Tutorial 7

CS 337 Artificial Intelligence & Machine Learning, Autumn 2019

October, 2019

Problem 1. Design a multilayer perceptron which will learn to recognize various forms of the letters C,L,T placed on a 3x3 grid (as shown in Figure 1) through backpropagation algorithm.

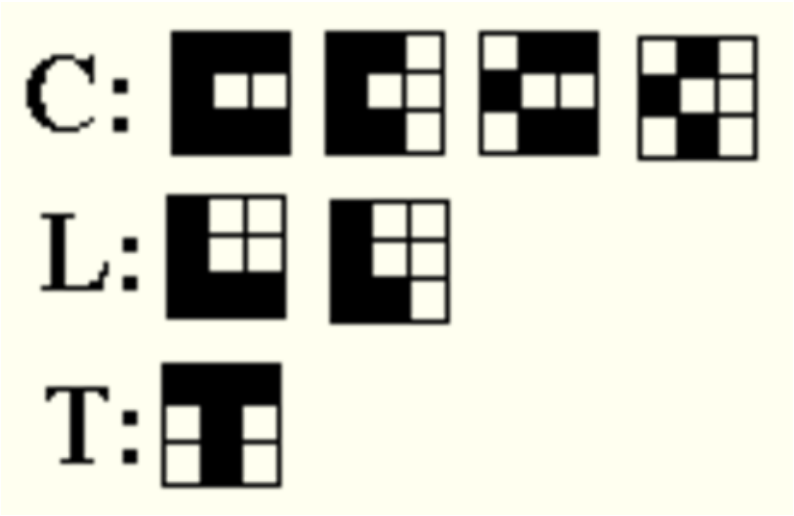


Figure 1: Various forms of the letters C,L,T placed on a 3×3 grid

1. Design a one layer network indicating what should be applied at the input layer and what should be expected at the output layer showing the number of neurons, the connections between them and the neuron's output function.
2. repeat (a) for two layer network by adding a hidden layer

Problem 2. Consider a perceptron (shown in Figure 2) for which $u \in R^2$ and

$$f(a) = \begin{cases} 1 & a > 0 \\ 0 & a = 0 \\ -1 & a < 0 \end{cases}$$

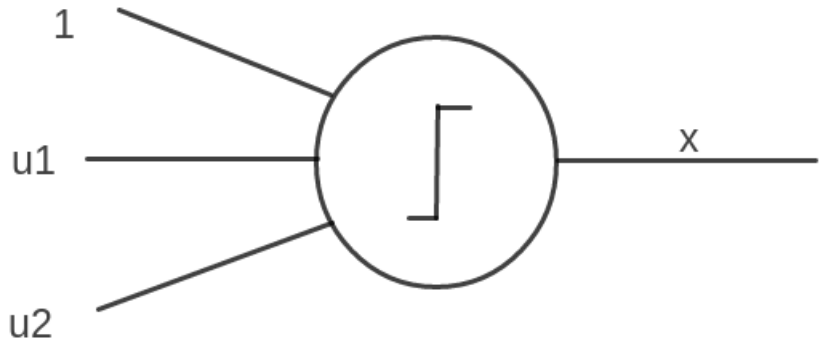


Figure 2: A simple perceptron depicting function f in Problem 2.

Let the desired output be 1 when elements of class A = {(1,2),(2,4),(3,3),(4,4)} is applied as input and let it be -1 for the class B = {(0,0),(2,3),(3,0),(4,2)}. Let the initial connection weights $w_0(0) = +1, w_1(0) = -2, w_2(0) = +1$ and learning rate be h = 0.5.

This perceptron is to be trained by perceptron convergence procedure, for which the weight update formula is $w(t+1) = w(t) + \eta(y^k - x^k(t))u^k$

1. (a) Mark the elements belonging to class A with x and those belonging to class B with o on input space.
   (b) Draw the line represented by the perceptron considering the initial connection weights w(0).
   (c) Find out the regions for which the perceptron output is +1 and –1
   (d) Which elements of A and B are correctly classified, which elements are misclassified and which are unclassified?
2. If u=(4,4) is applied at input, what will be w(1) ?
3. Repeat a) considering w(1).
4. If u=(4.2) is then applied at input, what will be w(2)?
5. Repeat 1) considering w(2).

