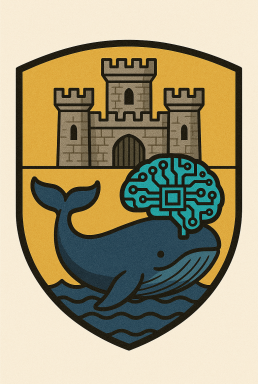
Resumen: Uso de datos subrogados II



Autor A. Romo ai.romo.moena@upm.es

1 Introducción

Este documento explica los detalles de implementación en la segunda recopilación de datos de ejecución de una Liquid State Machine (LSM) [4] utilizados para entrenar modelos subrogados. Se realizaron algunos cambios en el modelo con respecto a la primera versión:

1. Se cambio el simulador por un simulador propio (eBallena. Ver documentacion_eBallena.pdf en la carpeta de documentos). Esto con el objetivo de mejorar la velocidad de simulación.
2. Se agregó delay sinaptico al modelo neuronal (ver sección 3).
3. Se agregó periodo refractario al modelo neuronal (ver sección 3).
4. Se modificaron los parametros con los que se codificó el dataset para hacerlo más disperso (sparse, es decir, con menor cantidad de spikes).
5. Se considero como parametro a evaluar la constante τ (tau), la habia permanecido ija en la recopilación de datos 1, pero que tiene un peso importante en el desempeño del modelo (ver sección 3).

Como resultado de esas mejoras se recopiló datos de ejecuciones del modelo de LSM nuevamente. En la recopilacion de datos se considero el modelo de Liquid State Machine de la seccion 4 y el problema FSDD (free spoken digits dataset, ver sección 2). El modelo de LSM requiere varios hiperparametros. Se seleccionaron 4 de ellos (ver sección 5) los cuales se consideraron que tienen especial relevancia en el desempeño y eficiencia del modelo. Sin desmedro de lo anterior, otros hiperparametros tales como la probabilidad de conexiones input-reservoir tambien pueden ser criticos en el desempeño y eficiencia. Más adelante puede explorarse este hiperparametro tambien. Se considero una grilla utilizando los 4 hiperparametros y se evaluó cada combinacion 5 veces. Los resultados fueron guardados y se explica como interpretarlos en la sección 5.

2 Dataset

El dataset utilizado es "Free Spoken Digits Dataset" (FSDD) [3], el cual consiste en 3000 ejemplos de números (del 0 al 9) hablados en inglés. La duración de los audios es de 1 seg. Para procesarlos, los audios son cargados en Python y se extraen características mediante la técnica "coeficientes cepstrales en las frecuencias de Mel (MFCCs)". Este es un procedimiento estándar que aplica filtros a la señal de audio imitando la percepción del oído humano. Los canales de audio del FSDD luego son normalizados y convertidos a spikes utilizando "Rate encoding". Este dataset suele alcanzar un accuracy máximo de alrededor del 90%–92% luego de aplicar técnicas para modificar el algoritmo básico de una LSM.

3 Modelo neuronal LIF

El modelo LIF (leaky integrate and fire) es un modelo simplificado del funcionamiento de una neurona. En él, cada neurona mantiene un estado interno ($V_m(t)$, voltaje de su membrana). En ausencia de estímulos, el voltaje de la membrana tiende asintóticamente a un voltaje de equilibrio V_{rest}, con una velocidad determinada por la constante τ. Los estímulos ($I(t)$, spikes presinápticos representados como corriente que ingresa a la neurona) se modelan como una sumatoria de voltajes que se agregan al voltaje de la membrana. La ecuación 1 describe este comportamiento, que puede estudiarse con mayor profundidad en [2].

$$\tau \frac{d}{dt} V_m(t) = -\big(V_m(t) - V_{\text{rest}}\big) + R\, I(t), \tag{1}$$

$$s(t) = \begin{cases} 1, & \text{if } V_m(t^-) \geq V_{\text{thres}}, \\ 0, & \text{otherwise}, \end{cases} \qquad \text{and when } s(t) = 1, \text{ then } V_m(t) \leftarrow V_{\text{reset}}, \tag{2}$$

En la práctica, $RI(t)$ se implementa como una sumatoria de pesos sinápticos $\sum_{i=1}^{N} \omega_i \cdot s_i(t)$, donde ω_i representa el peso sináptico de la neurona presináptica i y $s_i(t)$ indica la presencia o ausencia de spikes (disparos) de la neurona i. Cuando el voltaje de una neurona supera el umbral v_{thres}, esta genera un spike y su voltaje se reinicia a v_{reset}. Es común también que, durante un tiempo después de haber disparado, la neurona se encuentre inhibida y no pueda recibir spikes presinápticos ni volver a disparar. A este periodo se le conoce como periodo refractario o refractory time. Cuando una neurona hace spike, la señal puede tardar un tiempo (delay de propagación) en sumarse al voltaje de la neurona post sináptica.

