Deep learning

Introduction to neural networks

Hamid Beigy

Sharif University of Technology

September 27, 2020
1. Brain
2. History of neural networks
3. Gradient based learning
4. Activation function
5. Deep feed-forward networks
6. Training feed-forward networks
7. Reading
Brain
Functions of different parts of Brain

As Figure 2-15 shows, the shape of the brainstem can be compared to the lower part of your arm held upright. The hindbrain is long and thick like your forearm, the midbrain is short and compact like your wrist, and the diencephalon at the end is bulbous like your hand forming a fist.

Each of these three major regions of the brainstem performs more than a single task. Each contains various subparts, made up of groupings of nuclei, that serve different purposes. All three regions, in fact, have both sensory and motor functions. However, the hindbrain is especially important in various kinds of motor functions, the midbrain in sensory functions, and the diencephalon in integrative tasks. Here we consider the central functions of these three regions; later chapters will contain more information about them.

The Hindbrain

The hindbrain, shown in Figure 2-16, controls various types of motor functions ranging from breathing to balance to the control of fine movements, such as those used in dancing. The most distinctive structure in the hindbrain is the cerebellum, which looks much like a cauliflower. Actually, the cerebellum is a separate structure lying above the rest of the hindbrain. In humans, it is one of the largest structures of the brain. As Figure 2-17 illustrates, the size of the cerebellum increases with the physical speed and dexterity of a species. Animals that move slowly (such as a sloth) have rather small cerebellums, whereas animals that can perform rapid, acrobatic movements (such as a hawk or a cat) have very large cerebellums. The cerebellum is apparently important in controlling complex movements.

### Cranial nerves

<table>
<thead>
<tr>
<th>Number</th>
<th>Cranial nerve</th>
<th>Name</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Olfactory</td>
<td>Smell</td>
<td>Sensation</td>
</tr>
<tr>
<td>2</td>
<td>Optic</td>
<td>Vision</td>
<td>Movement</td>
</tr>
<tr>
<td>3</td>
<td>Oculomotor</td>
<td>Eye movement</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Trochlear</td>
<td>Eye movement</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Trigeminal</td>
<td>Masticatory movements and facial sensation</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Abducens</td>
<td>Eye movement</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Facial</td>
<td>Facial movement and sensation</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Auditory</td>
<td>Vestibular hearing and balance</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Glossopharyngeal</td>
<td>Tongue and pharynx movement and sensation</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Vagus</td>
<td>Heart, blood vessels, viscera, movement of larynx and pharynx</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Spinal accessory</td>
<td>Neck muscles</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Hypoglossal</td>
<td>Tongue muscles</td>
<td></td>
</tr>
</tbody>
</table>
A neuron in a living biological system

- **Axon**
- **Cell body or Soma**
- **Nucleus**
- **Dendrite**
- **Synapses**
- **Axonal arborization**
- **Axon from another cell**
- **Synapse**

Signals are noisy "spike trains" of electrical potential

**Biological Background**

Diagram of a typical myelinated vertebrate motoneuron (source: Wikipedia, Ruiz-Villarreal 2007), showing the main parts involved in its signaling activity like the dendrites, the axon, and the synapses.
History of neural networks
1. The first model of a neuron was invented by McCulloch (physiologists) and Pitts (logician).
2. Inputs are binary.
3. This neuron has two types of inputs: Excitatory inputs (shown by a) and Inhibitory inputs (shown by b).
4. The output is binary: fires (1) and not fires (0).
5. Until the inputs summed up to a certain threshold level, the output would remain zero.
The McCulloch-Pitts Neuron

- The first mathematical model of a neuron (Warren McCulloch and Walter Pitts, 1943)
- Binary activation: fires (1) or not fires (0)
- Excitatory inputs: the $a_i$'s, and
- Inhibitory inputs: the $b_i$'s
- Unit weights and fixed threshold $\mu$
- Absolute inhibition

Computing with McCulloch-Pitts Neurons

Any task or phenomenon that can be represented as a logic function can be modelled by a network of MP-neurons

- $\{\text{OR, AND, NOT}\}$ is functionally complete
- Any Boolean function can be implemented using OR, AND and NOT
- Canonical forms: CSOP or CPOS forms
- MP-neurons, Finite State Automata
1. Problems with McCulloch and Pitts -neurons
   - Weights and thresholds are analytically determined (cannot learn them).
   - Very difficult to minimize size of a network.
   - What about non-discrete and/or non-binary tasks?

2. Perceptron solution.
   - Weights and thresholds can be determined analytically or by a learning algorithm.
   - Continuous, bipolar and multiple-valued versions.
   - Rosenblatt randomly connected the perceptrons and changed the weights in order to achieve learning.
   - Efficient minimization heuristics exist.
Perceptron (Frank Rosenblat (1958))

1. Let $y$ be the correct output, and $f(x)$ the output function of the network. Perceptron updates weights (Rosenblatt 1960)

$$w_j^{(t)} \leftarrow w_j^{(t)} + \alpha x_j (y - f(x))$$

2. McCulloch and Pitts’ neuron is a better model for the electrochemical process inside the neuron than the Perceptron.

3. But Perceptron is the basis and building block for the modern neural networks.
1. The model is same as perceptron, but uses different learning algorithm.

