Soutenance

Machine Learning for Scientific Computing and Numerical Analysis

Ethan Cohen, Jules Cognon

ÉCOLE POLYTECHNIQUE
INSTITUT POLYTECHNIQUE DE PARIS

Sommaire

Deep Operator Networks

Génération des données

1. **Fonctions Gaussiennes** :

$$\mu(x) = e^{-5(x-c)^2}$$

où $c$est un centre tiré aléatoirement dans un certain intervalle (ex. $[0.2, 0.8]$).

2. **Polynômes de Chebyshev** :

$$\mu(x) = \sum_{j=0}^{9} c_j T_j(x)$$

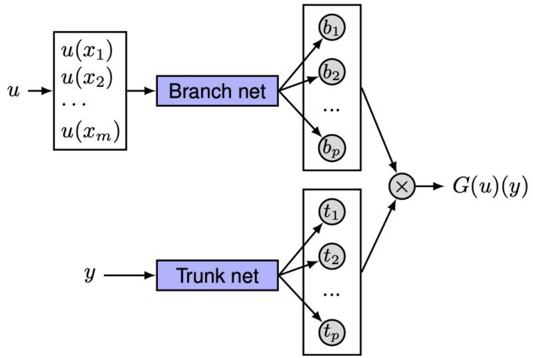
où $T_j(x)$ est le j-ième polynôme de Chebyshev et $c_j \sim \mathcal{N}(0, 1)$.

Hyperparamètre	Valeur
N_{train}	1000
N_{test}	200
d_p	50
d_v	50
Nombre de couches cachées	4
Nombre de neurones par couche	50
Fonction d'activation	ReLU
Optimiseur	Adam (lr $= 0.001$)
Fonction de perte	MSE
Nombre d'époques	100
Taille du batch	32

```python
# Génération des fonctions μ(x)
def generate_mu_samples(n_samples, type='gaussian_exemple'):
    mu_samples = []
    if type == 'polynomial':
        coeffs_list = [np.random.normal(0, 1, size=10) for _ in range(n_samples)]
        def mu_func(x, coeffs):
            return sum(coeffs[j] * eval_chebyt(j, x) for j in range(10))
        mu_samples = [lambda x, c=coeffs: mu_func(x, c) for coeffs in coeffs_list]
    elif type == 'gaussian_exemple':
        centers = np.random.uniform(0.2, 0.8, n_samples)
        mu_samples = [lambda x, c=c: np.exp(-5 * (x - c) ** 2) for c in centers]
    elif type == 'gaussian_exemple_2':
        centers = np.random.uniform(0.5, 1, n_samples)
        mu_samples = [lambda x, c=c: np.exp(-5 * (x - c) ** 2) for c in centers]
    return mu_samples[:n_samples]

# Génération des données
mu_train_funcs = generate_mu_samples(N_train, type=mu_type)
mu_test_funcs = generate_mu_samples(N_test, type=mu_type)
#mu_new_funcs= generate_mu_samples(N_test, type=mu_type)
mu_new_funcs = generate_mu_samples(N_test, type = 'gaussian_exemple_2')  #ligne à dé
```

