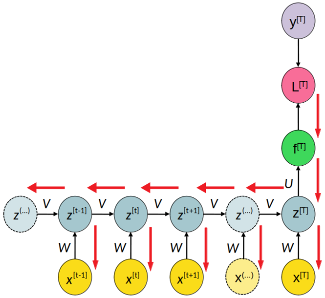
Deep Learning

Recurrent Neural Networks - Backpropogation



Learning goals

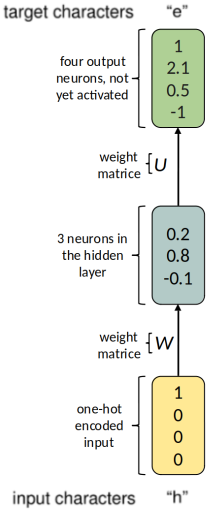
- How does Backpropagation work for RNNs?
- Exploding and Vanishing Gradients

SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL

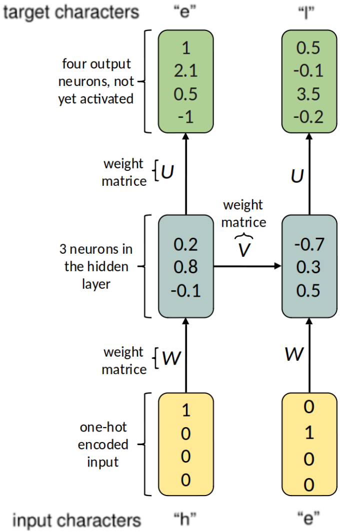
Task: Learn character probability distribution from input text

- Suppose we only had a vocabulary of four possible letters: “h”, “e”, “l” and “o”
- We want to train an RNN on the training sequence “hello”.
- This training sequence is in fact a source of 4 separate training examples:
    - The probability of “e” should be likely given the context of “h”
    - “l” should be likely in the context of “he”
    - “l” should also be likely given the context of “hel”
    - and “o” should be likely given the context of “hell”

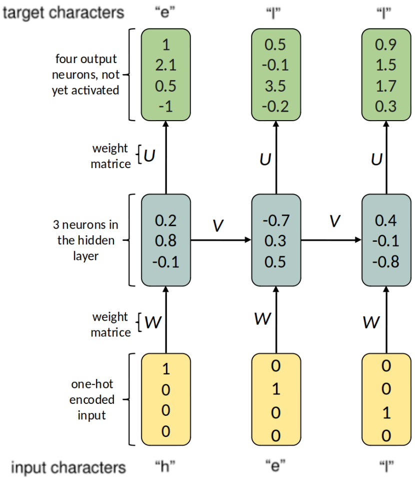
SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL



SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL

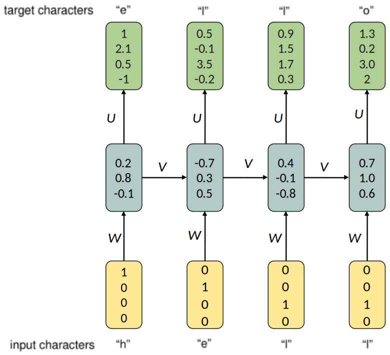


SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL



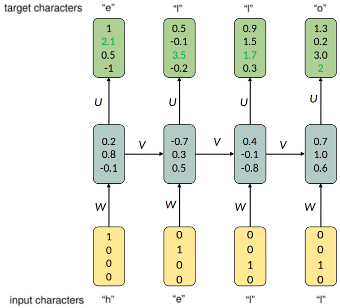
SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL

The RNN has a 4-dimensional input and output. The exemplary hidden layer consists of 3 neurons. This diagram shows the activations in the forward pass when the RNN is fed the characters “hell” as input. The output contains confidences the RNN assigns for the next character.



