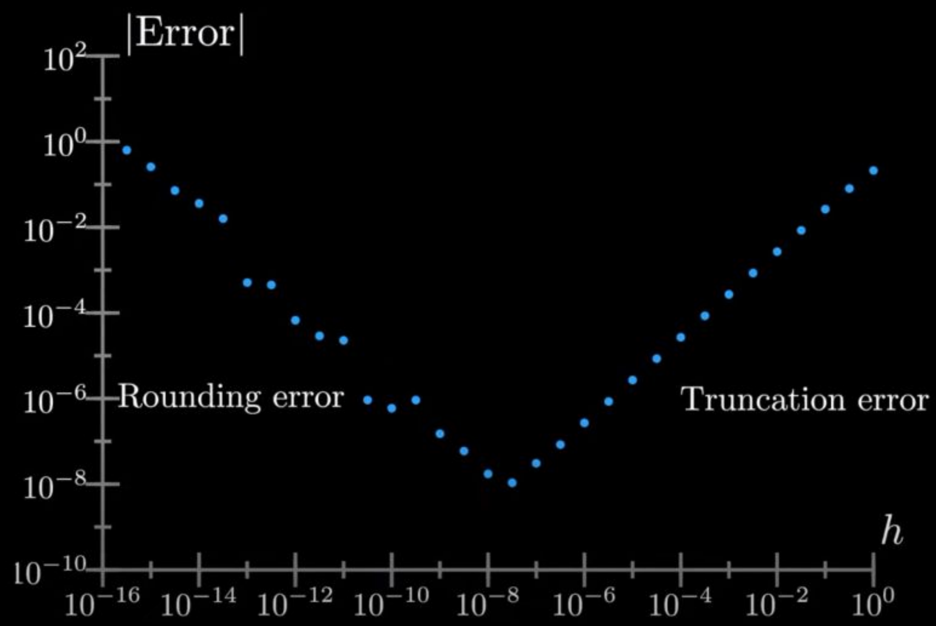
How to differentiate? Numerical

Finite differences

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$\frac{\partial f}{\partial x_i} \approx \frac{f(x + h\mathrm{e}_i) - f(x)}{h}$$

Problem: Truncation and Round-off Errors



Requires $O(n)$ evaluations:

$$\mathrm{e}_1, \mathrm{e}_2, \mathrm{e}_3, \ldots, \mathrm{e}_n$$

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

How to differentiate? Symbolic

Symbolic differentiation

Obtain $\frac{dz}{dx}$

```
[4]
dzdx = z.diff(x)
print('dzdx =', dzdx)

dzdx = y*cos(x)
```

Soft ReLU (Softplus) $\log(1 + e^{wx+b})$

Problem: Expression Swell

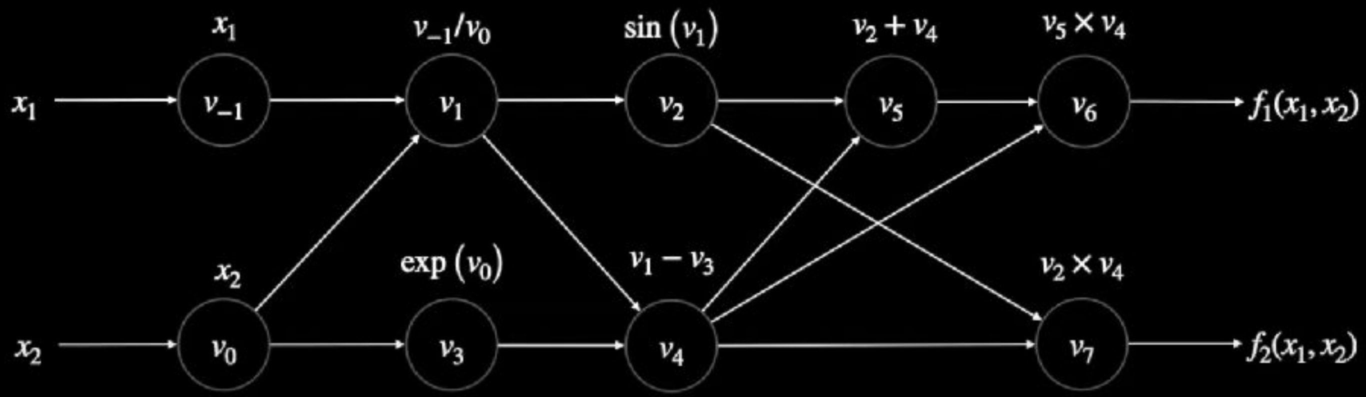
Derivative wrt w_1 for two layers

$$\frac{e^{b_1+b_2+w_1x+w_2\log\left[1+e^{b_1+w_1x}\right]}w_2x}{\left(1+e^{b_1+w_1x}\right)\left(1+e^{b_2+w_2\log\left[1+e^{b_1+w_1x}\right]}\right)}$$

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

How to differentiate? Automatic (Forward)

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2}\right] \times \left[\frac{x_1}{x_2} - e^{x_2}\right]$$



$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Primals

$v_{-1} = x_1$	$= 1.500$
$v_0 = x_2$	$= 0.500$
$v_1 = v_{-1}/v_0$	$= 3.000$
$v_2 = \sin(v_1)$	$= 0.141$
$v_3 = \exp(v_0)$	$= 1.649$
$v_4 = v_1 - v_3$	$= 1.351$
$v_5 = v_2 + v_4$	$= 1.492$
$v_6 = v_5 \times v_4$	$= 2.017$
$f(x_1, x_2) = v_6$	$= 2.017$

