Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

**Architecture** — The vocabulary around neural networks architectures is described in the figure below:

By noting \( i \) the \( i \)th layer of the network and \( j \) the \( j \)th hidden unit of the layer, we have:

\[
    z[i][j] = w[i][j]Tx + b[i][j] = w_j^i x + b_j^i
\]

where we note \( w, b, z \) the weight, bias and output respectively.

**Activation function** — Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:

<table>
<thead>
<tr>
<th>Sigmoid</th>
<th>Tanh</th>
<th>ReLU</th>
<th>Leaky ReLU</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(z) = 1 + e^{-z} )</td>
<td>( g(z) = \tanh(z) )</td>
<td>( g(z) = \max(0, z) )</td>
<td>( g(z) = \max(\epsilon z, z) ) with ( \epsilon \ll 1 )</td>
</tr>
</tbody>
</table>

where \( \epsilon \ll 1 \)
Cross-entropy loss — In the context of neural networks, the cross-entropy loss $L(z, y)$ is commonly used and is defined as follows:

$$L(z, y) = -[y \log(z) + (1-y) \log(1-z)]$$

Learning rate — The learning rate, often noted $\alpha$ or sometimes $\eta$, indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

Backpropagation — Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight $w$ is computed using chain rule and is of the following form:

$$\frac{\partial L(z, y)}{\partial w} = \frac{\partial L(z, y)}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w} = \frac{\partial L(z, y)}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial w}$$

As a result, the weight is updated as follows:

$$w \leftarrow w - \alpha \frac{\partial L(z, y)}{\partial w}$$

Updating weights — In a neural network, weights are updated as follows:

- **Step 1:** Take a batch of training data.
- **Step 2:** Perform forward propagation to obtain the corresponding loss.
- **Step 3:** Backpropagate the loss to get the gradients.
- **Step 4:** Use the gradients to update the weights of the network.

Dropout — Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability $p$ or kept with probability $1-p$.

**Convolutional Neural Networks**
**Convolutional layer requirement** — By noting $WW$ the input volume size, $FF$ the size of the convolutional layer neurons, $PP$ the amount of zero padding, then the number of neurons $NN$ that fit in a given volume is such that:

$$N=W-F+2PS+1 N=W-F+2PS+1$$

**Batch normalization** — It is a step of hyperparameter $γβy,β$ that normalizes the batch $(xi)$. By noting $μB,α2BμB,αB2$ the mean and variance of that we want to correct to the batch, it is done as follows:

$$xi⟵γxi−μB+α2Bγxi−μB+αB$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

**Recurrent Neural Networks**

**Types of gates** — Here are the different types of gates that we encounter in a typical recurrent neural network:

<table>
<thead>
<tr>
<th>Input gate</th>
<th>Forget gate</th>
<th>Gate</th>
<th>Output gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write to cell or not?</td>
<td>Erase a cell or not?</td>
<td>How much to write to cell?</td>
<td>How much to reveal cell?</td>
</tr>
</tbody>
</table>

**LSTM** — A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

*For a more detailed overview of the concepts above, check out the Deep Learning cheatsheets!* 

**Reinforcement Learning and Control**

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

**Definitions**

**Markov decision processes** — A Markov decision process (MDP) is a 5-tuple $(S,A,(Psa),γ,R)$ where:

- $SS$ is the set of states
- $AA$ is the set of actions
- $(Psa)(Psa)$ are the state transition probabilities for $s∈SS$ and $a∈AA$
- $γ∈[0,1]$ is the discount factor
- $R:S×A→R;R:S×A→R$ or $R:S→R;R:S→R$ is the reward function that the algorithm wants to maximize

**Policy** — A policy $ππ$ is a function $π:S→A$ that maps states to actions.

*Remark: we say that we execute a given policy $ππ$ if given a state $ss$ we take the action $a=π(s)a=π(s)$.*

**Value function** — For a given policy $ππ$ and a given state $ss$, we define the value function $VπVπ$ as follows:

$$Vπ(s)=E[R(s0)+γR(s1)+γ2R(s2)+...|s0=s,π]Vπ(s)=E[R(s0)+γR(s1)+γ2R(s2)+...|s0=s,π]$$

**Bellman equation** — The optimal Bellman equations characterizes the value function $Vπ∗Vπ∗$ of the optimal policy $π∗π∗$:

$$Vπ∗(s)=R(s)+maxa∈AY∑s′∈ESPsa(s′)Vπ∗(s′)Vπ∗(s)=R(s)+maxa∈AY∑s′∈ESPsa(s′)Vπ∗(s′)$$

*Remark: we note that the optimal policy $π∗π∗$ for a given state $ss$ is such that:*
\[ \pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} \gamma \mathbb{P}(s'|s,a)V(s) \]

**Value iteration algorithm** — The value iteration algorithm is in two steps:

1) We initialize the value:

\[ V^0(s) = 0 \]

2) We iterate the value based on the values before:

\[ V^{i+1}(s) = R(s) + \max_{a \in A} \left\{ \sum_{s' \in S} \gamma \mathbb{P}(s'|s,a)V^i(s') \right\} \]

**Maximum likelihood estimate** — The maximum likelihood estimates for the state transition probabilities are as follows:

\[ P(s|a,s') = \frac{\# \text{times took action } a \text{ in state } s \text{ and got to } s'}{\# \text{times took action } a \text{ in state } s} \]

**Q-learning** — Q-learning is a model-free estimation of \( Q \), which is done as follows:

\[ Q(s,a) \leftarrow Q(s,a) + \alpha [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \]

For a more detailed overview of the concepts above, check out the *States-based Models cheatsheets!*