CUCEI

Zeroing Neural Networks

Introducción y ejemplos

Ing. Ulises Hernandez-Venegas

4 de julio de 2024

UNIVERSIDAD DE GUADALAJARA

Red Universitaria e Institución Benemérita de Jalisco

Plan

Introducción

Las redes neuronales ZNN son un tipo especial de RNN, que logran encontrar los ceros en una función variante en el tiempo. Se basa en lo siguiente:

- Definir una función de error $e(t) = f(x(t), t)$.
- Modificar su dinámica mediante el diseño de su derivada, comunmente $\dot{e}(t) = \gamma\sigma(e(t))$
- Obtener el modelo dinámico $\frac{\partial f}{\partial x}\dot{x}(t) = -\gamma\sigma(f(x(t), t)) - \frac{\partial f}{\partial t}$

Funciones de activación

Para este tipo de problemas se suele usar funciones de activación impares y monóticamente crecientes.

Linear	$\sigma(x) = x$
Power	$\sigma(x) = x^k$ donde $k > 3$ es un entero impar
Bipolar sigmoid	$\sigma(x) = \dfrac{1 - \exp(-\xi x)}{1 + \exp(-\xi x)}$ with $\xi \geq 2$
Power sigmoid	$\sigma(x) = \begin{cases} x^p & \lvert x\rvert \geq 1 \\ \dfrac{1 + e^{-\xi} - e^{-\xi x}}{1 - e^{-\xi} + e^{-\xi x}} & \text{de otra forma} \end{cases}$
hyperbolic sine	$\frac{1}{2}(\exp(\xi x) - \exp(-\xi x))$ com $\xi > 1$

Plan

Inversa de una matrix

- Primero definimos nuestra función de error $E(t) = A(t)X(t) - I$

Inversa de una matrix

- Primero definimos nuestra función de error $E(t) = A(t)X(t) - I$
- Calculamos su derivada con respecto al tiempo $\dot{E}(t) = \dot{A}(t)X(t) + A(t)\dot{X}(t)$

- Primero definimos nuestra función de error $E(t) = A(t)X(t) - I$
- Calculamos su derivada con respecto al tiempo $\dot{E}(t) = \dot{A}(t)X(t) + A(t)\dot{X}(t)$
- Sustituimos $E(t)$ y $\dot{E}(t)$ en $\dot{E}(t) = \gamma\mathcal{F}\left(E(t)\right)$

$$A(t)\dot{X}(t) = -\gamma\left(A(t)X(t) - I\right) - \dot{A}(t)X(t) \tag{1}$$

Este modelo de dinámica implícita es conocido como el modelo en tiempo continuo de una red neuronal tipo zhang (CTZNN).

“*Zeroing Neural Network*” en tiempo continuo

Simulación del modelo CTZNN

Este modelo puede ser implementado en Matlab ® usando la función *ode45* estableciendo el modelo implícito (1) usando $A(t)$ como la masa inercial.

Si tenemos una matriz no singular $A(t) \in \mathrm{R}^{n\times n}$, comenzando de **cualquier estado inicial** $X(0)$, se puede llegar a la matriz inversa $A^{-1}(t)$. Además, si se usa una función de activación lineal se consigue convergencia global y exponencial con una tasa γ [1].

Usemos como ejemplo una matriz de rotación $A(t) \in \mathrm{R}^{2\times 2}$ cuya inversa teórica es A^T.

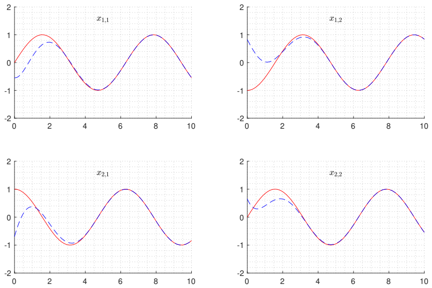
$$A(t) = \begin{bmatrix} \sin(t) & \cos(t) \\ -\cos(t) & \sin(t) \end{bmatrix}$$

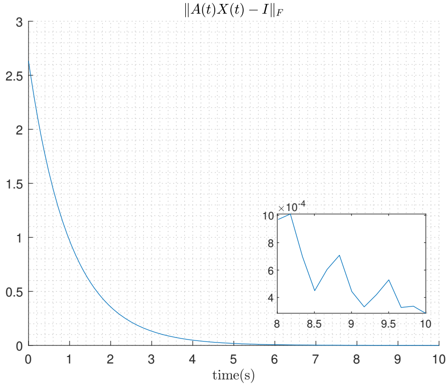
MatrixInversion/ctznn.m

[1] Yunong Zhang y Shuzhi Sam Ge. "Design and analysis of a general recurrent neural network model for time-varying matrix inversion". En: *IEEE Transactions on Neural Networks* 16.6 (2005), págs. 1477-1490