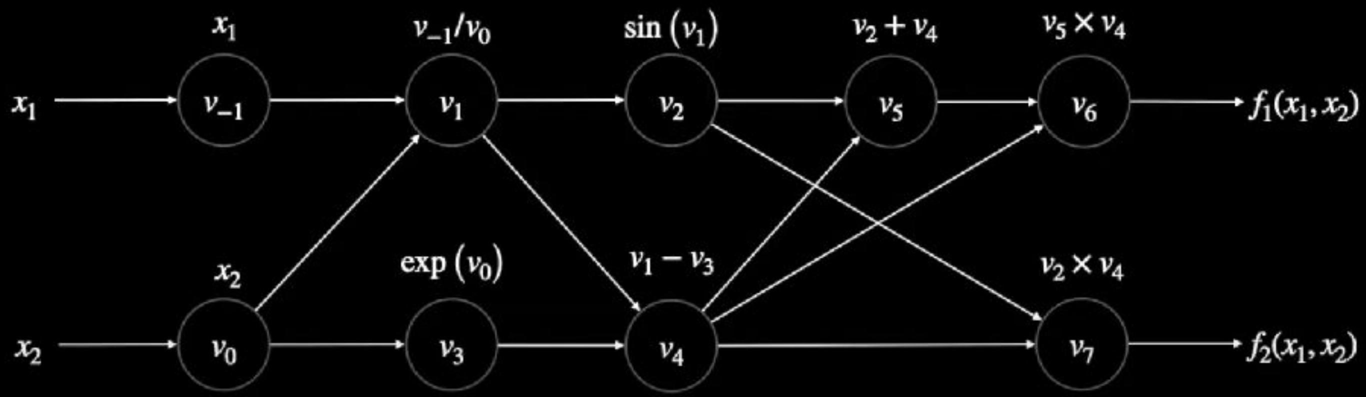
Tangents

$\dot{v}_{-1}$	$= 1.000$
$\dot{v}_0$	$= 0.000$
$\dot{v}_1 = (v_0 \dot{v}_{-1} - v_{-1} \dot{v}_0)/v_0^2$	$= 2.000$
$\dot{v}_2 = \cos(v_1) \times \dot{v}_1$	$= -1.980$
$\dot{v}_3 = v_3 \times \dot{v}_0$	$= 0.000$
$\dot{v}_4 = \dot{v}_1 - \dot{v}_3$	$= 2.000$
$\dot{v}_5 = \dot{v}_2 + \dot{v}_4$	$= 0.020$
$\dot{v}_6 = \dot{v}_5 v_4 - \dot{v}_4 v_5$	$= 3.012$
$\frac{\partial f}{\partial x_1} = \dot{v}_6$	$= 3.012$

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

How to differentiate? Automatic (Reverse)

$$f(x_1, x_2) = \left[\sin\left(\frac{x_1}{x_2}\right) + \frac{x_1}{x_2} - e^{x_2}\right] \times \left[\frac{x_1}{x_2} - e^{x_2}\right]$$



$$f : \mathbb{R}^n \to \mathbb{R}^m$$

$$J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Primals

$$\begin{aligned} v_{-1} &= x_1 & &= 1.500 \\ v_0 &= x_2 & &= 0.500 \\ v_1 &= v_{-1}/v_0 & &= 3.000 \\ v_2 &= \sin(v_1) & &= 0.141 \\ v_3 &= \exp(v_0) & &= 1.649 \\ v_4 &= v_1 - v_3 & &= 1.351 \\ v_5 &= v_2 + v_4 & &= 1.492 \\ v_6 &= v_5 \times v_4 & &= 2.017 \\ f(x_1, x_2) &= v_6 & &= 2.017 \end{aligned}$$

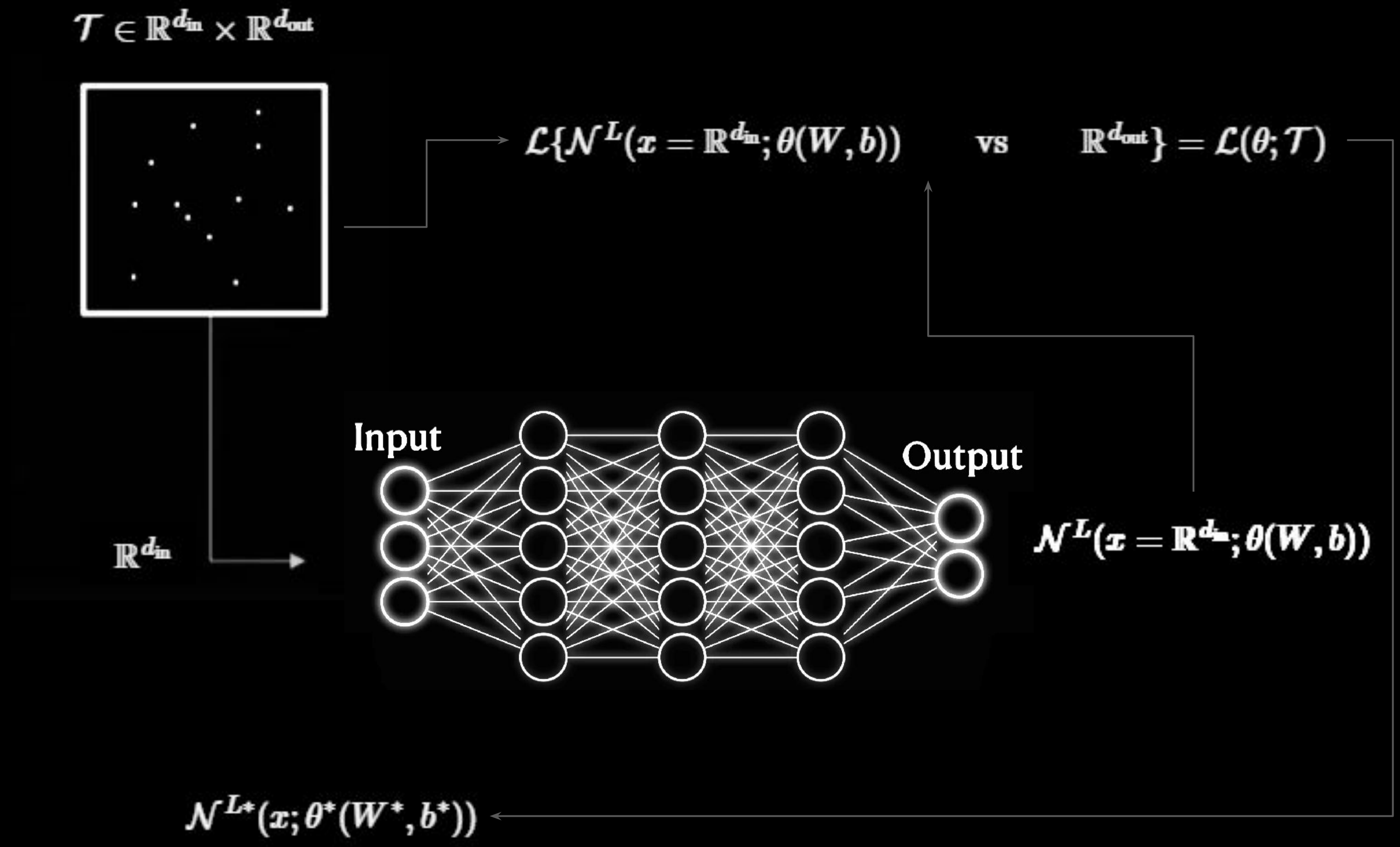
Adjoints

$$\begin{aligned} \bar{v}_6 &= \bar{f} & &= 1.000 \\ \bar{v}_5 &= v_4 \times \bar{v}_6 & &= 1.351 \\ \bar{v}_4 &= v_5 \times \bar{v}_6 + \bar{v}_5 & &= 2.844 \\ \bar{v}_3 &= -\bar{v}_4 & &= -2.844 \\ \bar{v}_2 &= \bar{v}_5 & &= 1.351 \\ \bar{v}_1 &= \bar{v}_2 \cos(v_1) + \bar{v}_4 & &= 1.506 \\ \bar{v}_0 &= \bar{v}_3 v_3 - \bar{v}_1 v_{-1}/v_0^2 & &= -13.724 \\ \bar{v}_{-1} &= \bar{v}_1/v_0 & &= 3.012 \\ \bar{x}_2 &= \bar{v}_0 & &= -13.724 \\ \bar{x}_1 &= \bar{v}_{-1} & &= 3.012 \end{aligned}$$

