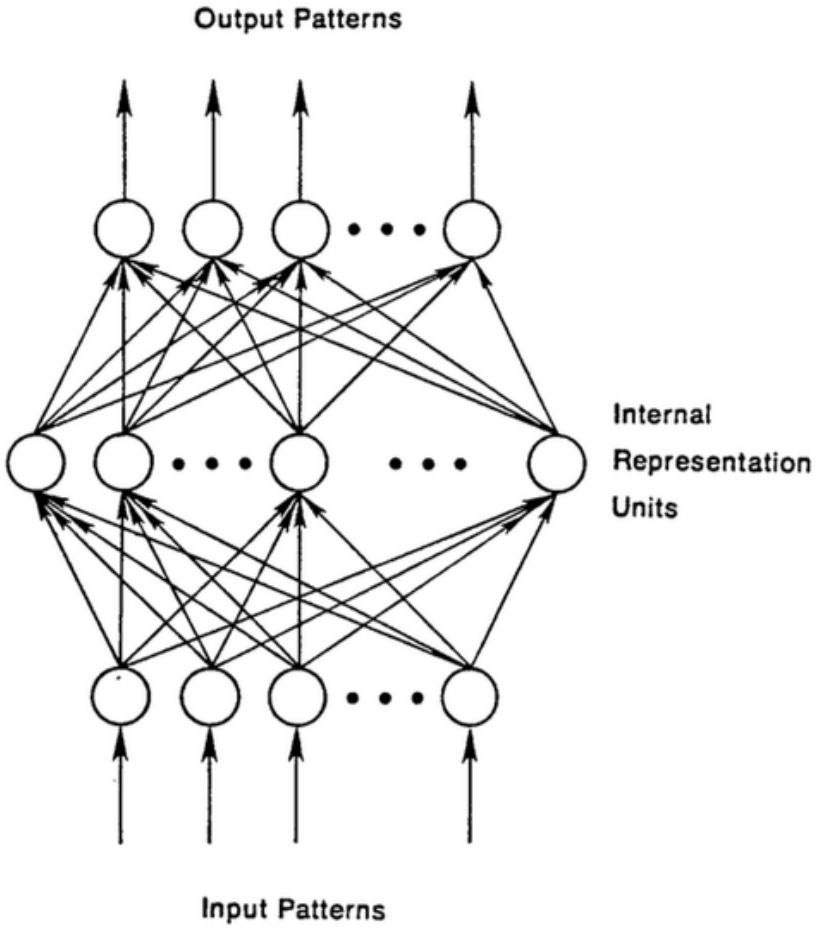
Введение в рекуррентные нейронные сети

Кнышов Александр.