2. A multilayer network of Adaline units is known as a MAdaline.
1. Let $y$ be the correct output, and $f(x) = \sum_{j=0}^{n} w_j x_j$. Adaline updates weights

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + \alpha x_j (y - f(x))$$

2. The Adaline converges to the least squares error which is $(y - f(x))^2$. This update rule is in fact the stochastic gradient descent update for linear regression.

3. In the 1960's, there were many articles promising robots that could think.

4. It seems there was a general belief that perceptron could solve any problem.
1. Minsky and Papert published their book *Perceptrons*. The book shows that perceptrons could only solve linearly separable problems.

2. They showed that it is not possible for perceptron to learn an XOR function.

3. After *Perceptrons* was published, researchers lost interest in perceptron and neural networks.
The first layer is a hidden layer.
1. Optimization

- In **1969**, Bryson and Ho described proposed Backpropagation as a multi-stage dynamic system optimization method.
- In **1972**, Stephen Grossberg proposed networks capable of learning XOR function.

2. In **1980s**, the field of artificial neural network research experienced a resurgence.


4. In **2010**, we are now able to train much larger networks using huge modern computing power such as GPUs.
Gradient based learning
1. The goal of machine learning algorithms is to construct a model (hypothesis) that can be used to estimate $y$ based on $x$.

2. Let the model be in form of

   $$h(x) = w_0 + w_1 x$$

3. The goal of creating a model is to choose parameters so that $h(x)$ is close to $y$ for the training data, $x$ and $y$.

4. We need a function that will minimize the parameters over our dataset. A function that is often used is mean squared error,

   $$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2$$

5. How do we find the minimum value of cost function?
1. Gradient descent is by far the most popular optimization strategy, used in machine learning and deep learning at the moment.
2. Cost (error) is a function of the weights (parameters).
3. We want to reduce/minimize the error.
4. Gradient descent: move towards the error minimum.
5. Compute gradient, which implies get direction to the error minimum.
6. Adjust weights towards direction of lower error.
Gradient descent

Initial weight

Gradient

Global cost minimum

\[ J_{\text{min}}(w) \]
Gradient descent (Linear Regression)

1. We have the following hypothesis and we need fit to the training data

\[ h(x) = w_0 + w_1 x \]

2. We use a cost function such Mean Squared Error

\[ J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h(x_i) - y_i)^2 \]

3. This cost function can be minimized using gradient descent.

\[
\begin{align*}
    w_0^{(t+1)} &= w_0^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_0} \\
    w_1^{(t+1)} &= w_1^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_1}
\end{align*}
\]

\( \alpha \) is step (learning) rate.
Gradient descent (effect of learning rate)
Gradient descent (landscape of cost function)
Challenges with gradient descent

1. **Local minimim:** A local minimum is a minimum within some neighborhood that need not be (but may be) a global minimum.

2. **Saddle points:** For non-convex functions, having the gradient to be 0 is not good enough.
   Example: \( f(x) = x_1^2 - x_2^2 \) at \( x = (0, 0) \) has zero gradient but it is clearly not a local minimum as \( x = (0, \epsilon) \) has smaller function value. The point \( (0, 0) \) is called a saddle point of this function.
Challenges with gradient descent
Considering the following single neuron

\[ h(x) = f \left( \sum_{i=0}^{n} w_i x_i + b \right) \]

- Inputs: \( x_1, x_2, x_3 \)
- Weights: \( w_1, w_2, w_3 \)
- Bias: \( b = x_0 \)
- Activate function: \( f \)
- Output: \( h(x) \)
1. We want to train this neuron to minimize the following cost function

\[ J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h(x^i) - y^i)^2 \]

2. Considering the sigmoid activation function \( f(z) = \frac{1}{1+e^{-z}} \)

3. We want to calculate \( \frac{\partial J(w)}{\partial w_i} \)
1. We want to calculate \( \frac{\partial J(w)}{\partial w_i} \)

2. By using the chain rule, we obtain

\[
\frac{\partial J(w)}{\partial w_j} = \frac{\partial J(w)}{\partial f(z)} \times \frac{\partial f(z)}{\partial z} \times \frac{\partial z}{\partial w_j}
\]

\[
\frac{\partial J(w)}{\partial f(z^i)} = \frac{1}{m} \sum_{i=1}^{m} (f(z^i) - y^i)
\]

\[
\frac{\partial f(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = f(z)(1 - f(z))
\]

\[
\frac{\partial z}{\partial w_j} = x^j
\]

\[
w_j^{(t+1)} = w_j^{(t)} - \alpha \frac{\partial J(w)}{\partial w_j}
\]

\( \alpha \) is the learning rate.
1. We want to train this neuron to minimize the following cost function

$$J(w) = \sum_{i=1}^{m} \left[ -y^i \ln h(x^i) - (1 - y^i) \ln(1 - h(x^i)) \right]$$

2. Computing the gradients of $J(w)$ with respect to $w$, we obtain

$$\nabla J(w) = \sum_{i=1}^{m} y^i x^i (h(x^i) - y^i)$$

3. Updating the weight vector using the gradient descent rule will result in

$$w^{(t+1)} = w^{(t)} - \alpha \sum_{i=1}^{m} y^i x^i (h(x^i) - y^i)$$

$\alpha$ is the learning rate.
1. We talked about batch gradient descent (BGD) learning.