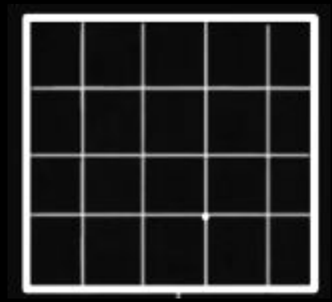
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Teaching it some physics

NN PDE Solution: Data-driven



Loss function is the absolute difference (squared) of the deviation between known numerical or analytical solution



For mesh-based numerical solution, we coincide the collocation points with the grid points

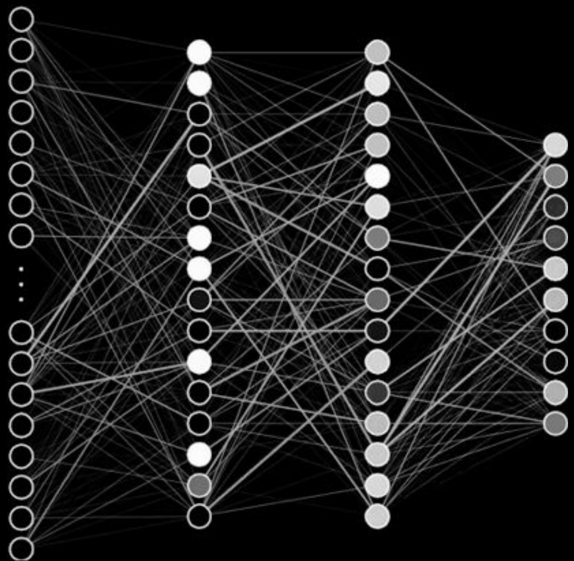
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Backpropagation is Automatic Differentiation

Gradient Descent Optimization

$$w^l_{jk} \rightarrow w^l_{jk} - \frac{\eta}{n} \sum_x \frac{\partial C_x}{\partial w^l_{jk}}$$

$$b^l_j \rightarrow b^l_j - \frac{\eta}{n} \sum_x \frac{\partial C_x}{\partial b^l_j}$$



Backpropagation

$$\begin{aligned} \delta^L_j &= \frac{\partial C_x}{\partial a^L_j} \sigma'(z^L_j), \\ \delta^\ell_j &= \sum_k w^{\ell+1}_{kj} \delta^{\ell+1}_k \sigma'(z^\ell_j), \\ \frac{\partial C_x}{\partial b^\ell_j} &= \delta^\ell_j, \\ \frac{\partial C_x}{\partial w^\ell_{jk}} &= \sum_k a^{\ell-1}_k \delta^\ell_j. \end{aligned}$$

NN PDE Solution: Physics-informed

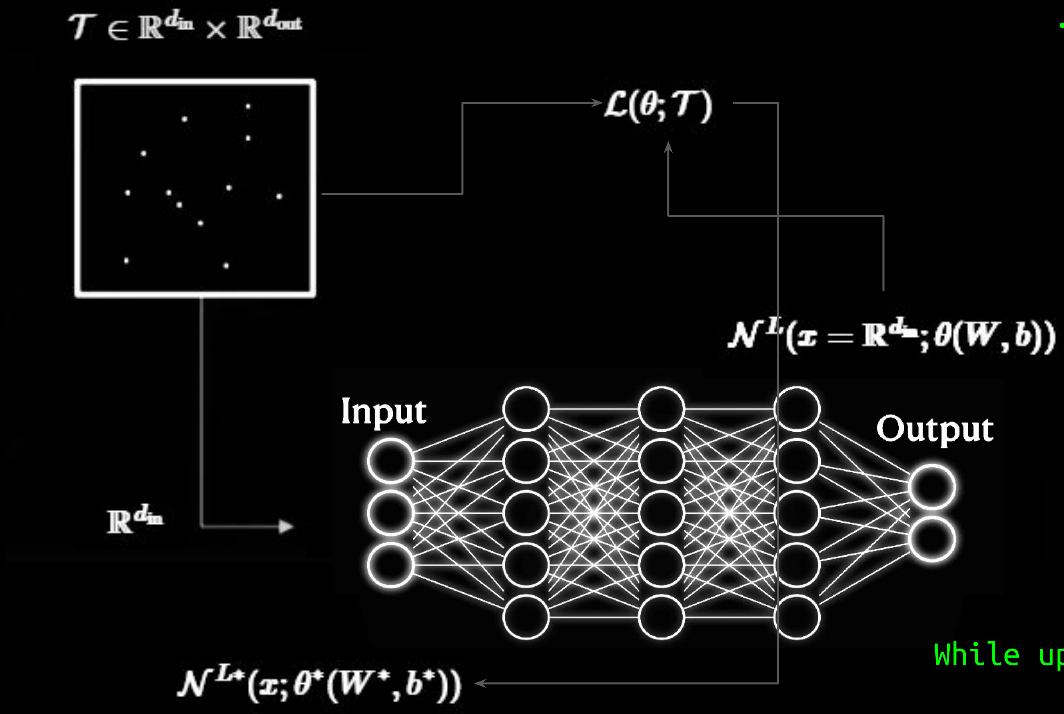
Training set consists of initial conditions and boundary conditions

The physics...