6. Do you expect the perceptron convergence procedure to terminate? Why?

Problem 3. Problems on Neural Networks

1. We have proved the following for the perceptron update rule:

   If $||\mathbf{w}^*|| = 1$ and if there exists $\theta > 0$ such that for all $i = 1, \ldots, n$, $y_i(\mathbf{w}^*)^T\phi(\mathbf{x}_i) \geq \theta$ and $||\phi(\mathbf{x}_i)||^2 \leq \Gamma^2$ then the perceptron algorithm will make atmost $\frac{\Gamma^2}{\theta^2}$ errors (that is take atmost $\frac{\Gamma^2}{\theta^2}$ iterations to converge)

   Point out specifically the challenge in extending step(s) in the proof to *(show) convergence of backpropagation in Neural Networks?

   Solution:

   When we look at the proof from the point of view of extending this claim or line of thought to neural networks, the first thing we note is that the update rule will change owing to the use of sigmoid activation and with backpropagation. As a result, the first inequality (reproduced below in red) will break.

   We showed the following: We know that $||\mathbf{w}^*||_2^2 = 1$ and $y_i\phi(\mathbf{x}_i)\mathbf{w}^* \geq \theta$ for all i. We assume that $\mathbf{w}^{(0)} = 0$

   Now consider $(\mathbf{w}^*)^T\mathbf{w}^{(k)} = (\mathbf{w}^*)^T(\mathbf{w}^{(k-1)} + y_i\phi(\mathbf{x}_i)) = (\mathbf{w}^*)^T\mathbf{w}^{(k-1)} + y_i(\mathbf{w}^*)^T\phi(\mathbf{x}_i) \geq (\mathbf{w}^*)^T\mathbf{w}^{(k-1)} + \theta \geq (\mathbf{w}^*)^T\mathbf{w}^{(k-2)} + 2\theta \geq (\mathbf{w}^*)^T\mathbf{w}^{(0)} + k\theta = k\theta$

   Thus,

   $$(\mathbf{w}^*)^T\mathbf{w}^{(k)} \geq k\theta$$

   and because

   $$||\mathbf{w}^*||||\mathbf{w}^{(k)}|| = ||\mathbf{w}^{(k)}|| \geq |(\mathbf{w}^*)^T\mathbf{w}^{(k)}|$$

   we must have

   $$||\mathbf{w}^{(k)}|| \geq k\theta$$

   Similarly,

   $||\mathbf{w}^{(k)}||_2^2 = ||\mathbf{w}^{(k-1)} + y_i\phi(\mathbf{x}_i)||_2^2 = ||\mathbf{w}^{(k-1)}||_2^2 + y_i^2||\phi(\mathbf{x}_i)||_2^2 + 2y_i(\mathbf{w}^{(k-1)})^T\phi(\mathbf{x}_i) < ||\mathbf{w}^{(k-1)}||_2^2 + \Gamma^2 < ||\mathbf{w}^{(k-2)}||_2^2 + 2\Gamma^2 < ||\mathbf{w}^{(0)}||_2^2 + k\Gamma^2 = k\Gamma^2$

   since $y_i^2 = 1$ and it must have been that (as per perceptron update rule)

   $y_i(\mathbf{w}^{(k-1)})^T\phi(\mathbf{x}_i) < 0$

   Thus,

   $$||\mathbf{w}^{(k)}||_2^2 < k\Gamma^2$$

   and

   $$||\mathbf{w}^{(k)}||_2^2 \geq k^2\theta^2$$

   which implies

   $$k^2\theta^2 < k\Gamma$$

that is,

$$k < \frac{\Gamma}{\theta^2}$$

which proves our claim.

2. Consider one layer of weights (edges) in a convolutional neural network (CNN) for grayscale images, connecting one layer of units to the next layer of units. Which type of layer has the fewest parameters to be learned during training? Explain/justify your answer.

