Devoir 2

IFT714 : Traitement automatique des langues naturelles

Enseignant : Amine Trabelsi

Étudiant : Mamadou Senghor

CIP : senm1912

Université de Sherbrooke

Département d'informatique

Hiver 2026

Date de remise : 31 janvier 2026

Table des matières

1 Question 1 : Régression logistique binaire

On considère un problème de classification binaire où $x = (x_1, \ldots, x_d)^T$, $y \in \{0, 1\}$ et $w = (w_1, \ldots, w_d)^T$. Le modèle est défini par :

$$p(y = 1 \mid x, w) = \sigma(z) = \frac{1}{1 + e^{-z}}, \qquad z = w^T x.$$

1.1 (a) Expression de la perte d'entropie croisée

La vraisemblance pour un exemple $(x^{(i)}, y^{(i)})$ est :

$$p(y^{(i)} \mid x^{(i)}, w) = \sigma(z^{(i)})^{y^{(i)}} (1 - \sigma(z^{(i)}))^{1-y^{(i)}}, \qquad z^{(i)} = w^T x^{(i)}.$$

La perte d'entropie croisée (log-vraisemblance négative) est :

$$L_{CE}^{(i)}(w) = -\log p(y^{(i)} \mid x^{(i)}, w).$$

En remplaçant :

$$L_{CE}^{(i)}(w) = -\log \left[\sigma(z^{(i)})^{y^{(i)}} (1 - \sigma(z^{(i)}))^{1-y^{(i)}} \right].$$

En utilisant $\log(ab) = \log a + \log b$ puis $\log(a^b) = b \log a$:

$$L_{CE}^{(i)}(w) = -\left[y^{(i)} \log \sigma(z^{(i)}) + (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})) \right].$$

Ainsi :

$$\boxed{L_{CE}^{(i)}(w) = -y^{(i)} \log \sigma(z^{(i)}) - (1 - y^{(i)}) \log(1 - \sigma(z^{(i)}))}$$

1.2 (b) Calcul du gradient

On part de :

$$L_{CE}^{(i)}(w) = -y^{(i)} \log \sigma(z^{(i)}) - (1 - y^{(i)}) \log(1 - \sigma(z^{(i)})), \qquad z^{(i)} = w^T x^{(i)}.$$

Par la règle de la chaîne :

$$\frac{\partial L_{CE}^{(i)}}{\partial w_j} = \frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_j}.$$

1) Calcul de $\dfrac{\partial z^{(i)}}{\partial w_j}$ Comme

$$z^{(i)} = \sum_{r=1}^{d} w_r x_r^{(i)},$$

on obtient :

$$\frac{\partial z^{(i)}}{\partial w_j} = x_j^{(i)}.$$

2) Calcul de $\frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}}$ On dérive $L_{CE}^{(i)}$ par rapport à $z^{(i)}$:

$$\frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}} = -y^{(i)} \frac{1}{\sigma(z^{(i)})} \sigma'(z^{(i)}) - (1 - y^{(i)}) \frac{1}{1 - \sigma(z^{(i)})} \frac{\partial}{\partial z^{(i)}} \Big(1 - \sigma(z^{(i)})\Big).$$

Or $\frac{\partial}{\partial z^{(i)}}(1 - \sigma(z^{(i)})) = -\sigma'(z^{(i)})$, donc :

$$\frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}} = -y^{(i)} \frac{\sigma'(z^{(i)})}{\sigma(z^{(i)})} + (1 - y^{(i)}) \frac{\sigma'(z^{(i)})}{1 - \sigma(z^{(i)})}.$$

On factorise $\sigma'(z^{(i)})$:

$$\frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}} = \sigma'(z^{(i)}) \left(-\frac{y^{(i)}}{\sigma(z^{(i)})} + \frac{1 - y^{(i)}}{1 - \sigma(z^{(i)})} \right).$$

Or la dérivée de la sigmoïde est :

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)).$$

En remplaçant :

$$\frac{\partial L_{CE}^{(i)}}{\partial z^{(i)}} = -y^{(i)}(1 - \sigma(z^{(i)})) + (1 - y^{(i)})\sigma(z^{(i)}) = \sigma(z^{(i)}) - y^{(i)}.$$

3) Résultat final Ainsi :

$$\boxed{\frac{\partial L_{CE}^{(i)}}{\partial w_j} = (\sigma(z^{(i)}) - y^{(i)}) x_j^{(i)}}$$

