Trabajo práctico 2 – Inteligencia artificial
UNaB 2023 cuat. 2

G. Sebastián Pedersen

sebastian.pedersen@unab.edu.ar

Lunes 16 de Octubre de 2022

1. Ponderación de este TP respecto a la nota final: 30 %.
2. Entregar las resoluciones de las partes teóricas en un único archivo `pdf` con nombre `tp2_teo_IA_apellido_nombre.pdf`.
3. Entregar las resoluciones de las partes de programación en un único archivo `zip` con nombre `tp2_prog_IA_apellido_nombre.zip`. El archivo `zip` debe contener únicamente los scripts y salidas de los mismos.
4. Entregar por el campus.
5. Fecha y hora límite de entrega: Dom 12/Nov/2023 23:59 hs. (GMT-3).
6. El TP es de entrega individual.
7. **El cumplimiento de las pautas de entrega es parte de la resolución del TP.**

1. **Una red neuronal simple**

   Sea $X = \{x^{(1)}, \ldots, x^{(m)}\}$ un conjunto de datos de m muestras con 2 características, es decir, $x^{(i)} \in \mathbb{R}^2$. Las muestras se clasifican en 2 categorías con etiquetas $y^{(i)} \in \{0, 1\}$. Un gráfico de dispersión del conjunto de datos se muestra en la figura 1.

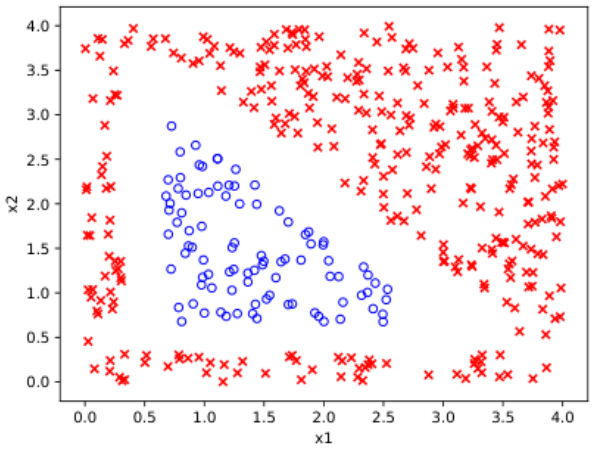

Figura 1. Scatter plot de los datos.

Los ejemplos en la clase 1 están marcados como "x" y los ejemplos en la clase 0 están marcados como "o". Queremos realizar una clasificación binaria utilizando una red neuronal simple con la arquitectura que se muestra en la figura 2.

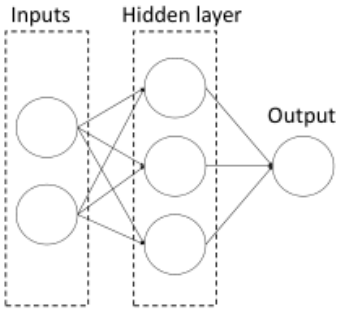

Figura 2. Arquitectura de la red neuronal.

Denote las dos características x_1 y x_2, las tres neuronas en la capa oculta h_1, h_2 y h_3, y la neurona de salida como o. Denotemos el peso de x_i a h_j como $w^1_{i,j}$ para $i \in \{1, 2\}$, $j \in \{1, 2, 3\}$, y el peso de h_j a o como w^2_j. Finalmente, denotemos el bias para h_j como $w^1_{0,j}$, y el bias o como w^2_0. Para la función de pérdida, usaremos la pérdida cuadrática promedio en lugar de la habitual (el negativo del log de la verosimilitud, NLL):

$$l = \frac{1}{m} \sum_{i=1}^{m} \left(o^{(i)} - y^{(i)} \right)^2$$

donde $o^{(i)}$ es el resultado de la neurona de salida del dato i.

a) Supongamos que usamos la función sigmoidea como activación para h_1, h_2, h_3 y o ¿Cómo queda la actualización de descenso por gradiente para $w^1_{1,2}$, suponiendo que usamos una tasa de aprendizaje α? La respuesta debe escribirse en términos de $x^{(i)}$, $o^{(i)}$, $y^{(i)}$ y los pesos.

b) Ahora, suponga que en lugar de usar la función sigmoidea para la función de activación para h_1, h_2, h_3 y o, en su lugar usamos la función escalonada f, definida como

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

¿Es posible tener un conjunto de pesos que permitan a la red neuronal clasificar este conjunto de datos con 100 % de accuracy? Si es posible, proporcione un conjunto de pesos que lo permitan y explique su razonamiento. Si no es posible, explique su razonamiento.