Simulación del modelo CTZNN

$$A(t)\dot{X}(t) = -\gamma\left(A(t)X(t) - I\right) - \dot{A}(t)X(t)$$





“*Zeroing Neural Network*” en tiempo discreto

CUCEI

Modelo en tiempo discreto DTZNN

- ***One-Step DTZNN model***: $\dot{X}_k \approx (X_{k+1} - X_k)/\tau.$ [2]

$$X_{k+1} = X_k - \tau X_k \dot{A}_k X_k - hX_k(A_k X_k - I) \tag{2}$$

[2] Dongsheng Guo y Yunong Zhang. “Zhang neural network, Getz–Marsden dynamic system, and discrete-time algorithms for time-varying matrix inversion with application to robots’ kinematic control”. En: *Neurocomputing* 97 (2012), págs. 22-32

[3] Yunong Zhang et al. “Taylor-type 1-step-ahead numerical differentiation rule for first-order derivative approximation and ZNN discretization”. En: *Journal of Computational and Applied Mathematics* 273 (2015), págs. 29-40

[4] Dongsheng Guo, Zhuoyun Nie y Laicheng Yan. “Novel discrete-time Zhang neural network for time-varying matrix inversion”. En: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.8 (2017), págs. 2301-2310

Modelo en tiempo discreto DTZNN

- ***One-Step DTZNN model***: $\dot{X}_k \approx (X_{k+1} - X_k)/\tau$. [2]

$$X_{k+1} = X_k - \tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I) \tag{2}$$

- ***Three-Step DTZNN model***: $\dot{X}_k \approx (2X_{k+1} - 3X_k + 2X_{k-1} - X_{k-2})/(2\tau)$. [3]

$$X_{k+1} = \frac{3}{2} X_k - X_{k-1} + \frac{1}{2} X_{k-2} - \tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I) \tag{3}$$

---

[2] Dongsheng Guo y Yunong Zhang. "Zhang neural network, Getz–Marsden dynamic system, and discrete-time algorithms for time-varying matrix inversion with application to robots' kinematic control". En: *Neurocomputing* 97 (2012), págs. 22-32

[3] Yunong Zhang et al. "Taylor-type 1-step-ahead numerical differentiation rule for first-order derivative approximation and ZNN discretization". En: *Journal of Computational and Applied Mathematics* 273 (2015), págs. 29-40

[4] Dongsheng Guo, Zhuoyun Nie y Laicheng Yan. "Novel discrete-time Zhang neural network for time-varying matrix inversion". En: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.8 (2017), págs. 2301-2310

Modelo en tiempo discreto DTZNN

- ***One-Step DTZNN model***: $\dot{X}_k \approx (X_{k+1} - X_k)/\tau.$ [2]

$$X_{k+1} = X_k - \tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I) \tag{2}$$

- ***Three-Step DTZNN model***: $\dot{X}_k \approx (2X_{k+1} - 3X_k + 2X_{k-1} - X_{k-2})/(2\tau).$ [3]

$$X_{k+1} = \frac{3}{2}X_k - X_{k-1} + \frac{1}{2}X_{k-2} - \tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I) \tag{3}$$

- ***Five-Step DTZNN model***: $\dot{X}_k \approx (24X_{k+1} - 5X_k - 12X_{k-1} - 6X_{k-2} - 4X_{k-3} + 3X_{k-4})/(48\tau).$ [4]

$$X_{k+1} = \frac{5}{24}X_k + \frac{1}{2}X_{k-1} + \frac{1}{4}X_{k-2} + \frac{1}{6}X_{k-3} - \frac{1}{8}X_{k-4} - 2\tau X_k \dot{A}_k X_k - h X_k (A_k X_k - I) \tag{4}$$

---

[2] Dongsheng Guo y Yunong Zhang. “Zhang neural network, Getz–Marsden dynamic system, and discrete-time algorithms for time-varying matrix inversion with application to robots’ kinematic control”. En: *Neurocomputing* 97 (2012), págs. 22-32

