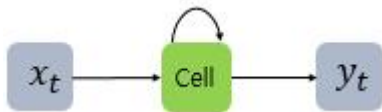


# RNN

Feed Forward Neural Network<sup>1)</sup> ≠ RNN

기존의 Neural Network은 맥락이 전혀 고려되지 않은 단순한 구조 >> hidden layer를 도입해 Speech Recognition, 언어모델링, 번역, 이미지캡셔닝 등.. 많은 문제를 해결

**RNN**은 hidden node에서 Activation Function을 통해 나온 결과값을 출력층 방향으로 보내면서, 동시에 다시 hidden node의 다음 계산의 입력으로 보내는 특징을 가진다.

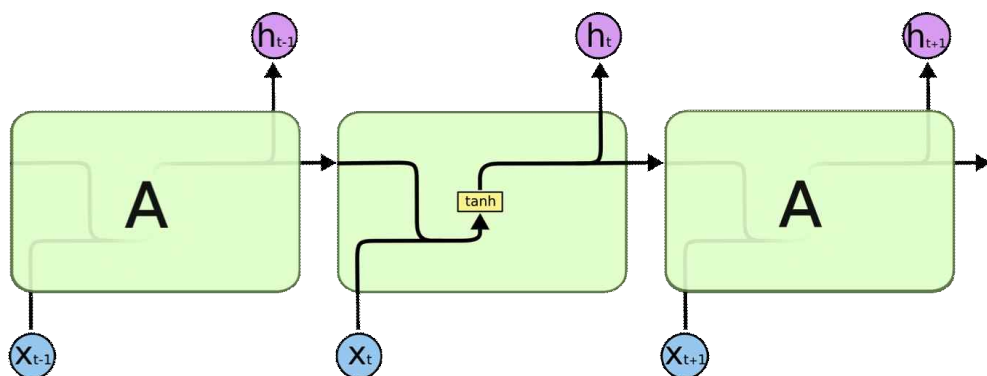


x는 입력층의 입력벡터, y는 출력층의 출력벡터

\*\* RNN에서는 hidden layer에서 활성화함수를 통해 결과를 내보내는 역할을 하는 노드를 셀(Cell)이라고 한다. 이 셀은 이전의 값을 기억하려고하는 일종의 메모리역할을 수행하므로 이를 **Memory-Cell** 또는 **RNN-Cell**이라고 표현한다.

hidden layer의 메모리셀(memory-cell)은 각각의 시점(time-step)에서 바로 이전 시점에서의 은닉층의 메모리 셀에서 나온 값을 자신의 입력으로 사용하는 재귀적(Recursive) 활동을 하고 있다.

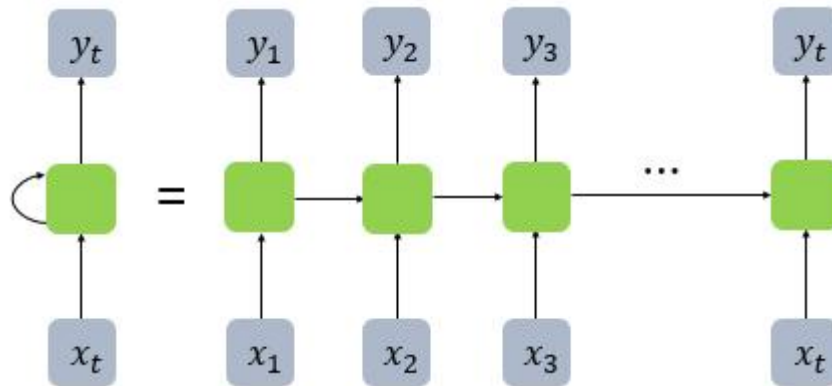
즉, 현재 시점 t에서의 메모리셀이 갖고있는 값은 과거의 메모리 셀들의 값에 영향을 받은 것 >> 메모리셀이 갖고 있는 이 값은? **은닉 상태(hidden state)**<sup>2)</sup>



