Apprentissage vue comme la minimisation d'une perte

- Le problème de l'apprentissage peut être formulé comme un problème d'optimisation
  - pour chaque exemple d'entraînement, on souhaite minimiser une certaine distance $Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))$ entre la cible y_t et la prédiction $h_{\mathbf{w}}(\mathbf{x}_t)$
  - on appelle cette distance une **perte**

- Dans le cas du perceptron:

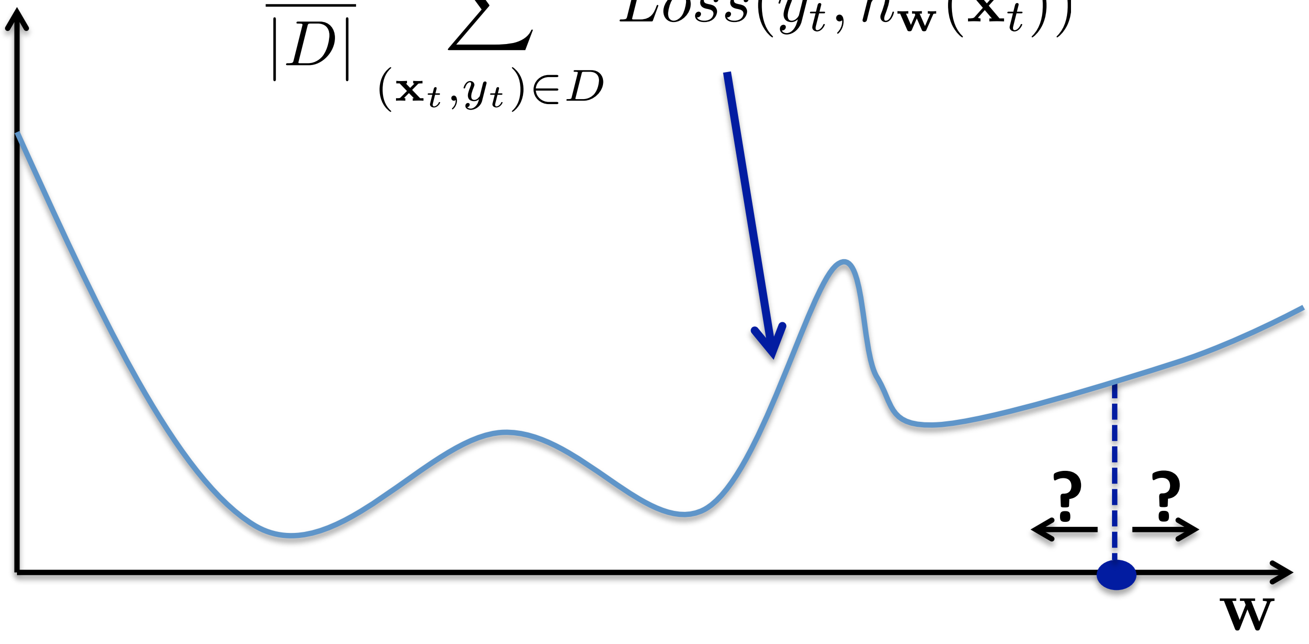
$$Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) = -(y_t - h_{\mathbf{w}}(\mathbf{x}_t))\mathbf{w} \cdot \mathbf{x}_t$$

  - si la prédiction est bonne, le coût est 0
  - si la prédiction est mauvaise, le perte est la distance entre $\mathbf{w} \cdot \mathbf{x}_t$ et le seuil à franchir pour que la prédiction soit bonne