La classe DeepONet

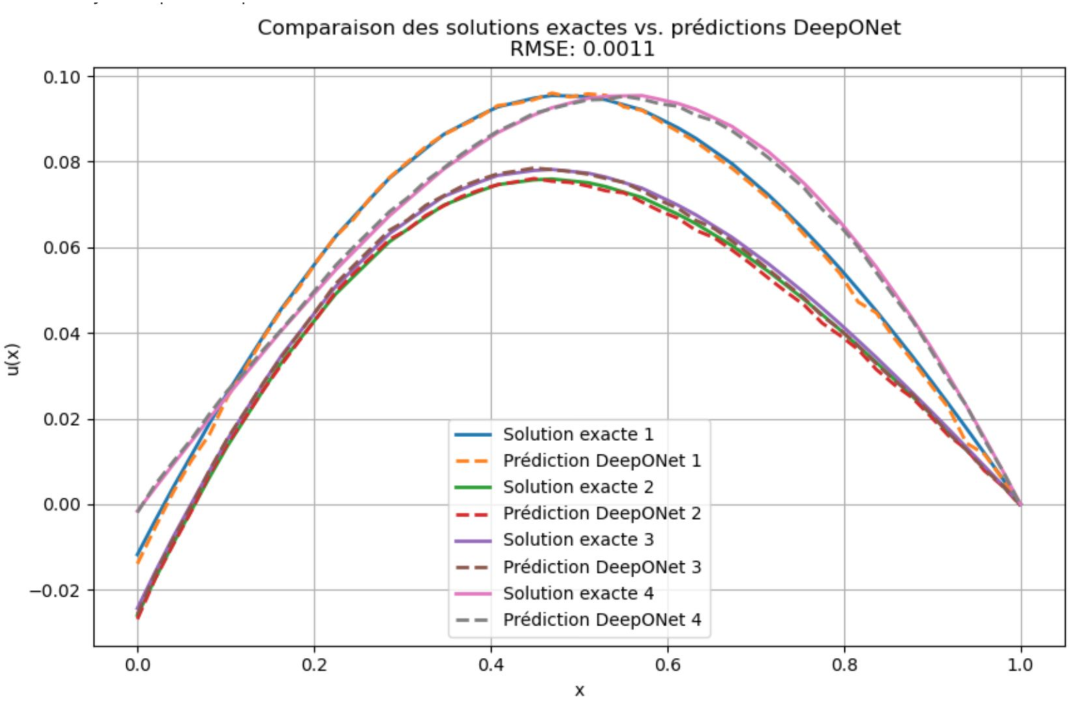


$$\hat{N}(\mu)(x) = \sum_{k=1}^{d_V} f_k(\mu(x_1), \ldots, \mu(x_{d_P}))g_k(x)$$

```python
class DeepONet(keras.Model):
    def __init__(self, branch_input_dim, trunk_input_dim, hidden_units=dv):
        super(DeepONet, self).__init__()
        self.branch_net = keras.Sequential([
            layers.Dense(hidden_units, activation='relu'),
            layers.Dense(hidden_units)
        ])
        self.trunk_net = keras.Sequential([
            layers.Dense(hidden_units, activation='relu'),
            layers.Dense(hidden_units)
        ])
        self.final_layer = layers.Dot(axes=1)

    def call(self, inputs):
        mu_input, x_input = inputs
        branch_out = self.branch_net(mu_input)
        trunk_out = self.trunk_net(x_input)
        return self.final_layer([branch_out, trunk_out])
```