Pista: hay tres lados en un triángulo y hay tres neuronas en la capa oculta.

c) Sean las funciones de activación para h_1, h_2, h_3 la función lineal $f(x) = x$ y la función de activación para o será la misma función escalonada de antes. ¿Es posible tener un conjunto de pesos que permitan a la red neuronal clasificar este conjunto de datos con 100 % de accuracy? Si es posible, proporcione un conjunto de pesos que lo permitan y explique su razonamiento. Si no es posible, explique su razonamiento.

d) [**Programación**]. En un archivo `p01_simple_nn.py` programe una pasada hacia adelante de esta arquitectura suponiendo todas funciones de activación sigmoideas. Es decir debe programar dos funciones:

- Una que mande el input x a la capa oculta.
- Otra que mande la salida de la capa oculta a la salida definitiva.

2. **Redes Neuronales: MNIST clasificación de imágenes.**

En este problema, implementará una red neuronal convolucional simple para clasificar imágenes (en escala de grises) de dígitos escritos a mano (0 - 9) del conjunto de datos MNIST. El conjunto de datos contiene 60.000 imágenes de entrenamiento y 10.000 imágenes de testeo. Cada imagen es 28×28 píxeles de tamaño (con un solo canal por es en blanco y negro). También incluye etiquetas para cada ejemplo, es decir indicando el número real escrito en la imagen.

En la figura 3 se pueden ver algunos ejemplos de las imágenes contenidas en el dataset MNIST.

Figura 3. Algunas imágenes del dataset MNIST.

Los datos para este problema se pueden encontrar en la carpeta `data` como `images_train.csv`, `images_test.csv`, `labels_train.csv` y `labels_test.csv`.

El código para esta problema se puede encontrar `/src/p02_cnn.py`.

El código divide el conjunto de 60.000 imágenes y etiquetas de entrenamiento en conjuntos de 59.600 ejemplos de entrenamiento y 400 ejemplos para validación.

Para empezar, implementará una red neuronal convolucional simple, con pérdida de entropía cruzada, y la entrenará con el conjunto de datos correspondiente.

La arquitectura es la siguiente:

I. La primera capa es una capa convolucional con 2 canales de salida con un tamaño de convolución de 4x4.

II. La segunda capa es una capa de max pooling con stride y ancho de 5x5.

III. La tercera capa es una capa de activación ReLU.

IV. Después de la cuarta capa, los datos se aplanan (flattening) en una sola dimensión.

V. La quinta capa es una sola capa lineal con tamaño de salida 10 (la cantidad de clases o etiquetas).

VI. La sexta capa es una capa softmax que calcula las probabilidades para cada clase.

VII. Finalmente, usamos una pérdida de entropía cruzada como nuestra función de pérdida.

Hemos proporcionado todas las funciones foward para estas capas. Su trabajo será implementar funciones que calculen los gradientes para estas capas.

Ahora un texto adicional que podría ser útil para comprender las funciones foward. La siguiente ecuación define lo que queremos decir con una convolución 2d:

$$output[out\_ch, x, y] = conv\_bias[out\_ch] + \sum_{di,dj,in_ch} input[in\_ch, x+di, y+dj] * conv\_weights[out\_ch, di, dj]$$

donde di y dj recorren ancho y alto de la convolución respectivamente.

La salida de una convolución es de tamaño (`#` canales de salida, ancho de entrada - ancho de convolución + 1, altura de salida - altura de convolución + 1). Tenga en cuenta que la dimensión de la salida es menor debido al padding.

Las capas max pooling simplemente toman el elemento máximo de la grilla, y están definidax por

$$output[out\_ch, x, y] = \max_{di,dj} input[in\_ch, x * ancho\_pool + di, y * alto\_pool + dj]$$

La ReLU es simplemente $max(0, x)$ donde x es la entrada.

Usamos la pérdida de entropía cruzada como nuestra función de pérdida. Recuerde que para un solo ejemplo (x, y), la entropía cruzada es

$$CE(y, \widehat{y}) = -\sum_{k=1}^{K} y_k \log \widehat{y}_k$$

donde $\widehat{y} \in \mathbb{R}^K$ es el vector de salidas softmax del modelo para el ejemplo de entrenamiento x, e $y \in \mathbb{R}^K$ es el vector real para el ejemplo de entrenamiento x tal que $y = (0, \ldots, 0, 1, 0, \ldots, 0)^t$ contiene un solo 1 en la posición de la clase correcta (llamada representación o codificación "one-hot").

