Curso ECI
Modelos Generativos Profundos para Imágenes
Entregable Autoencoders Variacionales y Aprendizaje Semi-Supervisado

Fecha de entrega: 25-08-25 23:59.

Problema 1: Implementación del Autoencoder Variacional (VAE) (33 puntos)

En este problema se utilizará PyTorch para implementar un autoencoder variacional (VAE) y aprender un modelo probabilístico del conjunto de datos MNIST de dígitos escritos a mano, binarizado. Formalmente, se observa una secuencia de píxeles binarios $\mathbf{x} \in \{0,1\}^d$, y se deja que $\mathbf{z} \in \mathbb{R}^k$ denote un conjunto de variables latentes. El objetivo es aprender un modelo de variables latentes $p_\theta(\mathbf{x})$ de la distribución de datos de alta dimensión $p_{\text{data}}(\mathbf{x})$.

El VAE es un modelo de variables latentes que aprende una parametrización específica $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$. Específicamente, el VAE se define mediante el siguiente proceso generativo:

$$
\begin{aligned}
p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; 0, I), \\
p_\theta(\mathbf{x}|\mathbf{z}) &= \prod_{j=1}^{d} \text{Bern}\left(x_j; [f_\theta(\mathbf{z})]_j\right).
\end{aligned}
$$

En otras palabras, se asume que las variables latentes $\mathbf{z}$ se muestrean de una distribución normal estándar multivariada $\mathcal{N}(\mathbf{z}; 0, I)$. Las variables latentes $\mathbf{z}$ luego se pasan a través de una red neuronal $f_\theta(\cdot)$, que es el decodificador, para obtener los logits ($logit(p) = \log(p) - \log(1-p)$) de las d variables aleatorias de Bernoulli que modelan los píxeles en cada imagen.

Aunque se desearía maximizar la verosimilitud marginal $p_\theta(\mathbf{x})$, el cálculo de $p_\theta(\mathbf{x}) = \int p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z}$ es generalmente intratable ya que implica integración sobre todos los valores posibles de $\mathbf{z}$. Por lo tanto, se plantea una aproximación variacional al verdadero posterior y se realiza inferencia amortizada como se ha visto en clase:

$$
q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mu_\phi(\mathbf{x}), \text{diag}\left(\sigma^2_\phi(\mathbf{x})\right)\right).
$$

Específicamente, se pasa cada imagen $\mathbf{x}$ a través de una red neuronal que produce la media $\mu_\phi(\mathbf{x})$ y la matriz de covarianza diagonal $\text{diag}\left(\sigma^2_\phi(\mathbf{x})\right)$ de la distribución gaussiana multivariada que aproxima la distribución sobre las variables latentes $\mathbf{z}$ dado $\mathbf{x}$. Luego se maximiza la cota inferior de la log-verosimilitud marginal para obtener una expresión conocida como la cota inferior de evidencia (ELBO):

$$
\log p_\theta(\mathbf{x}) \geq \text{ELBO}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right).
$$

Notar que la ELBO, como se muestra en el lado derecho de la expresión anterior, se descompone en dos términos: (1) el término de reconstrucción: $\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]$, y (2) el término de regularización: $-D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right)$.

El objetivo de este ejercicio es implementar el autoencoder variacional modificando los archivos `utils.py` y `vae.py`.

1. [9 puntos] Implementar el truco de reparametrización en la función `sample_gaussian` de `utils.py`. Específicamente, se debe tomar la media m y la varianza v de la distribución gaussiana $q_\phi(\mathbf{z}|\mathbf{x})$ y devolver una muestra $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$.

