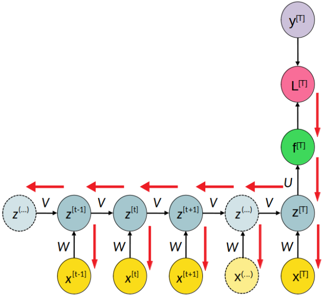
Deep Learning

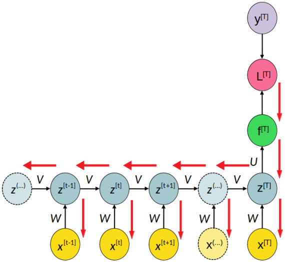
Recurrent Neural Networks - Backpropogation



Learning goals

- Backpropagation
- Exploding and Vanishing Gradients

BACKPROPAGATION THROUGH TIME



- For training the RNN we need to compute $\frac{dL}{du_{i,j}}$, $\frac{dL}{dv_{i,j}}$, and $\frac{dL}{dw_{i,j}}$.
- To do so, during backpropagation at time step t we may need to compute

$$\frac{dL}{d\boldsymbol{z}^{[1]}} = \frac{dL}{d\boldsymbol{z}^{[t]}} \frac{d\boldsymbol{z}^{[t]}}{d\boldsymbol{z}^{[t-1]}} \cdots \frac{d\boldsymbol{z}^{[2]}}{d\boldsymbol{z}^{[1]}}$$

LONG-TERM DEPENDENCIES

- Here, $\boldsymbol{z}^{[t]} = \sigma(\boldsymbol{V}^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}^\top \mathbf{x}^{[t]} + \boldsymbol{b})$
- It follows that:

$$\frac{d\boldsymbol{z}^{[t]}}{d\boldsymbol{z}^{[t-1]}} = \text{diag}(\sigma'(\boldsymbol{V}^\top \boldsymbol{z}^{[t-1]} + \boldsymbol{W}^\top \mathbf{x}^{[t]} + \boldsymbol{b}))\boldsymbol{V}^\top = \boldsymbol{D}^{[t-1]}\boldsymbol{V}^\top$$

$$\frac{d\boldsymbol{z}^{[t-1]}}{d\boldsymbol{z}^{[t-2]}} = \text{diag}(\sigma'(\boldsymbol{V}^\top \boldsymbol{z}^{[t-2]} + \boldsymbol{W}^\top \mathbf{x}^{[t-1]} + \boldsymbol{b}))\boldsymbol{V}^\top = \boldsymbol{D}^{[t-2]}\boldsymbol{V}^\top$$

$$\vdots$$

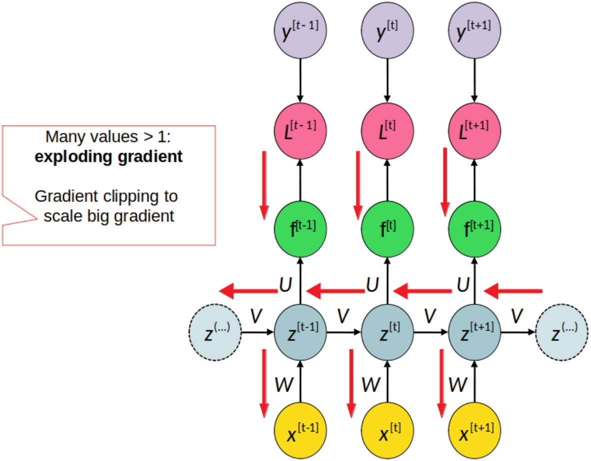
$$\frac{d\boldsymbol{z}^{[2]}}{d\boldsymbol{z}^{[1]}} = \text{diag}(\sigma'(\boldsymbol{V}^\top \boldsymbol{z}^{[1]} + \boldsymbol{W}^\top \mathbf{x}^{[2]} + \boldsymbol{b}))\boldsymbol{V}^\top = \boldsymbol{D}^{[1]}\boldsymbol{V}^\top$$

$$\frac{dL}{d\boldsymbol{z}^{[1]}} = \frac{dL}{d\boldsymbol{z}^{[t]}}\frac{d\boldsymbol{z}^{[t]}}{d\boldsymbol{z}^{[t-1]}} \cdots \frac{d\boldsymbol{z}^{[2]}}{d\boldsymbol{z}^{[1]}} = \boldsymbol{D}^{[t-1]}\boldsymbol{D}^{[t-2]} \ldots \boldsymbol{D}^{[1]}(\boldsymbol{V}^\top)^{[t-1]}$$