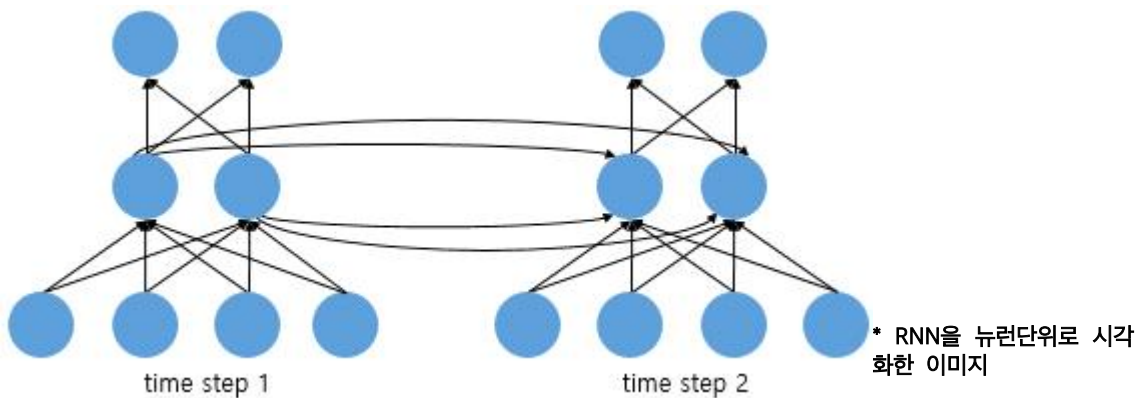
직전 데이터(t-1)와 현재 데이터(t)간의 상관관계를 고려해 다음의 데이터(t+1)를 예측하고자, 과거의 데이터도 반영한 신경망 모델을 만듦.

1) 은닉층에서 활성화함수를 지닌값은 오직 '출력층 방향'으로만 향하는 신경망

2) 메모리셀이 출력층 방향으로 또는 다음 시점인 t+1의 자신에게 보내는 값. 즉, t 시점의 메모리셀은 t-1 시점의 메모리셀이 보낸 은닉상태값을 t 시점의 은닉상태 계산을 위한 입력값으로 사용된다.

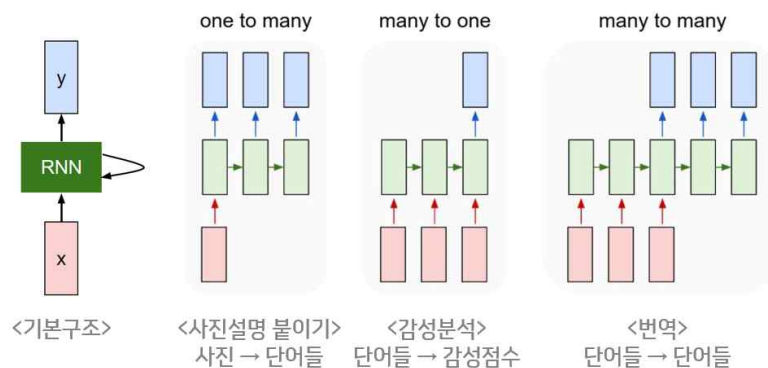


피드포워드신경망에서는 뉴런이란 단위를 사용했지만, RNN에서는 뉴런이란 단위보다는 입력층과 출력층에서는 입력벡터와 출력벡터, 은닉층에서는 은닉상태라는 표현을 주로 사용한다.



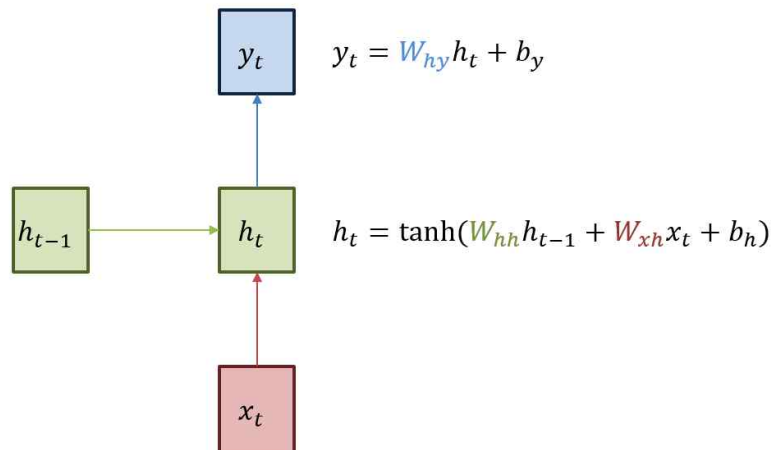
input\_dim = 4, hidden\_state = 2, output\_dim = 2, time\_step = 2  
RNN에선 2~3개의 layer를 쌓는 것이 일반적이다

## 구조



이와같이, RNN은 입력벡터(입력)와 출력벡터(출력)의 길이를 다르게 설계할 수 있으므로 다양한 용도로

사용할 수 있다. 위의 그림은 입력과 출력의 길이에 따라서 달라지는 RNN의 다양한 형태를 보여준다.  
(=시퀀스 길이와 관계없이 인풋과 아웃풋을 받아들일 수 있는 네트워크 구조이기 때문에 필요에 따라 다양하고 유연하게 구조를 만들 수 있다는 점이 RNN의 가장 큰 장점)



녹색박스 = 히든state / 빨간박스 = 인풋x / 파란박스 = 아웃풋y

현재 상태의 히든state  $h_t$ 는 직전 시점의 히든 state  $h_{t-1}$ 를 받아 갱신된다

아웃풋  $y_t$ 는  $h_t$ 를 전달받아 갱신되는 구조이며 히든state의 활성화함수는 비선형함수인 하이퍼볼릭탄젠트 (tanh)<sup>3)</sup>를 사용

이를 식으로 표현하면 다음과 같습니다.

