Neural Architectures for Image, Language, and Speech Processing

Karl Stratos

June 26, 2018

Overview

- Feedforward Networks
  - Need for Specialized Architectures
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
  - Long Short-Term Memory Networks (LSTMs)
  - Example: Bidirectional LSTM Network for POS Tagging
- Encoder-Decoder Models
  - Example: RNN-Based Seq2Seq
  - Bonus: Connectionist Temporal Classification (CTC)

What's a Neural Network?

What's a Neural Network?

Just a composition of linear/nonlinear functions.

$$f(x) = W^{(L)} \tanh\left(W^{(L-1)} \cdots \tanh\left(W^{(1)} x\right) \cdots\right)$$

What's a Neural Network?

Just a composition of linear/nonlinear functions.

$$f(x) = W^{(L)} \tanh\left(W^{(L-1)} \cdots \tanh\left(W^{(1)}x\right) \cdots\right)$$

More like a paradigm, not a specific model.

1. **Transform** your input $x \longrightarrow f(x)$.
2. Define **loss** between $f(x)$ and the target label y.
3. Train parameters by minimizing the loss.

You've Already Seen Some Neural Networks...

Log-linear model is a neural network with 0 hidden layer and a softmax output layer:

$$p(y|x) := \frac{\exp([Wx]_y)}{\sum_{y'} \exp([Wx]_{y'})} = \mathsf{softmax}_y(Wx)$$

Get W by minimizing $L(W) = -\sum_i \log p(y_i|x_i)$.

You've Already Seen Some Neural Networks...

Log-linear model is a neural network with 0 hidden layer and a softmax output layer:

$$p(y|x) := \frac{\exp([Wx]_y)}{\sum_{y'} \exp([Wx]_{y'})} = \mathsf{softmax}_y(Wx)$$

Get W by minimizing $L(W) = -\sum_i \log p(y_i|x_i)$.

Linear regression is a neural network with 0 hidden layer and the identity output layer:

$$f(x) := Wx$$

Get W by minimizing $L(W) = \sum_i (y_i - f_i(x))^2$.

Feedforward Network

Think: log-linear with extra transformation

Feedforward Network

Think: log-linear with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

Feedforward Network

Think: log-linear with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

With 2 hidden layers:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$h^{(2)} = \tanh(W^{(2)}h^{(1)})$$
$$p(y|x) = \mathsf{softmax}_y(h^{(2)})$$

Again, get parameters $W^{(l)}$ by minimizing $-\sum_i \log p(y_i|x_i)$.

Feedforward Network

Think: log-linear with extra transformation

With 1 hidden layer:

$$h^{(1)} = \tanh(W^{(1)}x)$$
$$p(y|x) = \mathsf{softmax}_y(h^{(1)})$$

With 2 hidden layers:

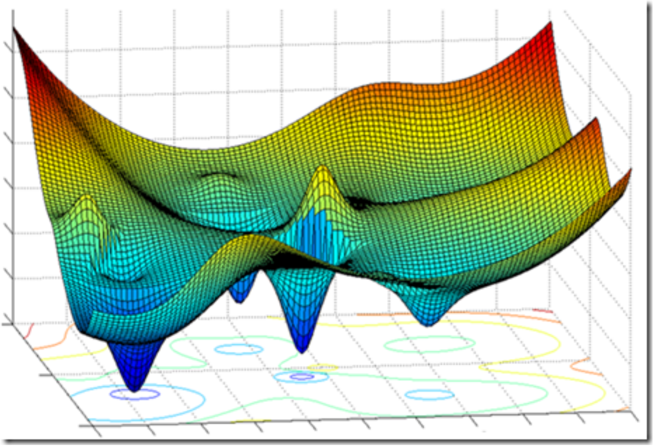
$$h^{(1)} = \tanh(W^{(1)}x)$$
$$h^{(2)} = \tanh(W^{(2)}h^{(1)})$$
$$p(y|x) = \mathsf{softmax}_y(h^{(2)})$$

Again, get parameters $W^{(l)}$ by minimizing $-\sum_i \log p(y_i|x_i)$.

- Q. What's the catch?

Training = Loss Minimization

We can decrease any continuous loss by following the subgradient.



1. Differentiate the loss wrt. model parameters (backprop)
2. Take a gradient step

Overview

- Feedforward Networks
  - Need for Specialized Architectures
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
  - Long Short-Term Memory Networks (LSTMs)
  - Example: Bidirectional LSTM Network for POS Tagging
- Encoder-Decoder Models
  - Example: RNN-Based Seq2Seq
  - Bonus: Connectionist Temporal Classification (CTC)

Neural Networks are (Finite-Sample) Universal Learners!

