Tutorial 8

Tuesday 11th October, 2016

Problem 1. In class, we saw the detailed derivation of backpropogation update rules when each of the activation units is a sigmoid. You need to derive all the update rules when each activation unit happens to be rectified linear unit (ReLU).

$$\sigma(s) = max(\theta, s)$$

(since we often represent σ).) by $g(.)$, this also means $g(s) = max(\theta, s)$)

Typically, $\theta = 0$. Note that ReLU is differentiable at all points except at $s = \theta$. But by using subgradient $\nabla_s \sigma$ instead of gradient $\nabla \sigma$, we can complete backpropagation as 'subgradient descent'. Note that subgradient is the same as gradient in regions in which the function is differentiable. Thus,

$$\nabla_s \sigma(s) = 1,\ s \in (\theta, \infty)\ \ ,\ \nabla_s \sigma(s) = 0 \text{ if } s < \theta \text{ and } \nabla_s \sigma(s) \in [0,1] \text{ if } s = \theta$$

The interval $[0, 1]$ is the subdifferential (denoted ∂), which is set of subgradients of σ at θ.

Is there a problem in cascading several layers of ReLU? Recall that we invoked subgradients in justifying the *Iterative Soft Thresholding Algorithm* for LASSO. And that LASSO gave sparsity owing to hard thresholding.

Solution:

All the gradients and partial derivatives in the backpropagation algorithm will remain unchanged except for the $\frac{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{p}}^{\mathbf{l+1}}}$ since σ is not differentiable now at all points. So the new

$$\frac{\partial \sigma_p^{l+1}}{\partial sum_p^{l+1}} = 1,\ sum_p^{l+1} \in [\theta, \infty)\ \ ,\ \frac{\partial \sigma_p^{l+1}}{\partial sum_p^{l+1}} = 0 \text{ if } sum_p^{l+1} < \theta$$

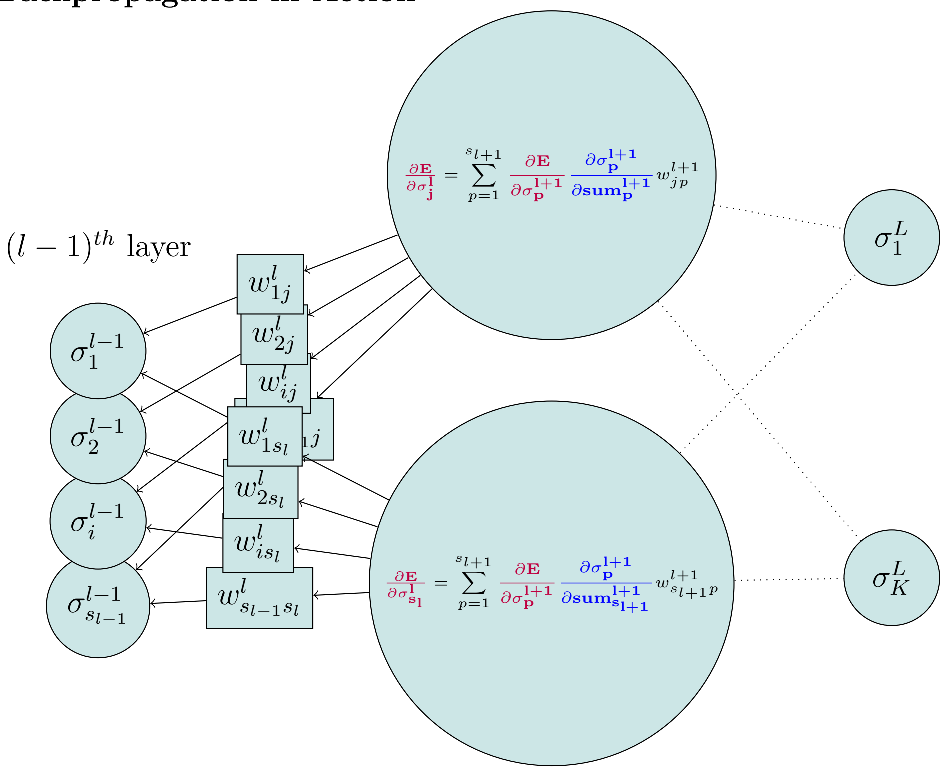
will be one possible choice

- For a single example $(\mathbf{x}, y)$:

