Deep Learning for Sequences
Lecture 1
David Kriegman
Computer Science & Engineering
UCSD

The Four Big Trends
...as seen by a CV Researcher

- Convolution Neural Networks (CNN)
- Recurrent Neural Networks (RNNs, LSTMs, GRUs)
- Generalized Adversarial Networks (GANs)
- Reinforcement Learning

Vision & Learning Got Hot

CVPR Attendance

Based on titles of conference papers. Source Jordi Pont-Tuset's Blog
Sequences

- Sequence: Ordered list of elements
- Infinite/finite
- Fixed or variable length
- Examples
  - Discrete samples of an electric signal
  - Characters in poem
  - ... To the blackboard for more.
- Characteristics of the sequences
  - Sampled "continuous" values (temperature, share price, audio, etc.)
  - Sampled uniformly in time (or not)
  - Discrete values from a small set of possibilities (e.g., Roman characters 10's)
  - Discrete values from a very large set of possibilities (e.g., words 100k)
  - High dimensional data (e.g., images)

A View of Sequence Processing

- Notation:
  - Input sequence of length $T_x$ is given by $x = (x_1, x_2, \ldots, x_{T_x})$
  - Output sequence of length $T_y$ is given by $y = (y_1, y_2, \ldots, y_{T_y})$
  - General sequence processing: A function $f$
    $$y = f(x)$$
  - Causal sequence processing: A function $g$ that is applied repeatedly for $t=1,\ldots,T_y$
    $$y_t = g(x_1, x_2, \ldots, x_t, y_1, y_2, \ldots, y_{t-1})$$
  - Deterministic causal sequence processing: A function $g'$
    $$y_t = g'(x_1, x_2, \ldots, x_t)$$
  - Recursive causal sequence processing: introduce a new sequence (state): $s_1, \ldots, s_{T_x}$ and two update equations $h_1$ and $h_2$
    $$y_t = h_1(s_{t-1}, x_t)$$
    $$s_t = h_2(s_{t-1}, x_t)$$

How are sequences processed?

- Digital signal processing
- Finite state machines
- Learning automata
- Markov models
- Hidden Markov models
- Neural networks
Kalman Filter

• A Kalman filter is an example of recursive causal sequence processing:
  • As a state it maintains a mean vector and covariance matrix
  • It uses an update equations with the Kalman gain to update the mean and covariance given a new measurement.
  • The output is just the mean vector (or a linear function of the main vector)

Input/Output Sequences

• Input & Sequence elements may be different
  • Input length: 0, 1 or arbitrary length \( T_x \)
  • Output length: 1 or arbitrary length \( T_y \)
  • Input length \( T_x \) and output length \( T_y \) do not have to be the same.

Supervised Learning of Sequence Models

• Training sample is a pair \( <x^{(i)}, y^{(i)}> \)
  • Given a training set \( \{<x^{(1)}, y^{(1)}>, <x^{(2)}, y^{(2)}>, \ldots, <x^{(N)}, y^{(N)}>\} \), estimate (learn) the appropriate function (model) \( f \) or \( g \) or \( h_1, h_2 \).
  • Different network architectures can be viewed in these categories with RNN's being an example of a recursive method.

Sequence Modeling Examples

• Music generation: none or context vector \( \rightarrow \) Audio, \( (T_x = 0,1) \)
• Speech recognition: Audio sequence \( \rightarrow \) word sequence \( (T_x = T_y) \)
• Sentiment classification: text sequence \( \rightarrow \) value (binary, score, stars) \( (T_y = 1) \)
• Machine translation: text sequence \( \rightarrow \) text sequence \( (T_x = T_y) \)
• Video activity recognition: sequence of video frames \( \rightarrow \) Category \( (T_y = 1) \)
• Named entity recognition: text sequence \( \rightarrow \) entity sequence \( (T_x = T_y) \)
• Prediction: Sequence \( \rightarrow \) Next element \( (T_y = 1) \)
  • Could also be interpolation rather than extrapolation
  • Applied recursively
  • Image captioning: Image \( \rightarrow \) Text sequence \( (T_x = 1) \)
Sentiment classification

Movie Reviews
- unbelievably disappointing
- Full of zany characters and richly applied satire, and some great plot twists
- this is the greatest screwball comedy ever filmed
- It was pathetic. The worst part about it was the boxing scenes.

Named entity recognition

- Input: Sequence of text
- Output: For each word, give it label:
  - 1: Named entity — person
  - 2: Named entity — place
  - 0: Not a named entity
- Example: Abbot & Costello “Who’s on First?”
  https://youtu.be/2ZksQd2fC6

  Costello: look abbot, if you’re the coach you must know all the players
  Input: ... Abbot: well, let's see who we have on the bags. who's on first, what's on second, i don’t know is on third
  Output: Costello: 0 1 0 0 0 0 0 0 0 0
  Input: ... Abbot: 0 0 0 0 0 0 0 1 0 2 1 0 2 1 1 0 0 2

Prediction

- When elements of sequence are real numbers, methods like linear prediction.
- When they're categorical such as text, learning is needed.
  - “The surfer rode the ___”
  - “The Beatles Abbey ___”
  - “Show me the ___”
  - “The Wolf of Wall ___”

We Have Great Tools for Fixed-Size Data (vector in, vector out)
The nodes of multilayered network

\[ y(x; w) = a(w^T x + w_0) \]