Conceptos del modelo		
Concepto	**Nombre**	**Descripcion**
Spike	Spike, disparo	Evento discreto que utilizan las neuronas para comunicarse
Sinapsis	Sinpasis, conexion	Conexiones entre 2 neuronas. Define una neurona presinaptica, otra post sinaptica y un peso sinaptico ω.
Resumen de variables del modelo LIF		
Variable	**Nombre**	**Descripcion**
$V_m(t)$	Voltaje de membrana	Estado interno de la neurona.
V_{rest}	Voltaje de descanso	Volaje al cual tiende la neurona en reposo.
V_{reset}	Voltaje de reset	Volaje de la neurona despues de hacer spike.
ω_i	Peso sinaptico	Voltaje que se suma a la neurona cuando la neurona presinaptica i hace spike.
τ	Constante de tiempo	Define que tan rapido se mueve la neurona en direccion a V_{rest}
Refractory time	Refractory time	Cuanto tiempo la neurona se encuentra inactiva despues de hacer spike.
Delay	Delay sináptico	Tiempo que tarda una señal en afectar a la neurona post sináptica.

4 Modelo de LSM

El modelo de Liquid State Machine utilizado puede encontrarse en [1]. Es un modelo estandar y simplificado que es utilizado por los autores como modelo base. También es común ver en la literatura modelos similares. El modelo consiste en:

- Una capa de entrada o **inputs**, los cuales codifican los datos del dataset. Cada input se conecta con las neuronas del reservoir con probabilidad p_{in-re}. La mitad de las sinapsis son generadas con peso sinaptico de $+w_{in}$ y la otra mitad con peso de $-w_{in}$. El valor de w_{in} es igual para todas las sinapsis input-reservoir.
- Un **reservoir** de neuronas LIF. El reservoir es un cubo con neuronas exitatorias e inhibitorias. Las sinapsis que salen de neuronas exitatorias tienen un peso positivo $+\omega_{re}$ mientras que las sinpsis cuyo origen es una neurona inhibitoria tienen peso negativo $-\omega_{re}$. El valor de ω_{re} es igual para todas las sinapsis del reservoir. Las sinapsis se generan de manera probabilistica basandose en el tipo de neurona (exitatoria o inhibitoria) y en la distancia entre neuronas dentro del cubo.
- Una capa de **readout** el cual es un modelo sencillo de machine learning que toma como entrada el estado del reservoir despues de simular una instancia del dataset y es entranado para predecir la etiqueta correspondiente.

En la seccion 5 se presentan los hiperparametros que se hicieron variar para generar los datos. Todos los demas hiperparametros del modelo se mantuvieron igual que en [1].

5 Hiperparámetro utilizados y datos recopilados

A continuacion se presentan los hiperparametros que se hicieron variar para generar los datos (tabla 1) y los parametros que se dejaron en valores fijos por defecto (tabla 2).

Parametro	**Definicion**	**Valores**
τ (tau)	Constante de tiempo del modelo LIF. Mientras más grande sea más se tardará el voltaje de la membrana en volver a v_{rest}. Esto significa más memoria, pero tambien mayor cantidad de spikes.	[80, 120, 160, 200]
ω_{in}	Peso sinaptico de las conexiones input-reservoir. Mayores valores generarán más spikes	[2, 3, 4, 5]
ω_{re}	Peso sinaptico de las conexiones reservoir-reservoir. Mayores valores generarán más spikes.	[0.5, 1.3, 2.1, 3.0]
p_{neu_ex}	Probabilidad de que una neurona del reservoir sea exitatoria. Las neuronas serán inhibitorias con probabilidad 1-p_{neu_ex}	[0.65, 0.72, 0.78, 0.85]

Table 1: Hiperparametros modificados

.

Parametro	**Definicion**	**Valor**
p_{in-re}	Probabilidad de que un input se conecte con una neurona del reservoir. En el paper [1] es encontrada mediante la técnica propuesta (MAdapeter). En este proyecto se fijó en 0.03 de manera empirica	0.03
n_{neu}	Cantidad de neuronas en el reservoir.	1000
L	Constante λ en [1]. Valores más altos harán que sea más probable que 2 neuronas se conecten.	3
$C_{ee}, C_{ei}, C_{ie}, C_{ii}$	Constante C. Valores más altos harán que sea más probable conectar dos neuronas. Su valor depende de si las neuronas son exitatorias o inhibitorias.	$C_{ee} = 0.2$, $C_{ei} = 0.1$, $C_{ie} = 0.05$, $C_{ii} = 0.3$

Table 2: Hiperparametros no modificados

El procedimimento para obtener las métricas fue el siguiente:

1. Los 4 hiperametros fueron combinandos generando $4^4 = 256$ combinaciones distintas.
2. Por cada combinacion de hiperparametros se construye un modelo y se simula todo el dataset.
3. Se obtiene el estado del reservoir para cada instancia. El estado corresponde a la cantidad de spikes de cada neurona durante la simulación. El estado por tanto es un vector de dimensión n_{neu}.
4. Los estados y las etiquetas de clase son divididos en 5 splits para validacion cruzada (5-fold cross validation). Cada split genera un conjunto de entrenamiento y test estratificado (es decir, preocupandose de balancear las clases).
5. Con dada split se entrena una regresión logística usando los estados y etiquetas de train. Con los estados del dataset de test y la regresión se obtienen prediciones (a cual categoria partenece cada instancia). En base a las predicciones se obtiene el accuracy y el accuracy sobre el conjunto de entramiento. A lo largo de los 5 splits se calculan las métricas acc, $std\ acc\_train$, std_train. Usando los estados del reservoir tambien se calculan num_{spikes} y std_{spikes}. (ver tabla 3).