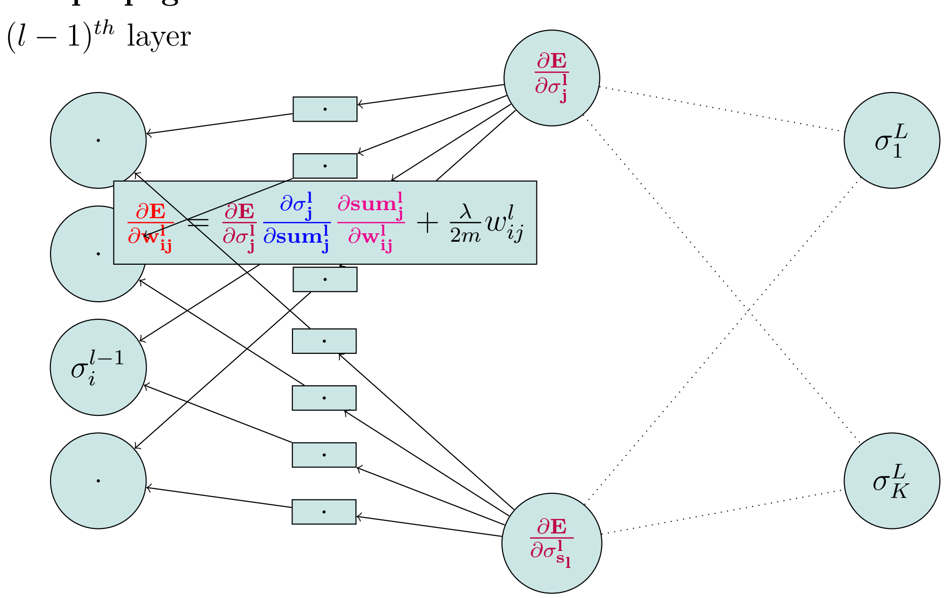
$$\begin{aligned} -\left[\sum_{k=1}^{K} y_k \log\left(\sigma_k^L(\mathbf{x})\right) + (1 - y_k)\log\left(1 - \sigma_k^L(\mathbf{x})\right)\right] \\ + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \end{aligned} \quad (1)$$

- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial E}{\partial sum_p^{l+1}} \frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}} \frac{\partial \sigma_{\mathbf{p}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{p}}^{\mathbf{l+1}}} w_{jp}^{l+1}$ since $\frac{\partial sum_p^{l+1}}{\partial \sigma_j^l} = w_{jp}^{l+1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{L}}} = -\frac{\mathbf{y_j}}{\sigma_{\mathbf{j}}^{\mathbf{L}}} - \frac{\mathbf{1-y_j}}{\mathbf{1-\sigma_j^L}}$

