# *Aide TP4 – Programmation

→ Jupyter Notebook : TP4.ipynb

→ Dans le terminal, vous pouvez rouler " jupyter notebook TP4.ipynb "

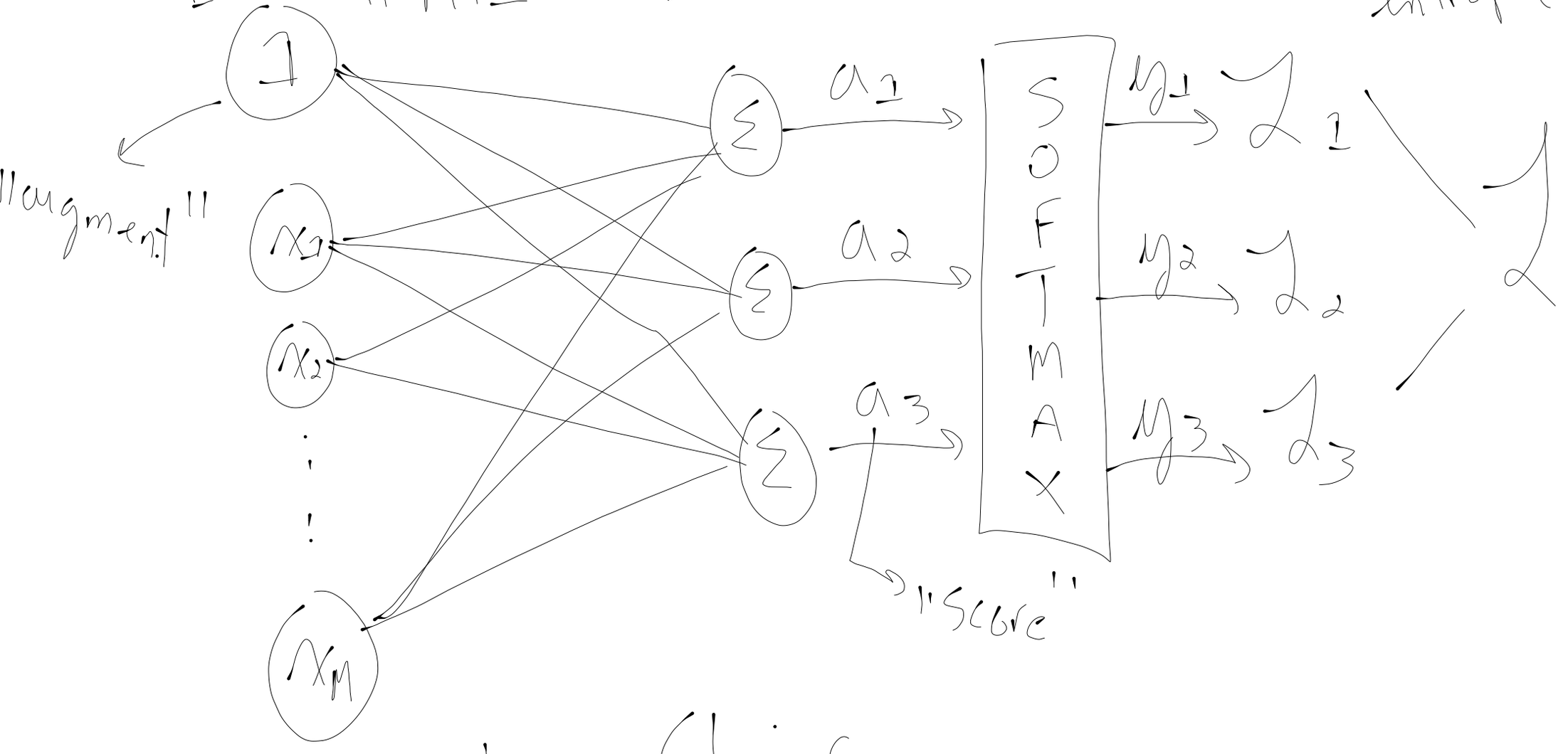
→ Pour rouler une cellule à la fois, vous pouvez utiliser : SHIFT + ENTER

# * PARTIE 1 : Classifieur linéaire à une couche

Entrée : # M+1

Sortie : # num_classes

Cross-entropie




Class : LinearClassifier

→ TODO 1 : global_accuracy_and_cross_entropy_loss

* Méthode haut-niveau pour calculer la loss et l'accuracy pour plusieurs points

→ Utilisation de "augment" pour le biais

→ Appel de la méthode "cross_entropy_loss"

→ Appel de la méthode "predict" pour prédire la bonne classe.