PDE + Boundaries

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega$$

$$\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on} \quad \partial\Omega$$



...is incorporated into the loss function

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b)$$

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) \right\|_2^2$$

$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2,$$

recasting solving the PDE into an optimization problem

$$\vec{u} \approx \hat{\vec{u}}(\theta^*), \qquad \theta = \{\vec{w}^l, B^l\}_{1 \le l \le L} \qquad \theta^* = \operatorname{argmin}_\theta \mathcal{L}(\theta; \mathcal{T})$$

While updating the collocation points via adaptive residual refinement

$$\mathcal{E}_r \approx \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \ldots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) \right\|$$

Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

PINN with DeepXDE: Heat Equation PDE

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

$$\Omega = [0,1]^2$$

$$u(0,t) = u(1,t) = 0$$

$$u(x,0) = \sin(\frac{n\pi x}{L}), \qquad 0 < x < L, \quad n = 1, 2, \ldots.$$

```python
def pde(x, y):
    """Expresses the PDE residual of the heat equation."""
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t - a * dy_xx
```

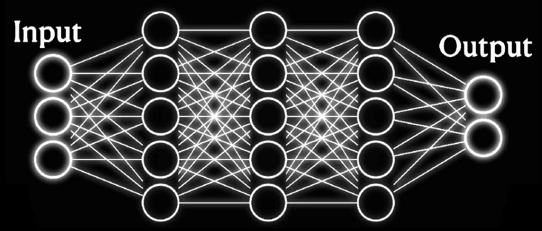
```python
# Computational geometry:
geom = dde.geometry.Interval(0, L)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

```python
# Initial and boundary conditions:
bc = dde.icbc.DirichletBC(
geomtime,
lambda x: 0,
lambda _,
on_boundary: on_boundary)
ic = dde.icbc.IC(
    geomtime,
    lambda x: np.sin(n * np.pi * x[:, 0:1] / L),
    lambda _, on_initial: on_initial,
)
```

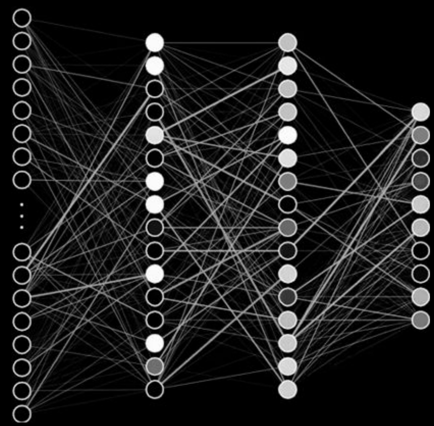
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

PINN with DeepXDE: Heat Equation PDE

```
# Define the PDE problem and configurations of the network:
data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic],
    num_domain=2540,
    num_boundary=80,
    num_initial=160,
    num_test=2540,
)
net = dde.nn.FNN([2] + [20] * 3 + [1], "tanh", "Glorot normal")
model = dde.Model(data, net)
```



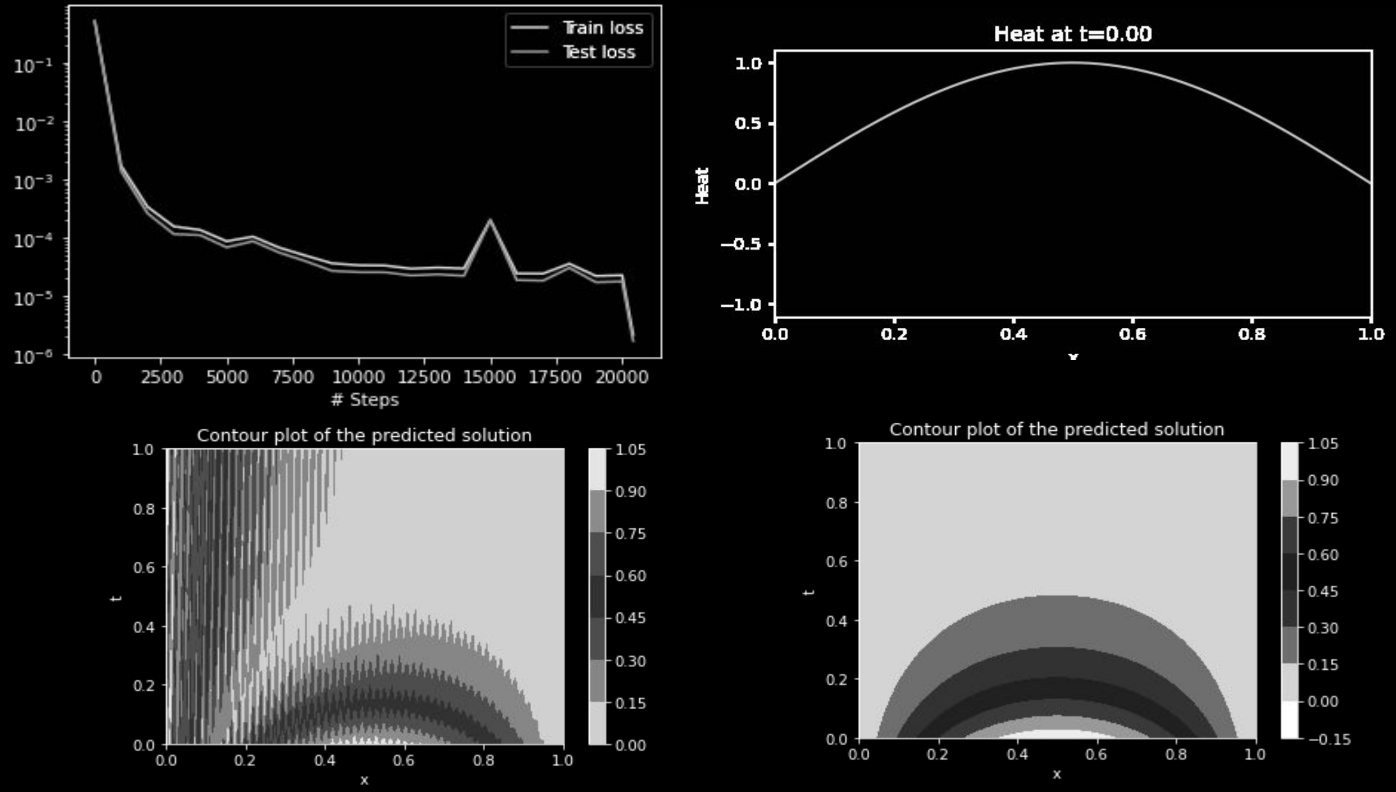

```
# Build and train the model:
model.compile("adam", lr=1e-3)
model.train(iterations=20000)
model.compile("L-BFGS")
losshistory, train_state = model.train()
```



Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

PINN with DeepXDE: Heat Equation PDE

