Overview

- Introduction Deep Learning
- Feedforward Neural Networks
General Introduction Deep Learning
Machine Learning Refresher

- Statistical machine learning algorithms rely on human-crafted representations of raw data and hand-designed features (e.g. POS tag, previous word, next word, TF-IDF, character n-gram, etc.)
- Main goal is to discover a mapping from the representation to the output
- Optimization of weights in the target function to optimize final prediction
- Most of the time goes into finding the optimal features for your task and optimizing the parameter settings of your algorithm
Features: Refreshing Example

Design matrix of the famous Iris dataset (flowers); S = Sepal; P = Petal

<table>
<thead>
<tr>
<th>Example</th>
<th>S.Length</th>
<th>S.Width</th>
<th>P.Length</th>
<th>P.Width</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>2</td>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>3</td>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4</td>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>6.2</td>
<td>3.4</td>
<td>5.4</td>
<td>2.3</td>
<td>virginica</td>
</tr>
<tr>
<td>150</td>
<td>5.9</td>
<td>3.0</td>
<td>5.1</td>
<td>1.8</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Dagmar Gromann, 18 June 2018
Curse of Dimensionality

Arises when analyzing data in high-dimensional space; the number of possible distinct configurations of a set of variables increases exponentially as the number of variables increases.

Deep learning algorithms are designed to overcome this and several other obstacles of traditional machine learning.
Deep Learning

- A subfield of machine learning
- Representation learning attempts to automatically learn good features from raw data
- Multiple levels of representations (here: 3 layers)
- Builds complex representations from simpler ones

Representation Learning
Brief History

- **Speech recognition breakthrough in 2010**

- **Computer vision breakthrough in 2012 (halving the error rate to 16%)**

- **Today: vast amounts of papers published on a daily basis; some help:** [http://www.arxiv-sanity.com/](http://www.arxiv-sanity.com/)
Application Examples of Deep Learning

- Neural Machine Translation
- Text summarization
- Text generation preserving the style
- Linguistic analysis (syntactic parsing, semantic parsing, etc.)
- ...

Dagmar Gromann, 18 June 2018
Image Captioning

Automatically generating image descriptions:

Lip Sync from Audio

Winning Atari Breakout

Google’s DeepMind learns to play the Atari game Breakout without any prior knowledge based on deep reinforcement learning.
Google’s CEO presented the future feature of Google’s Assistant at its I/O developer meeting in 2018. The assistant is able to make calls and arrange appointments, reserve tables, etc. for you. (Demo video)
Popular Software Libraries

- TensorFlow
- PyTorch
- PyLearn
- Theano
- Caffe
- MXNet
- many more
Feedforward Neural Networks
Feedforward Neural Networks

- approximate a function $f^*$
- defines a mapping $y = f(x, \theta)$ and learns the value of $\theta$ that represents the best function approximation
- feedforward means information flows from function being evaluated from $x$ through intermediate computation to define $f$ to output $y$
- no feedback connections from output that are fed back
- with feedback connections = recurrent neural network
- special kind of feedforward: Convolutional Neural Networks (CNN)
Why “network”? 

- compose together many different activation functions (directed, acyclic graph) 

- \( f_w \): network of basic units (e.g. three functions \( f_1, f_2, f_3 \) connected in a chain to form \( f(x) = f_3(f_2(f_1(x))) \))

- \( f_1 \) = first layer, \( f_2 \) = second layer, etc.; depth = length of chain (here: 3)

- training examples specify what the output layer must do for each point \( x \)

- training algorithm decides itself how to use the layers in between input and output layer to approximate the desired result \( y \); outputs of those layers not shown = hidden layers

- dimensionality of hidden layer = width of the model
Why “neural”? 

- inspired by neuroscience
- neurons: highly connected and perform computations by combining signals from other neurons
- deep learning: densely interconnected set of simple units (e.g. sigmoid units depending on activation function computed based on inputs from other units)
- vector-based representation:
  - $X$ features space: (vector of) continuous or discrete variables
  - $Y$ output space: (vector of) continuous or discrete variables
  - many layers of vector-valued representations, each unit: vector-to-scalar function
- more function approximation machine than model of the brain
Classical Neural Language Model


Dagmar Gromann, 18 June 2018

Semantic Computing
Main Design Decisions

Choosing the:

- optimizer
- cost function
- form of output units
- architecture design:
  - number of layers
  - type of connection between layers
  - number of units in each layer
Optimizer
Optimization

Most deep learning algorithms involve optimization.

Minimizing or maximizing some objective function $f(x)$ by altering $x$; when minimizing, the function is also called cost function, loss function, or error function; value that minimizes or maximizes a function usually denoted with $*$, e.g. $x^*$

How to optimize? Take the derivative $f'(x)$ because it gives you the slope of $f(x)$ at the point $x$; specifies which small change is needed in the input to obtain a small improvement in the output

List of optimizers and links: https://keras.io/optimizers/
Gradient Descent

Global minimum at $x = 0$. Since $f'(x) = 0$, gradient descent halts here.

