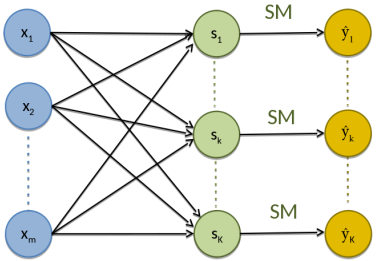
Neural Networks and Deep Learning: Gradient Error Backpropagation Algorithm

Nicolas Thome

Conservatoire National des Arts et Métiers (Cnam)
Département Informatique

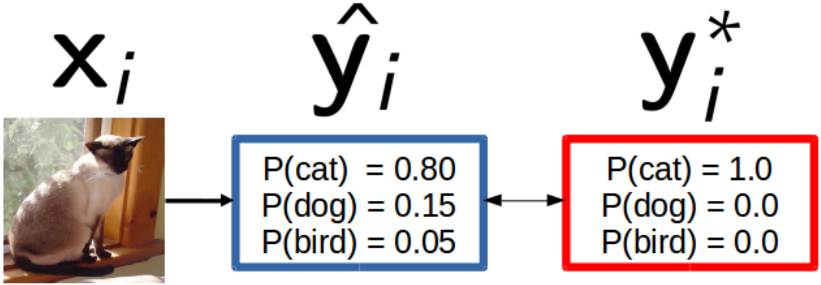
Supervised Learning: Multi-Class Classification

- Logistic Regression for multi-class classification
- $\mathbf{s}_i = \mathbf{x}_i\mathbf{W} + \mathbf{b}$
- Soft-Max (SM): $\hat{\mathbf{y}}_k \sim P(k/\mathbf{x_i}, \mathbf{W}, \mathbf{b}) = \frac{e^{\mathbf{s}k}}{\sum_{k'=1}^{K} e^{\mathbf{s}k'}}$
- Supervised loss function: $\mathcal{L}(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} \ell(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$



1. $\mathbf{y} \in \{1; 2; ...; K\}$
2. $\hat{\mathbf{y}}_i = \arg\max_k P(k/\mathbf{x}_i, \mathbf{W}, \mathbf{b})$
3. $\ell_{0/1}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = \begin{cases} 1 & \text{if } \hat{\mathbf{y}}_i \neq \mathbf{y}_i^* \\ 0 & \text{otherwise} \end{cases}$: **0/1 loss**

Logistic Regression Training Formulation



- **Input x_i, ground truth output supervision y_i^***
- One hot-encoding for y_i^*:

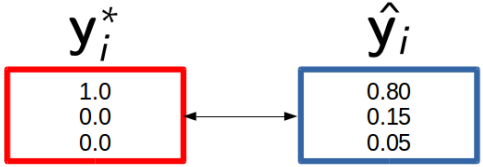
$$y_{c,i}^* = \begin{cases} 1 & \text{if c is the groud truth class for } x_i \\ 0 & \text{otherwise} \end{cases}$$

Logistic Regression Training Formulation

- Loss function: multi-class Cross-Entropy (CE) ℓ_{CE}
- ℓ_{CE}: Kullback-Leiber divergence between $\mathbf{y_i^*}$ and $\hat{\mathbf{y}}_\mathbf{i}$

$$\ell_{CE}(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i^*}) = KL(\mathbf{y_i^*}, \hat{\mathbf{y}}_\mathbf{i}) = -\sum_{c=1}^{K} y^*_{c,i} log(\hat{y}_{c,i}) = -log(\hat{y}_{c^*,i})$$

- ⚠ KL asymmetric: $KL(\hat{\mathbf{y}}_\mathbf{i}, \mathbf{y_i^*}) \neq KL(\mathbf{y_i^*}, \hat{\mathbf{y}}_\mathbf{i})$ ⚠

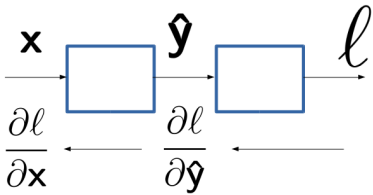


$$KL(\mathbf{y_i^*}, \hat{\mathbf{y}}_\mathbf{i}) = -log(\hat{y}_{c^*,i}) = -log(0.8) \approx 0.22$$