Los pasos del 2 al 5 son repetidos 5 veces, obteniendo así 5 conjuntos de metricas distintos por cada combinacion de hiperparametros. Esto se hace así porque los hiperparametros indican probabilidades y partir de los mismos hiperparametros pueden formarse distintos modelos con distintos resultados y performance. Para clarificar el uso de la validación cruzada se presenta el siguiente pseudocódigo.

```
resultados = []
for hiperparametros in combinaciones_hiperparametros: do
    for i = 1 to 5 do
        lsm = crear_lsm(hiperparametros)

        estados = []
        for instance in dataset do
            estado = simulate(lsm, instance)
            add estado to estados
        end for
        spikes = contatar spikes por instancia

        acc = []
        acc_train = []
        for split in 5-fold_splits do
            ajustar modelo con split.train

            new_acc = evaluar modelo con split.test
            add new_acc to acc

            new_acc = evaluar modelo con split.train
            add new_acc to acc_train
        end for

        metricas = mean(acc), std(acc), mean(acc_train), std(acc_train), mean(spikes), std(spikes)
        agregar a resultados hiperparametros, metricas
    end for # cierre for 1...5
end for # cierre for hiperparametros
```

Finalmente las métricas se definen en la tabla 3. Los datos se encuentran en el archivo datos_subrogados2_kfold.csv

. .

Parametro	Definicion
acc	Accuracy del modelo usando el dataset de test. Este es el resultado final. Es un promedio sobre el 5-fold cross validation
std	Es la desviación estandar del accuracy sobre el 5-fold cross validation
acc_{train}	Accuracy obtenido usando el mismo subconjunto del dataset usado para entrenar. Es un promedio sobre el 5-fold cross validation. Puede servir para detectar sobreajuste.
std_{train}	Es la desviación estandar del accuracy de entrenamiento sobre el 5-fold cross validation
num_{spikes}	Cantidad de spikes. Más spikes significa más tiempo de simulación y más gasto energetico. Es un promedio de todas las instancias del dataset de test.
std_{spikes}	Desviación estandar de la cantidad de spikes.

Table 3: Métricas obtenidas

6 Notas finales

Los valores de los hiperparámetros propuestos se obtuvieron de manera más analítica que en la primera recopilación de datos. En particular, el hiperparámetro τ resultó tener un papel más importante que no se había tenido en cuenta anteriormente.

Se incluyó también el número de spikes generado por cada combinación de hiperparámetros, el cual, en teoría, debería estar muy relacionado con ellos. Si todas las combinaciones de hiperparámetros tienen una accuracy similar, debería considerarse mejor aquella que genere menos spikes. Si se encuentra alguna forma de modelar esto, sería muy bueno.

Se incluyó también la accuracy obtenido en el conjunto de entrenamiento. La teoría dice que si la accuracy de entrenamiento es mucho mayor que la de test, esto significa sobreajuste. El sobreajuste se combate con más datos (lo cual no depende de nosotros en este caso) o con modelos más sencillos. En nuestro caso, ninguno de los hiperparámetros modifica directamente la complejidad del modelo, pero varios de ellos pueden influir de manera indirecta (por ejemplo, tener valores de pesos sinápticos muy pequeños es equivalente a eliminar conexiones. Lo mismo podría ocurrir al aumentar la proporción de neuronas inhibitorias).

Cualquier duda consultar a ai.romo.moena@upm.es!

References

[1] Anmol Biswas, Nivedya S Nambiar, Kushal Kejriwal, and Udayan Ganguly. Madapter: A multimodal adapter for liquid state machines configures the input layer for the same reservoir to enable vision and speech classification. *2023 International Joint Conference on Neural Networks (IJCNN)*, page 1–6, Jun 2023. doi:10.1109/ijcnn54540.2023.10191376.

[2] Wulfram Gerstner, Werner Kistler, Richard Naud, and Liam Paninski. *Introduction: Neurons and Mathematics*, chapter Introduction: Neurons and Mathematics. was published with Cambridge University Press, 2014. URL: https://neuronaldynamics.epfl.ch/online/Ch1.html.

[3] Zohar Jackson. Free spoken digits dataset. Accessed: 2025-09-29. URL: https://www.kaggle.com/datasets/joserzapata/free-spoken-digit-dataset-fsdd.

[4] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 11 2002. doi:10.1162/089976602760407955.