1.3 (c) Gradient pour l'ensemble d'entraînement

On considère un ensemble d'entraînement

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}.$$

La perte totale est la somme des pertes individuelles :

$$L_{CE}^{D}(w) = \sum_{i=1}^{m} L_{CE}^{(i)}(w).$$

En utilisant le résultat de la question (b), on obtient :

$$\frac{\partial L_{CE}^{D}}{\partial w_j} = \sum_{i=1}^{m} (\sigma(z^{(i)}) - y^{(i)}) x_j^{(i)},$$

où

$$z^{(i)} = w^T x^{(i)}, \qquad \sigma(z^{(i)}) = p(y^{(i)} = 1 \mid x^{(i)}, w).$$

Descente de gradient batch La mise à jour des poids en utilisant tous les exemples à chaque itération est :

$$w_j \leftarrow w_j - \eta \sum_{i=1}^{m} (\sigma(z^{(i)}) - y^{(i)}) x_j^{(i)}.$$

Descente de gradient stochastique (SGD) La mise à jour des poids après chaque exemple $(x^{(i)}, y^{(i)})$ est :

$$w_j \leftarrow w_j - \eta(\sigma(z^{(i)}) - y^{(i)})x_j^{(i)}.$$

1.4 (d) Interprétation

Le terme $(\sigma(z^{(i)}) - y^{(i)})$ représente l'erreur de prédiction du modèle pour l'exemple $(x^{(i)}, y^{(i)})$.

— Si $y^{(i)} = 1$ et $\sigma(z^{(i)}) < 1$, alors $(\sigma(z^{(i)}) - y^{(i)}) < 0$: la mise à jour augmente w_j lorsque $x_j^{(i)} > 0$, ce qui accroît la probabilité prédite pour la classe positive.

— Si $y^{(i)} = 0$ et $\sigma(z^{(i)}) > 0$, alors $(\sigma(z^{(i)}) - y^{(i)}) > 0$: la mise à jour diminue w_j lorsque $x_j^{(i)} > 0$, ce qui réduit la probabilité prédite pour la classe positive.

La règle de mise à jour :

$$w_j \leftarrow w_j - \eta(\sigma(z^{(i)}) - y^{(i)})x_j^{(i)}$$

ajuste donc les poids proportionnellement :

— à l'erreur de prédiction du modèle,

— à l'importance de la caractéristique $x_j^{(i)}$,

— au taux d'apprentissage η.

Ainsi, l'algorithme modifie les poids de façon à augmenter la probabilité de la classe correcte et à diminuer celle de la classe incorrecte.

2 Question 2 : Classification multiclasse (Multinomiale et Softmax)

2.1 (a) Écriture de $p(y)$ avec des indicatrices

On considère une variable aléatoire discrète $y \in \{1, \ldots, K\}$ telle que

$$\phi_k = p(y = k), \quad k = 1, \ldots, K, \qquad \sum_{k=1}^{K} \phi_k = 1.$$

On introduit la fonction indicatrice :

$$\mathbf{1}_{\{y=k\}} = \begin{cases} 1 & \text{si } y = k, \\ 0 & \text{sinon.} \end{cases}$$

Étape 1 : vérification de la forme produit dans le cas $y = r$ Supposons que $y = r$ pour un certain $r \in \{1, \ldots, K\}$. Alors :

$$\mathbf{1}_{\{y=r\}} = 1 \quad \text{et} \quad \mathbf{1}_{\{y=k\}} = 0 \ \ \forall k \neq r.$$

Considérons le produit :

$$\prod_{k=1}^{K} \phi_k^{\mathbf{1}_{\{y=k\}}}.$$

En remplaçant les exposants par leurs valeurs :

$$\prod_{k=1}^{K} \phi_k^{\mathbf{1}_{\{y=k\}}} = \phi_r^1 \prod_{k \neq r} \phi_k^0.$$

Étape 2 : utilisation de la propriété $a^0 = 1$ Comme $\phi_k^0 = 1$ pour tout $k \neq r$, on obtient :

$$\phi_r^1 \prod_{k \neq r} \phi_k^0 = \phi_r \cdot \prod_{k \neq r} 1 = \phi_r.$$

Étape 3 : conclusion Or, par définition, $\phi_r = p(y = r)$. Ainsi, pour tout $y \in \{1, \ldots, K\}$:

$$p(y) = \prod_{k=1}^{K} \phi_k^{\mathbf{1}_{\{y=k\}}} = \phi_1^{\mathbf{1}_{\{y=1\}}} \phi_2^{\mathbf{1}_{\{y=2\}}} \cdots \phi_K^{\mathbf{1}_{\{y=K\}}}.$$

$$\boxed{p(y) = \prod_{k=1}^{K} \phi_k^{\mathbf{1}_{\{y=k\}}}}$$

2.2 (b) Dérivation de la forme Softmax à partir de $\eta_k = \log \frac{\phi_k}{\phi_K}$

On définit, pour $k = 1, \ldots, K$:

$$\eta_k = \log \frac{\phi_k}{\phi_K}.$$

Pour $k = K$, on a directement $\eta_K = 0$.