은닉층 :  $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 :  $y_t = f(W_y h_t + b)$

단,  $f$ 는 비선형 활성화 함수 중 하나.

- 3) Vanishing Gradient problem 때문이다. 또한 활성화함수를 통해 기울기를 선형으로 유지시키기 위함이며, 2차 도함수가 긴 range동안 지속적으로 유지될수 있는 함수가 필요하기 때문이다. 값을 맵핑하는 관점에서 정규화하는 데에 더 강점을 보인다.

Graident 문제를 위해 ReLU함수를 사용할 수 있는데 왜 사용하지 않을까?

ReLU함수의 식은 매우 간단한데, Tanh와 Sigmoid함수는 복잡한 삼각법에의한 함수이며 다중곱셈을 시행한다. 단기의 딥러닝에선 아웃풋과 인풋간의 근사한 관계를 찾아야한다. 그러나, ReLU는 결과를 이러한 결과를 내는데 충분히 복잡하지 않은 함수. 또한 임의의 방식에 따라 값이 발산되는 경향이 있고, 선형 unit을 강하게 규제시키는 경향이 존재한다. 따라서 RNN모델에선 좀 더 복잡한 공식을 가지면서 -1에서 1의 범위를 가지는 함수인 Tanh를 채택하는 것이다.

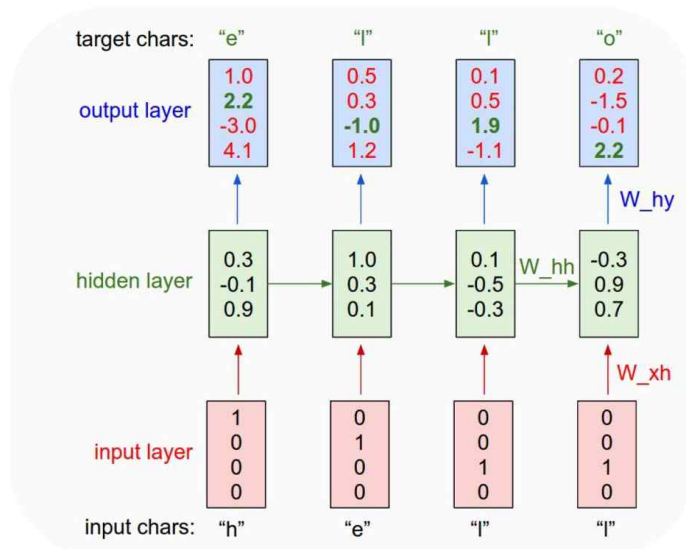
Tanh는 중간지점을 다루는 관점에서 더 괜찮은 값들을 도표화하는데 적합한 함수라 할 수 있다. 이는 재귀적 역학에서 매우 중요한 포인트인데 Monte Carlo 컨버전스에서 보이듯이, 분수함수(sigmoid)는 천천히 점진적으로 값이 갱신되며 true value값으로 점차 향할 것이다.

Tanh는 0 주변의 균형잡힌 값이 있는데 ReLU는 그렇지못하다. 이것이 의미하는 바는 무엇일까? ReLU함수를 취하게되면, 음수값의 weight는 0으로 리턴받을 것을 의미합니다. ReLU함수는 그저 빠르게 음수값을 0으로 만들려는 경향이 있고 이를 흔히 Deep Network에선, '**Dead ReLU Problem**'이라고 한다. 이 문제는 RNN에서 gradient값을 점점 더 낮추는 문제(vanishing gradient problem)를 야기할 뿐만 아니라, 이전의 타임스텝으로부터 전해 들어온 graident값을 취할 것이다. 필연적으로, RNN은 기존의 feed-forward network보다 몇몇 시간을 거쳐 더 깊게 들어가야한다. 이러한 과정에서 ReLU 뉴런은 빠르게 죽어나갈것이며 전체 뉴런이 동작하지 않게 될 수도 있다.

RNN이 학습하는 parameter값은 다음과 같다

- 1) 입력층(Input  $x$ )를 히든레이어  $h$ 로 보내는  $W_{xh}$
- 2) 이전 히든레이어  $h$ 에서 다음 히든레이어  $h$ 로 보내는  $W_{hh}$
- 3) 히든레이어  $h$ 에서 Output  $y$ 로 보내는  $W_{hy}$

이 값들은 **모든 시점에서 값을 동일하게 공유(shared weights)**한다. 만약 은닉층이 2개 이상일 경우에는 은닉층 2개의 가중치는 서로 다르게 될 것이다.