For $x < 0$, we have $f'(x) < 0$, so we can decrease $f$ by moving rightward.

For $x > 0$, we have $f'(x) > 0$, so we can decrease $f$ by moving leftward.

$\begin{align*}
  f(x) &= \frac{1}{2} x^2 \\
  f'(x) &= x
\end{align*}$

Gradient

Functions with multiple inputs require partial derivatives. The gradient of $f$ is a vector containing all the partial derivatives denoted $\nabla_x f(x)$.

1: Set initial parameters $\theta_1^0, \ldots, \theta_k^0$
2: $\epsilon = $ learning rate (step size)
3: while not converged calculate $\nabla f$ do
4: $\theta_1 = \theta_1 - \epsilon \frac{\partial f}{\partial \theta_1}$.
5:  ...  
6: $\theta_k = \theta_k - \epsilon \frac{\partial f}{\partial \theta_k}$
7: end while
8: small enough $\epsilon$ ensures that $f(\theta_1^i, \ldots, \theta_k^i) \leq f(\theta_1^{i-1}, \ldots, \theta_k^{i-1})$
Stochastic Gradient Descent (SGD)

- extension of gradient descent to reduce computational costs of large training sets
- We have a loss function $L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L(x_i, y_i, \theta)$, what we want to minimize is the expected loss:
  $$J(\theta) = \mathbb{E}_{x_i, y_i} L(x, y, \theta)$$
- Gradient descent requires computing
  $$\nabla_{\theta} J(\theta) = \frac{1}{m} \nabla_{\theta} L(x_i, y_i, \theta)$$
- Alternative: approximately estimate the expectation by randomly picking a small set of samples
- Estimate of the gradient is formed as ($m'$ is the small set of samples):
  $$g = \frac{1}{m'} \nabla_{\theta} \sum_{i=1}^{m'} L(x_i, y_i, \theta)$$
SGD continued

- needs more steps than GD but each step is cheaper (computation)
- there is not always a decrease for each step
- **minibatch:**
  - common way to uniformly draw examples from the training data
  - usually very small between one and a few hundred examples
  - batch size is held fixed even with an increasing training set size
- SGD outside of deep learning: e.g. main way to train linear models on large datasets
Cost function
Cost function

- similar to linear models: our model defines a distribution \( p(y|x; \theta) \) and we use the principle of maximum likelihood (negative log-likelihood)
- cost function = cross-entropy between training data and predictions
- or estimate of that cost function
- often combined with regularization term
Form of Output Units
Form of Output Units

The type of unit is defined by its activation function; Activation functions decide whether a neuron should be activated ("fire") or not, that is, whether the received information is relevant or should be ignored (= nonlinear transformation over the input signal)

- Sigmoid
- Tanh
- Rectified Linear Unit (ReLU)
- Leaky ReLU
- Softmax
- ...
Sigmoid Unit

- sigmoid function: $\sigma(x) = \frac{1}{1 + e^{-x}}$
- non-linear activation function, which means the output is non-linear as well (0 to 1 in the S-Shape)
- $g(x) = \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$ => high between values of -3 and 3: in this range small changes of $x$ would bring about large changes of $y$ (highly desirable property)
- with non-linear functions we can backpropagate (update weights) and have several layers (no difference with linear function)
Rectified Linear Unit (ReLU)

- function: \( f(x) = \text{max}(0, x) \)
- not all neurons are activated at the same time
- if input is negative => conversion to zero and neuron inactive
- advantage: more efficient computation
Architecture Design
Architecture Design

Architecture here means how many units and how are those units interconnected.

- usual organization:
  - groups of units called **layers**
  - layers are organized in a chain structure

- first layer: \( h_1 = g_1(W_1^T x + b_1) \)
- second layer: \( h_2 = g_2(W_2^T h_1 + b_2) \)
- ...

- **depth**: number of layers in the network
- **width**: number of units per layer
Units and Layers

• feedfoward network with a single layer is sufficient to represent any function
• but: might make the layer very large and fail to generalize correctly
• deeper networks can reduce the number of units needed to represent the desired function
• and can reduce the amount of generalization error
• implicit belief of deep networks: function we want to learn is the composition of simpler functions
Interconnection

- fully connected: (linear) transformation via the weight matrix $W$ where each input unit is connected to every output unit
- reducing the number of connections, reduces the number of parameters and the amount of computations
- more later in the lecture
In Keras

```python
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(12, activation='relu', input_shape=(11,)))
# other parameter: input_dim=100
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
model.compile(loss='binary_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
#Alternative specification
#model.compile(loss=keras.losses.binary_crossentropy,
#              optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
model.fit(X_train, y_train, epochs=5, batch_size=32)
y_pred = model.predict(X_test)
```

Keras documentation: https://keras.io/
Review of Lecture 9

- How does deep learning differ from statistical machine learning?
- What are some typical application scenarios of deep learning?
- Why is it called a network? And why neural?
- Which optimizer do you know and how does it work?
- What other design decisions are central in deep learning?
- How do you choose the correct number of layers?