CENG 501
Deep Learning
Week 3

Sinan Kalkan

https://xkcd.com/1838/
What is machine learning?

$x \in \mathbb{X}$

$y = f(x)$

$y \in \mathbb{Y}$
Outline for the Machine Learning Part

- Supervised Learning
- Unsupervised Learning
- Other Forms of Learning
- General Issues in Learning
- Evaluation
Neuron

- The pulses generated by the neuron travels along the axon as an electrical wave.
- Once these pulses reach the synapses at the end of the axon open up chemical vesicles exciting the other neuron.

Slide credit: Erol Sahin
Face selectivity in IT

http://www.billconnelly.net/?p=291
Fig. 2. Bain's diagram illustrating the way in which the connections in a neural network can channel activation in different directions.

It requires us to assume, not merely fibres multiplying by ramification through the cell junctions, but also an extensive arrangement of cross connections. We can typify it in this way. Suppose $a$ enters a cell junction, and is replaced by several branches, $a', a''$ etc; $b$ in like manner, is multiplied into $b', b''$ etc. Let one of the branches of $a$ or $a'$ pass into some second cell, and a branch of $b$, or $b'$, pass into the same, and let one of the emerging branches be $X$; we have then a means of connecting united $a$ and $b$ with $X$; and in some other crossing, a branch of $b$ may unite with a branch of $c$, from which crossing $Y$ emerges and so on. . . . By this plan we comply with the primary condition of assigning a separate outcome to every different combination of sensory impressions.

The diagram shows the arrangement. The fibre $a$ branches into two $a', a''$: the fibre $b$ into $b', b''$: $c', c''$. One of the branches of $a$ unites with one of the branches of $b$, or $a', a''$ in a cell $X$; $b', b''$ unite in $Y$; $c', c''$ unite in $Z$. (1873, pp. 110, 111)
McCulloch-Pitts Neuron

• Implement AND($x, y$):
  • Let $w_x$ and $w_y$ to be 1, and $w_{+1}$ to be -2.
  • When input is 1 & 1; net is 0.
  • When one input is 0; net is -1.
  • When input is 0 & 0; net is -2.

\[
f(\text{net}) = \begin{cases} 
0, & \text{net} < 0 \\
1, & \text{net} \geq 0 
\end{cases}
\]
THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN

F. ROSENBLATT
Cornell Aeronautical Laboratory

Fig. 1. Organization of a perceptron.
Perceptron training rule

Update weights

\[ w_i \leftarrow w_i + \Delta w_i \]

• How to determine \( \Delta w_i \)?

\[ \Delta w_i \leftarrow \eta(y - \hat{y})x_i \]

• \( \eta \): learning rate – can be slowly decreased

• \( y \): target/desired output

• \( \hat{y} \): current output, prediction
Linear classification

For class $j$: $f_j = f(x; W, b)_j = w_j \cdot x + b_j = \sum_{i=1}^{N} w_{ji} x_i + b_j$

For all classes: $f(x; W, b) = Wx + b$

Figure: http://cs231n.github.io/linear-classify/

One row per class

Sinan Kalkan
Hinge Loss, or Max-Margin Loss

\[
L = \frac{1}{N} \sum_{i} \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) + \lambda \sum_{i} \sum_{j} W_{i,j}^2
\]

http://cs231n.github.io/linear-classify/
Today

- Continue with linear classification & regression
  - MSE as a regression loss
  - Gradient descent
  - Calculating the gradients

- Non-linear classification & regression
- Multi-layer perceptrons
- Backpropagation
Administrative issues

• Project paper selection (15 April)

https://docs.google.com/spreadsheets/d/1u_4cYR5OFpJIwd8IXpH_SVLQuOh9i-Fgm6prz7eRdTs/edit?usp=sharing

• Midterm exam:
  • Tentative date: The week of 11-15 April.

• Programming Assignment 1 (PA1):
  • Tentative date: 28 March – 15 April.
Interactive Demo

http://vision.stanford.edu/teaching/cs231n/linear-classify-demo/
Regression loss

\[ L = \frac{1}{N} \sum_i \sum_j (s_{ij} - y_{ij})^2 + \lambda \sum_i \sum_j w_{i,j}^2 \]

In general:

\[ \sum_j |s_j - y_j|^q \]

• \( q = 1 \): Absolute Value Loss
• \( q = 2 \): Square Error Loss.