Étape 1 : isoler ϕ_k en fonction de ϕ_K En exponentiant des deux côtés :

$$e^{\eta_k} = \frac{\phi_k}{\phi_K} \quad \Longrightarrow \quad \phi_k = \phi_K \, e^{\eta_k}, \qquad k = 1, \ldots, K.$$

Étape 2 : utiliser la contrainte de normalisation Comme les probabilités somment à 1 :

$$\sum_{k=1}^{K} \phi_k = 1,$$

en remplaçant ϕ_k par $\phi_K e^{\eta_k}$:

$$\sum_{k=1}^{K} \phi_K e^{\eta_k} = 1 \quad \Longrightarrow \quad \phi_K \sum_{k=1}^{K} e^{\eta_k} = 1.$$

Étape 3 : isoler ϕ_K

$$\phi_K = \frac{1}{\sum_{l=1}^{K} e^{\eta_l}}.$$

Étape 4 : expression de ϕ_k En remplaçant ϕ_K dans $\phi_k = \phi_K e^{\eta_k}$:

$$\phi_k = \frac{e^{\eta_k}}{\sum_{l=1}^{K} e^{\eta_l}}, \qquad k = 1, \ldots, K.$$

$$\boxed{\phi_k = \frac{e^{\eta_k}}{\sum_{l=1}^{K} e^{\eta_l}}}$$

Conclusion (Softmax) Le vecteur $(\phi_1, \ldots, \phi_K)$ correspond exactement à la fonction Softmax appliquée au vecteur $(\eta_1, \ldots, \eta_K)$:

$$\text{Softmax}(\eta)_k = \frac{e^{\eta_k}}{\sum_{l=1}^{K} e^{\eta_l}}.$$

2.3 (c) Expression de $p(y \mid x, w_1, \ldots, w_K)$

On considère un vecteur d'entrée $x = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$ à classer dans l'une des K classes. Pour chaque classe k, on définit un score linéaire :

$$\eta_k(x) = w_k^T x, \quad \text{où} \quad w_k = (w_{k1}, \ldots, w_{kd})^T \in \mathbb{R}^d.$$

Étape 1 : partir de la forme Softmax obtenue en (b) D'après (b), pour des scores $(\eta_1, \ldots, \eta_K)$, on a :

$$p(y = k) = \phi_k = \frac{e^{\eta_k}}{\sum_{l=1}^{K} e^{\eta_l}}.$$

Étape 2 : rendre la distribution conditionnelle à x Ici, les scores dépendent de x, donc :

$$p(y = k \mid x, w_1, \ldots, w_K) = \frac{e^{\eta_k(x)}}{\sum_{l=1}^{K} e^{\eta_l(x)}}.$$

Étape 3 : remplacer $\eta_k(x)$ par $w_k^T x$ En utilisant $\eta_k(x) = w_k^T x$:

$$p(y = k \mid x, w_1, \ldots, w_K) = \frac{e^{w_k^T x}}{\sum_{l=1}^{K} e^{w_l^T x}}.$$

$$\boxed{p(y = k \mid x, w_1, \ldots, w_K) = \frac{e^{w_k^T x}}{\sum_{l=1}^{K} e^{w_l^T x}}, \qquad k = 1, \ldots, K.}$$

Forme vectorielle En posant $s(x) = (s_1(x), \ldots, s_K(x))$ avec $s_k(x) = p(y = k \mid x, w_1, \ldots, w_K)$, on obtient :

$$s(x) = \text{Softmax}\Big([w_1^T x, \ldots, w_K^T x]^T\Big).$$

2.4 (d) Perte d'entropie croisée pour l'ensemble d'entraînement

On considère un ensemble d'entraînement :

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m},$$

où chaque $y^{(i)} \in \{1, \ldots, K\}$.