   (A) A convolutional layer with ten 3×3 filters
   (B) A max-pooling layer that reduces a 10×10 image to 5×5
   (C) A convolutional layer with eight 5×5 filters
   (D) A fully-connected layer from 20 hidden units to 4 output units

   Solution:

   (B). It is based on simple parameter calculation similar to what will be done in class.

Problem 4. Neural logic

Our goal is to build a simple neural network which takes a point $(x_1, x_2) \in [0, 1]^2$ (i.e., a point in the unit square), and outputs $f(x_1, x_2) \in \{0, 1\}$, where f is as shown in the Figure 3 below. (The shaded region, including its boundaries, corresponds to all the points which are mapped to 1.)

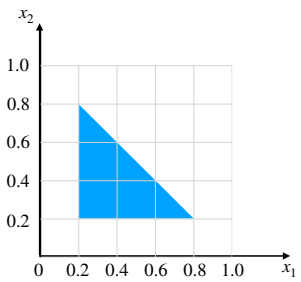


Figure 3: Figure showing function f that takes a point $(x_1, x_2) \in [0, 1]^2$ (i.e., a point in the unit square), and outputs $f(x_1, x_2) \in \{0, 1\}$

Suppose a 2-input neuron is defined as $N(x_1, x_2) = \tau(w_0 + w_1 x_1 + w_2 x_2)$ where $w_0, w_1, w_2 \in \mathbb{R}$ are real-valued weights, and $\tau : \mathbb{R} \to \{0, 1\}$ is defined so that $\tau(x) = 1$ iff $x \geq 0$.

Show a set of weights on a two-layer neural network as shown below in the Figure 4 to implement f. The 3-input neuron on the top layer should correspond to the boolean operator AND (when 0 is interpreted as false and 1 as true). Fill in all the blanks corresponding to the weights in the Figure 4 and justify your answers.

Solution:

F1: $w_0 = -0.2, w_1 = 1, w_2 = 0$
F2: $w_0 = -0.2, w_1 = 0, w_2 = 1$
F3: $w_0 = 1, w_1 = -1, w_2 = -1$
AND: $w_0 = -3, w_1 = 1, w_2 = 1, w_3 = 1$

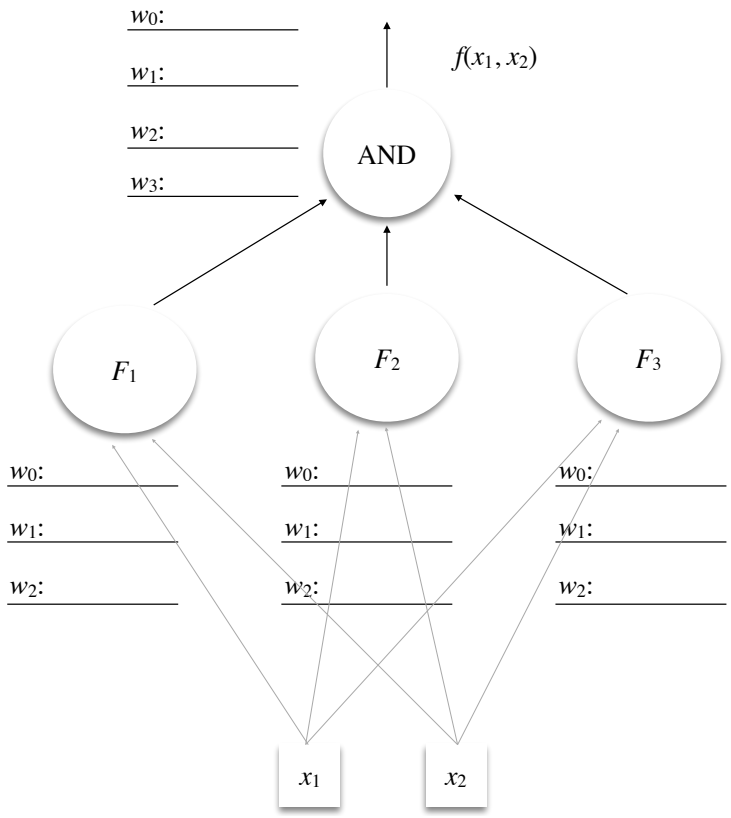


Figure 4: Fill in all the blanks corresponding to the weights in this figure and justify your answers.