2. [9 puntos] A continuación, implementar la `negative_elbo_bound` en el archivo `vae.py`. Varias de las funciones en `utils.py` serán útiles, así que por favor revisar lo que se proporciona. Tener en cuenta que se pide el ELBO negativo, ya que los optimizadores de PyTorch minimizan la función de pérdida. Además, dado que se está calculando el ELBO negativo sobre un mini-batch de datos $\left\{\mathbf{x}^{(i)}\right\}_{i=1}^{n}$, hay que asegurarse de calcular el promedio $-\frac{1}{n}\sum_{i=1}^{n} \text{ELBO}\left(\mathbf{x}^{(i)}; \theta, \phi\right)$ sobre el mini-batch. Finalmente, tener en cuenta que el ELBO en sí no se puede calcular exactamente ya que el cálculo exacto del término de reconstrucción es intratable. En cambio, se pide que se estime el término de reconstrucción a mediante Monte Carlo:

$$-\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] \approx -\log p_\theta\left(\mathbf{x}|\mathbf{z}^{(1)}\right),$$

   donde $\mathbf{z}^{(1)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ denota una sola muestra. La función `kl_normal` en `utils.py` será de utilidad. Nota: de `negative_elbo_bound` también se espera que se devuelva la pérdida de reconstrucción promedio y la divergencia KL promedio.

3. [5 puntos] Para probar la implementación, ejecutar python `run_vae.py` para entrenar el VAE. Una vez que se complete la ejecución (20000 iteraciones), se mostrarán los siguientes resultados (suponiendo que tu implementación es correcta): (1) la ELBO negativa promedio, (2) el término KL promedio, y (3) la pérdida de reconstrucción promedio evaluadas en un subconjunto de prueba. Informar los tres números que se obtuvieron como parte del informe. Dado que se está usando optimización estocástica, es deseable que se ejecute el modelo varias veces y se informe la media de cada métrica y el error estándar correspondiente. (Sugerencia: la ELBO negativa en el subconjunto de prueba debería estar alrededor de 100.)

4. [5 puntos] Visualizar 200 dígitos (generar una sola imagen en una cuadrícula de 10×20 dígitos) muestreados de $p_\theta(\mathbf{x})$.

5. [5 puntos] Una variante popular del VAE se llama β-VAE. La función objetivo del β-VAE es la siguiente:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \beta D_{\text{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})\right)\right),$$

donde β es un número real positivo. Dar una interpretación intuitiva del impacto de β en la optimización del VAE; específicamente, ¿qué sucede cuando $\beta = 1$? ¿Qué pasa cuando β se aumenta por encima de 1? Para esta pregunta, será suficiente una descripción cualitativa.

Problema 2: Implementación del VAE de Mezcla de Gaussianas (GMVAE) (34 puntos)

En el Problema 1, la densidad a priori del VAE es una gaussiana isotrópica sin parámetros $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|0, I)$. Si bien esta configuración original funciona bien, hay configuraciones en las que se desea más expresividad para modelar mejor los datos. En este problema se debe implementar el GMVAE, que tiene una mezcla de gaussianas como distribución a priori. Específicamente:

$$p_\theta(\mathbf{z}) = \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}\left(\mathbf{z}|\mu_i, \text{diag}\left(\sigma_i^2\right)\right),$$

donde $i \in \{1, \ldots, k\}$ denota el índice del clúster. Por simplicidad notacional, indicaremos los parámetros de mezcla de gaussianas del modelo generativo θ como: $\{\mu_i, \sigma_i\}_{i=1}^{k}$. También se asumen pesos uniformes fijos $1/k$ sobre los posibles diferentes clusters. Aparte del prior, el GMVAE tiene una configuración idéntica a la del VAE:

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mu_\phi(\mathbf{x}), \text{diag}\left(\sigma_\phi^2(\mathbf{x})\right)\right),$$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{d} \text{Bern}\left(x_j; [f_\theta(\mathbf{z})]_j\right).$$