```python
# Plot/print the results
dde.saveplot(losshistory, train_state, issave=True, isplot=True)
X, y_true = gen_testdata()
y_pred = model.predict(X)
f = model.predict(X, operator=pde)
```



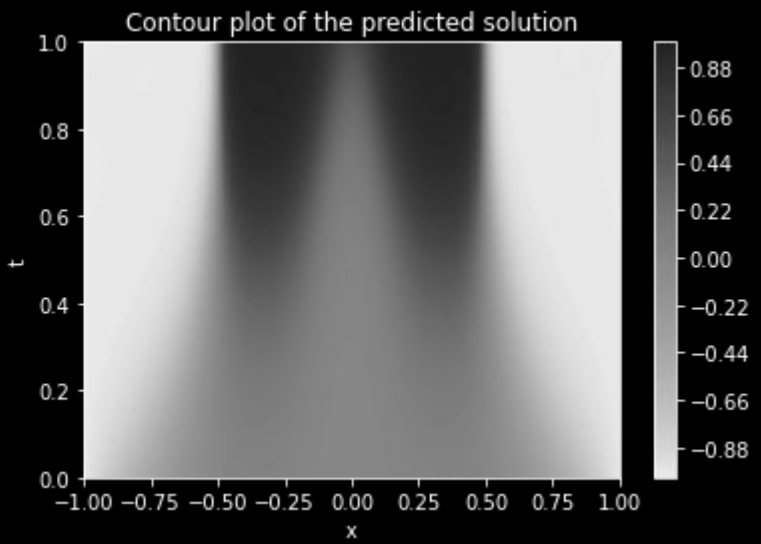
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

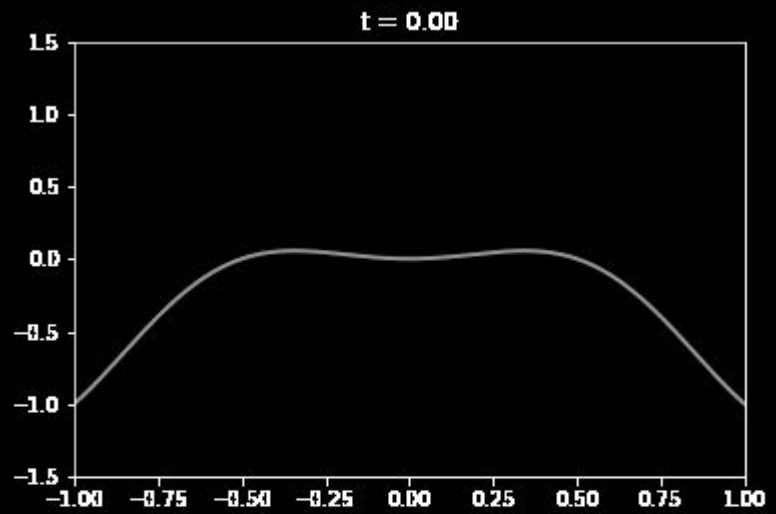
PINN with DeepXDE: Allen-Cahn Equation PDE

$$\frac{\partial u}{\partial t} = d\frac{\partial^2 u}{\partial x^2} + 5(u - u^3), \quad x \in [-1, 1], \quad t \in [0, 1]$$

$$u(x, 0) = x^2 \cos(\pi x)$$

$$u(-1, t) = u(1, t) = -1$$





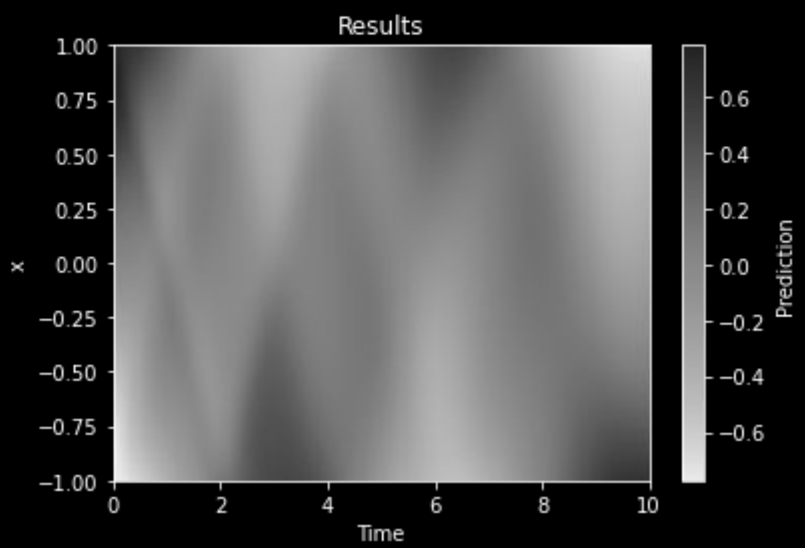
Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

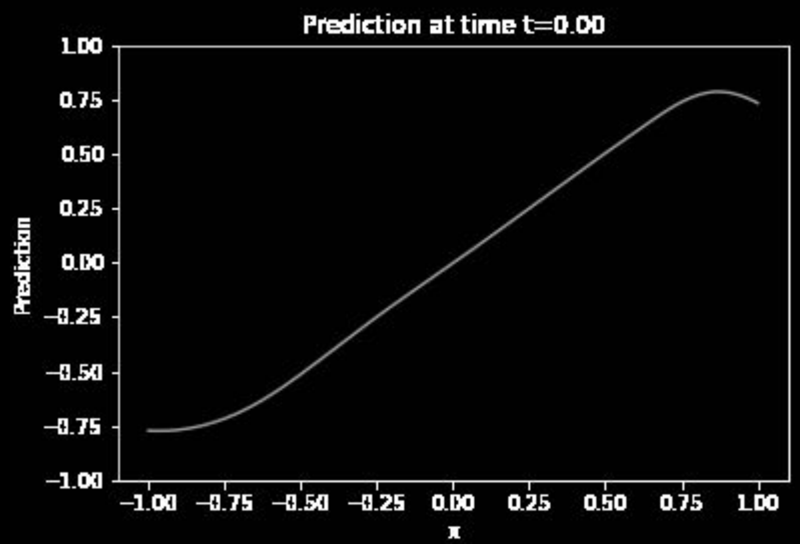
PINN with DeepXDE: Klein-Gordon Equation PDE

$$\frac{\partial^2 y}{\partial t^2} + \alpha \frac{\partial^2 y}{\partial x^2} + \beta y + \gamma y^k = -x\cos(t) + x^2\cos^2(t), \qquad x \in [-1, 1], \quad t \in [0, 10]$$

$$y(x,0) = x, \quad \frac{\partial y}{\partial t}(x,0) = 0 \qquad\qquad y(-1,t) = -\cos(t), \quad y(1,t) = \cos(t)$$

$$\alpha = -1, \beta = 0, \gamma = 1, k = 2.$$





Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

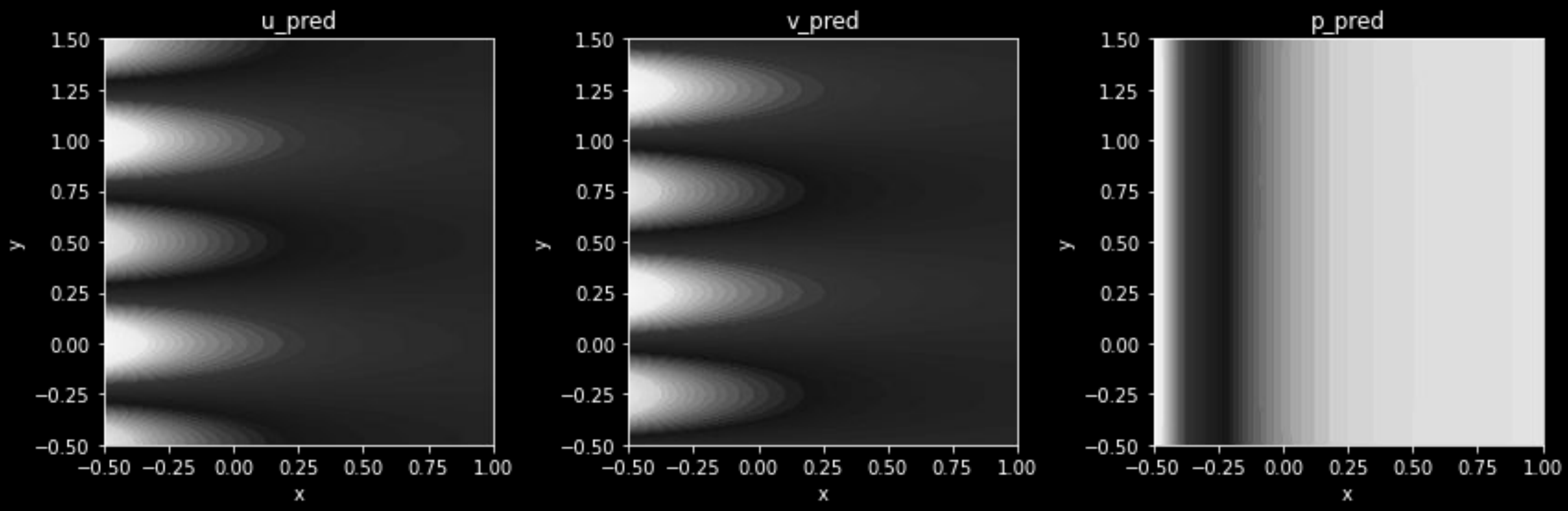
PINN with DeepXDE: Kovasznay Equation PDE

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),$$

$$\Omega = [0,1]^2$$

$$u(x,y) = 0, \qquad (x,y) \in \partial\Omega$$



Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

PINN for Inverse Problems