Problem 5. In the class, we discussed the probabilistic binary (class) logistic regression classifier. How will you extend logistic regression probabilistic model to multiple (say K) classes? Are their different ways of extending? What is the intuition behind each? Discuss and contrast advantages/disadvantages in each.

Solution:

One might suggest handling multi-class (K) classification via K one-vs-rest probabilistic classifiers. But there is no obvious probabilistic semantics associated with such a classifier (question asked for a probabilistic MODEL for multiple classes).

Basic idea is that each class c can have a different weight vector $[w_{c,1}, w_{c,2}, \ldots, w_{c,k}, \ldots, w_{c,K}]$.

Extension to multi-class logistic

1. Each class $c = 1, 2, \ldots, K-1$ can have a different weight vector $[\mathbf{w}_{c,1}, \mathbf{w}_{c,2}, \ldots, \mathbf{w}_{c,k}, \ldots, \mathbf{w}_{c,K-1}]$ and

$$p(Y = c|\phi(\mathbf{x})) = \frac{e^{-(\mathbf{w}_c)^T \phi(\mathbf{x})}}{1 + \sum_{k=1}^{K-1} e^{-(\mathbf{w}_k)^T \phi(\mathbf{x})}}$$

for $c = 1, \ldots, K-1$ so that

$$p(Y = K|\phi(\mathbf{x})) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{-(\mathbf{w}_k)^T \phi(\mathbf{x})}}$$

Alternative (equivalent) extension to multi-class logistic

1. Each class $c = 1, 2, \ldots, K$ can have a different weight vector $[\mathbf{w}_{c,1}, \mathbf{w}_{c,2} \ldots \mathbf{w}_{c,p}]$ and

$$p(Y = c|\phi(\mathbf{x})) = \frac{e^{-(\mathbf{w}_c)^T \phi(\mathbf{x})}}{\sum_{k=1}^{K} e^{-(\mathbf{w}_k)^T \phi(\mathbf{x})}}$$

for $c = 1, \ldots, K$.

This function is also the called the **softmax**[1] function.

Problem 6. Suppose you are provided a multi-layer neural network for a binary classification problem. Can you convert this network into an equivalent kernel perceptron (single node neural network only)? What will be the exact steps?

Solution:

1. Yes. You are already provided the NN classifier. So you need not worry about separability etc. In fact, the question was only about an equivalent classifier (even if not-separating completely) which can be constructed as follows:

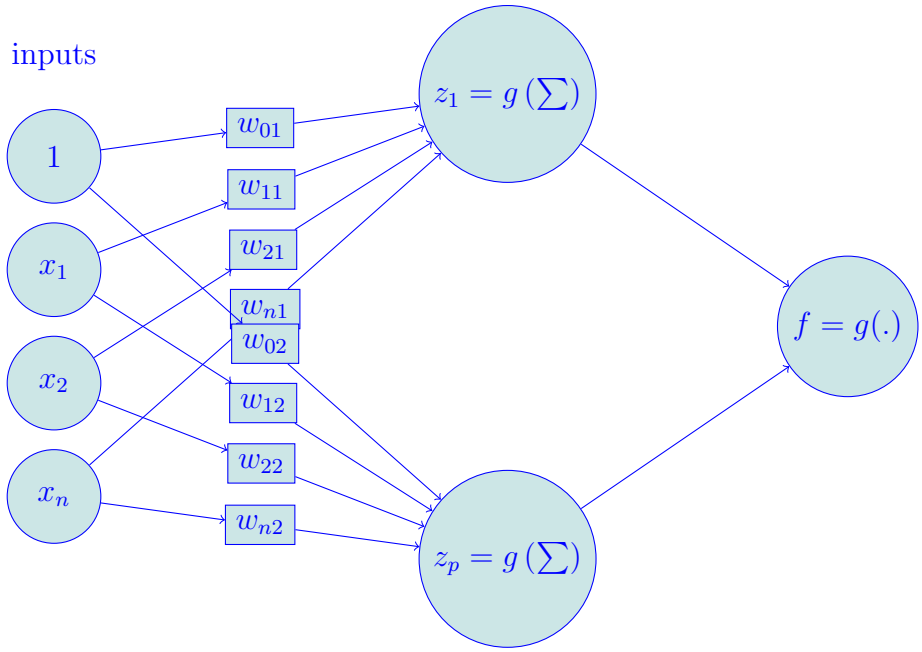
2. Let the neural network be as shown below with p nodes $(z_1, z_2, \ldots, z_p)$ in the last layer. You can develop an equivalent kernel preceptron by specifying