D'après la question (c), la probabilité conditionnelle est :

$$p(y = k \mid x^{(i)}, w_1, \ldots, w_K) = \frac{e^{w_k^T x^{(i)}}}{\sum_{l=1}^{K} e^{w_l^T x^{(i)}}}.$$

Étape 1 : vraisemblance pour un exemple En utilisant la représentation par indicatrices, on peut écrire :

$$p(y^{(i)} \mid x^{(i)}) = \prod_{k=1}^{K} p(y = k \mid x^{(i)})^{\mathbf{1}_{\{y^{(i)}=k\}}}.$$

Étape 2 : perte (log-vraisemblance négative)

$$L^{(i)} = -\log p(y^{(i)} \mid x^{(i)}).$$

Étape 3 : propriété du logarithme En utilisant $\log \prod = \sum \log$:

$$L^{(i)} = -\sum_{k=1}^{K} \mathbf{1}_{\{y^{(i)}=k\}} \log p(y = k \mid x^{(i)}).$$

Étape 4 : substitution de la forme Softmax

$$L^{(i)} = -\sum_{k=1}^{K} \mathbf{1}_{\{y^{(i)}=k\}} \log \left(\frac{e^{w_k^T x^{(i)}}}{\sum_{l=1}^{K} e^{w_l^T x^{(i)}}} \right).$$

Étape 5 : perte totale La perte totale sur l'ensemble D est :

$$L(w_1, \ldots, w_K) = \sum_{i=1}^{m} L^{(i)}.$$

Ainsi :

$$\boxed{L(w_1, \ldots, w_K) = -\sum_{i=1}^{m} \sum_{k=1}^{K} \mathbf{1}_{\{y^{(i)}=k\}} \log p(y = k \mid x^{(i)}, w_1, \ldots, w_K)}$$

Forme compacte Comme un seul terme de l'indicatrice vaut 1 pour chaque i, on peut aussi écrire :

$$\boxed{L(w_1, \ldots, w_K) = -\sum_{i=1}^{m} \log p(y^{(i)} \mid x^{(i)}, w_1, \ldots, w_K).}$$

3 Question 3 : Fonction Softmax

3.1 (a) Invariance de Softmax par translation

On considère la fonction Softmax définie pour $z \in \mathbb{R}^K$ par :

$$\text{Softmax}(z) = s = (s_1, \ldots, s_K), \qquad s_k = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}.$$

Soit $a \in \mathbb{R}$ une constante et $a\mathbf{1} = (a, \ldots, a)^T \in \mathbb{R}^K$. On veut montrer que :

$$\text{Softmax}(z + a\mathbf{1}) = \text{Softmax}(z).$$

Étape 1 : écrire la $k^{\text{ème}}$ composante de Softmax$(z + a\mathbf{1})$ On a $(z + a\mathbf{1})_k = z_k + a$, donc :

$$\Big(\text{Softmax}(z + a\mathbf{1})\Big)_k = \frac{e^{z_k+a}}{\sum_{l=1}^{K} e^{z_l+a}}.$$

Étape 2 : factoriser e^a au numérateur et au dénominateur Comme $e^{z_k+a} = e^{z_k}e^a$ et $e^{z_l+a} = e^{z_l}e^a$, on obtient :

$$\Big(\text{Softmax}(z + a\mathbf{1})\Big)_k = \frac{e^{z_k}e^a}{\sum_{l=1}^{K} e^{z_l}e^a}.$$

Étape 3 : simplifier en annulant le facteur commun e^a On factorise e^a dans le dénominateur :

$$\sum_{l=1}^{K} e^{z_l}e^a = e^a \sum_{l=1}^{K} e^{z_l}.$$

Donc :

$$\Big(\text{Softmax}(z + a\mathbf{1})\Big)_k = \frac{e^{z_k}e^a}{e^a \sum_{l=1}^{K} e^{z_l}} = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}} = s_k.$$

Conclusion Comme ceci est vrai pour tout $k = 1, \ldots, K$, on a :

$$\boxed{\text{Softmax}(z + a\mathbf{1}) = \text{Softmax}(z).}$$

3.2 (b) Choix pratique de $a = -\max_i(z_i)$

D'après la question (a), la fonction Softmax est invariante par translation :

$$\text{Softmax}(z + a) = \text{Softmax}(z)$$

pour toute constante $a \in \mathbb{R}$.