Figure 1.29  Plots of the quantity \( L_q - (y - \hat{y})^q \) for various values of \( q \).  Bishop
\[ L(x; \theta) = d(y, f(x; \theta)) \]

\[ \theta^* = \arg\min_{\theta} L(x; \theta) \]
\[ \theta^* = \arg \min_{\theta} L(x; \theta) \]

\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_{\theta} L(x; \theta) \]
\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L(x; \theta) \]

Learning Rate (Step Size)

\[ \nabla_\theta L(x; \theta) = \frac{\partial L(x; \theta)}{\partial \theta} \]

\[ t = \sum w_i x_i \rightarrow g(t) \rightarrow y \]
Gradient Descent

• True Gradient Descent
  • Calculate the loss & the gradient on the whole dataset
  • Then make the update

• Stochastic Gradient Descent
  • Calculate the loss & the gradient on examples one at a time
  • Update the weights after each example

• Batch Gradient Descent
  • Calculate the loss & the gradient on a set of examples (batch)
  • Update the weights after each bath
Gradient Descent

**Input:** Training set: \( \{(x_i, y_i)\}, \ i = 1, \ldots, N \)
- The network architecture.

**Output:** Network parameters, \( \theta \)

1. \( \theta_0 \leftarrow \) Random initial values
2. Until convergence:
   i. Take \( m \) samples from the dataset randomly
   ii. Calculate predictions, \( \hat{y} \), on \( m \) samples using the current parameters \( \theta_t \)
   iii. Calculate loss \( L() \) and \( \nabla_\theta L \)
   iv. Update the weights
      \[
      \theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L
      \]
Gradient descent

https://en.wikipedia.org/wiki/Gradient_descent

(Goodfellow vd., 2016)
Derive the gradients of hinge loss

$$\frac{\partial L_i}{\partial w_{mk}} = ?$$

$$\frac{\partial L_i}{\partial w_{mk}} = \frac{\partial L_i}{\partial e_m} \frac{\partial e_m}{\partial s_m} \frac{\partial s_m}{\partial w_{mk}}$$

$$\frac{\partial L_i}{\partial w_{mk}} = \mathbb{I}(s_m - s_{y_i} + \Delta > 0) \mathbb{I}(s_m - s_{y_i} + \Delta > 0) x_{ik}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

This assumed that $m \neq y_i$. What happens if that’s not the case? See the next page.
Derive the gradients of hinge loss

\[
\frac{\partial L_i}{\partial w_{y_i k}} = ?
\]

\[
\frac{\partial L_i}{\partial w_{y_i k}} = \sum_{j \neq y_i} \frac{\partial L_i}{\partial e_j} \frac{\partial e_j}{\partial s_{y_i}} \frac{\partial s_{y_i}}{\partial w_{y_i k}}
\]

\[
\begin{align*}
1 \\
\mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)
\end{align*}
\]

\[
\frac{\partial L_i}{\partial w_{y_i k}} = \sum_{j \neq y_i} \mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)x_{ik}
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)
\]

\[
L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) - \mathbb{I}(s_j - s_{y_i} + \Delta > 0)(-1)\]

Sinan Kalkan
Non-linear Classification/Regression
\[ t = \sum w_i x_i + b \]

\[ y = f(x) = f(x; \theta) \]

\[ = g \left( \sum_i w_i x_i + b \right) = g(w \cdot x) \]
Multi-layer Perceptrons
\[ a_1 = g(W_0 x^T) \quad a_2 = g(W_1 a_1^T) \]

\[ a_t = g(W_{t-1} a_{t-1}^T) \]

\[ f() = g \left( g \left( g \left( g(\ldots) \right) \right) \right) \]
\[ L(\mathbf{x}; \theta) = d(\mathbf{y}, f(\mathbf{x}; \theta)) \]

\[ \theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L(\mathbf{x}; \theta) \]

\[ \theta^* = \arg \min_\theta L(\mathbf{x}; \theta) \]

\[ \nabla_\theta L(\mathbf{x}; \theta) = \frac{\partial L(\mathbf{x}; \theta)}{\partial \theta} \]
\[
\frac{\partial L(x; \theta)}{\partial w_{ji}} = \frac{\partial L}{\partial \hat{y}_m} \frac{\partial \hat{y}_m}{\partial g_{mi}} \frac{\partial g_{mi}}{\partial t_m} \frac{\partial t_m}{\partial w_{mi}}
\]