También estamos haciendo un descenso por gradiente mini-batch con un tamaño de lote de 16. Normalmente lo haríamos iterar sobre los datos varias veces con varias épocas, pero para esta tarea solo hacemos 400 lotes para ahorrar tiempo.

a) **[Programación]**. Implemente las siguientes funciones dentro de `p02_cnn.py`. Le recomendamos que empiece por la parte superior de la lista y siga hacia abajo:

I. `backward_softmax`.
II. `backward_relu`
III. `backward_cross_entropy_loss`
IV. `backward_linear`
V. `backward_convolution`
VI. `backward_max_pool`

b) **[Programación]**. Ahora implemente una función que calcule la pasada backward completa.

I. `backward_propagation`.

3. **EM semisupervisado.**

EM es un algoritmo clásico para el aprendizaje no supervisado (es decir, aprendizaje con variables ocultas o latentes). En este problema exploraremos una de las formas en que EM se puede adaptar a la configuración semisupervisada, donde tenemos algunos ejemplos etiquetados. junto con ejemplos no etiquetados. En la configuración estándar no supervisada, tenemos m ejemplos no etiquetados $\{x^{(1)}, \ldots, x^{(m)}\}$. Deseamos conocer los parámetros de $p(x, z; \theta)$ a partir de los datos, pero no se observan los $z^{(i)}$. EM clásico está diseñado para este propósito, donde maximizamos el intractable $p(x; \theta)$ indirectamente realizando iterativamente el paso E y el paso M, cada vez maximizando un límite inferior tractable de $p(x; \theta)$. Nuestro objetivo se puede escribir concretamente como:

$$l_{\text{unsup}}(\theta) = \sum_{i=1}^{m} \log p\left(x^{(i)}; \theta\right) = \sum_{i=1}^{m} \log \sum_{z^{(i)}} p\left(x^{(i)}, z^{(i)}; \theta\right)$$

Ahora, intentaremos construir una extensión de EM al entorno semisupervisado. Supongamos que tenemos $\widetilde{m}$ ejemplos adicionales etiquetados $\{(\widetilde{x}^{(1)}, \widetilde{z}^{(1)}), \ldots, (\widetilde{x}^{(\widetilde{m})}, \widetilde{z}^{(\widetilde{m})})\}$ donde se observan tanto x como z. Queremos maximizar simultáneamente la probabilidad marginal de los parámetros usando los ejemplos no etiquetados, y la probabilidad total de los parámetros usando los ejemplos etiquetados, optimizando su suma ponderada (con algún hiperparámetro α). Más concretamente:

$$l_{\text{sup}}(\theta) = \sum_{i=1}^{\widetilde{m}} \log p\left(\widetilde{x}^{(i)}, \widetilde{z}^{(i)}; \theta\right)$$

$$l_{\text{semi-sup}}(\theta) = l_{\text{unsup}} + \alpha l_{\text{sup}}(\theta)$$

Podemos derivar los pasos de EM para el entorno semisupervisado usando el mismo enfoque y pasos como antes. Le recomendamos fuertemente que se convenza usted mismo (no es necesario incluirlo) que queda así:

E-step (semi-supervisado) Para cada $i \in \{1, \ldots, m\}$ sea

$$Q_i^t(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^t)$$

M-step (semi-supervisado)

$$\theta^{t+1} := \underset{\theta}{\text{argmax}} \left[\sum_{i=1}^{m} \left(\sum_{z^{(i)}} Q_i^t(z^{(i)}) \log \frac{p(z^{(i)}, x^{(i)}; \theta)}{Q_i^t(z^{(i)})} \right) + \alpha \left(\sum_{i=1}^{\widetilde{m}} \log p(\widetilde{x}^{(i)}, \widetilde{z}^{(i)}; \theta) \right) \right]$$

a) **Convergencia**. Mostrar que este algoritmo finalmente converge. Para esto observar que es suficiente mostrar que nuestro objetivo semi-supervisado $l_{\text{semi-sup}}(\theta)$ aumenta monótonamente con cada iteración de los pasos E y M. Específicamente, sea θ^t los parámetros obtenidos al final de t EM-pasos. Demuestre que $l_{\text{semi-sup}}(\theta^{t+1}) \geq l_{\text{semi-sup}}(\theta^t)$.