Review of Feedforward Networks

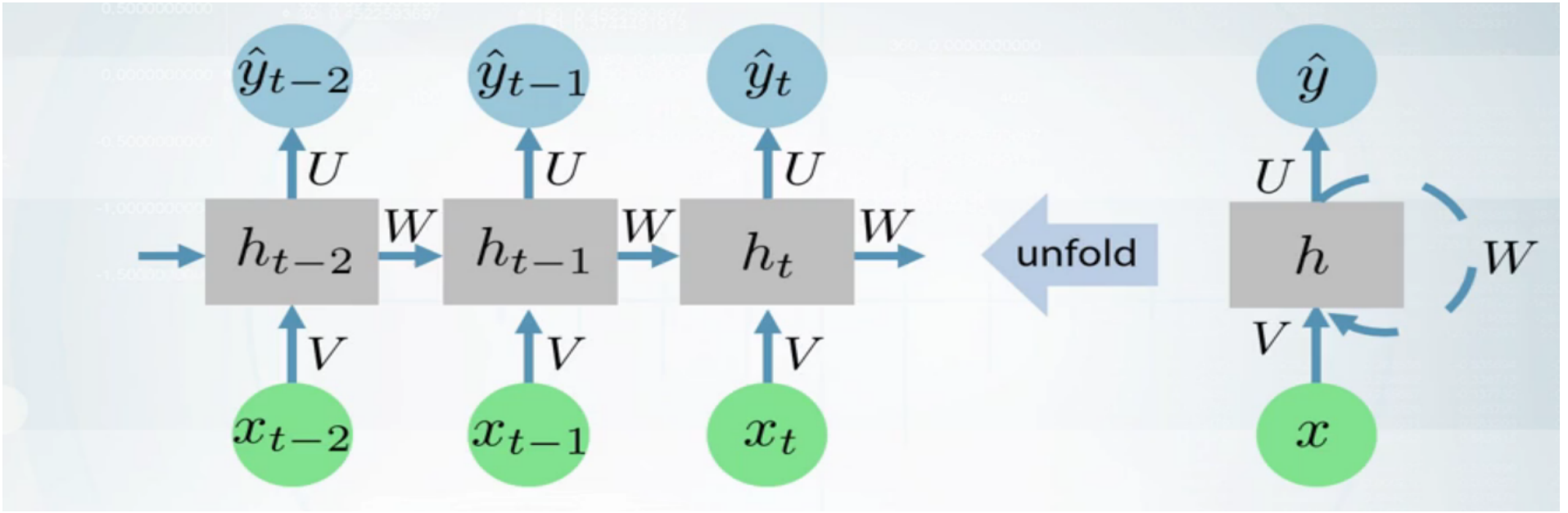


- Информация идет строго вперед, т.е не заходим в вершину дважды
- Тренируем на размеченных данных, до тех пор, пока не минимизируем ошибку
- Обучив параметры(матрицу весов), можем работать с неразмеченными данными
- Входные данные должны быть независимы

Почему feedforward NN нас не всегда устраивают?

- В реальной жизни мы запоминаем информацию и действуем относительно ее
- Для обычных нейросетей все входы считаются независимыми
- Хочется как-то использовать предыдущие знания для решения задачи

Recurrent Network(RNN)



x_t — вход в момент времени t

U, W, V — матрицы весов

$h_t = f(Vx_t + Wh_{t-1} + b_h)$ — скрытый слой

$\hat{y}_t = g(Uh_t + b_y)$ — выход в момент времени t

b_h, b_y — векторы сдвига

Пример функций:

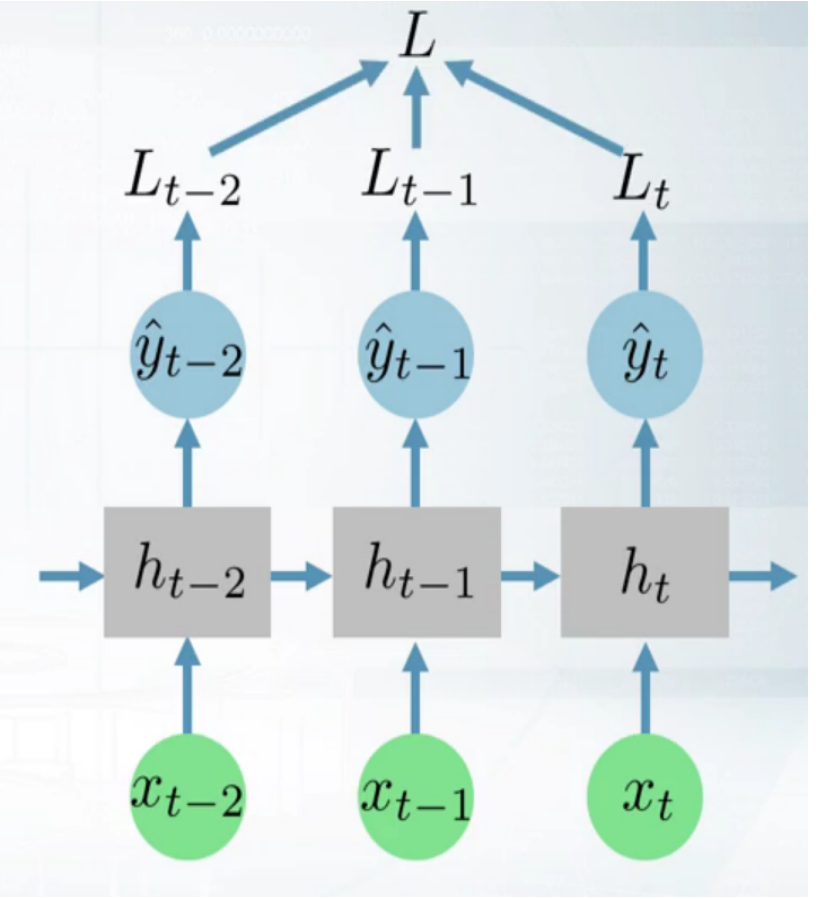
$f = \tanh, g = softmax$

Как обучать RNN?

