The path to LSTMs

A succinct introduction

Bernardo Pérez Orozco
ber@robots.ox.ac.uk

Department of Engineering Sciences
University of Oxford

MLRG Tea Talk Series, 2016

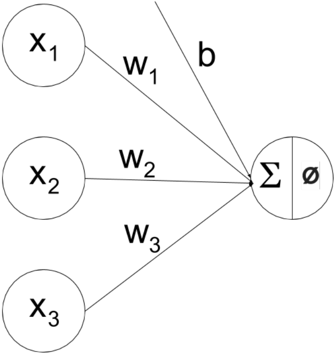
Introduction

- Deep Learning revolution: neural networks (NN)
    - "Deep" models = "many" layers
- LSTM is state-of-the-art in sequence learning
- LSTM is a specific case of Recurrent NNs (RNNs)
- LSTM exists because:
    - NNs are most often trained by gradient descent...
    - ... and you can't train general RNNs by gradient descent!

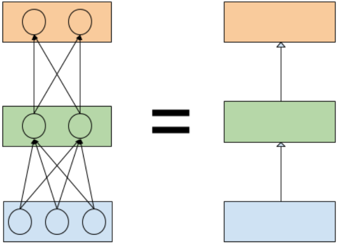
Outline

Neural networks

- A NN is a collection of mappings called **layers**
- Each layer contains a number of **neurons** (hidden units)
- Neurons are feature detectors, i.e. they react to stimuli
- **Example:** Dense unit outputs $h = \phi(\mathbf{w}^T\mathbf{x} + b)$, where ϕ is called the activation function of the unit.



- We don't **think of NNs** in terms of neurons, but rather **in terms of layers** with parameters θ:



Dense layer

A dense layer is a mapping $H : \mathbb{R}^n \to \mathbb{R}^m$ given by $H(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x} + \mathbf{b})$, where:

- $\mathbf{x}$ is the input to the layer
- $\mathbf{h} = H(\mathbf{x})$ are the output or activations
- ϕ is called activation function
- m is number of neurons in the layer
- $\theta = (\mathbf{W}, \mathbf{b})$ are the parameters of the layer

- Output $\mathbf{h}$ indicates amount of activity in the neuron

- Some common activation functions are:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\mathrm{ReLU}(x) = \max(0, x)$$

Deep learning

- Models have "many" layers
  - Convolutional networks
  - Recurrent networks
  - Autoencoders
- Advances in optimisation
  - Developments in gradient descent
  - New regularisation techniques, e.g. dropout
- Why is depth important?
  - Lower layers detect simple features, e.g. edges
  - Upper layers learn complex features by combining simpler ones, e.g. combine edges to get shapes
- Why now?
  - DL has been around since the 80s!
  - Better hardware: GPUs
  - Better software: specialised libraries

Gradient descent developments

Vanilla gradient descent

Gradient descent

Iterative optimisation method. Update rule is given by:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta)$$

- Learning rate: α
- Loss function or criterion: $L(\theta)$

Gradient descent developments

Recent developments

- Stochastic minibatch gradient descent
  - Approximate gradient over random minibatch for multiple updates per epoch
- Adaptive methods
  - Learning rate changes with time
- Momentum
  - New updates consider previous updates, i.e. gain momentum on descent
- AutoDiff
  - Program P computes $f(\theta)$. AutoDiff finds program P' that computes $\nabla_\theta f(\theta)$ efficiently.
- Examples: Adam, RMSprop, Adadelta

Recurrent Neural Networks

The RNN layer

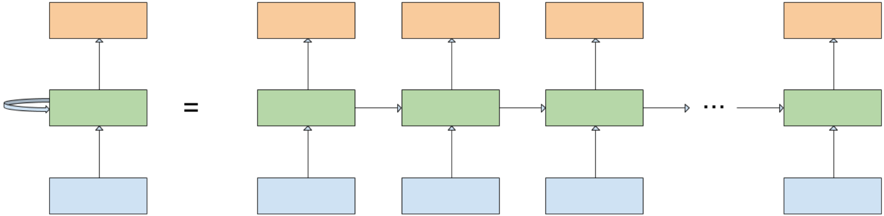
The recurrent neural network layer is given by:

$$\mathbf{h}_t = \mathbf{W_r}\phi(\mathbf{h_{t-1}}) + \mathbf{W_i}\mathbf{x}_t + \mathbf{b}$$

with $\theta = (\mathbf{W}_i, \mathbf{W}_r, \mathbf{b})$

- From the equation above, it is easy to see that $\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t, \theta)$, and furthermore $\mathbf{h}_t = f(f(...f(\mathbf{h}_0, \mathbf{x}_1, \theta)..., \mathbf{x}_{t-1}, \theta), \mathbf{x}_t, \theta)$, where $\mathbf{h}_0$ is the initial state of the recurrent network.