Logistic Regression Training

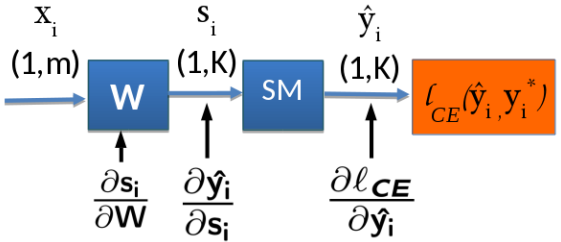
- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$
- ℓ_{CE} smooth convex upper bound of $\ell_{0/1}$
  $\Rightarrow$ **gradient descent optimization**
- Gradient descent: $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}}$
  $(\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} - \eta \frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{b}})$
- Computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}}$?
  $\Rightarrow$ **Backpropagation of gradient error!**
  $\Rightarrow$ **Key Property:** chain rule $\frac{\partial \mathbf{x}}{\partial z} = \frac{\partial \mathbf{x}}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial z}$

Chain Rule



$$\frac{\partial \ell}{\partial \mathbf{x}} = \frac{\partial \ell}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}}$$

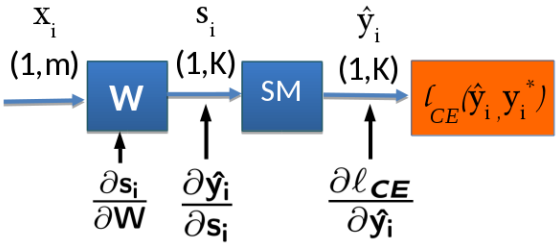
- Logistic regression: $\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_i} \frac{\partial \hat{\mathbf{y}}_i}{\partial \mathbf{s}_i} \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}}$



Logistic Regression Training: Backpropagation

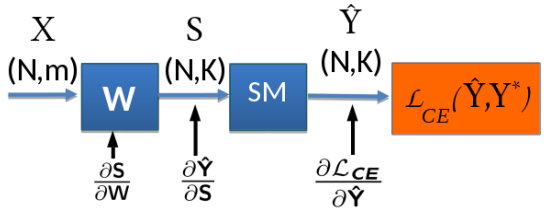
$\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} \frac{\partial \hat{\mathbf{y}}_\mathbf{i}}{\partial \mathbf{s}_\mathbf{i}} \frac{\partial \mathbf{s}_\mathbf{i}}{\partial \mathbf{W}}$, $\ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*) = -log(\hat{y}_{c^*,i}) \Rightarrow$ Update for 1 example:

- $\frac{\partial \ell_{CE}}{\partial \hat{\mathbf{y}}_\mathbf{i}} = \frac{-1}{\hat{y}_{c^*,i}} = \frac{-1}{\hat{\mathbf{y}}_\mathbf{i}} \odot \delta_{\mathbf{c},\mathbf{c}^*}$
- $\frac{\partial \ell_{CE}}{\partial \mathbf{s}_\mathbf{i}} = \hat{\mathbf{y}}_\mathbf{i} - \mathbf{y}_\mathbf{i}^* = \delta_\mathbf{i}^\mathbf{y}$
- $\frac{\partial \ell_{CE}}{\partial \mathbf{W}} = \mathbf{x}_\mathbf{i}{}^T \delta_\mathbf{i}^\mathbf{y}$



Logistic Regression Training: Backpropagation

- Whole dataset: data matrix $\mathbf{X}$ $(N \times m)$, label matrix $\hat{\mathbf{Y}}$, $\mathbf{Y}^*$ $(N \times K)$
- $\mathcal{L}_{CE}(\mathbf{W}, \mathbf{b}) = -\frac{1}{N} \sum_{i=1}^{N} log(\hat{y}_{c^*,i})$, $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_{CE}}{\partial \hat{\mathbf{Y}}} \frac{\partial \hat{\mathbf{Y}}}{\partial \mathbf{S}} \frac{\partial \mathbf{S}}{\partial \mathbf{W}}$

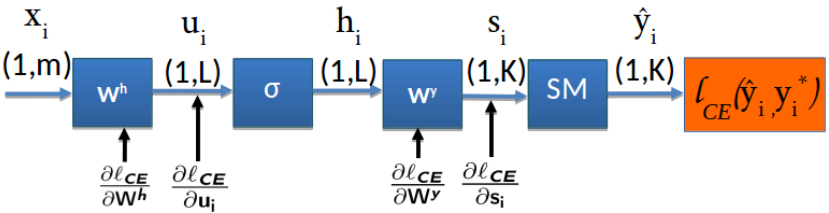


- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{s}} = \hat{\mathbf{Y}} - \mathbf{Y}^* = \Delta^y$
- $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}} = \mathbf{X}^T \Delta^y$

Perceptron Training: Backpropagation

- Perceptron *vs* Logistic Regression: adding hidden layer (sigmoid)
- **Goal:** Train parameters $\mathbf{W}^y$ and $\mathbf{W}^h$ (+bias) with Backpropagation

  $\Rightarrow$ computing $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^y} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}^y}$ and $\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{W}^h} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \ell_{CE}}{\partial \mathbf{W}^h}$