Theorem. (Zhang et al., 2016) Give me any

1. Set of n samples $S = \left\{ \boldsymbol{x}^{(1)} \dots \boldsymbol{x}^{(n)} \right\} \subset \mathbb{R}^d$
2. Function $f : S \to \mathbb{R}$ that assigns some arbitrary value $f(\boldsymbol{x}^{(i)})$ to each $i = 1 \dots n$

Then I can specify a 1-hidden-layer feedforward network $C : S \to \mathbb{R}$ with $2n + d$ parameters such that $C(\boldsymbol{x}^{(i)}) = f(\boldsymbol{x}^{(i)})$ for all $i = 1 \dots n$.

Neural Networks are (Finite-Sample) Universal Learners!

Theorem. (Zhang et al., 2016) Give me any

1. Set of n samples $S = \left\{\boldsymbol{x}^{(1)} \ldots \boldsymbol{x}^{(n)}\right\} \subset \mathbb{R}^d$
2. Function $f : S \to \mathbb{R}$ that assigns some arbitrary value $f(\boldsymbol{x}^{(i)})$ to each $i = 1 \ldots n$

Then I can specify a 1-hidden-layer feedforward network $C : S \to \mathbb{R}$ with $2n + d$ parameters such that $C(\boldsymbol{x}^{(i)}) = f(\boldsymbol{x}^{(i)})$ for all $i = 1 \ldots n$.

Proof.

Define $C(\boldsymbol{x}) = \boldsymbol{w}^\top \mathrm{relu}((\boldsymbol{a}^\top \boldsymbol{x} \ldots \boldsymbol{a}^\top \boldsymbol{x}) + \boldsymbol{b})$ where $\boldsymbol{w}, \boldsymbol{b} \in \mathbb{R}^n$ and $\boldsymbol{a} \in \mathbb{R}^d$ are network parameters. Choose $\boldsymbol{a}, \boldsymbol{b}$ so that the matrix $A_{i,j} := [\max\left\{0, \boldsymbol{a}^\top \boldsymbol{x}^{(i)} - b_j\right\}]$ is triangular. Solve for $\boldsymbol{w}$ in

$$\begin{bmatrix} f(\boldsymbol{x}^{(1)}) \\ \vdots \\ f(\boldsymbol{x}^{(n)}) \end{bmatrix} = A\boldsymbol{w}$$

So Why Not Use a Simple Feedforward for Everything?

So Why Not Use a Simple Feedforward for Everything?

Computational reasons

- For example, using a GIANT feedforward to cover instances of different sizes is clearly inefficient.

So Why Not Use a Simple Feedforward for Everything?

Computational reasons

- For example, using a GIANT feedforward to cover instances of different sizes is clearly inefficient.

Empirical reasons

- In principle, we can learn any function.
- This tells us nothing about *how to get there*. How many samples do we need? How can we find the right parameters?
- Specializing an architecture to a particular type of computation allows us to incorporate **inductive bias**.
- "Right" architecture is **absolutely critical** in practice.

Overview

Feedforward Networks
- Need for Specialized Architectures

Convolutional Neural Networks (CNNs)

Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Example: Bidirectional LSTM Network for POS Tagging

Encoder-Decoder Models
- Example: RNN-Based Seq2Seq
- Bonus: Connectionist Temporal Classification (CTC)

Image = Cube

$X \in \mathbb{R}^{w \times h \times d}$: width w, height h, depth d (e.g., 3 RGB values)

(scene from *Your Name* (2016))

Convolutional Neural Network (CNN)

Think: Slide various types of "filters" across image

Convolutional Neural Network (CNN)

Think: Slide various types of "filters" across image

We say we apply a **filter** $F^i \in \mathbb{R}^{f \times f \times d}$ with **stride** $s \in \mathbb{N}$ and **zero-padding size** $p \in \mathbb{N}$ to an image $X \in \mathbb{R}^{w \times h \times d}$ and obtain a slice $Z^i \in \mathbb{R}^{w' \times h' \times 1}$ where $w' = (w - f + 2p)/s + 1$ and $h' = (h - f + 2p)/s + 1$.