En pratique, on choisit :

$$a = -\max_i(z_i).$$

Étape 1 : analyse du problème numérique La fonction Softmax contient des exponentielles :

$$s_k = \frac{e^{z_k}}{\sum_{l=1}^{K} e^{z_l}}.$$

Si certaines composantes z_k sont très grandes ($z_k \gg 0$), alors

$$e^{z_k}$$

peut dépasser la capacité de représentation des nombres en machine (phénomène d'*overflow*). Inversement, si z_k est très négatif, alors e^{z_k} peut devenir numériquement nul (*underflow*).

Étape 2 : application de la translation On remplace z par $z+a$ avec $a = -\max_i(z_i)$:

$$(z + a)_k = z_k - \max_i(z_i).$$

Ainsi, la plus grande composante devient :

$$\max_k(z_k - \max_i(z_i)) = 0,$$

et pour tout k :

$$z_k - \max_i(z_i) \leq 0.$$

Étape 3 : effet sur les exponentielles Après translation :

$$e^{z_k - \max_i(z_i)} \leq e^0 = 1.$$

Toutes les exponentielles sont donc bornées entre 0 et 1, ce qui permet d'éviter :
- les débordements numériques (*overflow*),
- les pertes de précision dues à des nombres trop grands ou trop petits (*underflow*).

Étape 4 : invariance du résultat Comme montré en (a), cette translation ne modifie pas la valeur finale de Softmax :

$$\text{Softmax}(z - \max_i(z)) = \text{Softmax}(z).$$

Conclusion On choisit en pratique :

$$\boxed{a = -\max_i(z_i)}$$

afin d'assurer la stabilité numérique du calcul de Softmax sans modifier les probabilités produites par le modèle.

3.3 (c) Dérivée $\dfrac{\partial s_k}{\partial z_l}$

On rappelle la définition de la fonction Softmax :

$$s_k = \frac{e^{z_k}}{\sum_{r=1}^{K} e^{z_r}}, \qquad k = 1, \ldots, K.$$

Posons

$$S = \sum_{r=1}^{K} e^{z_r},$$

alors

$$s_k = \frac{e^{z_k}}{S}.$$

On cherche la dérivée partielle $\frac{\partial s_k}{\partial z_l}$ pour $k, l \in \{1, \ldots, K\}$.

Étape 1 : règle du quotient Comme $s_k = \frac{u}{v}$ avec $u = e^{z_k}$ et $v = S$, on a :

$$\frac{\partial s_k}{\partial z_l} = \frac{\frac{\partial u}{\partial z_l} v - u \frac{\partial v}{\partial z_l}}{v^2}.$$

Donc :

$$\frac{\partial s_k}{\partial z_l} = \frac{\frac{\partial}{\partial z_l}(e^{z_k})\, S - e^{z_k} \frac{\partial S}{\partial z_l}}{S^2}.$$

Étape 2 : dérivée du numérateur On obtient :

$$\frac{\partial}{\partial z_l}(e^{z_k}) = e^{z_k} \delta_{kl},$$

où δ_{kl} est le delta de Kronecker.

Étape 3 : dérivée du dénominateur Comme

$$S = \sum_{r=1}^{K} e^{z_r},$$

on a :

$$\frac{\partial S}{\partial z_l} = e^{z_l}.$$

Étape 4 : substitution On remplace dans l'expression précédente :

$$\frac{\partial s_k}{\partial z_l} = \frac{e^{z_k}\delta_{kl}S - e^{z_k}e^{z_l}}{S^2}.$$

On factorise e^{z_k} :

$$\frac{\partial s_k}{\partial z_l} = \frac{e^{z_k}}{S^2}\left(\delta_{kl}S - e^{z_l}\right).$$

Étape 5 : réécriture en fonction de s_k et s_l On remarque que :

$$\frac{e^{z_k}}{S} = s_k, \qquad \frac{e^{z_l}}{S} = s_l.$$

Donc :

$$\frac{\partial s_k}{\partial z_l} = s_k(\delta_{kl} - s_l).$$

Forme finale (deux cas)

$$\boxed{\frac{\partial s_k}{\partial z_l} = \begin{cases} s_k(1 - s_k) & \text{si } k = l, \\ -s_k s_l & \text{si } k \neq l. \end{cases}}$$

4 Question 4 : Programmation

4.1 Implémentation

Nous avons implémenté un classifieur de régression logistique binaire pour la classification de sentiments (positif = 1, négatif = 0) appliquée à des critiques de films. Les phrases sont représentées à l'aide de **features unigrammes** (sac-à-mots, Bag-of-Words). L'optimisation des paramètres est réalisée par **descente de gradient stochastique (SGD)** en minimisant la perte d'entropie croisée.