Backpropagation
\[
\frac{\partial L(x; \Theta)}{\partial a^l_i} = \frac{\partial L(x; \Theta)}{\partial a^l_i} \frac{\partial a^l_i}{\partial w_{ij}}
\]

\[
\frac{\partial L(x; \Theta)}{\partial a^{l-1}_i} = \sum_j \frac{\partial L(x; \Theta)}{\partial a^l_j} \frac{\partial a^l_j}{\partial a^{l-1}_i}
\]
Importance of increasing layers

• Continuous functions:
  • Every bounded continuous function can be approximated with small error with two layers

• Arbitrary functions:
  • Three layers can approximate any arbitrary function

• Why do we need deep layers then?
  • If the problem has a hierarchical nature, more layers yield better performance
  • Lin vd., “Why does deep and cheap learning work so well?”, 2017.

Importance of increasing layers

Multi-layered Fully-connected Networks

• To be able to have solutions for linearly non-separable cases, we need a non-linear and differentiable unit, e.g.:
  \[ \hat{y} = \sigma(w \cdot x) \]

where

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

- Sigmoid (logistic) function
- Output is in range (0,1)
- Since it maps a large domain to (0,1) it is also called squashing function
- Alternatives: tanh
Multi-layered Networks

Derivative of the sigmoid:

\[
\frac{d\sigma(x)}{dx} = \frac{d}{dx} \left( \frac{1}{1 + e^{-x}} \right) = \frac{0 \cdot (1 + e^{-x}) - 1 \cdot (-e^{-x})}{(1 + e^{-x})^2} = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} = \frac{1}{1 + e^{-x}} \cdot (1 - \frac{1}{1 + e^{-x}}) = \sigma(x) \cdot (1 - \sigma(x))
\]
A neuron with sigmoid function

\[ \hat{y} = \sigma(\text{net}) = \frac{1}{1 + e^{-\text{net}}} \]
backpropagation
Why do we need to learn backpropagation?

• “Many frameworks implement backpropagation for us, why do we need to learn?”
  • This is not a blackbox. There are many problems/issues involved. You can only deal with them if you have a good understanding of backpropagation.

https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b#.7zawffou2
Black Magic in Deep Learning: How Human Skill Impacts Network Training

Kanav Anand\textsuperscript{1}  
anandkanav92@gmail.com  
Ziqi Wang\textsuperscript{1}  
z.wang-8@tudelft.nl  
Marco Loog\textsuperscript{1,2}  
M.Loog@tudelft.nl  
Jan van Gemert\textsuperscript{1}  
c.vangemert@tudelft.nl

\textsuperscript{1}Delft University of Technology, Delft, The Netherlands  
\textsuperscript{2}University of Copenhagen, Copenhagen, Denmark

Abstract

How does a user’s prior experience with deep learning impact accuracy? We present an initial study based on 31 participants with different levels of experience. Their task is to perform hyperparameter optimization for a given deep learning architecture. The results show a strong positive correlation between the participant’s experience and the final performance. They additionally indicate that an experienced participant finds better solutions using fewer resources on average. The data suggests furthermore that participants with no prior experience follow random strategies in their pursuit of optimal hyperparameters. Our study investigates the subjective human factor in comparisons of state of the art results and scientific reproducibility in deep learning.
The Model

Hidden activations:
\[ h_{ij} = \sigma(W^h_j \cdot x_i) = \sigma(net^h_{ij}) \]

Output layer:
\[ \hat{y}_ic = \sigma(W^o_c \cdot h_i) = \sigma(net^o_{ic}) \]

The loss function:
\[ L(\theta) = \frac{1}{2} \sum_{i=1}^N \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]
- For one sample:
\[ L_i(\theta) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]
Backpropagation

For each output unit $c$, calculate its grad term $\delta_c^o$:

$$\delta_c^o = \frac{\partial L_i}{\partial \text{net}_c^o} = \frac{\partial L_i}{\partial \hat{y}_{ic}} \frac{\partial \hat{y}_{ic}}{\partial \text{net}^o_{ic}} = (\hat{y}_{ic} - y_{ic})\hat{y}_{ic}(1 - \hat{y}_{ic})$$