Add a loss function term with known points on the domain and optimize it together with the parameters

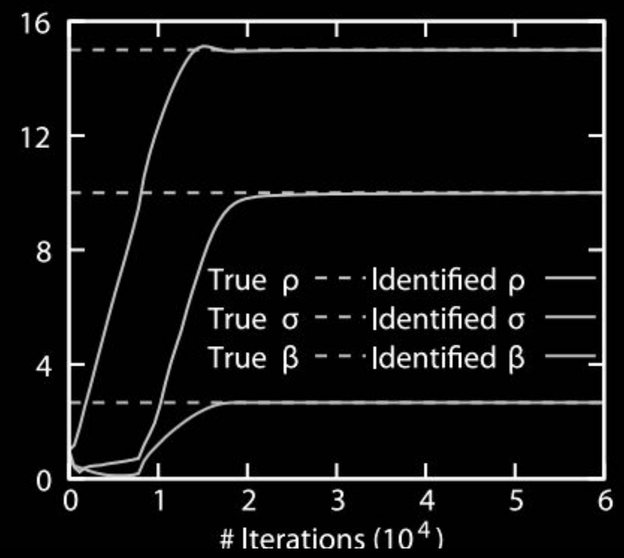
$$\mathcal{I}(u, \mathbf{x}) = 0 \quad \text{for} \quad \mathbf{x} \in \mathcal{T}_i.$$

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_b) + w_i \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i)$$

$$\mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \|\mathcal{I}(\hat{u}, \mathbf{x})\|_2^2.$$

$$\boldsymbol{\theta}^*, \boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T})$$

$$\frac{dx}{dt} = \rho(y - x), \quad \frac{dy}{dt} = x(\sigma - z) - y, \quad \frac{dz}{dt} = xy - \beta z$$

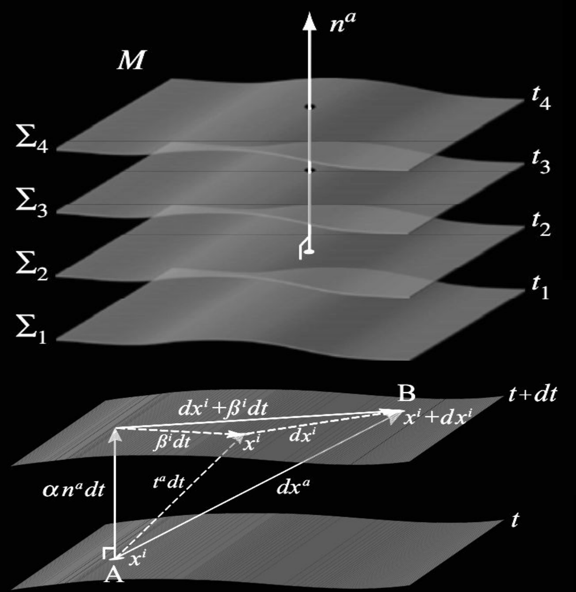


Source: https://arxiv.org/abs/1907.04502 (DeepXDE: A deep learning library for solving differential equations)

Future applications

(???)

Canonical 3+1 Decomposition (ADM)



$$R + K^2 - K_{ij}K^{ij} = 16\pi\rho \qquad \text{(Hamiltonian Constraint)}$$

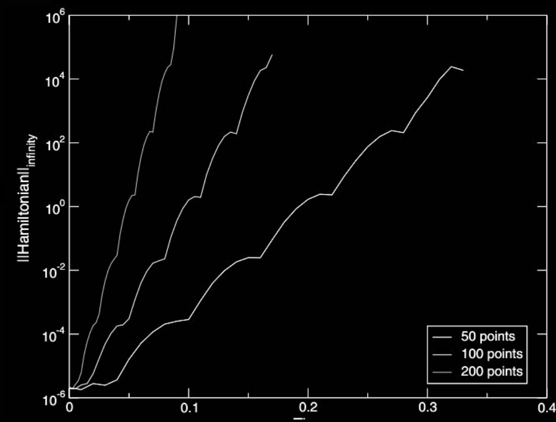
$$D_j(K^{ij} - \gamma^{ij}K) = 8\pi j^i \qquad \text{(Momentum Constraint)}$$

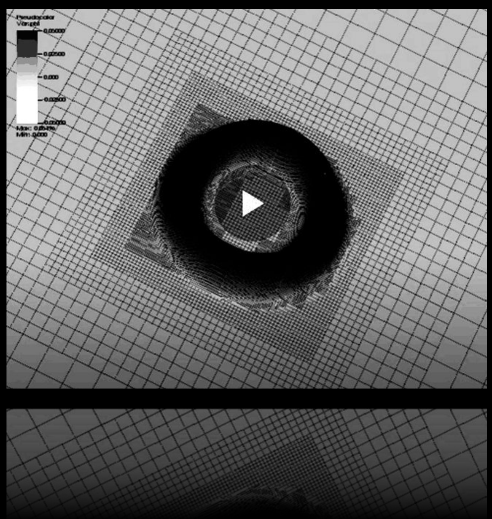
$$\partial_t \gamma_{ij} = -2\alpha K_{ij} + \mathcal{L}_\beta \gamma_{ij}$$

$$\partial_t K_{ij} = \alpha(R_{ij} - 2K_{ij}K^k_j + KK_{ij}) - D_iD_j\alpha + 4\pi\alpha M_{ij} + \mathcal{L}_\beta K_{ij}$$

12 evolution equations + 4 constraint equations

Constraint-violating modes



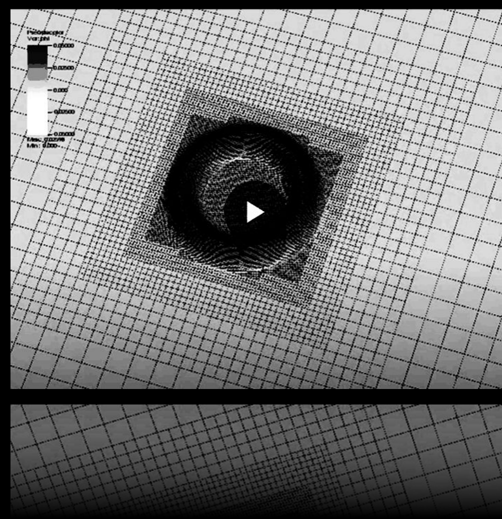


