Exercises
Computational Intelligence Lab
SS 2019

Machine Learning Institute
Dept. of Computer Science, ETH Zürich
Prof. Dr. Thomas Hofmann
Web http://cil.inf.ethz.ch/

Series 8, April 11-12, 2019 (Neural Networks)

Problem 1 (Linear MLP):

1. Let's write down the output, o'':

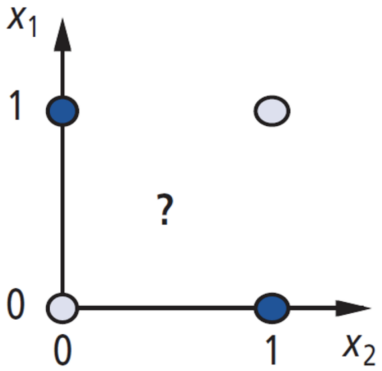
$$h_1 = a \times x_1 + c \times x_2$$
$$h_2 = b \times x_1 + d \times x_2$$
$$o'' = e \times h_1 + f \times h_2$$

   Plugging in h_1 and h_2 in the last equation you obtain:

$$o'' = (ae + bf) \times x_1 + (ce + df) \times x_2$$

   Hence setting $\omega_1 = ae + bf$ and $\omega_2 = (ce + df)$ the two outputs o' and o'', are the same.
   In general any multilayer linear perceptron can be squeezed as single layer linear perceptron with according weights, hence there is no gain in adding layers.

2. Intuitively XOR outputs are not linearly separable. This can be seen in the figure below, there are no straight lines that divides the blue points, where the XOR output is 1, to the light blue ones in which the XOR output is 0.



   More formally, any linear MLP can be seen as a single layer linear perceptron, i.e. the output is a linear transformation of its input: $o = \omega_1 x_1 + \omega_2 x_2$. If we want the output to imitate the XOR function then: $o([1,0]) = 1 = \omega_1$ and $o([0,1]) = 1 = \omega_2$. However $o([1,1]) = \omega_1 + \omega_2 = 2 \neq 0$, hence there are no ω_1, ω_2 that satisfy the conditions.

Problem 2 (Neural Networks):

1.
$$\forall x \in \mathbb{R},\ s'(x) = -\frac{-e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = s(x) \cdot (1 - s(x)).$$

2.
$$h_1 = \omega'_{11} \times x_1 + \omega'_{12} \times x_2 = 0.18$$
$$h_2 = \omega'_{21} \times x_1 + \omega'_{22} \times x_2 = 0.14$$

$$out(h_1) = s(h_1) = \frac{1}{1+\exp(-h_1)} = 0.54$$

$$out(h_2) = s(h_2) = \frac{1}{1+\exp(-h_2)} = 0.53$$

$$o = \omega_1'' \times out(h_1) + \omega_2'' \times out(h_2) = 0.22$$

$$\hat{y} = s(o) = \frac{1}{1+\exp(-o)} = 0.55.$$

The loss is then:

$$L(\mathbf{w}', \mathbf{w}'') = -y \log \hat{y} - (1-y)\log(1-\hat{y}) = -\log(1-\hat{y}) = 0.81$$

3. (a) For one training example $\mathbf{x} = (x_i)_{1\leq i\leq d}$ and m hidden layers $(h_j)_{1\leq j\leq m}$ then:

$$h_j = s(\sum_{i=0}^{d} \omega_{ji} x_i)$$

$$\hat{y} = \sum_{j=0}^{m} \omega_j h_j = \sum_{j=0}^{m} \omega_j s(\sum_{i=0}^{d} \omega_{ji} x_i) = \mathbf{w}^T \mathbf{s}(W\mathbf{x})$$

where $\mathbf{w} = (w_1, w_2, \cdots, w_m)$ and $W_{ji} = w_{ji}$ and $\mathbf{s}(\mathbf{y}) = (s(y_1), \cdots, s(y_m))$.

(b) The MSE for one training example with label y is given by

$$E(\mathbf{w}) = (\hat{y} - y)^2 = (\sum_{j=0}^{m} \omega_j s(\sum_{i=0}^{d} \omega_{ji} x_i) - y)^2,$$

and its partial derivative with respect to the single parameter $w_{j'}$ is given by:

$$\frac{\partial}{\partial w_{j'}} E(\mathbf{w}) = 2(\hat{y} - y) \cdot \frac{\partial}{\partial w_{j'}} \hat{y}$$

$$\frac{\partial}{\partial w_{j'}} \hat{y} = h_j.$$

Hence we obtain:

$$\frac{\partial}{\partial w_{j'}} E(\mathbf{w}) = 2(\sum_{j=0}^{m} \omega_j s(\sum_{i=0}^{d} \omega_{ji} x_i) - y) \cdot s(\sum_{i=0}^{d} \omega_{j'i} x_i)$$

.

The partial derivative w.r.t. the parameter $w_{j'i'}$ is instead:

$$\frac{\partial}{\partial w_{j'i'}} E(\mathbf{w}) = 2(\hat{y} - y) \cdot \frac{\partial}{\partial w_{j'i'}} \hat{y}$$

$$\frac{\partial}{\partial w_{j'i'}} \hat{y} = \omega_{j'} \cdot \frac{\partial}{\partial w_{j'i'}} (h_{j'})$$

$$\frac{\partial}{\partial w_{j'i'}} (h_{j'}) = s'(\sum_{i=0}^{d} \omega_{ji} x_i) \cdot x_{i'} = s(\sum_{i=0}^{d} \omega_{ji} x_i)(1 - s(\sum_{i=0}^{d} \omega_{ji} x_i)) x_{i'}$$

Notice that this closed-form formula doesn't require us to actually compute any derivative.