4.1.1 Modèle

Pour un exemple x représenté par un vecteur de caractéristiques, le modèle prédit :

$$p(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-z}}, \qquad z = w^T x + b,$$

où w est le vecteur de poids associé aux features et b est le biais.

4.1.2 Entraînement (SGD avec régularisation L2)

Pour chaque exemple $(x^{(i)}, y^{(i)})$, on calcule :

$$p^{(i)} = \sigma(w^T x^{(i)} + b).$$

La mise à jour des paramètres par descente de gradient stochastique est donnée par :

$$w \leftarrow w - \eta\Big((p^{(i)} - y^{(i)})x^{(i)} + \lambda w\Big), \qquad b \leftarrow b - \eta(p^{(i)} - y^{(i)}),$$

où η est le taux d'apprentissage et λ le coefficient de régularisation L2.

Une implémentation numériquement stable de la fonction sigmoïde est utilisée afin d'éviter les problèmes de débordement numérique (*overflow*).

4.1.3 Composantes principales

- **Classe `LogisticRegressionClassifier`** :
  - `__init__` : initialise les poids et le biais à zéro,
  - `train` : entraîne le modèle par SGD avec régularisation L2,
  - `predict` : retourne 1 si $p(y = 1 \mid x) \geq 0.5$, sinon 0.
- **Fonction `train_lr`** : instancie le modèle et lance l'entraînement,
- **Features** : unigrammes extraits par `UnigramFeatureExtractor`.

4.2 Hyperparamètres du modèle final

Hyperparamètre	**Valeur**
Nombre d'époques (`num_iters`)	100
Taux d'apprentissage (η)	0.01
Régularisation L2 (λ)	0.1
Type de features	Unigrammes (Bag-of-Words)
Optimisation	Descente de Gradient Stochastique (SGD)

Table 1 – Hyperparamètres du modèle final

4.3 Résultats du modèle final

Ensemble	**Précision**	**Exemples corrects**
Entraînement	98.82%	6838 / 6920
Développement	80.16%	699 / 872

Table 2 – Résultats du modèle final de régression logistique

4.4 Expérimentation : influence de la régularisation

Nous avons étudié l'impact du paramètre de régularisation λ sur les performances du modèle en maintenant les autres hyperparamètres constants ($\eta = 0.01$, `num_iters`=100).

λ	**Train Acc.**	**Dev Acc.**	**Écart**
0.0	98.38%	76.83%	21.55%
0.001	98.42%	76.95%	21.47%
0.01	98.47%	77.98%	20.49%
0.05	98.79%	79.70%	19.09%
0.1	**98.82%**	**80.16%**	**18.66%**

TABLE 3 – Impact de la régularisation L2 sur les performances ($\eta = 0.01$, 100 époques)

4.5 Analyse

4.5.1 Surapprentissage et généralisation

Un écart significatif est observé entre les performances sur l'ensemble d'entraînement et sur l'ensemble de développement, indiquant un phénomène de **surapprentissage**. Ce comportement est attendu en présence de features unigrammes de très grande dimension. Néanmoins, la précision obtenue sur l'ensemble de développement (80.16%) constitue un **baseline raisonnable** pour un modèle linéaire simple.

4.5.2 Impact de la régularisation

La régularisation L2 permet de limiter le surapprentissage en pénalisant les poids de grande amplitude. Les résultats montrent une amélioration progressive lorsque λ augmente :

— $\lambda = 0.0$: précision dev = 76.83%, écart = 21.55%,
— $\lambda = 0.1$: précision dev = 80.16%, écart = 18.66%.

Ainsi, la régularisation améliore la précision sur l'ensemble de développement de **+3.33%** et réduit l'écart train/dev de 2.89 points de pourcentage, indiquant une meilleure capacité de généralisation.

4.5.3 Choix des hyperparamètres

Les hyperparamètres finaux ont été sélectionnés empiriquement :

— $\eta = 0.01$ pour assurer une convergence plus stable,
— 100 époques pour garantir la convergence avec ce taux d'apprentissage,
— $\lambda = 0.1$ comme compromis optimal biais–variance sur l'ensemble de développement.

4.6 Conclusion

Le classifieur de régression logistique entraîné par descente de gradient stochastique avec régularisation L2 atteint une précision de **80.16%** sur l'ensemble de développement. Ces résultats constituent une base solide pour la classification automatique de sentiments à l'aide de features unigrammes.