$$\frac{1}{a}\frac{da}{dr} + \frac{a^2-1}{2r} - 2\pi r(\Pi^2 + \Phi^2) = 0$$

$$\frac{1}{\alpha}\frac{d\alpha}{dr} - \frac{1}{a}\frac{da}{dr} - \frac{a^2-1}{r} = 0$$

$$\partial_t \Phi = \partial_r\left(\frac{\alpha}{a}\Pi\right)$$

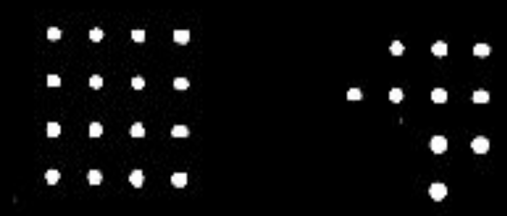
$$\partial_t \Pi = \frac{1}{r^2}\partial_r\left(r^2\frac{\alpha}{a}\Phi\right)$$



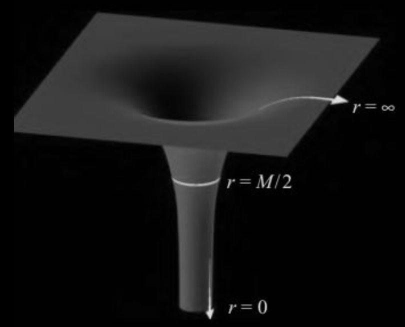
Source: Numerical Relativity: Starting from Scratch, Baumgarte and Shapiro

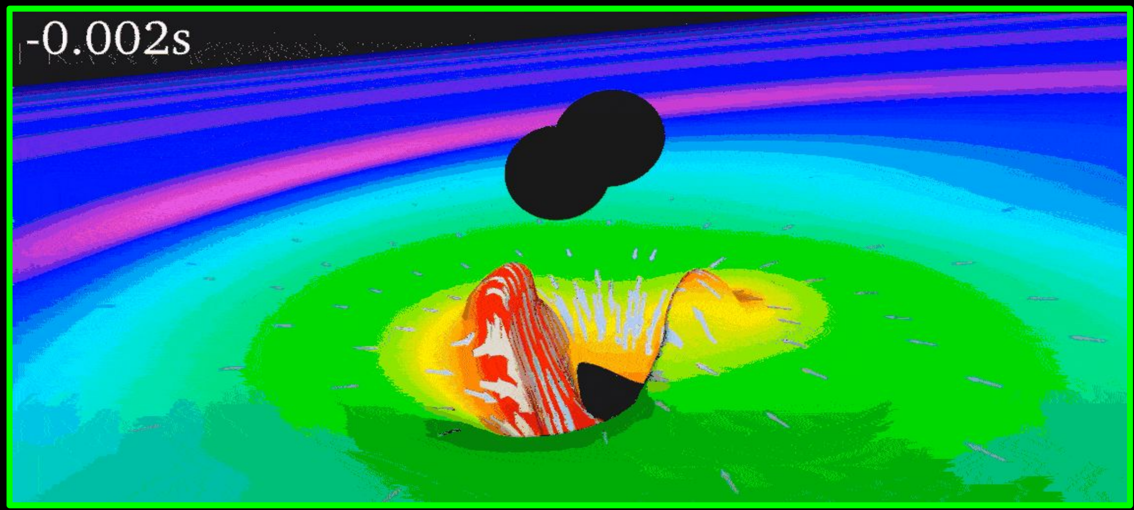
Binary Mergers: Singularity Avoidance

Black Hole Excision



Moving Puncture Gauge





Source: Nhttps://www.youtube.com/watch?v=adQumfzldUM&t=2812s (Numerical Relativity - Saul Teukolsky)

Binary Mergers: Puncture Initial Data

Hamiltonian + Momentum Constraint

$$\bar{D}^2\psi + \frac{1}{8}\psi^{-7}\bar{A}^{\mathrm{L}}_{ij}\bar{A}^{ij}_{\mathrm{L}} = 0,$$

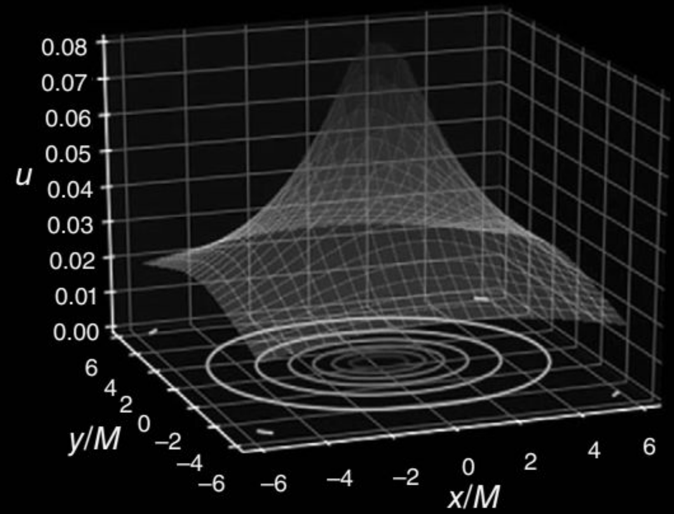
Bowen-York Solutions

$$\bar{A}^{ij}_{\mathrm{L}} = \frac{6}{s^3} l^{(i}\,\epsilon^{j)kl} J_k l_l, \qquad \bar{A}^{ij}_{\mathrm{L}} = \frac{3}{2s^2}\left(P^i l^j + P^j l^i - (\eta^{ij} - l^i l^j) l_k P^k\right).$$

Puncture Initial Data

$$\psi = 1 + \frac{1}{\alpha} + u, \qquad \frac{1}{\alpha} = \sum_n \frac{\mathcal{M}_n}{2s_n}$$

$$\bar{D}^2 u = -\beta\,(\alpha + \alpha u + 1)^{-7} \qquad \beta \equiv \frac{1}{8}\alpha^7 \bar{A}^{\mathrm{L}}_{ij}\bar{A}^{ij}_{\mathrm{L}}.$$



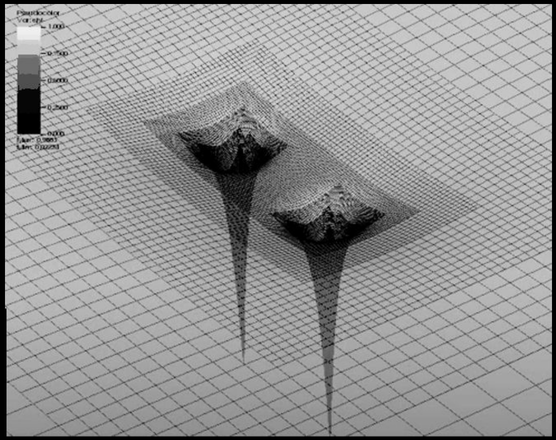

