Exercise 6

Protein Prediction I for CS

Neural Networks in Numpy v1
Motivation

Why should we want to implement a neural network from scratch?

- High-level modules s.a. SciKit require specific I/O specifications & allow only certain architectures
- Parameters might be used without understanding their impact
- Demystifying the Black Box
Feed-Forward ANNs

Umbrella term for all networks which do not form cycles; include: Fully-connected MLPs, CNNs,..

⇒ Let’s break down the complexity to the easiest case: a single perceptron.
1. **Forward pass**: Let $x_1, x_2, \ldots, x_n$ be the input $X$ for one sample and $w_1, w_2, \ldots, w_n$ be the weight associated with each input node. Then the forward pass is defined as $OR = W \cdot X$

2. **Loss function**: E.g. Hinge loss $l(y) = \max(0, 1 - y \ast y_{\text{pred}})$

3. **Backward pass**: Use the backpropagation algorithm to update weights based on negative gradient of loss. Let $\mu$ be the learning rate and $\delta$ be the gradient of the loss function. Without activation function and only a single layer, the update rule simplifies to:

$$W_{\text{new}} = W_{\text{old}} - \mu \ast \delta \ast X$$
Why do we need a bias term?

Without bias:
\[ y(X) = W^*X \]

With bias:
\[ y(X) = W^*X + b \]
The XOR Gate and MLP

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>XOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>
Let $i$ be a linear node in network layer $k$: 
$$a_i^k = \sum w_{ij}^k x_j^k$$
with $x_j^k$ being inputs, $a$ the output and $w$ the weights.

In matrix form: $A^k = W^k X^k$

In the next layer, the outputs $a^k$ will become inputs, s.t. $x_j^{k+1} = a_j^k$; giving for the next layer:

$$a_i^{k+1} = \sum w_{ij}^{k+1} a_j^k = \sum w_{ij}^{k+1} w_{jm}^k x_m^k$$

In matrix form: $A^{k+1} = W^{k+1} X^{k+1} = (W^{k+1} W^k) X^k$

$\Rightarrow$ Multiple linear transformations (layers) can be replaced by a single linear transformation if no non-linearity (s.a. sigmoid or Rectified Linear Unit) is introduced in between these layers.
⇒ Introduce a non-linearity after each linear transformation to find non-linear solutions.

⇒ Remember to take derivative of the non-linearity into account when calculating the backward pass.
General update rule: $W_{ij} = W_{ij} - \mu * \delta_j * X_i$ with $X_i$ being the inputs from the previous layer (closer to the inputs) and $\delta_j$ being the error which has already been backpropagated until layer $j$ (closer to the output).

How to calculate $\delta$ for multiple layers?

- $\delta$ for output layer: $\frac{\partial C}{\partial g(a)} * \frac{\partial g(a)}{\partial (a)}$

- $\delta_i$ for all other layers: $(\sum_i w_{ij} \delta_j) \frac{\partial g(a_i)}{\partial (a_i)}$

With $C$ being the cost function, $g(a)$ being a non-linearity and $a$ being the output from the previous layer.

Please note that we do not use a non-linear function in the output layer which simplifies the $\delta$ expression for the output layer as $\frac{\partial g(x)}{\partial x}$ equals 1 for $g(x) = x$. 
1. Calculate update-rate (delta) based on derivative of Loss function.

2. Backpropagate update-rate ($\delta$) based on weights used during forward pass.

3. Backpropagate new update-rate to layers closer to the input.

4. Update weights based on $\delta$ at each layer, the learning rate $\eta$ and derivative of activation function.

Figure thankfully taken from: http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html, on 20.06.2018.
Thank you!

QUESTIONS?