В каждый момент времени у нас есть:

- y_t – *реальная метка*
- $\hat{y}_t$ – *наше предсказание*
- $L_t(\hat{y}_t, y_t)$ – *функция потерь*

Минимизируем функционал: $L(\hat{y}, y) = \sum_i L_i(\hat{y}_i, y_i)$



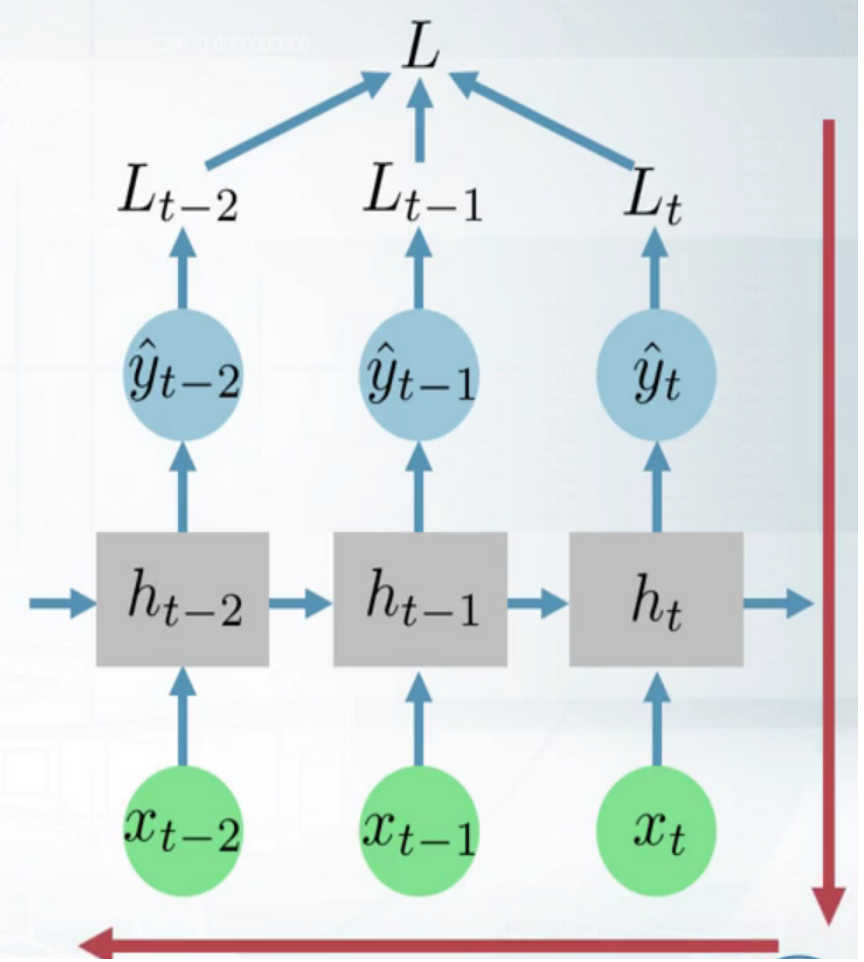
Backpropagation Through Time(BPPT)

Forward pass:

$$\hat{y}_t, h_t, L_t, L$$

Backward pass:

$$\frac{\partial L}{\partial V}, \frac{\partial L}{\partial U}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b_y}, \frac{\partial L}{\partial b_h}$$