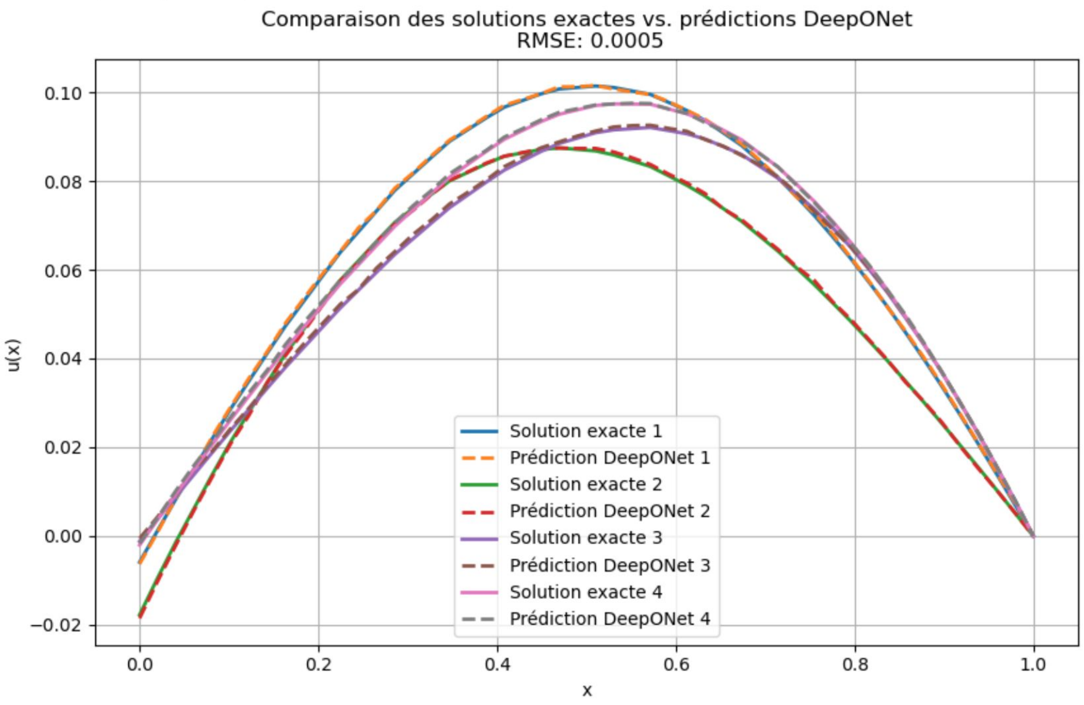
Résultats
Gaussian exemple avec 2*2 couches (entraîné et testé sur le même set)



Temps d'entraînement: 20s

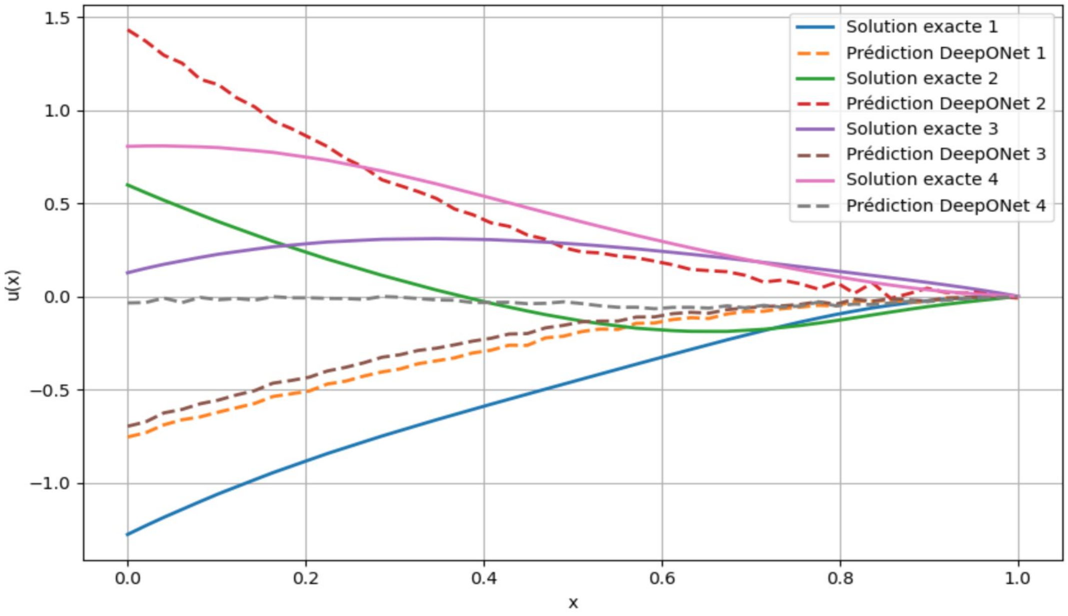
L'erreur RMSE: 10e-3

Résultats
Gaussian exemple avec 2*3 couches (entraîné et testé sur le meme set)