- Last hidden layer ~ Logistic Regression
- First hidden layer: $\frac{\partial \ell_{CE}}{\partial \mathbf{W}^h} = \mathbf{x_i}^T \frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} \Rightarrow$ **computing** $\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \delta_{\mathbf{i}}^{\mathbf{h}}$

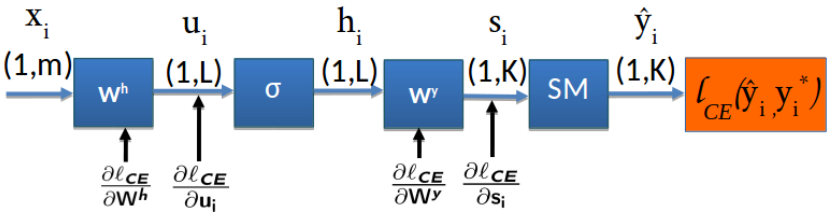
Perceptron Training: Backpropagation

- Computing $\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \delta_{\mathbf{i}}^{\mathbf{h}} \Rightarrow$ use chain rule:

$$\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \sum_{k=1}^{K} \frac{\partial \ell_{CE}}{\partial \mathbf{s_k}} \frac{\partial \mathbf{s_k}}{\partial \mathbf{u_i}}$$

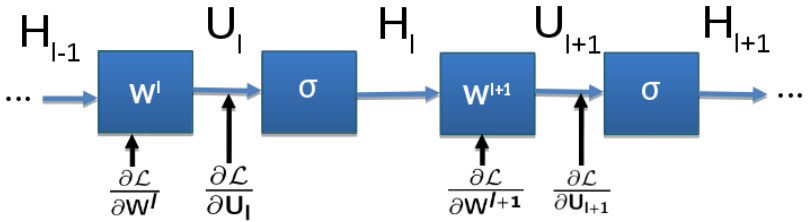
- ... Leading to:

$$\frac{\partial \ell_{CE}}{\partial \mathbf{u_i}} = \delta_{\mathbf{i}}^{\mathbf{h}} = \delta_{\mathbf{i}}^{\mathbf{y}\,T} \mathbf{W^y} \odot \sigma'(\mathbf{h_i}) = \delta_{\mathbf{i}}^{\mathbf{y}\,T} \mathbf{W^y} \odot (\mathbf{h_i} \odot (1 - \mathbf{h_i}))$$



Deep Neural Network Training: Backpropagation

- Multi-Layer Perceptron (MLP): adding more hidden layers
- Backpropagation update ~ Perceptron: **assuming** $\frac{\partial \mathcal{L}}{\partial \mathbf{U}_{l+1}} = \Delta^{l+1}$ **known**
  - $\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l+1}} = \mathbf{H_l}^T \Delta^{l+1}}$
  - $\boxed{\text{Computing } \frac{\partial \mathcal{L}}{\partial \mathbf{U_l}} = \Delta^l}$ $(= {\Delta^{l+1}}^T \mathbf{W^{l+1}} \odot \mathbf{H_l} \odot (1 - \mathbf{H_l})$ sigmoid$)$
  - $\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} = \mathbf{H_{l-1}}^T \Delta^{h_l}}$



Backpropagation: Conclusion

- **Backpropagation: solution for end-to-end training of deep neural networks**
  - **Core for deep learning**
- **Long history and paternity:**
  - Chain rule [Leibniz, 1676], gradient descent optimization [Cauchy, 1847]
  - First efficient back-prop applications, [Kelley, 1960, Bryson and Ho, 1969]
  - Neural networks training [Lecun, 1985, Rumelhart et al., 1986]
  - Overview [Schmidhuber, 2014]
- **Training issues with back-prop? $\Rightarrow$ following!**

References I

Bryson, E. and Ho, Y. C. (1969).
Applied optimal control: optimization, estimation, and control.
Blaisdell Publishing Company.

Cauchy, A.-L. (1847).
Méthode générale pour la résolution des systèmes d'équations simultanées, pages 399–402.

Kelley, H. J. (1960).
Gradient theory of optimal flight paths.
ARS Journal, 30(10):947–954.

Lecun, Y. (1985).
Une procedure d'apprentissage pour reseau a seuil asymmetrique (A learning scheme for asymmetric threshold networks), pages 599–604.

Leibniz, G. W. (1676).
Memoir using the chain rule (cited in TMME 7:2&3 p 321-332, 2010), 1676.

Rumelhart, D., Hinton, G., and Williams, R. (1986).
Learning representations by back-propagating errors.
Nature, 323:533–536.

Schmidhuber, J. (2014).
Deep learning in neural networks: An overview.
CoRR, abs/1404.7828.

References II