LONG-TERM DEPENDENCIES

- Therefore, for an arbitrary time-step i in the past, $\frac{d\boldsymbol{z}^{[t]}}{d\boldsymbol{z}^{[i]}}$ will contain the term $(\boldsymbol{V}^\top)^{t-i}$ within it (this follows from the chain rule).
- Based on the largest eigenvalue of $\boldsymbol{V}^\top$, the presence of the term $(\boldsymbol{V}^\top)^{t-i}$ can either result in vanishing or exploding gradients.
- This problem is quite severe for RNNs (as compared to feedforward networks) because the **same** matrix $\boldsymbol{V}^\top$ is multiplied several times. ▸ Click here
- As the gap between t and i increases, the instability worsens.
- It is quite challenging for RNNs to learn long-term dependencies. The gradients either **vanish** (most of the time) or **explode** (rarely, but with much damage to the optimization).
- That happens simply because we propagate errors over very many stages backwards.

LONG-TERM DEPENDENCIES



Many values > 1:
exploding gradient

Gradient clipping to
scale big gradient

Figure: Exploding of gradients

LONG-TERM DEPENDENCIES

- Recall, that we can counteract exploding gradients by implementing gradient clipping.
- To avoid exploding gradients, we simply clip the norm of the gradient at some threshold *h* (see chapter 4):

$$\text{if } ||\nabla W|| > \text{h} : \nabla W \leftarrow \frac{h}{||\nabla W||}\nabla W$$

LONG-TERM DEPENDENCIES

Many values < 1:
vanishing gradient

1. Activation function
2. Weight initialization
3. Network architecture

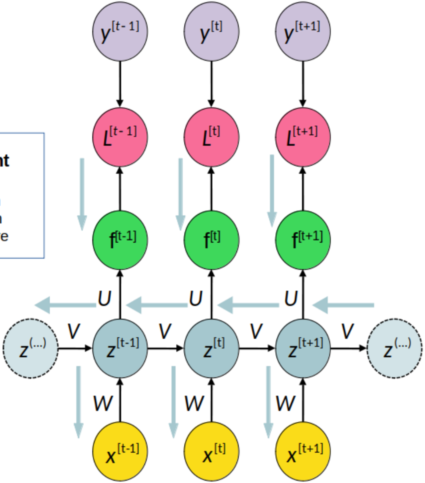


Figure: Vanishing gradients

LONG-TERM DEPENDENCIES

- Even if we assume that the parameters are such that the recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones.
- A more sophisticated solution is needed for the vanishing gradient problem in RNNs.
- The **vanishing gradient problem** is heavily dependent on the parameter initialization method, but in particular on the choice of the activation functions.
  - For example, the sigmoid maps a real number into a “small” range (i.e. $[0, 1]$).
  - As a result, large regions of the input space are mapped into a very small range.

LONG-TERM DEPENDENCIES

- Even a huge change in the input will only produce a small change in the output. Hence, the gradient will be small.
- This becomes even worse when we stack multiple layers of such non-linearities on top of each other (For instance, the first layer maps a large input to a smaller output region, which will be mapped to an even smaller region by the second layer, which will be mapped to an even smaller region by the third layer and so on..).
- We can avoid this problem by using activation functions which do not have the property of “squashing” the input.
- The most popular choice is obviously the Rectified Linear Unit (ReLU) which maps x to $max(0, x)$.
- The really nice thing about ReLU is that the gradient is either 0 or 1, which means it never saturates. Thus, gradients can’t vanish.

LONG-TERM DEPENDENCIES

- The downside of this is that we can obtain a “dead” ReLU. It will always return 0 and consequently never learn because the gradient is not passed through.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)
Deep Learning
http://www.deeplearningbook.org/

Andrej Karpathy (2015)
The Unreasonable Effectiveness of Recurrent Neural Networks
http://karpathy.github.io/2015/05/21/rnn-effectiveness/