=> TODO2 : cross_entropy_loss [voir kit-survie.pdf]

* Méthode pour calculer la loss et le gradient des paramètres par rapport à la loss pour un point

(1) Forward pass

(2) + Softmax

## ② Loss

$$\mathcal{L}_{total} = \underbrace{\mathcal{L}_{pred}}_{\text{(prédiction)}} + \underbrace{\mathcal{L}_{reg}}_{\text{(régularisation)}}$$

$$= -\ln(y) + \frac{\lambda}{2}\|\vec{w}\|^2$$

## (3) $\nabla_{\vec{w}} J_{total}$

$$\rightarrow \frac{\partial J_{total}}{\partial w} = \frac{\partial J_{pred}}{\partial w} + \frac{\partial J_{reg}}{\partial w}$$

$$= \frac{\partial J_{pred}}{\partial a} \cdot \frac{\partial a}{\partial w} + \frac{\partial J_{reg}}{\partial w}$$

$\rightarrow$ Nous avons déterminé que

$$\frac{\partial J_{pred}}{\partial a} = (\vec{y} - \vec{t})$$

$$\rightarrow \frac{\partial a}{\partial w} \rightarrow a = \vec{w}^T \vec{x}$$

$$\boxed{\frac{\partial u}{\partial w} = \vec{x}}$$

$$\rightarrow \frac{\partial J_{reg}}{\partial w} = \lambda \vec{w}$$

* Ainsi, nous pouvons écrire que

$$\nabla_{\vec{w}} J_{total} = \left( \vec{y} - \vec{t} \,\middle|\, \vec{x} \right) + \lambda \vec{w}$$