Recurrent Neural Networks
Unfolding the network



- Compute gradient to train by gradient descent

$$
\begin{aligned}
\frac{\partial \mathbf{h}_t}{\partial \theta_j} &= \frac{\partial \mathbf{W}_r}{\partial \theta_j}\phi(\mathbf{h_{t-1}}) + \mathbf{W}_r \frac{\partial \phi(\mathbf{h}_{t-1})}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \theta_j} \\
&= \frac{\partial \mathbf{W}_r}{\partial \theta_j}\phi(\mathbf{h_{t-1}}) + \mathbf{W}_r \phi'(\mathbf{h}_{t-1}) \frac{\partial \mathbf{h}_{t-1}}{\partial \theta_j} \\
&= \frac{\partial \mathbf{W}_r}{\partial \theta_j}\phi(\mathbf{h_{t-1}}) + \color{red}{\mathbf{W}_r \phi'(\mathbf{h}_{t-1})} \color{black}{\left(\frac{\partial \mathbf{W}_r}{\partial \theta_j}\phi(\mathbf{h_{t-2}}) + \color{red}{\mathbf{W}_r \phi'(\mathbf{h}_{t-2})} \color{black}{\frac{\partial \mathbf{h}_{t-2}}{\partial \theta_j}} \right)}
\end{aligned}
$$

- Exploding gradient
    - Can be handled by heuristics, e.g. gradient clipping
- Vanishing gradient
    - Currently solved by means of architectural changes, i.e. LSTM

The LSTM layer

The LSTM layer

The LSTM layer directly addresses the issue of the vanishing gradient by means of a gating mechanism, and is given by the following equations[1]:

$$
\begin{aligned}
\mathbf{x}'_t &= [\mathbf{x}_t, \mathbf{h}_{t-1}] \\
\mathbf{S}_t &= \tanh(\mathbf{W}_S \mathbf{x}'_t + \mathbf{b}_S) \\
\mathbf{C}_t &= \mathbf{i}_t \odot \mathbf{S}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1} \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{C}_t)
\end{aligned}
$$

where $\mathbf{i}_t, \mathbf{o}_t, \mathbf{f}_t$ are called the input, output and forget gates given by:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}'_t + \mathbf{b}_i) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}'_t + \mathbf{b}_o) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}'_t + \mathbf{b}_f)
\end{aligned}
$$

[1] Where $\mathbf{x} \odot \mathbf{y}$ refers to the element-wise product of vectors $\mathbf{x}, \mathbf{y}$.

The LSTM layer

- We call $\mathbf{C}_t$ the *memory cell* at time t, and it is arguably the main component of the LSTM architecture.
- We can think of the LSTM in the following manner:

$$\mathbf{C}_t = \mathbf{i}_t \odot \mathbf{S}_t + \mathbf{f}_t \odot \mathbf{C}_{t-1}$$
$$\text{memory}_t = \text{read} \odot \text{input}_t + \text{remember} \odot \text{memory}_{t-1}$$

- Similar reasoning:

$$\frac{\partial \mathbf{h}_t}{\partial \theta_j} = \frac{\partial \mathbf{o}_t}{\partial \theta_j} \odot \tanh(\mathbf{C}_t) + \mathbf{o}_t \odot \frac{\partial \tanh(\mathbf{C}_t)}{\partial \mathbf{C}_t} \frac{\partial \mathbf{C}_t}{\partial \theta_j}$$

$$\frac{\partial \mathbf{C}_t}{\partial \theta_j} = \frac{\partial \mathbf{i}_t}{\partial \theta_j} \odot \mathbf{S}_t + \mathbf{i}_t \odot \frac{\partial \mathbf{S}_t}{\partial \theta_j} + \frac{\partial \mathbf{f}_t}{\partial \theta_j} \odot \mathbf{C}_{t-1} + \mathbf{f}_t \odot \frac{\partial \mathbf{C}_{t-1}}{\partial \theta_j}$$

- The term $\frac{\partial \mathbf{C}_t}{\partial \theta_j}$ is also a function of $\frac{\partial \mathbf{C}_{t-1}}{\partial \theta_j}$
  - $\mathbf{C}_{t-1}$ does not go through non-linear ϕ
  - Weighted by the element-wise product of the previous k forget gates $\prod_{T=k}^{t} \mathbf{f}_T$. The LSTM can only forget once $\mathbf{f}_k$ tends to zero.

LSTM applications

- Speech recognition
- Sequence generation (handwriting and Shakespeare texts)
- Financial forecasting
- Reinforcement learning
- Models of attention (scene labelling)
- Check reading list in notes!

- Integrates routines for: training, testing, data preprocessing, activation, regularisation and loss functions, and optimisation procedures.
- Inherits AutoDiff, GPU compatibility from Theano/TensorFlow (backends).
- Paradigm: stack modules (Torch-like)
- Well documented
- Runs on Python

1. Prepare data: xTrain, yTrain, xTest, yTest
2. Init empty layer: model = Sequential()
3. Stack layers: model.add(...)
4. Compile model: model.compile(loss, optimiser)
5. Train: model.fit(xTrain, yTrain, ...)
6. Test: model.predict(xTest)

Open questions

- No probabilistic reasoning
    - Measuring uncertainty?
    - Variational methods?
- Little interpretability
    - What do its hidden representations mean?
- Most applications are classification problems
    - More time series forecasting!