- **\( x \)**: input vector
- **\( w \)**: weights
- **\( w_0 \)**: bias term
- **\( a \)**: activation function

**Activation Function: Sigmoid**

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

- As \( z \) goes from \(-\infty\) to \(\infty\), \( \sigma(z) \) goes from 0 to 1
- It has a “sigmoid” or S-like shape
- \( \sigma(0) = 0.5 \)

**Activation Function: Tanh**

\[ \text{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1 \]

- As \( x \) goes from \(-\infty\) to \(\infty\), tanh(\( x \)) goes from -1 to 1
- It has a "sigmoid" or S-like shape
- \( \text{tanh}(0) = 0 \)

**Activation Function: ReLU**

\[ g(z) = \max(0, z) \]

Let \( f = \frac{dg}{dz} \)
Two Layer Network

\[ y(x; W) = a_2(W_2 a_1(W_1 x + w_0)) \]

- Two sets of weights
- Two activation functions

Feedforward Networks

- These networks are composed of functions represented as “layers”

\[ y(x) = a_i(a_{i-1}(x; w_i); w_j) \]

with weights \( w_i \) associated with layer \( i \) and \( a_i \) is the activation function for layer \( i \).

- \( y(x) \) can be a scalar or a vector function.

Classification Networks and Softmax

- To classify the input \( x \) into one of \( c \) classes, we have \( c \) outputs.

- Output \( y_i \) can be viewed as \( p(\omega_i | x) \). That is the posterior probability of the class, given the input.

- Classification decision is \( \text{arg max } p(\omega_i | x) \).

- If the network were certain about the class, one output would be 1 and the rest would be zero.

- More generally \( \sum_{i=1}^{c} p(\omega_i | x) = 1 \), the \( c \) outputs must sum to 1.

- This can be implemented with a softmax layer: \( y_i(z) = \frac{e^{z_i}}{\sum_{j=1}^{c} e^{z_j}} \)

Universal Approximation Theorem

- tldr: if we have enough hidden units we can approximate “any” function! … but we may not be able to train it.

- Universal Approximation Theorem: A feedforward neural network with a linear output layer and one or more hidden layers with ReLU \[ \text{[Leshno et al. '93]}, \text{or sigmoid or some other “squashing” activation function [Hornik et al. '89, Cybenko '89]} \] can approximate any continuous function on a closed and bounded subset of \( \mathbb{R}^n \). This holds for functions mapping finite dimensional discrete spaces as well.
Universal Approximation Theorem: Caveats

- So even though “any” function can be approximated with a network as described with single hidden layer, the network may fail to train, overfit, fail to generalize, or require so many hidden units as to be infeasible.
- This is both encouraging and discouraging!
- However, [Montufar et al. 2014] showed that deeper networks are more efficient in that a deep rectified net can represent functions that would require an exponential number of hidden units in a shallow one hidden layer network.
- Deep networks composed on many rectified hidden layers are good at approximating functions that can be composed from simpler functions. And lots of tasks such as image classification may fit nicely into this space.

The loss function

- The loss function is really important. It determines how we compare what the network produces to our labels.
- Common ones:
  - Regression problems:
    - Distance: $L(y, \hat{y}) = ||y - \hat{y}||^p$, usually $p = 1$ or $2$
  - Classification
    - Softmax: $y(z) = \frac{e^z}{\sum_{j=1}^c e^z_j}$
    - Cross entropy between $y$ and $\hat{y}$

High level view of evaluation and training

- Training data: $\{<x^{(i)}, y^{(i)}>: 1 \leq i \leq n\}$
- Total Loss: $\sum_{i=1}^n L(f(x^{(i)}; W), y^{(i)})$
- Training: Find a $W$ that makes total loss small.

Cross Entropy

- Cross entropy between the output of a network $\hat{y}$ and ground truth $y$ is commonly used when $n$ classes are mutually exclusive. That is:
  - $y$ is a vector with one 1 and the rest 0’s.
  - $\hat{y}$ is a vector with positive floats that sum to 1: $\sum_{j=1}^n y_j = 1$
- Cross entropy between $y$ and $\hat{y}$ is
  \[
  H(y, \hat{y}) = \sum_{j=1}^n y_j \log \frac{1}{\hat{y}_j} = - \sum_{j=1}^n y_j \log \hat{y}_j
  \]
- Since $y$ is “one hot”, only one term of the sum is non-zero, and there is only a positive loss back propagating from that one output.
- Cross entropy is the “number of bits we’ll need if we encode symbols from $y$ using the wrong distribution of $\hat{y}$. Minimizing cross entropy is minimizes the number of bits.
- It’s the same as minimizing the KL divergence $KL(y, \hat{y})$ between $y$ and $\hat{y}$.
- Maximizing the likelihood of the output $P(y|\hat{y}) = \prod_{i=1}^n P(y_i|\hat{y}_i)$ is the same as minimizing the cross entropy.
Training

- Back propagation using Stochastic Gradient Descent
- Adagrad, RMSprop, ADAM
- Regularization: Dropout, Batch/Group/Instance Normalization
- Early Stopping

CSE291G Details

- Piazza: Join Piazza Group, even if you’re on waitlist
- Course “web page” is using Dropbox Paper.
  - [bit.ly/cse291g](bit.ly/cse291g)
- Wait list