$$\phi_z(x) = [z_1(x), z_2(x), \ldots, z_p(x)]$$

and using the following Kernel:

$$K_z(x, x') = \phi_z^T(x)\phi_z(x')$$

[1] `https://en.wikipedia.org/wiki/Softmax_function`



Problem 7. 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

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

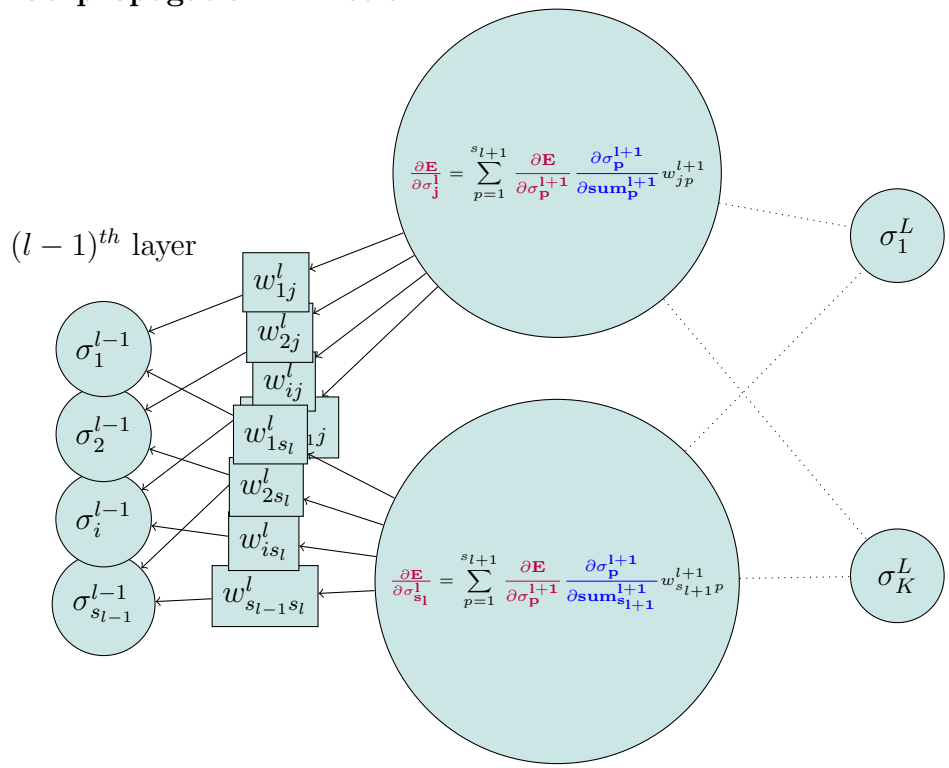
will be one possible choice

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

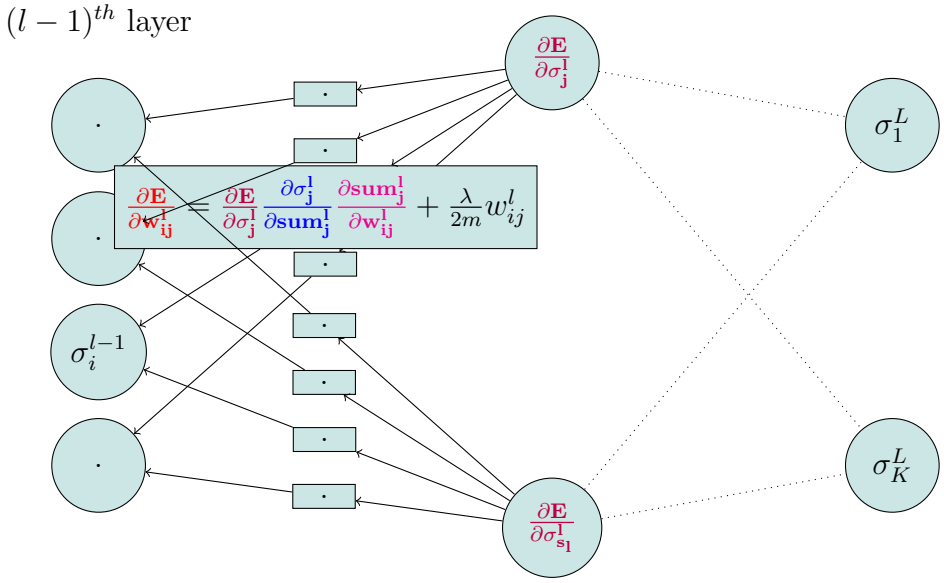
$$-\left[\sum_{k=1}^{K} y_k \log\left(\sigma_k^L(\mathbf{x})\right) + (1-y_k)\log\left(1-\sigma_k^L(\mathbf{x})\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L}\sum_{i=1}^{s_{l-1}}\sum_{j=1}^{s_l}\left(w_{ij}^l\right)^2 \tag{1}$$