Si bien la ELBO para el GMVAE se define de la misma forma, i.e., $\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})\right)$, la divergencia KL $D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})\right)$ entre una gaussiana $q_\phi(\mathbf{z}|\mathbf{x})$ y una mezcla de gaussiana $p_\theta(\mathbf{z})$ no se puede calcular analíticamente. Sin embargo, se puede obtener un estimador insesgado mediante muestreo de Monte Carlo:

$$\begin{aligned} D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})\right) &\approx \log q_\phi\left(\mathbf{z}^{(1)}|\mathbf{x}\right) - \log p_\theta\left(\mathbf{z}^{(1)}\right) \\ &= \log \mathcal{N}\left(\mathbf{z}^{(1)}|\mu_\phi(\mathbf{x}), \mathrm{diag}\left(\sigma_\phi^2(\mathbf{x})\right)\right) - \log \sum_{i=1}^{k} \frac{1}{k} \mathcal{N}\left(\mathbf{z}^{(1)}|\mu_i, \mathrm{diag}\left(\sigma_i^2\right)\right), \end{aligned}$$

donde $\mathbf{z}^{(1)} \sim q_\phi(\mathbf{z}|\mathbf{x})$ denota una sola muestra.

1. [22 puntos] Implementar las funciones: (1) `log_normal` y (2) `log_normal_mixture` en `utils.py`, y la función `negative_elbo_bound` en `gmvae.py`. La función `log_mean_exp` en `utils.py` ayuda a asegurar que la implementación sea numéricamente estable.

2. [6 puntos] Para probar la implementación, ejecutar python `run_gmvae.py` para entrenar el GM-VAE. Una vez que se complete la ejecución (20000 iteraciones), se mostrarán los valores de la función de pérdida al igual que en el parte 3 del Problema 1. Informar los tres números tal como se pidió en dicha parte.

3. [6 puntos] Visualizar 200 dígitos muestreados de $p_\theta(\mathbf{x})$.

Problema 3: Implementación del VAE Semi-Supervisado (SSVAE) (33 puntos)

Hasta ahora se han tratado modelos generativos en el contexto no supervisado. Ahora se considera el aprendizaje semi-supervisado en el conjunto de datos MNIST, donde se dispone de un pequeño número de pares etiquetados $\mathbf{x}_\ell = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{100}$ en los datos de entrenamiento y una gran cantidad de datos no etiquetados $\mathbf{x}_u = \left\{\mathbf{x}^{(i)}\right\}_{i=101}^{60000}$. Una etiqueta $y^{(i)}$ para una imagen es simplemente el número que representa la imagen $\mathbf{x}^{(i)}$. El objetivo es construir un clasificador que prediga la etiqueta y dado el ejemplo $\mathbf{x}$. Una posible aproximación es entrenar un clasificador utilizando métodos estándar únicamente con los datos etiquetados. Sin embargo, se busca aprovechar la gran cantidad de datos no etiquetados disponibles para mejorar el rendimiento del clasificador.

Se utiliza un modelo generativo de variables latentes (VAE), donde las etiquetas y están parcialmente observadas y $\mathbf{z}$ son muestras no observadas. La ventaja de un modelo generativo es que permite incorporar de manera natural los datos no etiquetados en la función objetivo de máxima verosimilitud simplemente marginalizando y cuando no está observado. Se implementará el VAE Semi-Supervisado (SSVAE) para esta tarea, siguiendo el proceso generativo especificado a continuación:

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; 0, I), \\ p(y) &= \mathrm{Categorical}(y; \pi) = \frac{1}{10}, \\ p_\theta(\mathbf{x}|\mathbf{z}) &= \prod_{j=1}^{d} \mathrm{Bern}\left(x_j; [f_\theta(\mathbf{z})]_j\right), \end{aligned}$$

donde $\pi = (1/10, \ldots, 1/10)$ es una distribución uniforme fija sobre las 10 posibles etiquetas, y cada secuencia de píxeles $\mathbf{x}$ se modela como un vector de variables aleatorias de Bernoulli, con vector de parámetros parametrizado por la salida de un decodificador basado en una red neuronal $f_\theta(\cdot)$.