Each entry of Z^i is given by a dot product between F^i and the corresponding **receptive field** in X (with zero-padding):

$$Z^i_{t,t',1} = \sum_{a,b,c} F^i_{a,b,c} \times X_{t+a,t'+b,c}$$

Convolutional Neural Network (CNN)

Think: Slide various types of "filters" across image

We say we apply a **filter** $F^i \in \mathbb{R}^{f\times f\times d}$ with **stride** $s \in \mathbb{N}$ and **zero-padding size** $p \in \mathbb{N}$ to an image $X \in \mathbb{R}^{w\times h\times d}$ and obtain a slice $Z^i \in \mathbb{R}^{w'\times h'\times 1}$ where $w' = (w - f + 2p)/s + 1$ and $h' = (h - f + 2p)/s + 1$.

Each entry of Z^i is given by a dot product between F^i and the corresponding **receptive field** in X (with zero-padding):

$$Z^i_{t,t',1} = \sum_{a,b,c} F^i_{a,b,c} \times X_{t+a,t'+b,c}$$

We use multiple filters and stack the slices: if m filters are used, the output is $Z \in \mathbb{R}^{w'\times h'\times m}$.

An Example ConvNet Architecture

Input: image $X \in \mathbb{R}^{w \times h \times d}$

- **Convolutional layer**: use m filters to obtain $Z \in \mathbb{R}^{w' \times h' \times m}$.

An Example ConvNet Architecture

Input: image $X \in \mathbb{R}^{w \times h \times d}$

- **Convolutional layer**: use m filters to obtain $Z \in \mathbb{R}^{w' \times h' \times m}$.
- **Relu layer**: Apply element-wise relu to Z.

An Example ConvNet Architecture

Input: image $X \in \mathbb{R}^{w \times h \times d}$

- **Convolutional layer**: use m filters to obtain $Z \in \mathbb{R}^{w' \times h' \times m}$.
- **Relu layer**: Apply element-wise relu to Z.
- **Max pooling layer**: use 2×2 filter with stride 2 and appropriate zero-padding size to "downsample" Z to $P \in \mathbb{R}^{(w'/2) \times (h'/2) \times m}$.

An Example ConvNet Architecture

Input: image $X \in \mathbb{R}^{w \times h \times d}$

- **Convolutional layer**: use m filters to obtain $Z \in \mathbb{R}^{w' \times h' \times m}$.
- **Relu layer**: Apply element-wise relu to Z.
- **Max pooling layer**: use 2×2 filter with stride 2 and appropriate zero-padding size to "downsample" Z to $P \in \mathbb{R}^{(w'/2) \times (h'/2) \times m}$.
- **Fully-connected layer**: equivalent to convolutional layer with K giant filters $Q^k \in \mathbb{R}^{(w'/2) \times (h'/2) \times m}$

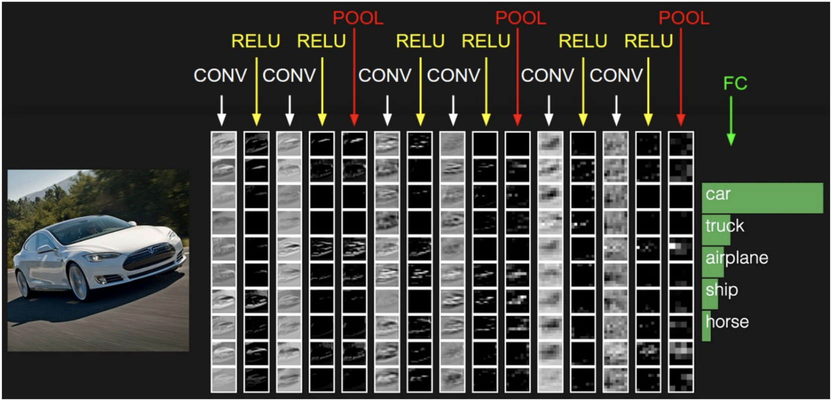
An Example ConvNet Architecture

Input: image $X \in \mathbb{R}^{w \times h \times d}$

- **Convolutional layer**: use m filters to obtain $Z \in \mathbb{R}^{w' \times h' \times m}$.
- **Relu layer**: Apply element-wise relu to Z.
- **Max pooling layer**: use 2×2 filter with stride 2 and appropriate zero-padding size to "downsample" Z to $P \in \mathbb{R}^{(w'/2) \times (h'/2) \times m}$.
- **Fully-connected layer**: equivalent to convolutional layer with K giant filters $Q^k \in \mathbb{R}^{(w'/2) \times (h'/2) \times m}$