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

Backpropagation in Action



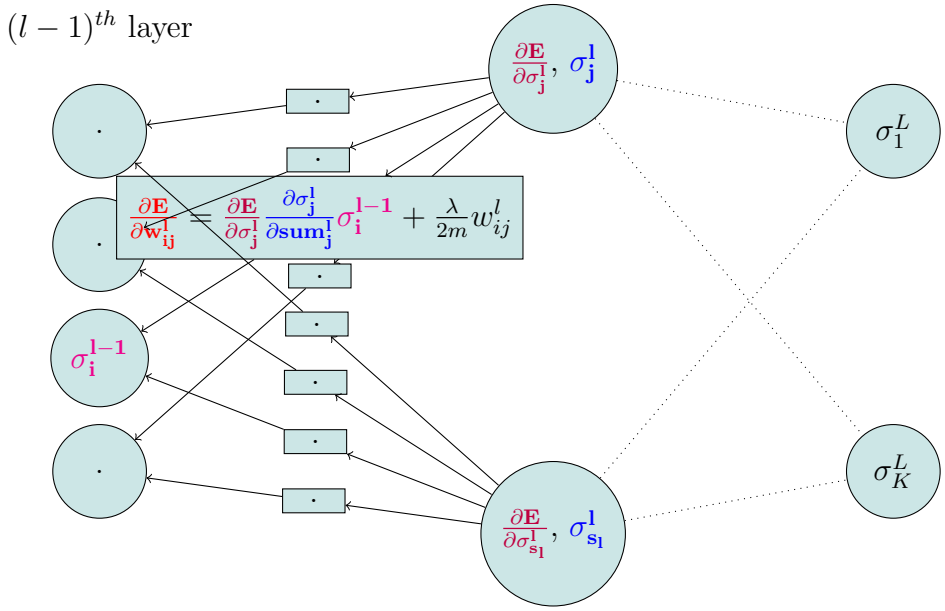
Backpropagation in Action



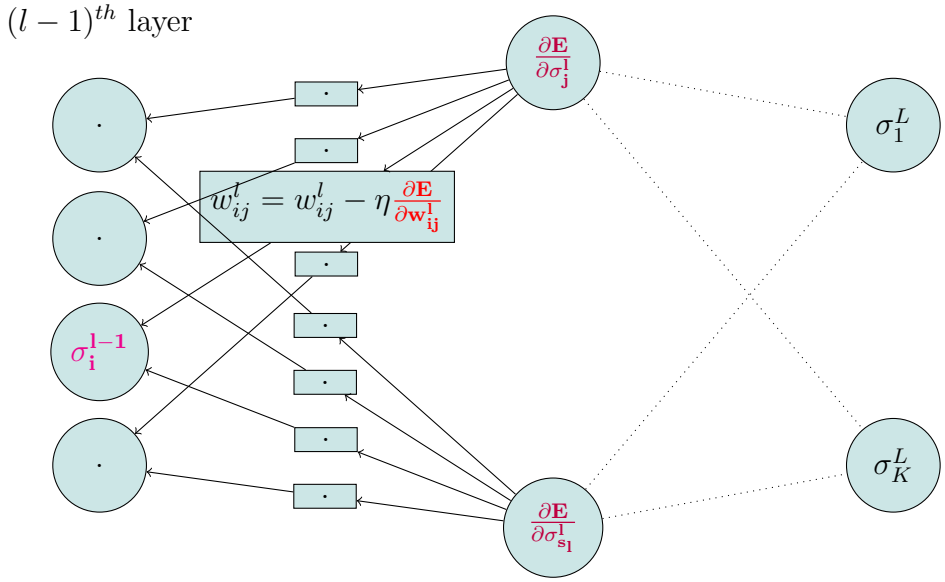
Recall and Substitute

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

Backpropagation in Action



Backpropagation in Action



The Backpropagation Algorithm for Training NN

1. Randomly initialize weights w_{ij}^{l} 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. $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{L}}} = -\frac{\mathbf{y_j}}{\sigma_{\mathbf{j}}^{\mathbf{L}}} - \frac{\mathbf{1-y_j}}{\mathbf{1}-\sigma_{\mathbf{j}}^{\mathbf{L}}}$ for $j = 1$ to s_L.
5. For $l = L-1$ down to 2:
   (a) $\frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}} = \mathbf{1}$, **if** $\mathbf{sum}_{\mathbf{j}}^{\mathbf{l}} \in [\theta, \infty)$, $\frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}} = \mathbf{0}$ if $\mathbf{sum}_{\mathbf{j}}^{\mathbf{l}} < \theta$

(b) $\frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} = \sum_{p=1}^{s_{l+1}} \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l+1}}} \frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l+1}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l+1}}} w_{jp}^{l+1}$

(c) $\frac{\partial \mathbf{E}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}} = \frac{\partial \mathbf{E}}{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}} \frac{\partial \sigma_{\mathbf{j}}^{\mathbf{l}}}{\partial \mathbf{sum}_{\mathbf{j}}^{\mathbf{l}}} \sigma_{\mathbf{i}}^{\mathbf{l-1}} + \frac{\lambda}{2m} w_{ij}^{l}$

(d) $w_{ij}^{l} = w_{ij}^{l} - \eta \frac{\partial \mathbf{E}}{\partial \mathbf{w}_{\mathbf{ij}}^{\mathbf{l}}}$

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 8. 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 9. 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.

Problem 8. Extend the backpropagation algorithm, developed on general Neural Networks to Convolutional Neural Networks (with and without Max Pooling).

Advanced and optional: Also extend the backpropagation algorithm to Recurrent Neural Networks and LSTMs.

Problem 10. ConvNetJS (`http://cs.stanford.edu/people/karpathy/convnetjs/`) is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser. Try different choices of network configurations which include the choice of the stack of convolution, pooling, activation units, number of parallel networks, position of fully connected layers and so on. You can also save some network snapshots as JSON objects. What does the network visualization of the different layers reveal?

Also try out the demo at `http://places.csail.mit.edu/demo.html` to understand the heat maps and their correlations with the structure of the neural network.

Problem 11. Discuss the advantages and disadvantages of different activation functions: tanh, sigmoid, ReLU, softmax. Explain and illustrate when you would choose one activation function in lieu of another in a Neural Network. You can also include any experiences from Problem 5 in your answer.