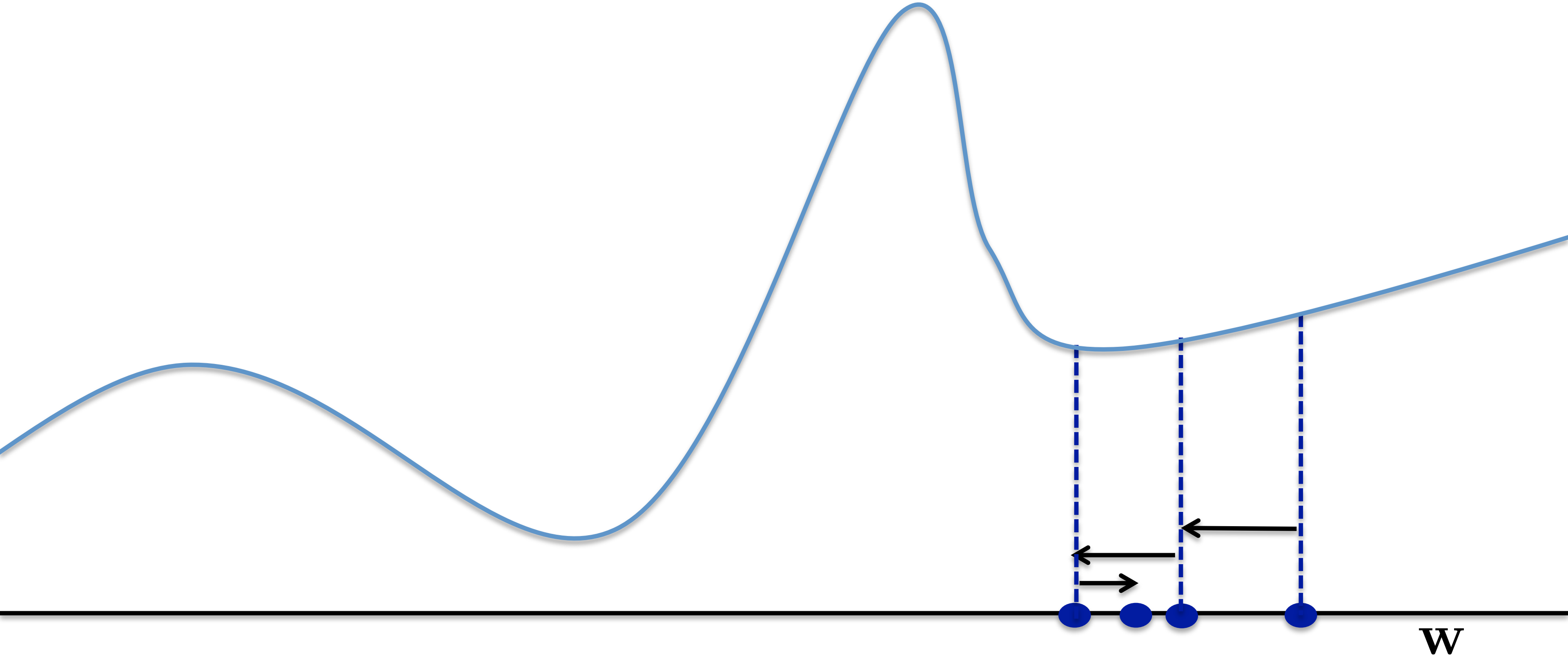
Hugo Larochelle et Froduald Kabanza

Recherche locale pour la minimisation d'une perte

$$\frac{1}{|D|} \sum_{(\mathbf{x}_t, y_t) \in D} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))$$