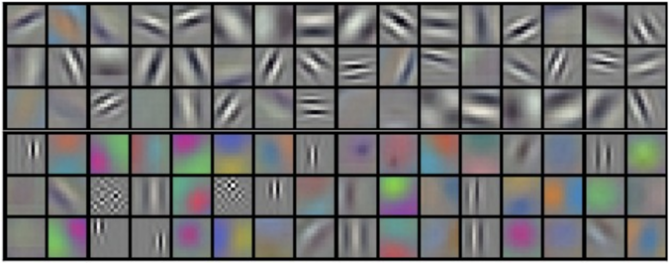
Output: vector $v \in \mathbb{R}^{1 \times 1 \times K}$ used for K-way classification of X

In Practice, Many Such Layers



http://cs231n.github.io/convolutional-networks

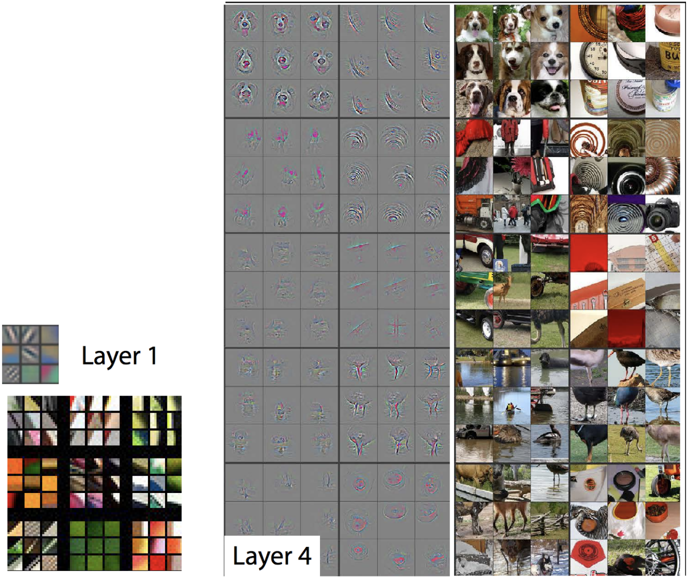
Filters Correspond to Patterns



http://cs231n.github.io/convolutional-networks

Filters at Lower Layer vs Higher Layer

Zeiler and Fergus (2013)



Vanishing/Exploding Gradient Problem in Deep Networks

$h^l = \mathsf{Conv}(\mathsf{Conv}(\mathsf{Conv}(\mathsf{Conv}(\mathsf{Conv}(\cdots(x)\cdots)))))$

ResNet (He et al., 2015): at each layer l,

$$h^l = \mathsf{Conv}(h^{l-1}) + h^{l-1}$$

Overview

Feedforward Networks
- Need for Specialized Architectures

Convolutional Neural Networks (CNNs)

Recurrent Neural Networks (RNNs)
- Long Short-Term Memory Networks (LSTMs)
- Example: Bidirectional LSTM Network for POS Tagging

Encoder-Decoder Models
- Example: RNN-Based Seq2Seq
- Bonus: Connectionist Temporal Classification (CTC)

Language/Speech = Sequence

$x_1 \dots x_N \in \mathbb{R}^d$: each x_i is a d-dimensional embedding of the i-th input

- Text: x_i is a word embedding (to be also learned)
- Speech: x_i is an acoustic feature vector

Two properties of this type of data

1. Length N not fixed
2. Typically processed from left to right (most of the time)

Recurrent Neural Network (RNN)

Think: HMM (or Kalman filter) with extra transformation

Recurrent Neural Network (RNN)

Think: HMM (or Kalman filter) with extra transformation

Input: sequence $x_1 \dots x_N \in \mathbb{R}^d$

- For $i = 1 \dots N$,

$$h_i = \tanh\left(W x_i + V h_{i-1}\right)$$

Output: sequence $h_1 \dots h_N \in \mathbb{R}^{d'}$

RNN $\approx$ Deep Feedforward

Unroll the expression for the last output vector h_N:

$$h_N = \tanh\left(Wx_N + V\left(\cdots + V\tanh\left(Wx_1 + Vh_0\right)\cdots\right)\right)$$

It's just a deep "feedforward network" with one important difference: **parameters are reused**

- (V, W) are applied N times

Training: do backprop on this unrolled network, update parameters

Overview

- Feedforward Networks
  - Need for Specialized Architectures
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
  - Long Short-Term Memory Networks (LSTMs)
  - Example: Bidirectional LSTM Network for POS Tagging
- Encoder-Decoder Models
  - Example: RNN-Based Seq2Seq
  - Bonus: Connectionist Temporal Classification (CTC)

LSTM

- RNN produces a sequence of output vectors

$$x_1 \dots x_N \quad \longrightarrow \quad h_1 \dots h_N$$

- LSTM produces "memory cell vectors" along with output

$$x_1 \dots x_N \quad \longrightarrow \quad c_1 \dots c_N,\ h_1 \dots h_N$$

- These $c_1 \dots c_N$ enable the network to keep or drop information from previous states.