[3] Yunong Zhang et al. “Taylor-type 1-step-ahead numerical differentiation rule for first-order derivative approximation and ZNN discretization”. En: *Journal of Computational and Applied Mathematics* 273 (2015), págs. 29-40

[4] Dongsheng Guo, Zhuoyun Nie y Laicheng Yan. “Novel discrete-time Zhang neural network for time-varying matrix inversion”. En: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.8 (2017), págs. 2301-2310

Simulación del modelo DTZNN

A diferencia del modelo en tiempo continuo, no se garantiza la convergencia exponencial para cualquier estado inicial. X_0 debe ser inicializado cerca del vecindario de A_0^{-1}. Se ha confirmado convergencia exponencial para cualquier matriz no singular iniciando en $X_0 = 2A_0^T/\text{tr}(A_0 A_0^T)$, en caso requerir más estados iniciales se usa:

$$\begin{cases} X_1 = X_0 - \tau X_0 \dot{A}_0 X_0 - h X_0 \left(A_0 X_0 - I\right) \\ X_2 = X_1 - \tau X_1 \dot{A}_1 X_1 - h X_1 \left(A_1 X_1 - I\right) \\ X_3 = X_2 - \tau X_2 \dot{A}_2 X_2 - h X_2 \left(A_2 X_2 - I\right) \\ X_4 = X_3 - \tau X_3 \dot{A}_3 X_3 - h X_3 \left(A_3 X_3 - I\right). \end{cases}$$

MatrixInversion/setup.slx

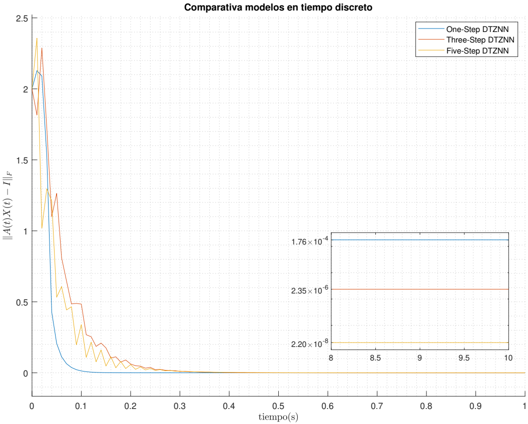
MatrixInversion/One_Step_DTZNN.slx

MatrixInversion/Three_Step_DTZNN.slx

MatrixInversion/Five_Step_DTZNN.slx

Resultados modelos DTZNN

Resultados obtenidos usando $\tau = 0{,}01$ y $h = 0{,}4$



Simulación en tiempo real

“*Zeroing Neural Network*” tolerante a ruidos

Modelo NTZNN

Para los modelos tolerantes a ruidos se ha propuesto cambiar la fórmula de diseño por $\dot{E}(t) = \alpha E(t) - \beta \int_0^t E(\tau)\mathrm{d}\tau$, con los parámetros $\alpha, \beta > 0$.

En este ejemplo usaremos de ejemplo el modelo con parámetros variables VPNTZNN cuya fórmula de diseño es la siguiente[5].

$$\dot{E}(t) = -\mu_1(t)E(t) - \mu_2(t)\int_0^t E(\tau)\mathrm{d}\tau$$

dode $\mu_1(t)$ y $\mu_2(t)$ están definidos como

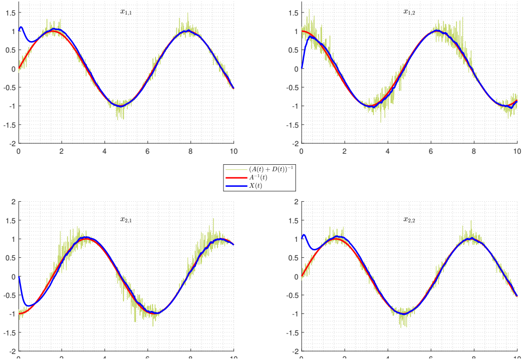
$$\begin{cases} \mu_1(t) = 3\exp\left(\frac{\alpha t}{2}\right) - \frac{\alpha}{2} \\ \mu_2(t) = \exp(\alpha t) \end{cases}$$

[5] Lin Xiao et al. “A variable-parameter noise-tolerant zeroing neural network for time-variant matrix inversion with guaranteed robustness”. En: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (2020), págs. 1535-1545

Simulación del modelo VPNTZNN

El modelo dinámico del sistema nos queda de la siguiente manera:

$$A(t)\dot{\chi}(t) = -\mu_1(t)(A(t)\chi(t) - I) - \dot{A}(t)\chi(t) - \mu_2(t)\int_0^t (A(\tau)\chi(\tau) - I)d\tau + D(t)$$