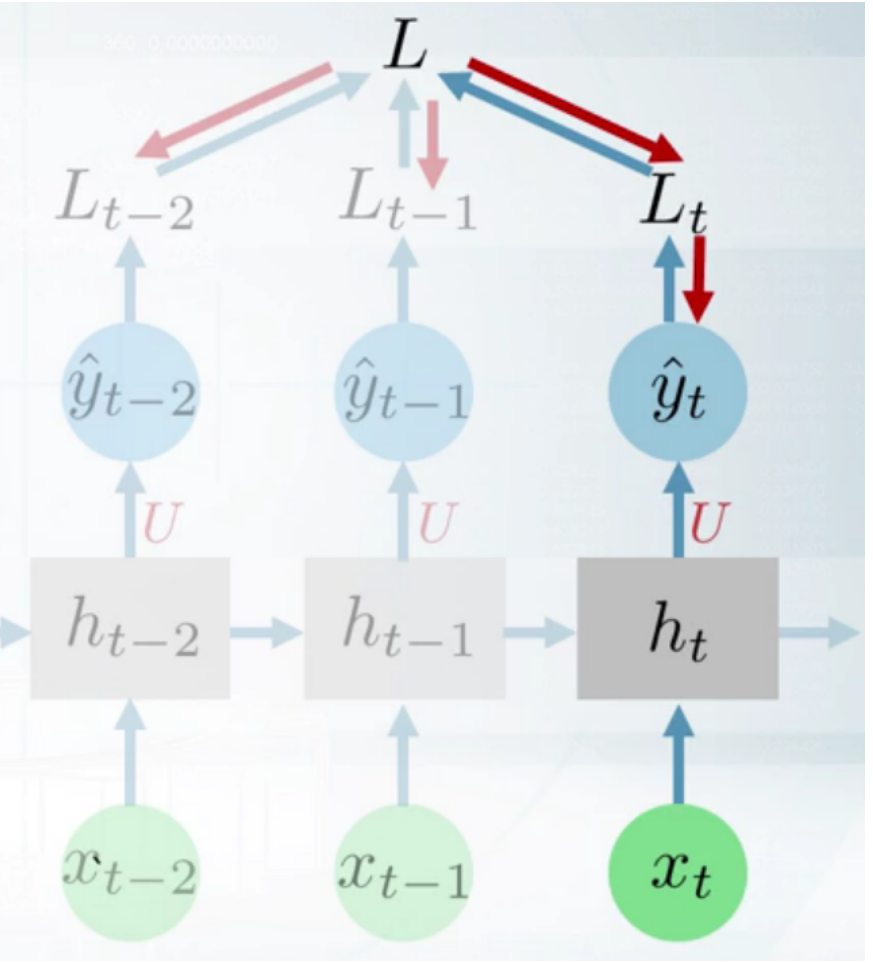
BPPT

$$\frac{\partial L}{\partial U} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial U}$$

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial U}$$

$$\hat{y}_t = g(U h_t + b_y)$$

Лишь одна зависимость



BPPT

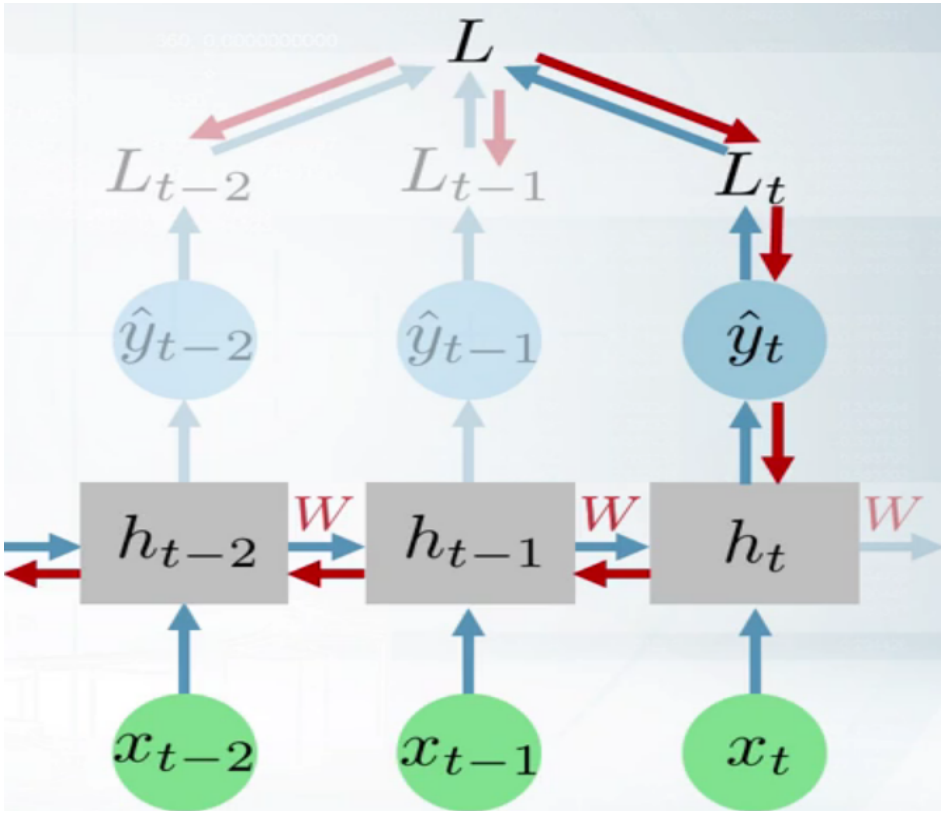
$$\frac{\partial L}{\partial W} = \sum_{i=0}^{T} \frac{\partial L_i}{\partial W}$$

