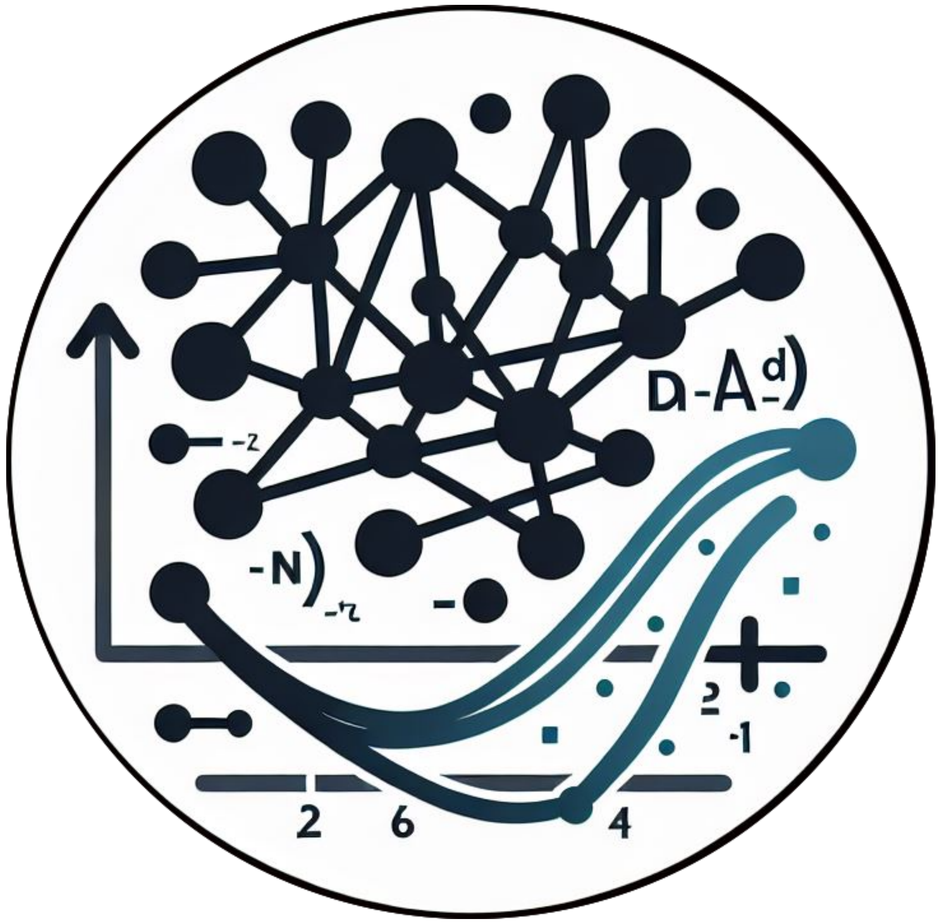
Neural Koopman



Deep Learning for Engineers

Andrew Ning
aning@byu.edu

Dynamic systems

nonlinear

linear

$$\frac{dy(t)}{dt} = f(y(t))$$ (continuous time) $$\frac{dy}{dt} = Ay$$

$$y_{k+1} = F(y_k)$$ (discrete time) $$y_{k+1} = Ly_k$$

Koopman operator

$$g(y_{k+1}) = \mathscr{K} g(y_k)$$

$$z_{k+1} = K z_k \qquad \text{(finite dimensional approximation)}$$

Simple Example

$$y_{k+1} = y_k^2$$

Simple Example

$$y_{k+1} = y_k^2$$

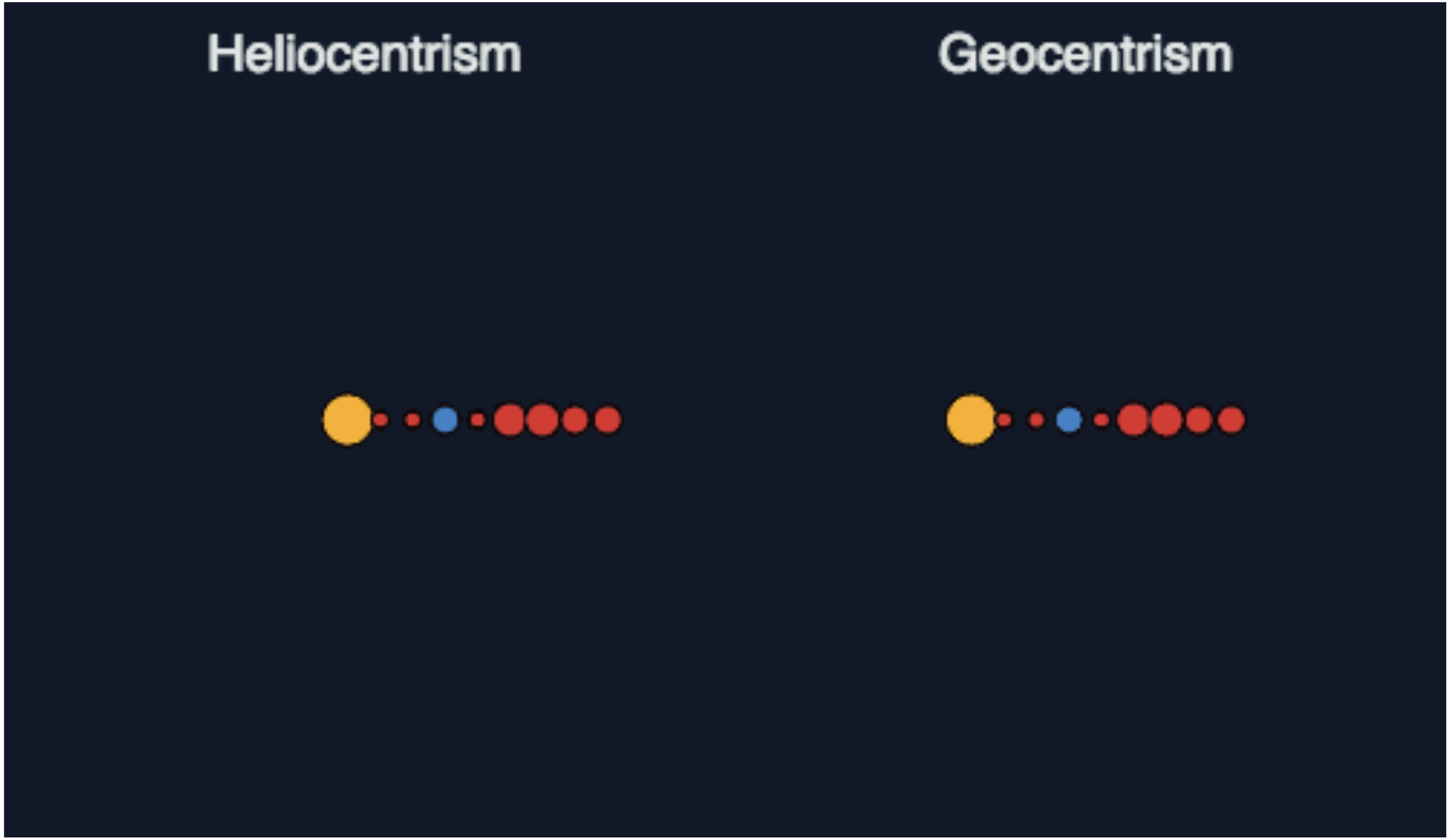
$$g(y) = \log|y|$$

Simple Example

$$y_{k+1} = y_k^2$$

$$g(y) = \log|y|$$

$$g(y_{k+1}) = \log|y_{k+1}| = \log|y_k^2| = 2\log|y_k| = 2g(y_k)$$



Malin Christersson

Dynamic Mode Decomposition

$$y_{k+1} = Ay_k \quad \text{(assume linear)}$$

$$Y = \begin{bmatrix} | & | & & | \\ y_1 & y_2 & \cdots & y_{m-1} \\ | & | & & | \end{bmatrix}$$

$$Y' = \begin{bmatrix} | & | & & | \\ y_2 & y_3 & \cdots & y_m \\ | & | & & | \end{bmatrix}$$

$$Y' \approx AY$$

Dynamic Mode Decomposition

$$y_{k+1} = Ay_k \quad \text{(assume linear)}$$

$$Y = \begin{bmatrix} | & | & & | \\ y_1 & y_2 & \cdots & y_{m-1} \\ | & | & & | \end{bmatrix}$$