Para entrenar un modelo en los conjuntos de datos $\mathbf{X}_\ell$ y $\mathbf{X}_u$, el principio de máxima verosimilitud sugiere encontrar el modelo p_θ que maximice la verosimilitud sobre ambos conjuntos de datos. Suponiendo que las muestras de $\mathbf{X}_\ell$ y $\mathbf{X}_u$ son independientes e idénticamente distribuidas (i.i.d.), esto se traduce en el siguiente objetivo:

$$\max_\theta \sum_{x \in \mathbf{X}_u} \log p_\theta(\mathbf{x}) + \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \log p_\theta(\mathbf{x}, y),$$

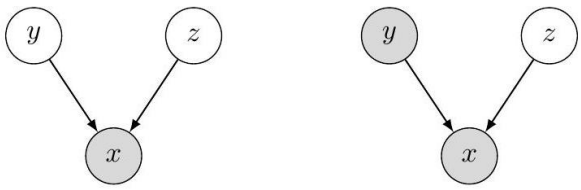

Figure 1: Modelo gráfico para el SSVAE. Los nodos grises representan variables observadas; los nodos sin sombrear representan variables latentes. Izquierda: SSVAE para el caso donde las etiquetas y no están observadas; Derecha: SSVAE donde algunos puntos de datos (x, y) tienen etiquetas observadas.

donde

$$p_\theta(\mathbf{x}) = \sum_{y \in \mathcal{Y}} \int p_\theta(\mathbf{x}, y, \mathbf{z}) d\mathbf{z},$$
$$p_\theta(\mathbf{x}, y) = \int p_\theta(\mathbf{x}, y, \mathbf{z}) d\mathbf{z}.$$

Para superar la intratabilidad de la marginalización exacta de las variables latentes $\mathbf{z}$, se maximiza en su lugar la ELBO:

$$\max_{\theta,\phi} \sum_{x \in \mathbf{X}_u} \text{ELBO}(\mathbf{x}; \theta, \phi) + \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \text{ELBO}(\mathbf{x}, y; \theta, \phi),$$

donde se introduce un modelo de inferencia amortizada $q_\phi(y, \mathbf{z}|\mathbf{x}) = q_\phi(y|\mathbf{x}) q_\phi(\mathbf{z}|\mathbf{x}, y)$. Específicamente:

$$q_\phi(y|\mathbf{x}) = \text{Categorical}\left(y; f_\phi(\mathbf{x})\right),$$
$$q_\phi(\mathbf{z}|\mathbf{x}, y) = \mathcal{N}\left(\mathbf{z}; \mu_\phi(\mathbf{x}, y), \text{diag}\left(\sigma^2_\phi(\mathbf{x}, y)\right)\right),$$

donde los parámetros de la distribución gaussiana se obtienen a través de una pasada hacia adelante del codificador. Se observa que $q_\phi(y|\mathbf{x}) = \text{Categorical}\ (y|f_\phi(\mathbf{x}))$ es en realidad un clasificador basado en una red neuronal multicapa (MLP) que también forma parte del modelo de inferencia, y predice la probabilidad de la etiqueta y dado el dato observado $\mathbf{x}$.

Se utilizará este modelo de inferencia amortizada para construir las ELBOs:

$$\text{ELBO}(\mathbf{x}; \theta, \phi) = \mathbb{E}_{q_\phi(y, \mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x}, y, \mathbf{z})}{q_\phi(y, \mathbf{z}|\mathbf{x})}\right],$$
$$\text{ELBO}(\mathbf{x}, \mathbf{y}; \theta, \phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}, y)}\left[\log \frac{p_\theta(\mathbf{x}, y, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x}, y)}\right].$$

Sin embargo, Kingma et al. (2014)[1], observaron que maximizar la cota inferior de la log-verosimilitud no es suficiente para aprender un buen clasificador. En cambio, propusieron introducir una señal de entrenamiento adicional que entrene directamente al clasificador con los datos etiquetados

$$\max_{\theta,\phi} \sum_{x \in \mathbf{X}_u} \text{ELBO}(\mathbf{x}; \theta, \phi) + \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \text{ELBO}(\mathbf{x}, y; \theta, \phi) + \alpha \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \log q_\phi(y|\mathbf{x}).$$

donde $\alpha \geq 0$ pondera la importancia de la precisión de clasificación. En este problema, se considerará una variante más simple de este objetivo que funciona igual de bien en la práctica,

$$\max_{\theta,\phi} \sum_{x \in \mathbf{X}} \text{ELBO}(\mathbf{x}; \theta, \phi) + \alpha \sum_{\mathbf{x}, y \in \mathbf{X}_\ell} \log q_\phi(y|\mathbf{x}).$$