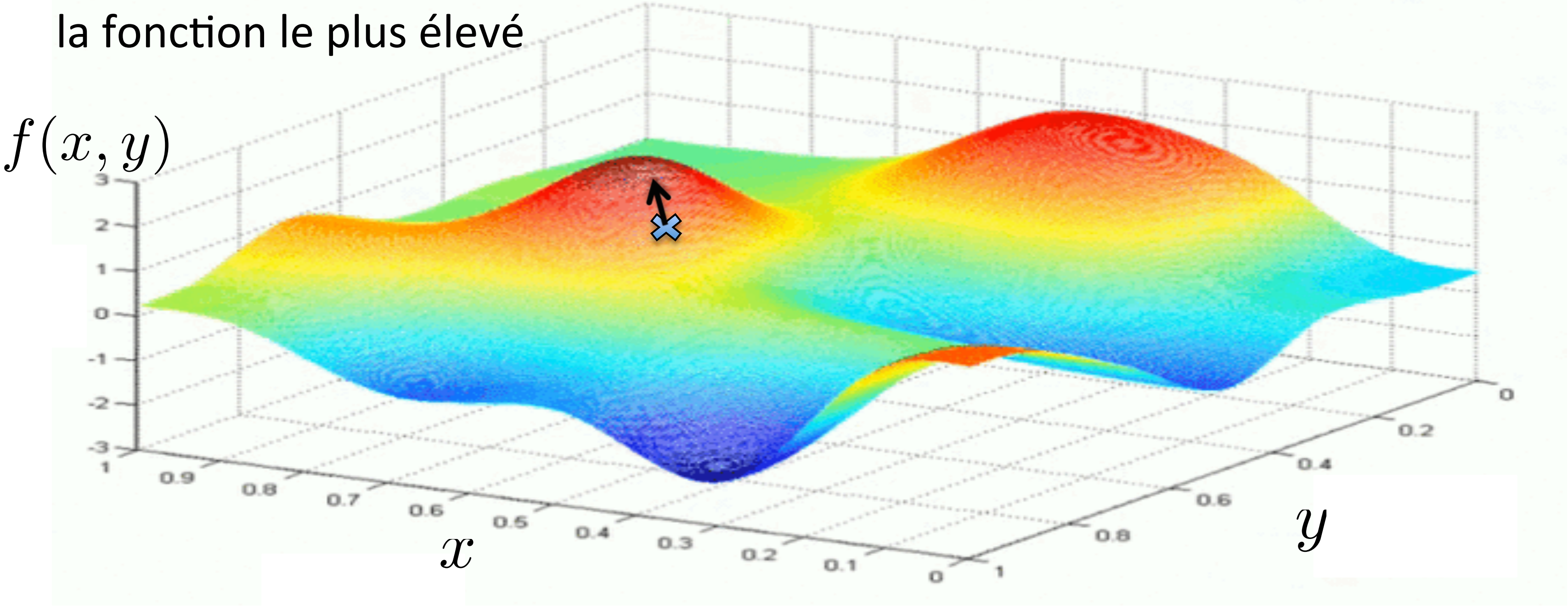


Algorithme de descente de gradient



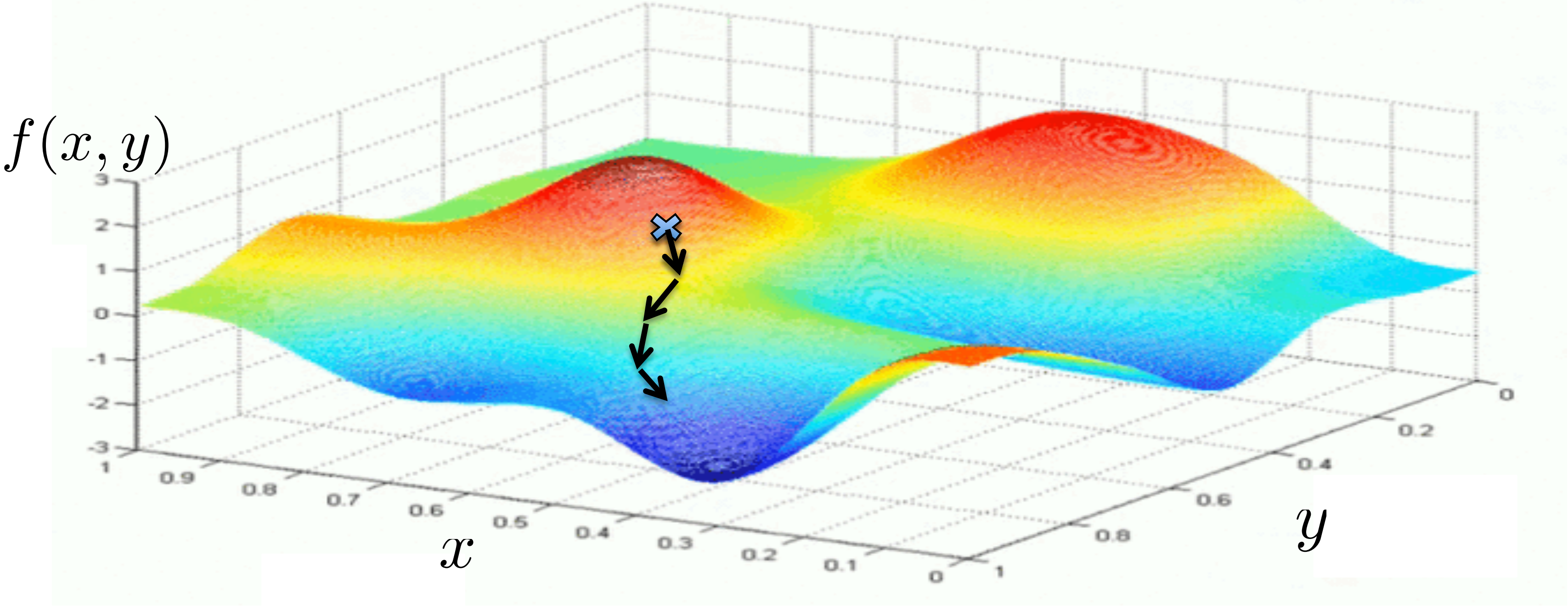
Descente de gradient

- Le gradient donne la direction (vecteur) ayant le taux d'accroissement de la fonction le plus élevé



Descente de gradient

- La direction opposée au gradient nous donne la direction à suivre



Apprentissage vue comme la minimisation d'une perte

- En apprentissage automatique, on souhaite optimiser:

$$\frac{1}{|D|} \sum_{(\mathbf{x}_t, y_t) \in D} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t))$$

- Le gradient par rapport à la perte moyenne contient les dérivées partielles:

$$\frac{1}{|D|} \sum_{(\mathbf{x}_t, y_t) \in D} \frac{\partial}{\partial w_i} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i$$

- Devrait calculer la moyenne des dérivées sur tous les exemples d'entraînement avant de faire une mise à jour des paramètres!

Descente de gradient stochastique

- **Descente de gradient stochastique**: mettre à jour les paramètres à partir du gradient (c.-à-d. des dérivées partielles) d'un seul exemple, choisi aléatoirement:

> - Initialiser $\mathbf{w}$ aléatoirement
> - Pour *T* itérations
>   - Pour chaque exemple d'entraînement $(\mathbf{x}_t, y_t)$
>   - $w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i$

- Cette procédure est plus efficace lorsque l'ensemble d'entraînement est grand