$$Y' = \begin{bmatrix} | & | & & | \\ y_2 & y_3 & \cdots & y_m \\ | & | & & | \end{bmatrix}$$

$$Y' \approx AY$$

Extended DMD

$$z = \phi(y)$$

$$\phi(y) = [y_1, y_2, y_1 y_2, y_1^2, \log(y_1)]$$

$$z_{k+1} = Kz_k$$

Neural Koopman

$$z = \phi(y) \qquad \text{encoder}$$

$$\|z_{k+1} - Kz_k\| \qquad \text{enforce linearity}$$

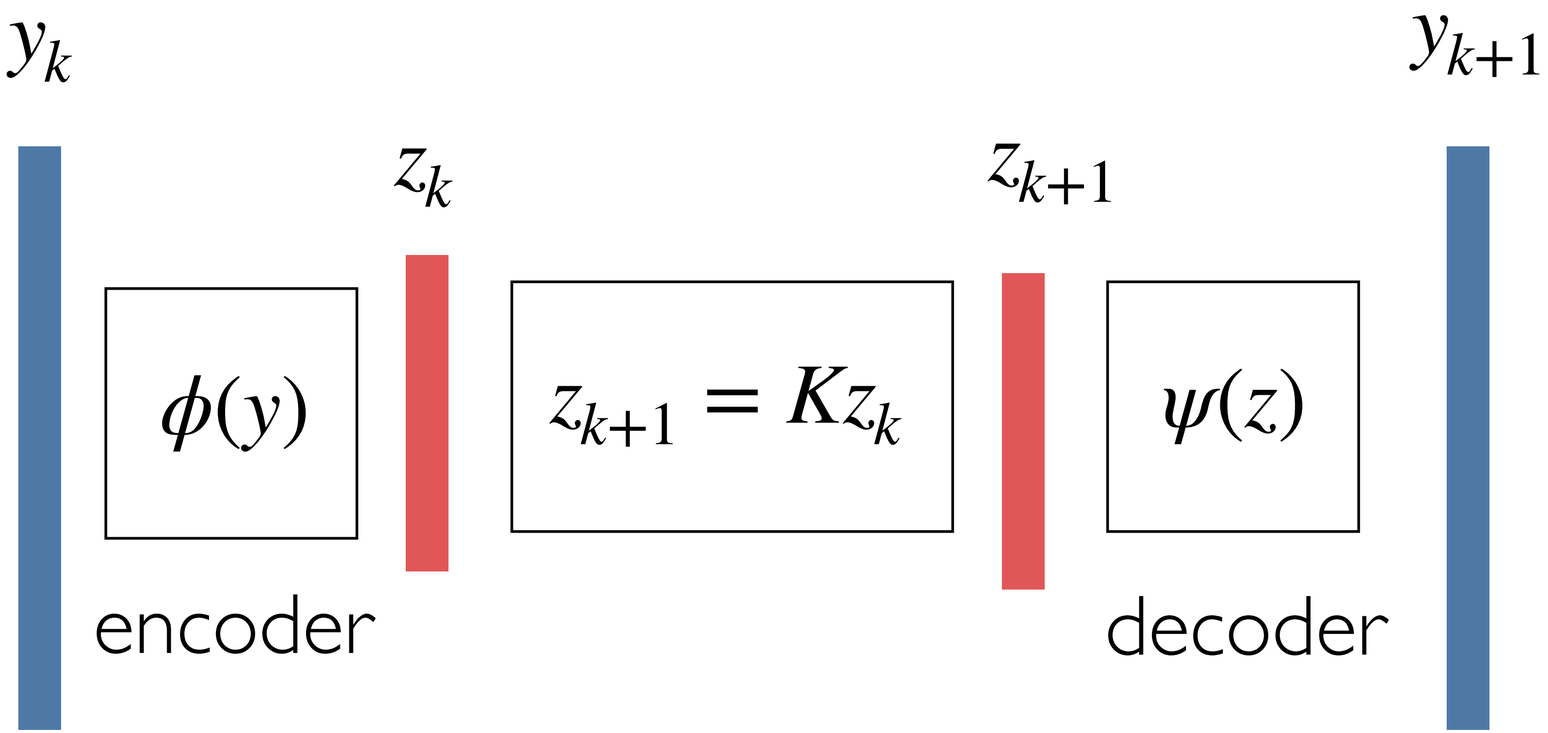
$$\|y_{k+1} - \psi(z_{k+1})\| \qquad \text{state prediction}$$