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

- Compute *another masking vector* for the output:

$$o_i = \sigma\left(U^o x + V^o h_{i-1} + W^o c_i\right) \in [0,1]^{d'}$$

LSTM: Details

At each time step i,

- Compute a *masking vector* for the memory cell:

$$q_i = \sigma\left(U^q x + V^q h_{i-1} + W^i c_{i-1}\right) \in [0,1]^{d'}$$

- Use q_i to keep/forget dimensions in previous memory cell:

$$c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x + V^c h_{i-1}\right)$$

- Compute *another masking vector* for the output:

$$o_i = \sigma\left(U^o x + V^o h_{i-1} + W^o c_i\right) \in [0,1]^{d'}$$

- Use o_i to keep/forget dimensions in current memory cell:

$$h_i = o_i \odot \tanh(c_i)$$

Overview

Feedforward Networks

- Need for Specialized Architectures

Convolutional Neural Networks (CNNs)

Recurrent Neural Networks (RNNs)

- Long Short-Term Memory Networks (LSTMs)
- Example: Bidirectional LSTM Network for POS Tagging

Encoder-Decoder Models

- Example: RNN-Based Seq2Seq
- Bonus: Connectionist Temporal Classification (CTC)

Build Your Network

Model Parameters

- Embedding e_c for each character type c
- 2 LSTMs (forward/backward) at character level
- Embedding e_w for each word type w
- 2 LSTMs (forward/backward) at word level
- Feedforward network (U, V) at the output

`the` `dog` `saw` `the` `cat`

1. Character-Level LSTMs

Forward character-level LSTM

$$e_{\mathsf{d}}\ e_{\mathsf{o}}\ e_{\mathsf{g}} \longrightarrow f_1^c\ f_2^c\ f_3^c \in \mathbb{R}^{d_c}$$

Backward character-level LSTM

$$e_{\mathsf{g}}\ e_{\mathsf{o}}\ e_{\mathsf{d}} \longrightarrow b_1^c\ b_2^c\ b_3^c \in \mathbb{R}^{d_c}$$

Get a character-aware encoding of `dog`:

$$x_{\mathsf{dog}} = \begin{bmatrix} f_3^c \\ b_3^c \\ e_{\mathsf{dog}} \end{bmatrix} \in \mathbb{R}^{2d_c+d_w}$$

2. Word-Level LSTMs

Forward word-level LSTM

$$x_{\text{the}} \;\; x_{\text{dog}} \;\; x_{\text{saw}} \;\; x_{\text{the}} \;\; x_{\text{cat}} \longrightarrow f_1^w \;\; f_2^w \;\; f_3^w \;\; f_4^w \;\; f_5^w \in \mathbb{R}^d$$

Backward word-level LSTM

$$x_{\text{cat}} \;\; x_{\text{the}} \;\; x_{\text{saw}} \;\; x_{\text{dog}} \;\; x_{\text{the}} \longrightarrow b_1^w \;\; b_2^w \;\; b_3^w \;\; b_4^w \;\; b_5^w \in \mathbb{R}^d$$

Get a sentence-aware encoding of dog:

$$z_{\text{dog}} = \begin{bmatrix} f_2^w \\ b_4^w \end{bmatrix} \in \mathbb{R}^{2d}$$

3. Feedforward

The final vector $h_{\text{dog}} \in \mathbb{R}^m$ has **dimension equal to the the number of tag types** m, computed by feedforward

$$h_{\text{dog}} = V \tanh(U z_{\text{dog}})$$

Think

$$[h_{\text{dog}}]_{\texttt{N}} \approx \text{score of tag } \texttt{N} \text{ for dog}$$

Greedy Model

Define distribution over tags at each position as:

$$p(\texttt{N}|\text{dog}) = \text{softmax}_{\texttt{N}}(h_{\text{dog}})$$

Given tagged words $(x^{(1)}, y^{(1)}) \dots (x^{(n)}, y^{(n)})$, minimize the loss

$$L(\Theta) = -\sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)})$$

References

- CNN tutorial: http://cs231n.github.io/convolutional-networks/
- LSTM tutorial: http://colah.github.io/posts/2015-08-Understanding-LSTMs/