SIMPLE EXAMPLE: CHARACTER LEVEL LANGUAGE MODEL

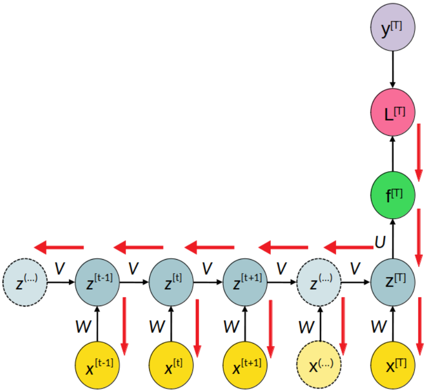
The RNN has a 4-dimensional input and output. The exemplary hidden layer consists of 3 neurons. This diagram shows the activations in the forward pass when the RNN is fed the characters “hell” as input. The output contains confidences the RNN assigns for the next character.



Our goal is to increase the confidence for the correct letters (green digits) and decrease the confidence of all others (we could also use a softmax activation to squash the digits to probabilities $\in [0, 1]$). How can we now train the network?

Backpropagation through time!

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 for an arbitrary RNN, we need to compute

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

LONG-TERM DEPENDENCIES

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

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

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

$$\vdots$$

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

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

LONG-TERM DEPENDENCIES

- In general, for an arbitrary time-step $i < t$ in the past, $\frac{d\mathbf{z}^{[t]}}{d\mathbf{z}^{[i]}}$ will contain the term $(\mathbf{V}^\top)^{t-i}$ (this follows from the chain rule).
- Based on the largest eigenvalue of $\mathbf{V}^\top$, the presence of the term $(\mathbf{V}^\top)^{t-i}$ can either result in vanishing or exploding gradients.
- This problem is quite severe for RNNs (as compared to feedforward networks) because the **same** matrix $\mathbf{V}^\top$ is multiplied several times. Click here
- As the gap between t and i increases, the instability worsens.
- It is thus 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

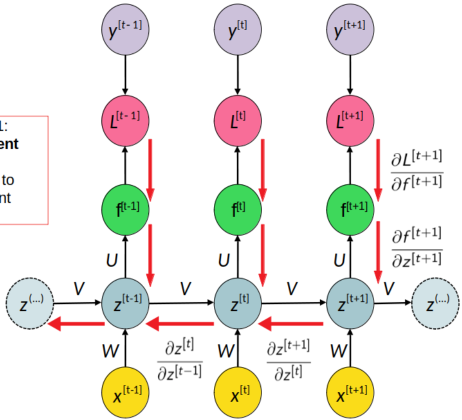


Figure: Exploding gradients

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

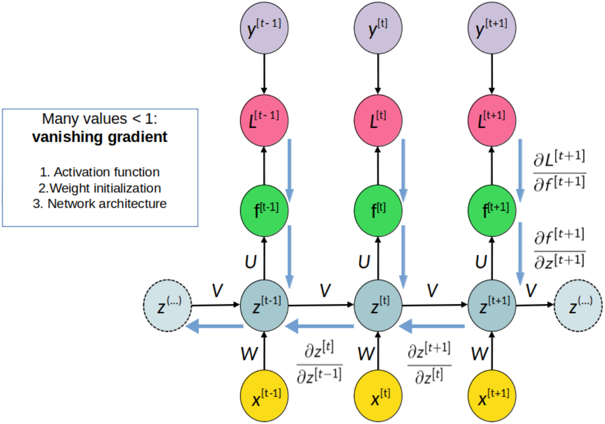


Figure: Vanishing gradients

LONG-TERM DEPENDENCIES

- Even for a stable RNN (gradients not exploding), there will be exponentially smaller weights for long-term interactions compared to short-term ones and a more sophisticated solution is needed for this vanishing gradient problem (discussed in the next chapters).
- The vanishing gradient problem heavily depends on the choice of the activation functions.
  - Sigmoid maps a real number into a “small” range (i.e. $[0, 1]$) and thus even huge changes 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.
  - We can avoid this problem by using activation functions which do not “squash” the input.
  - The most popular choice is ReLU with gradients being either 0 or 1, i.e., they never saturate and thus don’t vanish.
  - The downside of this is that we can obtain a “dead” ReLU.

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/