→ Descente de gradient

↳ $\vec{w} \leftarrow \vec{w} - \eta \nabla_{\vec{w}} J_{total}$

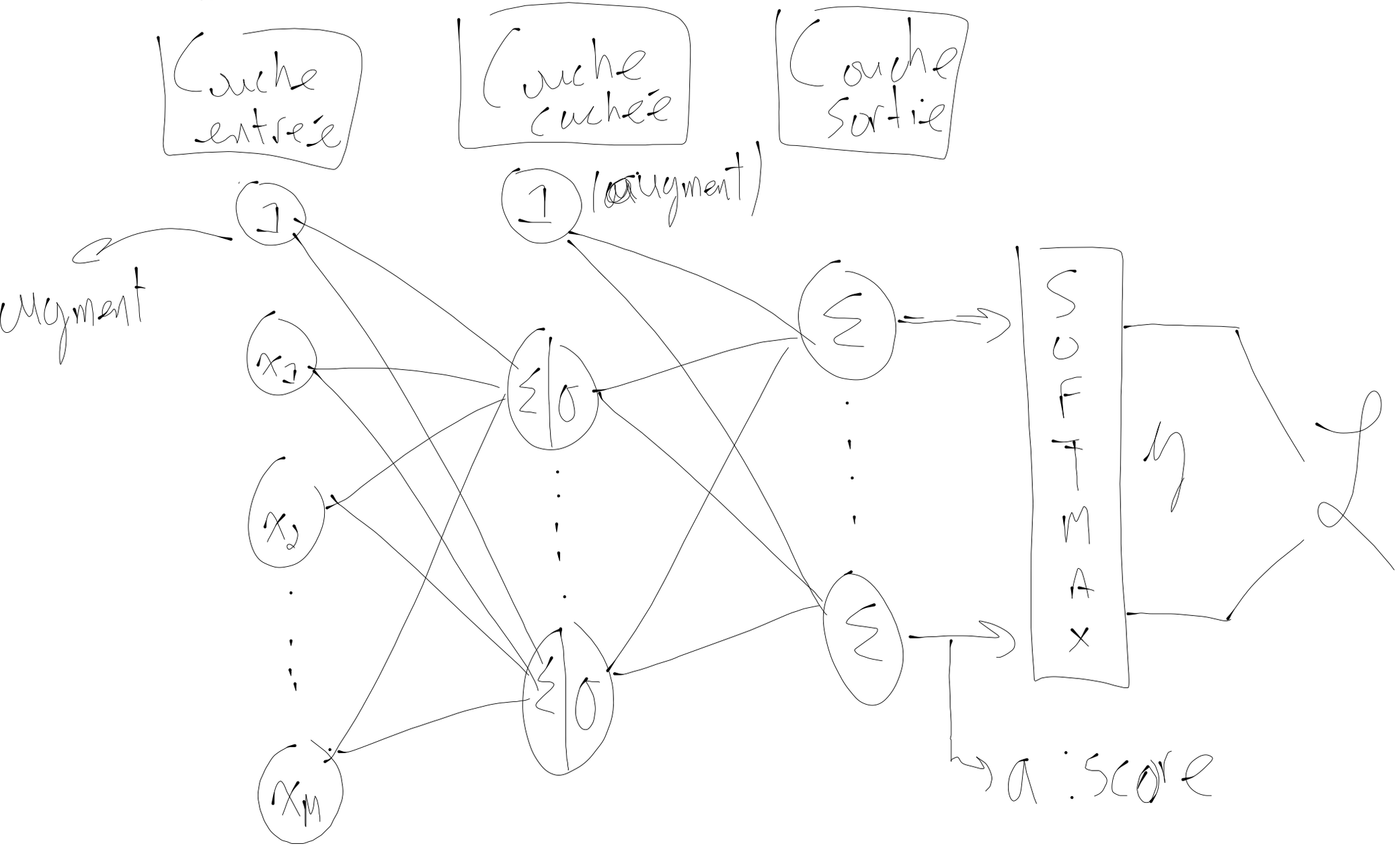
⇒ TODO3 : predict

→ Le score maximal peut directement être obtenu avec "a" ( max a <=> max y)

→ Utisation de "augment" pour le biais

→ Calcul du score maximal

# * PARTIE 2 : Classifieur non-linéaire à deux couches



* Class TwoLayerClassifier
  → Class TwoLayerNet
    → Class DenseLayer

---

=> TODO 1 : Forward pass → DenseLayer

* Forward pass → pour une couche
  → Forward TwoLayerNet est déjà implementé

→ Utilisation de "augment" pour le biais

→ Score : dot product

→ Activation : Sigmoïde ou ReLU
(voir vidéos)

(Notez que la backpropagation est déjà implementée)

## ⇒ TODO2 : cross-entropy-loss (TwoLayerNet)

→ Très similaire à la partie 1

→ Output 1 : loss totale

$$L_{total} = L_{pred} + L_{reg}$$

→ Output 2 : gradient de la loss "pred"

seulement

↳ $\frac{\partial L_{pred}}{\partial a}$

* $\partial L_{reg}$ / $\partial W$ est déjà implementée dans DenseLayer.backward

=> TODO3 : momentum_update

* Veuillez noter : @property decorator facilite la tâche. Vous devez simplement mettre à jour V et W

* Voir le lien dans les commentaires du code pour faire un "momentum update".

=> TODO 4 : global_accuracy_and_cross_entropy_loss (Class TwoLayerClassifier)

→ Très similaire à la partie 1 & mais avec une "forward pass" pour tout le reseau (TwoLayerNet).

## => TODO 5 : predict (TwoLayerClassifier)

* Similaire à la partie 4, mais avec une forward pass pour tout le réseau et la possibilité de faire des prédictions pour plusieurs points à la fois

NOTE : En guise d'apprentissage des bonnes pratiques, portez attention aux "Sanity checks" dans les Jupyter Notebooks

→ voir les vidéos aussi.

# * Mot Final : Les concepts importants du "Deep Learning"

- DeepLearning
- Neurone
- Couche
- Loss
- Couche dense
- Couche à convolution
- Forward pass
- Backward pass
- Epoch
- Optimisation
- Maxpooling
- Softmax
- Dropout
- GPU
- Over-Fitting
- Unité - ...

"End-to-End" training (IFT780)