    - on fait $|D|$ mises à jour des paramètres après chaque parcours de l'ensemble d'entraînement, plutôt qu'une seule mise à jour avec la descente de gradient normale

Retour sur le perceptron

- On utilise le gradient (dérivée partielle) pour déterminer une direction de mise à jour des paramètres:

$$\frac{\partial}{\partial w_i} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) = \frac{\partial}{\partial w_i} - (y_t - h_{\mathbf{w}}(\mathbf{x}_t))\mathbf{w} \cdot \mathbf{x}_t \cong -(y_t - h_{\mathbf{w}}(\mathbf{x}_t))x_{t,i}$$

- Par définition, le gradient donne la direction (locale) d'augmentation la plus grande de la perte
  - pour mettre à jour les paramètres, on va dans la direction opposée à ce gradient:

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(y_t, h_{\mathbf{w}}(\mathbf{x}_t)) \quad \forall i$$

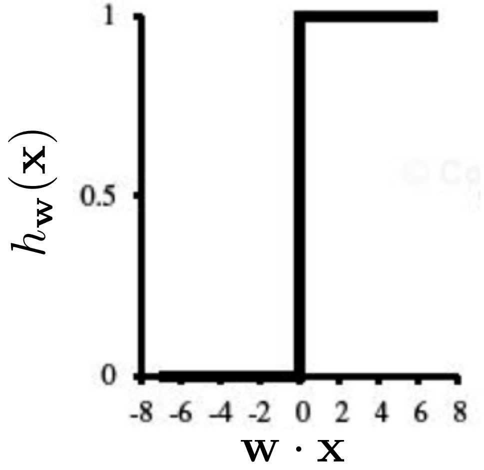
  - on obtient à la règle d'apprentissage du perceptron

$$w_i \leftarrow w_i + \alpha(y_t - h_{\mathbf{w}}(\mathbf{x}_t))x_{t,i} \quad \forall i$$

Apprentissage vue comme la minimisation d'une perte

- La procédure de descente de gradient stochastique est applicable à n'importe quelle perte dérivable partout

- Dans le cas du perceptron, on a un peu triché:
  - la dérivée de $h_{\mathbf{w}}(\mathbf{x})$ n'est pas définie lorsque $\mathbf{w} \cdot \mathbf{x} = 0$



- L'utilisation de la fonction *Threshold* (qui est constante par partie) fait que la courbe d'entraînement peut être instable