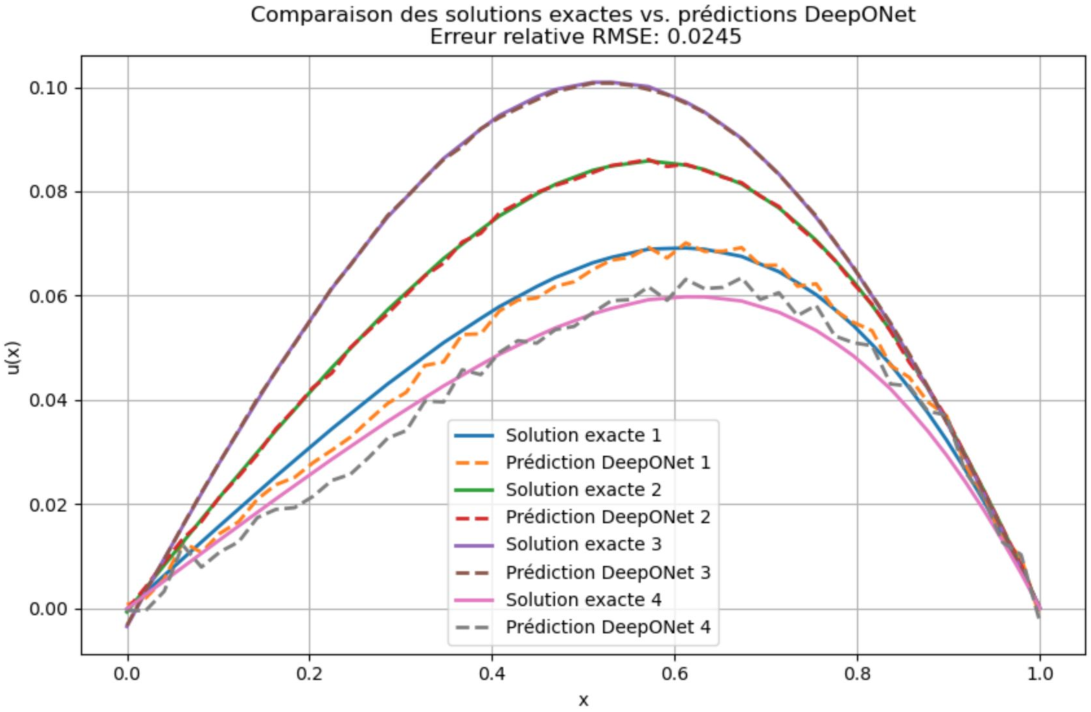


Temps d’entraînement: +- 25 s

Résultats
Polynomial avec 2*3 couches (entrainé et testé sur le meme set)



Résultats
Gaussian exemple avec 2*3 couches (entraîné et testé sur set différent)



II

Fourier Neural Operators

Génération des données

Hyperparamètre	Valeur
N_{train}	100
N_{test}	20
Taille de la grille (N)	256
Temps final (T)	1.0
Viscosité (ε)	0.1
Nombre de modes (modes)	4
Nombre de neurones par couche (width)	16
Nombre de couches (num_layers)	2
Fonction d'activation	ReLU
Optimiseur	Adam (lr = 1e-3)
Fonction de perte	MSE
Nombre d'époques (epochs)	50
Taille du batch	10

```python
def generate_burgers_data(n_samples, N, T, eps):
    x_grid = np.linspace(-1, 1, N, endpoint=False)
    u0_data = np.zeros((n_samples, N, 1))
    uT_data = np.zeros((n_samples, N, 1))

    for i in range(n_samples):
        c = np.random.uniform(-0.5, 0.5)
        u0_func = lambda x, c=c: np.exp(-100 * (x - c) ** 2)
        u0_sample = u0_func(x_grid)
        uT_sample = compute_burgers_reference(N, eps, T, u0_func)

        u0_data[i, :, 0] = u0_sample
        uT_data[i, :, 0] = uT_sample

    return x_grid, u0_data, uT_data


N, T, eps = 256, 1.0, 0.1
n_train, n_test, epochs, batch_size = 100, 20, 50, 10
learning_rate, modes, width, num_layers = 1e-3, 4, 16, 2

x_grid, u0_train, uT_train = generate_burgers_data(n_train, N, T, eps)
_, u0_test, uT_test = generate_burgers_data(n_test, N, T, eps)
```