For each hidden unit $j$, calculate its grad term $\delta_j^h$:

$$\delta_j^h = \frac{\partial L_i}{\partial \text{net}_j^h} = \frac{\partial L_i}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial \text{net}^h_{ij}} = \left(\sum_{c \in C} \frac{\partial L_i}{\partial \text{net}^o_{ic}} \frac{\partial \text{net}^o_{ic}}{\partial h_{ij}}\right) h_{ij} (1 - h_{ij})$$

$$= (\sum_{c \in C} \delta_{ic}^o w_{cj}) h_{ij} (1 - h_{ij})$$

Update weight $w_{jk}^o$ in the output layer:

$$w_{jk}^o = w_{jk}^o - \eta \delta_{ij}^o h_{ik}$$

Update weight $w_{jk}^h$ in the hidden layer:

$$w_{jk}^h = w_{jk}^h - \eta \delta_{ij}^h x_{ik}$$

The Model

Hidden activations: $h_{ij} = \sigma(w_{ij}^h \cdot x_i) = \sigma(net_{ij}^h)$

Output layer: $\hat{y}_{ic} = \sigma(w_{ic}^o \cdot h_i) = \sigma(net_{ic}^o)$

The loss function:

$$L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$

- For one sample:

$$L_t(\theta) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2$$

Sinan Kalkan
Derivation of backpropagation

Derivation of the output unit weights

\[ \Delta w^o_{ck} = -\eta \frac{\partial L_i}{\partial w^o_{ck}} \]

1. Derivative of sigmoid:
\[ \hat{y}_{ic}(1 - \hat{y}_{ic}) \]

2. Derivative of the output layer:
\[ h_{ik} \]

3. For one sample:
\[ L_i(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]

The loss function:
\[ L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]

Sinan Kalkan
Derivation of backpropagation

Derivation of the hidden unit weights

\[ \Delta w_{jk}^h = -\eta \frac{\partial L_i}{\partial w_{jk}^h} \]

\[ \frac{\partial L_i}{\partial w_{jk}^h} = \frac{\partial L_i}{\partial \text{net}_{ij}^h} \frac{\partial \text{net}_{ij}^h}{\partial w_{jk}^h} \rightarrow x_{ik} \]

\[ \frac{\partial L_i}{\partial \text{net}_{ij}^h} = \left( \sum_c \frac{\partial L_i}{\partial \hat{y}_{ic}^c} \frac{\partial \hat{y}_{ic}^c}{\partial \text{net}_{ij}^d} \right) \frac{\partial \text{net}_{ij}^d}{\partial h_{ij}} \frac{\partial h_{ij}}{\partial \text{net}_{ij}^h} \]

\[ = \left( \sum_c \delta_{ic}^o \cdot w_{cj} \right) h_{ij}(1 - h_{ij}) \]

\[ \Delta w_{jk}^h = -\eta \frac{\partial L_i}{\partial w_{jk}^h} = -\eta \left( \sum_c \delta_{ic}^o \cdot w_{cj} \right) h_{ij}(1 - h_{ij}) x_{ik} = -\eta \delta_{ij}^h x_{ik} \]

The Model

Hidden activations: \( h_{ij} = \sigma(w_j^h \cdot x_i) = \sigma(\text{net}_{ij}^h) \)

Output layer: \( \hat{y}_{ic} = \sigma(w_{ci}^o \cdot h_i) = \sigma(\text{net}_{ic}^o) \)

The loss function:

\[ L(\theta) = \frac{1}{2} \sum_{i=1}^{N} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]

- For one sample:

\[ L_i(\theta) = \frac{1}{2} \sum_{c \in C} (\hat{y}_{ic} - y_{ic})^2 \]

Sinan Kalkan
Forward pass

```python
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

https://cs231n.github.io/
Backward pass

\[
\text{loss} = 0.5 \cdot \text{np.sum}((\text{out} - \text{y})^2)
\]
\[
\text{dout} = (\text{out} - \text{y})
\]
\[
\text{dW3} = \text{np.dot}((\text{out} - \text{y}), \text{h2.T})
\]

...
Backpropagation vs. numerical differentiation

What are their complexities?

• Backpropagation:
  • $O(|\theta|)$

• Numerical differentiation
  • $O(|\theta|^2)$
Neural Engineering

• Loss functions
• On optimization
• Activation functions
• Capacity, convergence
• Preprocessing
• ...