2. The batch update refers to the fact that the cost function is minimized based on the complete training data set.

3. We can update weights after each individual training sample.

4. Updating weights is also called stochastic gradient descent (SGD) because it approximates the gradient.

5. SGD versus BGD
1. Mini-batch gradient descent (MBGD) is a trade-off between SGD and BGD.

2. In MBGD, the cost function (and therefore gradient) is averaged over a small number of samples, from around 10-500.

3. This is opposed to the SGD batch size of 1 sample, and the BGD size of all the training samples.

4. Benefits of MBGD
   - It smooths out some of the noise in SGD.
   - The mini-batch size is small and keeps the performance benefits of SGD.
Mini-batch gradient descent (comparison)
1. If $\alpha$ is too high, the algorithm diverges.
2. If $\alpha$ is too low, makes the algorithm slow to converge.
3. A common practice is to make $\alpha_k$ a decreasing function of the iteration number $k$. e.g.

$$\alpha_k = \frac{c_1}{k + c_2}$$

where $c_1$ and $c_2$ are two constants.
4. The first iterations cause large changes in the $w$, while the later ones do only fine-tuning.
1. SGD with momentum remembers the update $\Delta w$ at each iteration\(^1\).

2. Each update is as a (convex) combination of the gradient and the previous update.

\[
\Delta w^{(k)} = \alpha_k \nabla^{(k)} J(w) + \beta \Delta w^{(k-1)}
\]

\[
w^{(k)} = w^{(k)} - \Delta w^{(k)}.
\]

3. A common practice is to make $\alpha_k$ a decreasing function of the iteration number $k$. e.g.

\[
\alpha_k = \frac{c_1}{k + c_2}
\]

where $c_1$ and $c_2$ are two constants.

4. The first iterations cause large changes in the $w$, while the later ones do only fine-tuning.

Activation function
Properties of identity activation function

1. Output of this functions will not be confined between any range.

2. It doesn’t help with the complexity or various parameters of usual data that is fed to the neural networks.

3. It doesn’t increase the complexity of hypothesis space of neural network
Sigmoid activation function

Properties of sigmoid activation function

1. The sigmoid function is in interval $(0, 1)$.
2. It is used to predict the probability as an output.
3. The function is differentiable.
4. The function is monotonic but its derivative is not.
5. This function can cause a neural network to get stuck at the training time.
Hyperbolic tangent activation function

Properties Hyperbolic tangent activation function

1. The Tanh function is in interval \((-1, 1)\).
2. It is used for classification of two classes.
3. The function is differentiable.
4. The function is monotonic but its derivative is not.
5. This function can cause a neural network to get stuck at the training time.
6. Both tanh and logistic sigmoid activation functions are used in feed-forward nets.
Properties Rectified linear unit (ReLU)

1. The ReLU is the most used activation function in the world right now.
2. The function is differentiable except at the origin.
3. The function and its derivative both are monotonic
4. All the negative values become zero immediately which decreases the ability of the model to train from the data properly.
Leaky ReLU activation function

Properties Leaky

1. The leaky ReLU helps to increase the range of the ReLU function.

2. Usually, the value of $a$ is 0.01. $a$ is the slope of the negative part.

3. When $a \neq 0.01$, then it is called Randomized ReLU.

4. Both Leaky and Randomized ReLU functions are monotonic in nature. Also, their derivatives are monotonic in nature.
Deep feed-forward networks
Deep feed-forward networks
Deep feed-forward networks
1. What is the decision surface of perceptron?
1. What is the network structure for the following decision surface?
Designing network for more complex decision boundaries

- Build a network of units with a single output that fires if the input is in the coloured area.

Can now be composed into "networks" to compute arbitrary classification boundaries.

![Diagram of a network and decision boundary](image)
Can you build such region with one hidden layer network?
1. What is the topology of network for the given problem?
2. Can we build a network to create every decision boundary?
3. Neural networks are universal approximators.
4. Can we build a network without local minimia in cost function?
Training feed-forward networks
1. Specifying the topology of network and the cost function
   - #-layers
   - #-nodes in each layer
   - function of each node
   - activation of each node

2. We use gradient decent algorithm for training the network.

3. But, we don’t have the true output of each hidden unit.
Reading
1. Chapter 6 of Deep Learning Book\textsuperscript{2}


Questions?