```
# location of puncture
bh_loc = ( loc_x, loc_y, loc_z )

# linear momentum
lin_mom = ( p_x, p_y, p_z )

# set up Puncture solver
black_hole = Puncture(bh_loc, lin_mom, n_grid, x_out)

# and construct solution
black_hole.construct_solution(tol, it_max)

# and write results to file
black_hole.write_to_file()
```

Source: Numerical Relativity: Starting from Scratch, Baumgarte and Shapiro

Binary Mergers: Moving Punctures

BSSN = conformal + trace-tracefree + transverse-longitudinal decompositions {ADM}

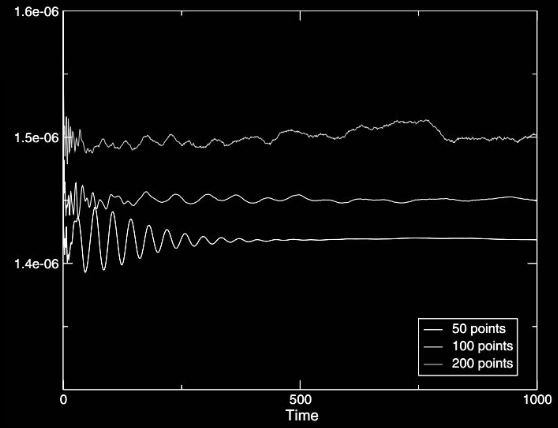
$$\phi = \tfrac{1}{12}\ln(\gamma/\bar{\gamma}),$$
$$K = \gamma^{ij}K_{ij},$$
$$\tilde{\gamma}_{ij} = e^{-4\phi}\gamma_{ij},$$
$$\tilde{A}_{ij} = e^{-4\phi}(K_{ij} - \tfrac{1}{3}\gamma_{ij}K),$$



20 evolution equations + 6 constraint equations

$$\begin{aligned}
\partial_\perp \phi &= \tfrac{1}{6}\bar{D}_i\beta^i - \tfrac{1}{6}\alpha K,\\
\partial_\perp \tilde{\gamma}_{ij} &= -\tfrac{2}{3}\tilde{\gamma}_{ij}\bar{D}_k\beta^k - 2\alpha\tilde{A}_{ij},\\
\partial_\perp K &= \alpha(\tilde{A}_{ij}\tilde{A}^{ij} + \tfrac{1}{3}K^2) - \gamma^{ij}D_iD_j\alpha,\\
\partial_\perp \tilde{A}_{ij} &= -\tfrac{2}{3}\tilde{A}_{ij}\bar{D}_k\beta^k + \alpha(K\tilde{A}_{ij} - 2\tilde{A}_{ik}\tilde{A}^k_j)\\
&\quad + e^{-4\phi}[\alpha R_{ij} - D_iD_j\alpha]^{\mathrm{TF}},\\
\partial_\perp \tilde{\Lambda}^i &= \tilde{\gamma}^{k\ell}\bar{D}_k\bar{D}_\ell\beta^i + \tfrac{2}{3}\tilde{\gamma}^{jk}\Delta\tilde{\Gamma}^i{}_{jk}\bar{D}_\ell\beta^\ell\\
&\quad + \tfrac{1}{3}\tilde{D}^i(\bar{D}_k\beta^k) - 2\tilde{A}^{ik}\bar{D}_k\alpha + 2\alpha\tilde{A}^{k\ell}\Delta\tilde{\Gamma}^i{}_{k\ell}\\
&\quad + 12\alpha\tilde{A}^{ik}\bar{D}_k\phi - \tfrac{4}{3}\alpha\tilde{D}^iK,
\end{aligned}$$

$$\begin{aligned}
\mathcal{H} &= e^{-4\phi}(\tilde{R} - 8\tilde{D}^i\tilde{D}_i\phi - 8\tilde{D}^i\phi\tilde{D}_i\phi) + \tfrac{2}{3}K^2\\
&\quad - \tilde{A}_{ij}\tilde{A}^{ij} = 0,\\
\tilde{\mathcal{M}}^i &= \tilde{D}_j\tilde{A}^{ij} + 6\tilde{A}^{ij}\partial_j\phi - \tfrac{2}{3}\tilde{D}^iK = 0,\\
\mathcal{G}^i &= \tilde{\Lambda}^i - \tilde{\gamma}^{jk}\Delta\tilde{\Gamma}^i_{jk} = 0,
\end{aligned}$$



Moving puncture gauge: 1 + log slicing + Γ driver condition

$$\begin{aligned}
\partial_t\alpha &= \beta^a\partial_a\alpha - 2\alpha K\\
\partial_t\beta^a &= \frac{3}{4}B^a + \beta^c\partial_c\beta^a\ ,\\
\partial_t B^a &= \partial_t\Gamma^a + \beta^c\partial_c B^a - \beta^c\partial_c\Gamma^a - \eta B^a
\end{aligned}$$

Source: Numerical Relativity: Starting from Scratch, Baumgarte and Shapiro