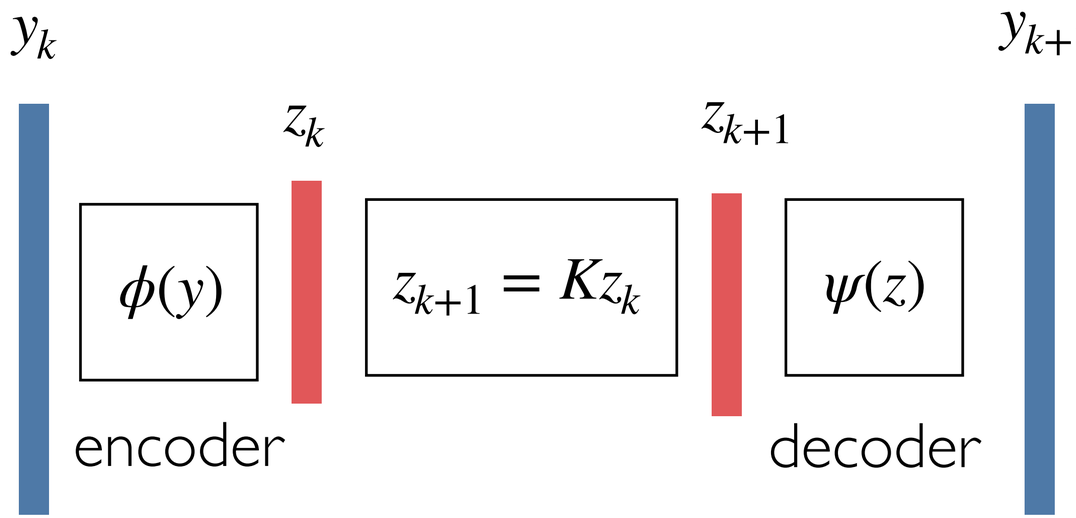
ARTICLE

DOI: 10.1038/s41467-018-07210-0 OPEN

Deep learning for universal linear embeddings of nonlinear dynamics

Bethany Lusch [1,2], J. Nathan Kutz[1] & Steven L. Brunton[1,2]


y_k
y_{k+1}
z_k
z_{k+1}
$\phi(y)$
$z_{k+1} = Kz_k$
$\psi(z)$
encoder
decoder



reconstruction loss $$\|y - \psi(\phi(y))\|$$

linear dynamics $$\frac{1}{n_t - 1} \sum_{i=1}^{n_t - 1} \|\phi(y_i) - K\phi(y_{i-1})\|$$

state prediction $$\frac{1}{n_t - 1} \sum_{i=1}^{n_t - 1} \|y_i - \psi(K\phi(y_{i-1}))\|$$

```
class Network(nn.Module):
    def __init__():
        self.encoder = nn.Sequential(…)
        self.K = nn.Parameter(K0, requires_grad=True)
```

```
model = Network()
optimizer = Adam(model.parameters(), lr=lr)
```