Vale la pena señalar que la introducción de la pérdida de clasificación tiene una interpretación natural como maximizar la ELBO sujeta a la restricción blanda de que el clasificador $q_\phi(y|\mathbf{x})$ (que es un componente del modelo de inferencia amortizada) logre un buen rendimiento en el conjunto de datos etiquetados. Este enfoque de controlar el modelo generativo mediante la restricción de su modelo de inferencia es, por lo tanto, una forma de regularización de la inferencia amortizada.[2]

[1] Kingma, et al. Semi-Supervised Learning With Deep Generative Models. NeurIPS, 2014.
[2] Shu, et al. Amortized Inference Regularization. NeurIPS, 2018.

1. [4 puntos] Ejecutar `python run_ssvae.py -gw 0`. La bandera gw indica cuánto peso poner en el término ELBO(x) en la función objetivo; escalar el término a cero corresponde a un escenario de aprendizaje supervisado tradicional solo con el pequeño conjunto de datos etiquetados, donde se ignoran los datos no etiquetados. Reportar la precisión de clasificación en el conjunto de prueba una vez que finalice la ejecución (30000 iteraciones).

2. [21 puntos] Implementar la función `negative_elbo_bound` en `ssvae.py`. Tener en cuenta que la función espera como salida la cota inferior de evidencia negativa, así como su descomposición en los siguientes términos,

$$
\begin{aligned}
-\operatorname{ELBO}(\mathbf{x};\theta,\phi) &= -\mathbb{E}_{q_\phi(y|\mathbf{x})} \log \frac{p(y)}{q_\phi(y|\mathbf{x})} - \mathbb{E}_{q_\phi(y|\mathbf{x})}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},y)} \left(\log \frac{p(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x},y)} + \log p_\theta(\mathbf{x}|\mathbf{z},y) \right) \\
&= \underbrace{D_{\mathrm{KL}}\left(q_\phi(y|\mathbf{x}) \| p(y)\right)}_{\mathrm{KL}_y} + \underbrace{\mathbb{E}_{q_\phi(y|\mathbf{x})} D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x},y) \| p(\mathbf{z})\right)}_{\mathrm{KL}_z} + \underbrace{\mathbb{E}_{q_\phi(y,\mathbf{z}|\mathbf{x})}\left[-\log p_\theta(\mathbf{x}|\mathbf{z},y)\right]}_{\text{Reconstruction}}.
\end{aligned}
$$

   Dado que solo hay diez etiquetas, se calcularán las esperanzas con respecto a $q_\phi(y|\mathbf{x})$ exactamente, mientras que se utilizará una sola muestra de Monte Carlo de las variables latentes $\mathbf{z}$ muestreadas de cada $q_\phi(\mathbf{z}|\mathbf{x},y)$ al tratar con el término de reconstrucción. En otras palabras, aproximamos la ELBO negativa con

$$
D_{\mathrm{KL}}\left(q_\phi(y|\mathbf{x}) \| p(y)\right) + \sum_{y\in\mathcal{Y}} q_\phi(y|\mathbf{x}) \left[D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x},y) \| p(\mathbf{z})\right) - \log p_\theta\left(\mathbf{x}|\mathbf{z}^{(y)},y\right) \right],
$$

   donde $\mathbf{z}^{(y)} \sim q_\phi(\mathbf{z}|\mathbf{x},y)$ denota una muestra de la distribución de inferencia cuando se condiciona a una posible pareja $(\mathbf{x},y)$. Las funciones `kl_normal` y `kl_cat` en `utils.py` serán de utilidad.

3. [8 puntos] Ejecutar `python run_ssvae.py`. Esto ejecutará el SSVAE con el término ELBO(x) incluido, y realizará un aprendizaje semi-supervisado. Reportar la precisión de clasificación en el conjunto de prueba una vez que se complete la ejecución.

Instrucciones para la Entrega de Programación

Consultar el archivo README incluido en la tarea de programación.