La classe Fourier Layer

```python
class FourierLayer(layers.Layer):
    def __init__(self, modes, width):
        super(FourierLayer, self).__init__()
        self.modes = modes
        self.width = width

        self.w_linear = self.add_weight(
            shape=(self.width, self.width),
            initializer="random_normal",
            trainable=True,
            name="W_linear"
        )
        self.b_linear = self.add_weight(
            shape=(self.width,),
            initializer="zeros",
            trainable=True,
            name="b_linear"
        )

        # Poids complexes pour la convolution
        self.weights_real = self.add_weight(
            shape=(self.width, self.width, self.modes),
            initializer="random_normal",
            trainable=True,
            name="w_real"
        )
        self.weights_imag = self.add_weight(
            shape=(self.width, self.width, self.modes),
            initializer="random_normal",
            trainable=True,
            name="w_imag"
        )
```

```python
def call(self, x):
    x_linear = tf.einsum('bij,jk->bik', x, self.w_linear) + self.b_linear
    x_perm = tf.transpose(x, perm=[0, 2, 1])
    x_ft = tf.signal.rfft(x_perm)
    x_ft_cut = x_ft[:, :, :self.modes]
    weight = tf.complex(self.weights_real, self.weights_imag)
    out_ft_low = tf.einsum('bjm,jim->bim', x_ft_cut, weight)
    n_fft = tf.shape(x_ft)[-1]
    pad_size = n_fft - self.modes
    zeros = tf.zeros((tf.shape(x)[0], self.width, pad_size), dtype=tf.complex64)
    out_ft = tf.concat([out_ft_low, zeros], axis=-1)
    x_ifft = tf.signal.irfft(out_ft, fft_length=[tf.shape(x)[1]])
    x_out = tf.transpose(x_ifft, perm=[0, 2, 1])
    return tf.nn.relu(x_linear + x_out)
```

$$a^{(\ell)} : \{v : D \to \mathbb{R}^{d_{\ell-1}}\} \to \{w : D \to \mathbb{R}^{d_\ell}\}, \qquad a^{(\ell)}(v) = \sigma\left(W^{(\ell-1)}v + \boldsymbol{b}^{(\ell-1)} + K^{(\ell-1)}v\right), \quad 1 \le \ell \le L, \tag{26}$$

$$w_{\boldsymbol{k}}^j = \frac{1}{n_1 \dots n_d} \sum_{j_1=1}^{n_1} \dots \sum_{j_d=1}^{n_d} f_j(x_{j_1}, \dots, x_{j_d}) e^{-i\boldsymbol{k}\cdot(x_{j_1}, \dots, x_{j_d})}, \quad 1 \le j \le d', \tag{21}$$

$$(K^{(\ell-1)}v)_j(\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^d} \left(\sum_{j'=1}^{d_{\ell-1}} c_{\boldsymbol{k}}^{j'} d_{\boldsymbol{k}}^{jj'} \right) e^{i\boldsymbol{k}\cdot\boldsymbol{x}}, \quad 1 \le j \le d_\ell. \tag{30}$$

La classe FNO

```python
class FNO(tf.keras.Model):
    def __init__(self, modes, width, num_layers=4):
        super(FNO, self).__init__()
        self.modes = modes
        self.width = width
        self.num_layers = num_layers
        self.lifting = layers.Dense(width)
        self.fourier_layers = [FourierLayer(modes, width) for _ in range(num_layers)]
        self.projection = layers.Dense(1)

    def call(self, x):
        x = self.lifting(x)
        for f_layer in self.fourier_layers:
            x = f_layer(x)
        x = self.projection(x)
        return x
```