Backpropagation in Action



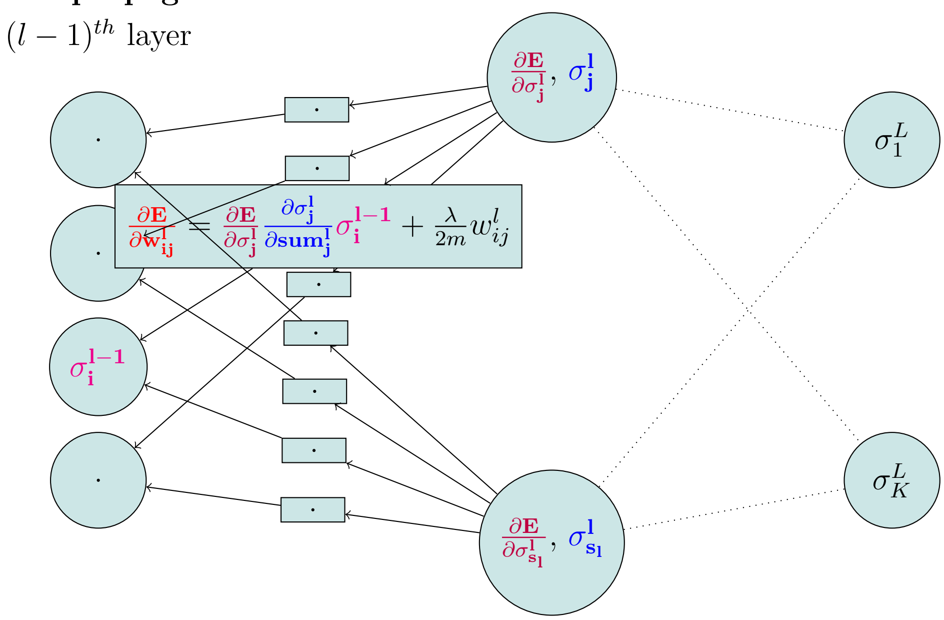
Backpropagation in Action



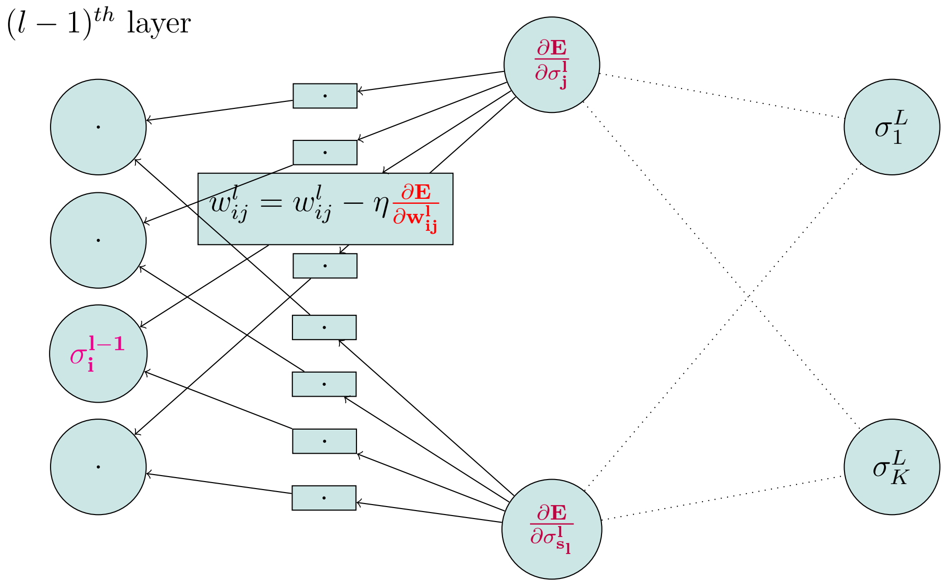
Recall and Substitute

- $sum_j^l = \sum_{k=1}^{s_{l-1}} w_{kj}^l \sigma_k^{l-1}$ and $\sigma_i^l = \frac{1}{1+e^{-sum_i^l}}$
- $\frac{\partial \mathbf{E}}{\partial \mathbf{w_{ij}^l}} = \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} \frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} + \frac{\lambda}{2m} w_{ij}^l$
- $\frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} = \mathbf{1}$, **if** $\mathbf{sum_j^l} \in [\theta, \infty)$, $\frac{\partial \sigma_\mathbf{j}^\mathbf{l}}{\partial \mathbf{sum_j^l}} = \mathbf{0}$ if $\mathbf{sum_j^l} < \theta$
- $\frac{\partial \mathbf{sum_j^l}}{\partial \mathbf{w_{ij}^l}} = \sigma_\mathbf{i}^\mathbf{l-1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{l+1}} \frac{\partial \sigma_\mathbf{j}^\mathbf{l+1}}{\partial \mathbf{sum_j^{l+1}}} w_{jp}^{l+1}$
- $\frac{\partial \mathbf{E}}{\partial \sigma_\mathbf{j}^\mathbf{L}} = -\frac{\mathbf{y_j}}{\sigma_\mathbf{j}^\mathbf{L}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_\mathbf{j}^\mathbf{L}}$

Backpropagation in Action



Backpropagation in Action



The Backpropagation Algorithm for Training NN

1. Randomly initialize weights w^l_{ij} for $l = 1, \ldots, L$, $i = 1, \ldots, s_l$, $j = 1, \ldots, s_{l+1}$.
2. Implement **forward propagation** to get $f_{\mathbf{w}}(\mathbf{x})$ for every $\mathbf{x} \in \mathcal{D}$.
3. Execute **backpropagation** on any misclassified $\mathbf{x} \in \mathcal{D}$ by performing gradient descent to minimize (non-convex) $E(\mathbf{w})$ as a function of parameters $\mathbf{w}$.
4. $\frac{\partial \mathbf{E}}{\partial \sigma^{\mathbf{L}}_{\mathbf{j}}} = -\frac{\mathbf{y_j}}{\sigma^{\mathbf{L}}_{\mathbf{j}}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma^{\mathbf{L}}_{\mathbf{j}}}$ for $j = 1$ to s_L.
5. For $l = L-1$ down to 2:
   (a) $\frac{\partial \sigma^{\mathbf{l}}_{\mathbf{j}}}{\partial \mathbf{sum^l_j}} = \mathbf{1}, \ \textbf{if } \mathbf{sum^l_j} \in [\theta, \infty) \ , \ \frac{\partial \sigma^{\mathbf{l}}_{\mathbf{j}}}{\partial \mathbf{sum^l_j}} = \mathbf{0} \ \text{if } \mathbf{sum^l_j} < \theta$
   (b) $\frac{\partial \mathbf{E}}{\partial \sigma^{\mathbf{l}}_{\mathbf{j}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma^{\mathbf{l+1}}_{\mathbf{j}}} \frac{\partial \sigma^{\mathbf{l+1}}_{\mathbf{j}}}{\partial \mathbf{sum^{l+1}_j}} w^{l+1}_{jp}$
   (c) $\frac{\partial \mathbf{E}}{\partial \mathbf{w^l_{ij}}} = \frac{\partial \mathbf{E}}{\partial \sigma^{\mathbf{l}}_{\mathbf{j}}} \frac{\partial \sigma^{\mathbf{l}}_{\mathbf{j}}}{\partial \mathbf{sum^l_j}} \sigma^{\mathbf{l-1}}_{\mathbf{i}} + \frac{\lambda}{2m} w^l_{ij}$
   (d) $w^l_{ij} = w^l_{ij} - \eta \frac{\partial \mathbf{E}}{\partial \mathbf{w^l_{ij}}}$
6. Keep picking misclassified examples until the cost function $E(\mathbf{w})$ shows significant reduction; else resort to some random perturbation of weights $\mathbf{w}$ and restart a couple of times.

Problem 2. Compute the minimum number of multiplications and additions for a single backpropagation while also estimating the memory required for the minimum number of such multiplications and additions to become possible.

Problem 3. Solve the assignment at `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/udacity/4_convolutions.ipynb`

Follow the instructions to implement and run each indicated step. Some steps have been implemented for you. This is a self-evaluated assignment. Make sure you are able to solve each problem and answer any posed questions and save the answers/solutions wherever possible.