;

NoiseTolerantMatrixInversion/setup.slx

NoiseTolerantMatrixInversion/VPNTZNN.slx

Plan

“*Zeroing Neural Network*” para resolver un problema de optimización cuadrática dinámico con restricciones

Formulación del problema

$$\begin{aligned} &\min f(\mathbf{x}(t), t) = \mathbf{x}^{\mathrm{T}}(t)P(t)\mathbf{x}(t) + \mathbf{q}^{\mathrm{T}}(t)\mathbf{x}(t) \\ &\text{s.t. } A(t)\mathbf{x}(t) = \mathbf{b}(t) \end{aligned}$$

Para resolver este problema usamos los multiplicadores de lagrange, definimos la función de lagrange

$$L(\mathbf{x}(t), \boldsymbol{\lambda}(t), t) = f(\mathbf{x}(t), t) + \boldsymbol{\lambda}^{\mathrm{T}}(t)(A(t)\mathbf{x}(t) - \mathbf{b}(t))$$

Podemos solucionar el problema cuadrático variante en el tiempo si llevamos a cero las siguientes ecuaciones en todo instante de tiempo

$$\begin{cases} \frac{\partial L(\mathbf{x}(t),\boldsymbol{\lambda}(t),t)}{\partial \mathbf{x}(t)} = P(t)\mathbf{x}(t) + \mathbf{q}(t) + A^{T}(t)\boldsymbol{\lambda}(t) = 0 \\ \frac{\partial L(\mathbf{x}(t),\boldsymbol{\lambda}(t),t)}{\partial \boldsymbol{\lambda}(t)} = A(t)\mathbf{x}(t) - \mathbf{b}(t) = 0. \end{cases}$$

Formulación del problema

Podemos reescribir las ecuaciones como

$$W(t)\mathbf{y}(t) + \mathbf{u}(t) = 0$$

Donde

$$W(t) = \begin{bmatrix} P(t) & A^T(t) \\ A(t) & \mathbf{0} \end{bmatrix} \in \mathrm{R}^{(n+m)\times(n+m)}$$

$$\mathbf{y}(t) = \begin{bmatrix} \mathbf{x}(t) \\ \boldsymbol{\lambda}(t) \end{bmatrix} \in \mathrm{R}^{n+m}, \qquad \mathbf{u}(t) = \begin{bmatrix} \mathbf{q}(t) \\ -\mathbf{b}(t) \end{bmatrix} \in \mathrm{R}^{n+m}.$$

Formulación del problema

Tenemos definida entonces nuestra función de error $E(t) = W(t)\mathbf{y}(t) + \mathbf{u}(t)$, utilizando la fórmula de diseño $\dot{E}(t) = -\gamma E(t)$ obtenemos el siguiente modelo de dinámica implícita en tiempo continuo[6][7].

$$W(t)\dot{\mathbf{y}}(t) = -\dot{W}(t)\mathbf{y}(t) + \gamma(W(t)\mathbf{y}(t) + \mathbf{u}(t)) + \dot{\mathbf{u}}(t)$$

[6] Bolin Liao, Yunong Zhang y Long Jin. "Taylor $O(h^3)$ discretization of ZNN models for dynamic equality-constrained quadratic programming with application to manipulators". En: *IEEE transactions on neural networks and learning systems* 27.2 (2015), págs. 225-237

[7] Yunong Zhang y Zhan Li. "Zhang neural network for online solution of time-varying convex quadratic program subject to time-varying linear-equality constraints". En: *Physics Letters A* 373.18-19 (2009), págs. 1639-1643