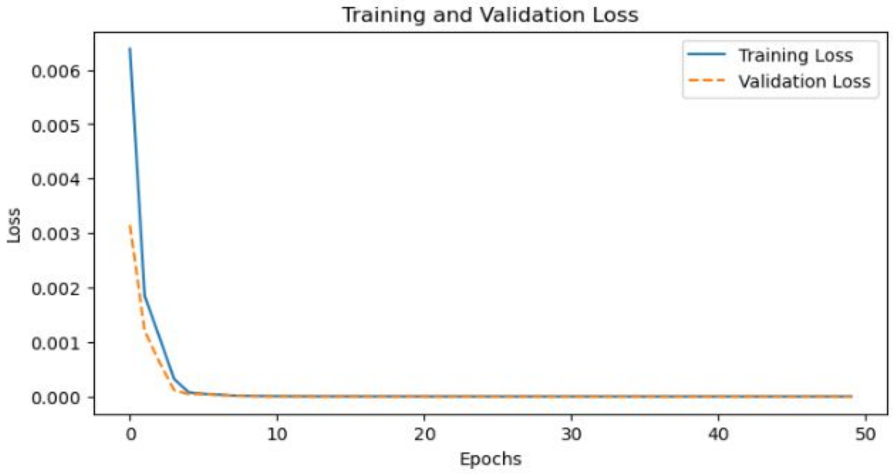
$$a^{(0)} : \{v : D \to \mathbb{R}\} \to \{w : D \to \mathbb{R}^{d_0}\}, \qquad a^{(0)}(v) = \hat{R}(v), \tag{25}$$

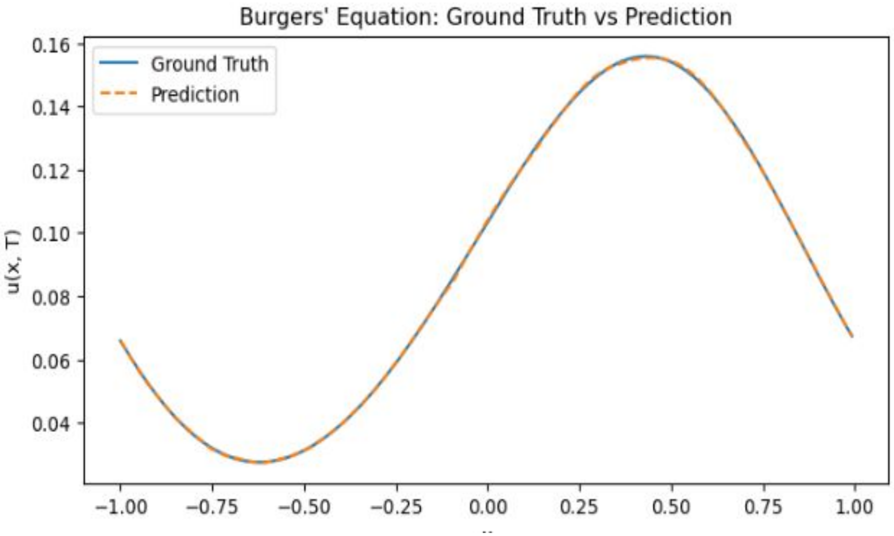
$$a^{(\ell)} : \{v : D \to \mathbb{R}^{d_{\ell-1}}\} \to \{w : D \to \mathbb{R}^{d_\ell}\}, \qquad a^{(\ell)}(v) = \sigma\left(W^{(\ell-1)}v + \boldsymbol{b}^{(\ell-1)} + K^{(\ell-1)}v\right), \quad 1 \le \ell \le L, \tag{26}$$

$$a^{(L+1)} : \{v : D \to \mathbb{R}^{d_L}\} \to \{w : D \to \mathbb{R}\}, \qquad a^{(L+1)}(v) = \hat{Q}(v), \tag{27}$$

Résultats

Test L2 Error: 2.2361e-03
Test L∞ Error: 3.0006e-03





III

Retour sur le cours

Points positifs et négatifs

1

Positif: Format des cours

2

Positif: 6 pages par poly

3

Amélioration: Rendu des PCs