(c) The partial derivative of the loss w.r.t. the input $x_{i'}$ is:

$$\frac{\partial}{\partial xi'} E(\mathbf{w}) = 2(\hat{y} - y) \cdot \frac{\partial}{\partial x_{i'}} \hat{y}$$

$$\frac{\partial}{\partial x_{i'}} \hat{y} = \sum_{j=0}^{m} \frac{\partial \hat{y}}{\partial h_{j'}} \cdot \frac{\partial h_{j'}}{\partial x_{i'}} = \sum_{j=0}^{m} \omega_j s'(\sum_{i=0}^{d} \omega_{ji} x_i) \cdot \frac{\partial}{\partial x_{i'}} (\sum_{i=0}^{d} \omega_{ji} x_i) = \sum_{j=0}^{m} \omega_j s(\sum_{i=0}^{d} \omega_{ji} x_i)(1 - s(\sum_{i=0}^{d} \omega_{ji} x_i)) \omega_{ji'}$$

Problem 3 (Convolutional Neural Networks):

1. The convolutional layer has two 2-channels filters K^1 and K^2, with each filter channel K^l_c having an odd number of pixels, so that we can index them as $(K^l_c)_{i,j}$ for $-k \leq i,j \leq k$. The convolutional layer can be seen as taking as input the image $(I_c)_{1\leq c\leq 2}$ where $(I_c)_{i,j}$ is the pixel of position (i,j) and channel c, for $1 \leq i,j \leq 4$, and giving as output the following 4x4 2-channels image, whose pixel (i',j') of channel l is given by
$$(I \star K^l)_{i',j'} = \sum_{1\leq c\leq 2} \sum_{-k\leq i,j\leq k} (I_c)_{i'+i,j'+j}(K^l_c)_{i,j},$$
where $(I_c)_{a,b} = 0$ for (a,b) outside of the range of pixels $\{1,...,4\} \times \{1,...,4\}$ of I, because of the zero-padding.

2. The ReLU activation function is defined by $ReLU(x) = \max(0,x)$. Applying such a real function to an image consists of applying it to each pixel. Hence after the convolutional layer and the non-linearity, we get the following two images, for $l \in \{1,2\}$, whose pixels (i',j') are given by
$$(ReLU(I \star K^l))_{i',j'} = \max\left(0, \sum_{1\leq c\leq 2} \sum_{-k\leq i,j\leq k} (I_c)_{i'+i,j'+j}(K^l_c)_{i,j}\right).$$
Then, applying a 3x3 max-pooling with stride 1 to such a 1-channel image gives us the following image, whose pixel (i,j) is given by
$$\max_{-1\leq i',j'\leq 1} (ReLU(I \star K^l))_{i+i',j+j'}.$$

3. We have
$$K^1_1 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -1 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$
and
$$I_1 = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 3 & 5 & 7 & 9 \\ 4 & 7 & 10 & 13 \\ 5 & 9 & 13 & 17 \end{pmatrix},\ I_2 = \begin{pmatrix} 3 & 4 & 5 & 6 \\ 4 & 6 & 8 & 10 \\ 5 & 8 & 11 & 14 \\ 6 & 10 & 14 & 18 \end{pmatrix},$$
hence
$$(I \star K^1)_{1,1} = (-2+3+3) + (-6+4+4) = 6.$$
Similarly,
$$(I \star K^1)_{1,2} = (-3+2+4+5) + (-8+3+5+6) = 14,$$
$$(I \star K^1)_{2,1} = (-3+2+4+5) + (-8+3+5+6) = 14,$$
$$(I \star K^1)_{2,2} = (-5+3+3+7+7) + (-12+4+4+8+8) = 27,$$
etc.

4. In this case, all pixel values happen to be non-negative, hence applying ReLU doesn't change the values.

5. For the first coefficient, we have
$$\max_{-1\leq i',j'\leq 1} (ReLU(I \star K^l))_{1+i',1+j'} = \max(0,6,14,27) = 27.$$

Problem 4 (Getting Started with TensorFlow):

The solution is provided in the notebook `solution08.ipynb`. Here are the answers to the questions:

- The linear model fits `blobs` properly because the dataset is linearly separable.
- On the other hand, `circles` is not linearly separable, so a simple linear model is not expressive enough.
- Adding a hidden layer with non-linear activations (e.g. ReLU) allows modeling arbitrary distributions. One hidden layer with 10–100 neurons is enough for `circles`.
- SGD converges faster than batch GD because it exploits the redundancy of the data to approximate the gradient, resulting in more weight updates per epoch (epoch = single pass over the entire dataset) with a much lower computational cost. Moreover, the gradient noise allows the optimizer to escape sharp local minima and converge to wider minima (which has an implicit regularization effect, i.e. improves generalization). A drawback of SGD is that it reduces parallelism, which can be a problem for GPUs as they rely on parallel computations. This can however be tackled by using sufficiently large minibatches.