Modelos en tiempo discreto

Taylor-type DTZNNK	$\mathbf{y}_{k+1} = 1{,}5\mathbf{y}_k - \mathbf{y}_{k-1} + 0{,}5\mathbf{y}_{k-2} - W_k^{-1}\left(\tau \dot{W}_k \mathbf{y}_k + h\left(W_k \mathbf{y}_k + \mathbf{u}_k\right) + \tau \dot{\mathbf{u}}_k\right)$
Taylor-type DTZNNU	$\mathbf{y}_{k+1} = 1{,}5\mathbf{y}_k - \mathbf{y}_{k-1} + 0{,}5\mathbf{y}_{k-2} - W_k^{-1}\left(\Delta W_k \mathbf{y}_k + h\left(W_k \mathbf{y}_k + \mathbf{u}_k\right) + \Delta \mathbf{u}_k\right)$
Euler-type DTZNNK	$\mathbf{y}_{k+1} = \mathbf{y}_k - W_k^{-1}\left(\tau \dot{W}_k \mathbf{y}_k + h\left(W_k \mathbf{y}_k + \mathbf{d}_k\right) + \tau \dot{\mathbf{u}}_k\right)$
Euler-type DTZNNU	$\mathbf{y}_{k+1} = \mathbf{y}_k - W_k^{-1}\left(\bar{\Delta} W_k \mathbf{y}_k + h\left(W_k \mathbf{y}_k + \mathbf{u}_k\right) + \bar{\Delta} \mathbf{u}_k\right)$
	$\bar{\Delta} W_k = W_k - W_{k-1}, \bar{\Delta}\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$
	$\Delta W_k = (3W_k - 4W_{k-1} + W_{k-2})/2, \Delta \mathbf{u}_k = (3\mathbf{u}_k - 4\mathbf{u}_{k-1} + \mathbf{u}_{k-2})/2$

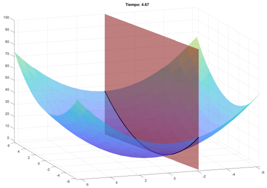
Problema de ejemplo

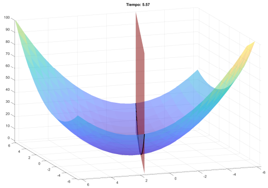
Usaremos el siguiente problema como ejemplo:

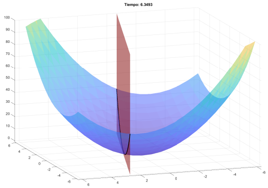
$$
\begin{aligned}
\min \quad & ((\sin t)/4+1)x_1^2(t) + ((\cos t)/4+1)x_2^2(t) \\
& \quad + \cos t x_1(t)x_2(t) + \sin 3t x_1(t) + \cos 3t x_2(t), \\
\text{s. t} \quad & \sin 4t x_1(t) + \cos 4t x_2(t) = \cos 2t.
\end{aligned}
$$

- DynamicQP/setup.slx
- DynamicQP/EULER_TYPE_ZNNK.slx
- DynamicQP/EULER_TYPE_ZNNU.slx
- DynamicQP/TAYLOR_TYPE_ZNNK.slx
- DynamicQP/TAYLOR_TYPE_ZNNU.slx

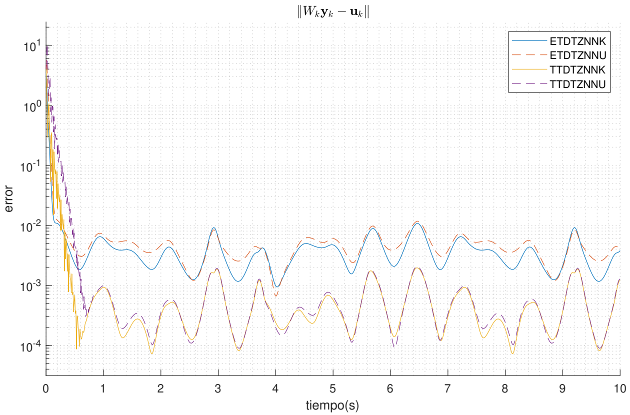
Problema variante en el tiempo







Resultados problema QP



Plan

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.
- Establecer nuestra función de error.

CUCEI

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.
- Establecer nuestra función de error.
- Elegir o proponer una función para regir el comportamiento dinámico de nuestra función de error.

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.
- Establecer nuestra función de error.
- Elegir o proponer una función para regir el comportamiento dinámico de nuestra función de error.
- Obtener el modelo dinámico en tiempo continuo.

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.
- Establecer nuestra función de error.
- Elegir o proponer una función para regir el comportamiento dinámico de nuestra función de error.
- Obtener el modelo dinámico en tiempo continuo.
- En caso de requerirlo, usar algún método de discretización o proponer uno nuevo.

Recapitulando

- Identificar el problema y ver si podemos convertirlo en un sistema de ecuaciones.
- Establecer nuestra función de error.
- Elegir o proponer una función para regir el comportamiento dinámico de nuestra función de error.
- Obtener el modelo dinámico en tiempo continuo.
- En caso de requerirlo, usar algún método de discretización o proponer uno nuevo.
- Realizar el análisis de estabilidad.

Zeroing Neural Networks

Gracias por su atención !