GMM semi-supervisado Ahora revisaremos el modelo GMM, para aplicar nuestro algoritmo EM semi-supervisado. Consideremos un escenario en el que los datos se generan a partir de k distribuciones gaussianas, con medias desconocidas $\mu_j \in \mathbb{R}^d$ y covarianzas $\Sigma_j \in \mathbb{S}_+^d$ donde $j \in \{1, \ldots, k\}$. Tenemos m datos $x^{(i)} \in \mathbb{R}^d$, y cada dato tiene un valor latente correspondiente (oculto/desconocido) variable $z^{(i)} \in \{1, \ldots, k\}$ indicando a qué distribución pertenece $x^{(i)}$. Específicamente, $z^{(i)} \sim \text{Multinomial}(\phi)$, tal que $\sum_{j=1}^{k} \phi_k = 1$ y $\phi_j \geq 0$ para todo j, y $x^{(i)} | z^{(i)} \sim N(\mu_{z^{(i)}}, \Sigma_{z^{(i)}})$ i.i.d. Entonces, μ, Σ y ϕ son los parámetros del modelo.

También tenemos $\widetilde{m}$ datos adicionales $\widetilde{x}^{(i)}$, y una variable asociada observada $\widetilde{z}$ indicando la distribución a la que pertenece $\widetilde{x}^{(i)}$. Nótese que $\widetilde{z}^{(i)}$ son constantes conocidas (en contraste con $z^{(i)}$ que son variables aleatorias desconocidas). Como antes, suponemos $\widetilde{x}^{(i)} | \widetilde{z}^{(i)} \sim N(\mu_{\widetilde{z}^{(i)}}, \Sigma_{\widetilde{z}^{(i)}})$ i.i.d.

En resumen, tenemos $m + \widetilde{m}$ ejemplos, de los cuales m son datos x no etiquetados con valores no observados z, y $\widetilde{m}$ son datos etiquetados $\widetilde{x}^{(i)}$ con las correspondientes etiquetas observadas $\widetilde{z}^{(i)}$.

El algoritmo EM tradicional está diseñado para tomar solo los m ejemplos no etiquetados como entrada y aprender los parámetros μ, Σ y ϕ.

Nuestra tarea ahora será aplicar el algoritmo EM semisupervisado a los GMM para aprovechar los $\widetilde{m}$ ejemplos etiquetados adicionales, y crear una actualización semisupervisada de E-step y M-step específicas para los GMM. Siempre que sea necesario, puede citar notas de clase para derivaciones y pasos.

b) **E-Step semi-supervisado**. Indique claramente cuáles son todas las variables latentes que necesitan ser reestimadas en el paso E. Derive el paso E que reestimar todas esas variables latentes. Su expresión final del paso E solo debe incluir x, z, μ, Σ, ϕ y constantes universales.

c) **M-Step semi-supervisado**. Indique claramente cuáles son todos los parámetros que necesitan ser reestimados en el paso M. Derive el paso M que reestimar todos esos parámetros. Específicamente, derive expresiones de forma cerrada para las reglas de actualización de parámetros para μ^{t+1}, Σ^{t+1} y ϕ^{t+1}.

d) [**Programación**] **Implementación EM clásica (no supervisada)**. Para este ítem, solo vamos a considerar los m ejemplos no etiquetados. Siga las instrucciones en `src/p03_gmm.py` para implementar el algoritmo EM tradicional y ejecútelo en el conjunto de datos no etiquetados hasta la convergencia.

Realice tres pruebas y utilice la función graficadora provista para construir un gráfico de dispersión de la asignaciones resultantes a cada cluster (una gráfico para cada prueba). Su gráfico debe indicar las asignaciones a cada cluster con colores a los que se les asignó (es decir, los clusters que tenían mayor probabilidad en el paso E final).

e) [**Programación**] **Implementación de EM semisupervisada**. Ahora considere tanto los ejemplos etiquetados como los no etiquetados (un total de $m+\widetilde{m}$), con 5 ejemplos etiquetados por cluster. Hemos proporcionado un código de inicio para dividir el conjunto de datos en matrices x de ejemplos etiquetados y $\widetilde{x}$ de ejemplos no etiquetados. Modifique `src/p03_gmm.py` para implementar el algoritmo EM modificado y ejecutarlo en el conjunto de datos hasta convergencia.

Cree un gráfico para cada prueba, como en el ítem anterior.

f) **Comparación de EM no supervisado y semisupervisado**. Describa resumidamente las diferencias que vio en EM no supervisado versus semi-supervisado para:

i. Número de iteraciones necesarias para converger.
ii. Estabilidad (es decir, ¿cuánto cambiaron las asignaciones con diferentes inicializaciones aleatorias?)
iii. Calidad general de los clusters.

Comentario: El conjunto de datos se muestreó de una mezcla de tres distribuciones gaussianas de baja varianza, y una cuarta distribución gaussiana de alta varianza. Esto debería ser útil para determinar la calidad general de los clusters hallados por cada algoritmo.