Intro to Sequential Models

GRUs and LSTMs

Sources: A. Ng, F. Lee, J. Johnson, Mach. Learn. Mastery (book)

RNNs are imperfect...

- Great step forward for sequential data predictions, but…
  - Vanishing gradient problems!
  - Optimization routine can break down
  - Solution:
    - Carry forward information earlier in network to use later in network!

Enter idea of Gated Recurrent Units (or GRUs)

RNNs are imperfect...

- Great step forward for sequential data predictions, but…
  - Vanishing gradient problems!
  - Optimization routine can break down
  - Early sequential data can be cancelled out in explanations of later sequential data (early time steps / later time steps)
  - Solution:
    - Carry forward information earlier in network to use later in network!

Enter idea of **Gated Recurrent Units (or GRUs)**

Enter idea of **Gated Recurrent Units (or GRUs)**

RNNs:

$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$
$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

Memory Cell Idea:

Add ability to carry forward information a<n> hidden nodes later in matrix

Use new update weights learned to ***optionally update*** a<n> in next time step dependent upon x input.

Enter idea of **Gated Recurrent Units (or GRUs)**

Defining GRU formula elements:

C= memory cell

c<t> = a<t>

We calculate candidate values for updating c<t> labelled: $\tilde{c}^{<t>}$

And we use a vector of values generated using further weights that output to values between zero and one.

These “gates” are used to adjust c<t> with candidate values in $\tilde{c}^{<t>}$

Gate label: Γ_u

Enter idea of **Gated Recurrent Units (or GRUs)**

Defining GRU formulas:

$$\tilde{c}^{<t>} = \tanh(W_c[\, c^{<t-1>}, x^{<t>}] + b_c)$$ ⟸

Note:
same structure as **a<t>** from RNNs,
but new weights Wc

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$ ⟸

Note:
same structure as **a<t>** from RNNs,
but new weights Wu AND
Outputs between >0 and <1.

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$ ⟸

Note:
Element-wise multiplication used to updated c<t> and candidate for c<t> using gates.

Enter idea of **Gated Recurrent Units (or GRUs)**

Defining GRU formulas:

$$\tilde{c}^{<t>} = \tanh(W_c[\, c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

What's happening?

Take previous hidden node ouputs

Calculate new candidate outputs for next step in sequence.

Use gate to update each element of c<t> with element wise multiplication.

Gates allow some information from previous steps to be carried forward in network!

Newer **Gated Recurrent Units (or GRUs)**

Adds another gate here with new weights used to multiply with c<t-1>

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

Why? Same idea, but performs a bit better in practice.

Another option: Long Short Term Memory Cells (LSTMs)

GRUs and LSTMs are different ways to achieve the same goals...RNN improvement

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

Update Gate =>>>> $\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$

Forget Gate =>>>> $\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$

Output Gate =>>>> $\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$

Update and forget gates affect c<t> =>> $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$

c<t> and output gate affect a<t> =>> $a^{<t>} = \Gamma_o * \tanh\, c^{<t>}$

LSTM or GRU, Which to Choose?

- LSTM has been tested more in production
  - Works well and is typical approach used, but…
  - More complexity means slightly more comput time
- GRU less complex, but newer
  - Means less evidence of success than older LSTM

Out of the box use LSTMs, but both tend to work well and neither is guaranteed to perform better on particular data (therefore, experiment!)