As always, be careful with sizes

$$Z_{next} = KZ$$

As always, be careful with sizes

$$Z_{next} = KZ$$

$$K : n_z \times n_z$$

$$Z : n_z \times n_b$$

As always, be careful with sizes

$$Z_{next} = KZ$$

$$K : n_z \times n_z$$

$$Z : n_z \times n_b$$

$$X_{data} : n_b \times n_t \times n_x$$

```
Z0 = encode(X[:, 0, :])
```

As always, be careful with sizes

$$Z_{next} = KZ$$

$$K : n_z \times n_z$$

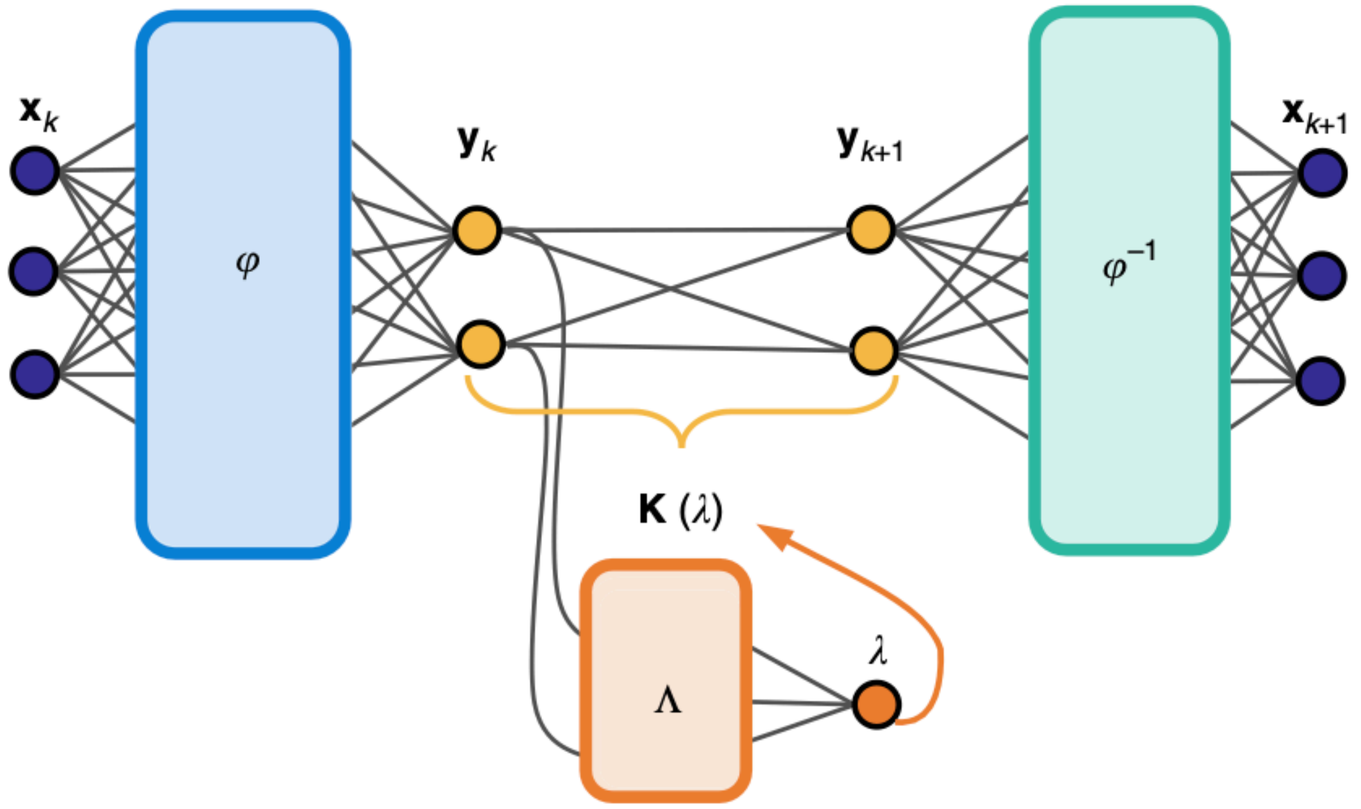
$$Z : n_z \times n_b$$

$$X_{data} : n_b \times n_t \times n_x$$

```
Z0 = encode(X[:, 0, :])
```

$$Z_{next}^T = Z^T K^T$$

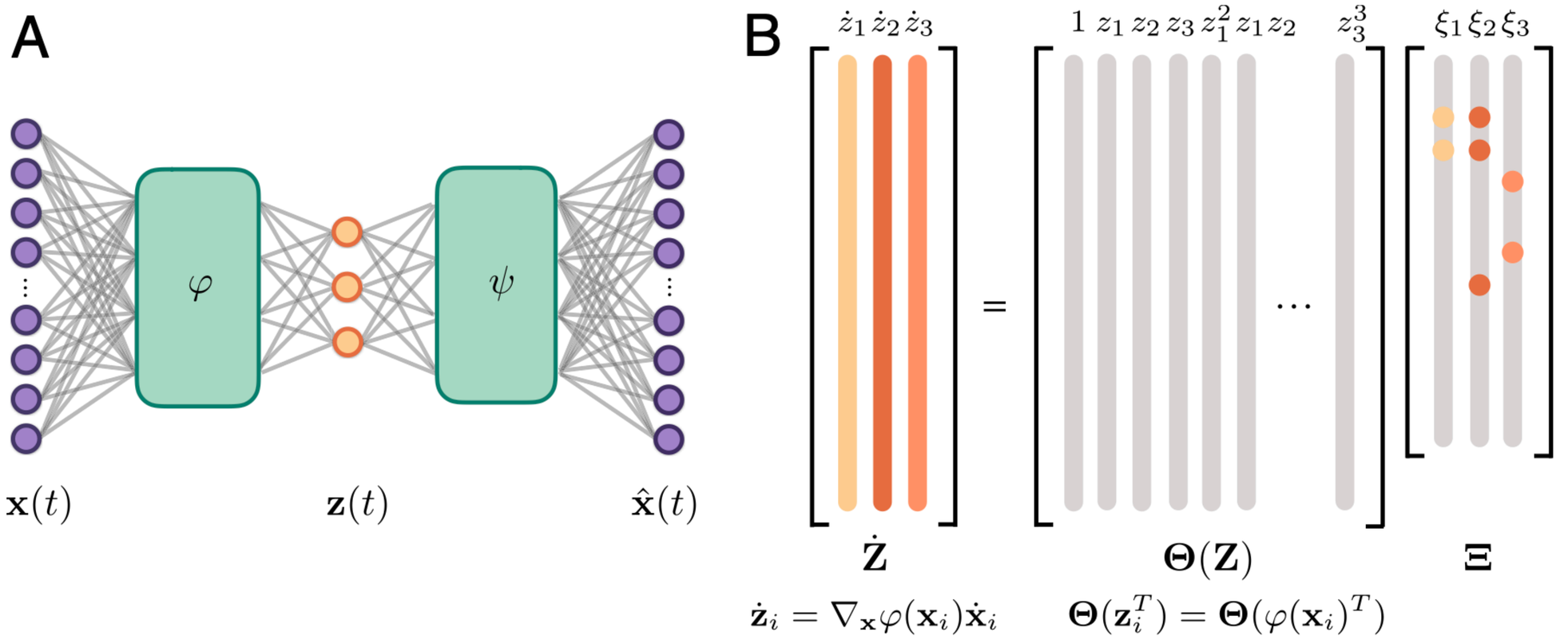
```
Z1 = torch.matmul(Z0, K.T)
```


$\mathbf{x}_k$
φ
$\mathbf{y}_k$
$\mathbf{y}_{k+1}$
φ^{-1}
$\mathbf{x}_{k+1}$
$\mathbf{K}(\lambda)$
Λ
λ

SINDy plus autoencoder

Data-driven discovery of coordinates and governing equations

Kathleen Champion[a,1], Bethany Lusch[b], J. Nathan Kutz[a], and Steven L. Brunton[a,c]



$$\underbrace{\left\|\mathbf{x}-\psi(\mathbf{z})\right\|_2^2}_{\text{reconstruction loss}} + \underbrace{\lambda_1\left\|\dot{\mathbf{x}}-\left(\nabla_{\mathbf{z}}\psi(\mathbf{z})\right)\left(\boldsymbol{\Theta}(\mathbf{z}^T)\boldsymbol{\Xi}\right)\right\|_2^2}_{\text{SINDy loss in } \dot{\mathbf{x}}} + \underbrace{\lambda_2\left\|\left(\nabla_{\mathbf{x}}\mathbf{z}\right)\dot{\mathbf{x}}-\boldsymbol{\Theta}(\mathbf{z}^T)\boldsymbol{\Xi}\right\|_2^2}_{\text{SINDy loss in } \dot{\mathbf{z}}} + \underbrace{\lambda_3\left\|\boldsymbol{\Xi}\right\|_1}_{\substack{\text{SINDy}\\ \text{regularization}}}$$