$$h_t = f(Vx_t + Wh_{t-1} + b_h)$$

Тоже зависит от W

$$z = z(x(t), y(t))$$

$$\frac{dz}{dt} = \frac{dz}{dx}\frac{dx}{dt} + \frac{dz}{dy}\frac{dy}{dt}$$



$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\frac{\partial h_t}{\partial W} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W} + \frac{\partial h_t}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W} + \cdots\right) =$$

$$= \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

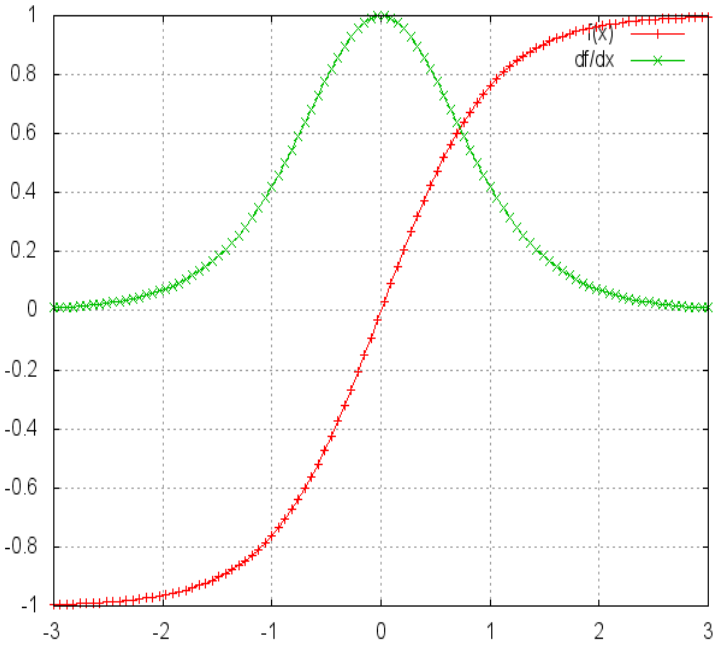
Недостатки BPTT.

- Затухающие/взрывающиеся градиенты
- Не можем поддерживать длинные зависимости
- Работает довольно долго

The vanishing gradient problem

$$\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\sum_{k=0}^{t}\left(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_k}{\partial W}$$

- Производная вектор-функции по вектору – Якобиан
- Может быть много маленьких производных
- Градиент уменьшается экспоненциально быстро
- Становится тяжело обучать нейросеть



The vanishing gradient problem, пути решения

- Хорошая инициализация матриц весов
- Регуляризация
- Использование ReLU, как активационной функции
- Long Short-Term Memory (LSTM)

Truncated BPTT

- Задаем два гиперпараметра: k1, k2
- Каждый раз, когда мы приняли k1 входов, запускаем BPTT, откатываясь не более чем на k2 слоев назад
- Обновляем параметры
- Canonical TBPTT: k1 = k2
- Хорошо работает при коротких зависимостях(например, слова в предложении)
- Работает быстрее, чем обычный BPTT
- Немного решает проблему затухающих/взрывающихся градиентов

Итог

- Vanilla RNN неплохо работают с короткими зависимостями
- Некоторые модификации делают их довольно полезными(особенно LSTM), позволяя использовать длинные зависимости
- Используются в задачах NLP(text generating, speech recognition, machine translation), в задачах компьютерного зрения(совместно с CNN)

Материалы.

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
https://www.coursera.org/learn/intro-to-deep-learning/lecture/zGHtr/simple-rnn-and-backpropagation
https://deeplearning4j.org/lstm.html