학습데이터의 글자는 h, e, l, o 4개 >> one-hot-vector >> [1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]

$x_1$ 은 [1,0,0,0] >>  $h_1$  [0.3,-0.1,0.9] 생성 >>  $y_1$  [1.0,2.2,-3.0,4.1] 생성 // 마찬가지로 두 번째, 세 번째, 네 번째 단계들도 모두 갱신, 이 과정을 **순전파(Foward Propagation)**라고 칭한다







RNN도 정답을 필요로하는데, 모델에 정답을 알려줘야 모델이 Parameter를 적절히 갱신해 나갈 것. 이 경우엔 바로 다음 글자가 정답, 예를들어 'h'의 다음정답은 'e' 그 다음정답은 'l' 그 다음은 'l' 그 다음은 'o'가 되겠다.  $y_1$ 에 진한 녹색으로 표시된 숫자가 있는데 정답에 해당하는 인덱스를 의미한다 >> **이 정보를 바탕으로 역전파(Backpropagation)을 수행해 parameter값들을 갱신해 나간다**


이와 같이, 다른 신경망과 마찬가지로 RNN 역시 경사 하강법(Gradient Descent)과 오차 역전파(backpropagation)를 이용해 학습한다. 정확하게는 시간 흐름에 따른 작업을 하기 때문에 역전파를 확장한 **BPTT<sup>4)</sup>(Back-Propagation Through Time)를 사용해서 학습한다.**

4) BPTT는 필연적으로 gradient가 발산(Exploding), 소실(Vanishing)문제가 발생. 에러(학습해야하는 분량)가 이전의 시간으로 전달될수록 점차 희석될 것이다. 에러가 학습 Parameter(weight, bias)에 잘 전달되지 않으면 모델링이 제대로 이루어질 수 없음. 반대로 필요하지 않은 이전 정보들까지도 학습하게 돼서 예측이 잘 되지 않을 수 있다.





\*\* **Truncated BPTT** - 장기기억을 적절한 단위로 쪼개서 학습시키자는 것이 아이디어이며, 이는 앞서 기술했던 vanishing and exploding graident problem을 해결하면서, 동시에 특정 인풋에 적합한 시간window를 사용하는 것

## Activation Function








[Long Short-term Memory](#) [Artificial Neural Networks](#) [Machine Learning](#) 

### In an LSTM unit, what is the reason behind the use of a tanh activation?





[Answer](#) [Follow · 30](#) [Request](#)    

### Why does an LSTM with ReLU activations diverge?

[Answer](#) [Follow · 12](#) [Request](#)    

[Recurrent Neural Networks \(RNNs\)](#) [Artificial Neural Networks](#) [Deep Learning](#)  
[Artificial Intelligence](#) [+1](#) 

### Why are tanh activation functions more common than ReLU in Recurrent Neural Networks (RNNs)?

[Answer](#) [Follow · 15](#) [Request](#)    

---

## Use ReLU with MLPs, CNNs, but Probably Not RNNs

The ReLU can be used with most types of neural networks.

It is recommended as the default for both Multilayer Perceptron (MLP) and Convolutional Neural Networks (CNNs).

The use of ReLU with CNNs has been investigated thoroughly, and almost universally results in an improvement in results, initially, surprisingly so.

---

##truncated BPTT에선 BPTT를 얼마나(시간적으로) 긴 단위에 걸쳐서 할 것인지(=window\_size), 그리고 얼마나 좁하게 할 것인지(=shifting\_size)를 설정할 수 있다. 해당 파라미터 값은 각각 k1, k2라 표현된다. 즉, 얼마나 BPTT를 자주 할지가 k1이고, 얼마나 뒤까지 에러를 전달할지가 k2이다.

“For example, the rectified linear function  $g(z) = \max\{0, z\}$  is not differentiable at  $z = 0$ . This may seem like it invalidates  $g$  for use with a gradient-based learning algorithm. In practice, gradient descent still performs well enough for these models to be used for machine learning tasks.

— Page 192, [Deep Learning](#), 2016.

“ReLU함수는 gradient-based learning algorithm에선 만족할만한 성능을 뽑아내지 못 할 것”

“At first sight, ReLUs seem inappropriate for RNNs because they can have very large outputs so they might be expected to be far more likely to explode than units that have bounded values.

— [A Simple Way to Initialize Recurrent Networks of Rectified Linear Units](#), 2015.

Nevertheless, there has been some work on investigating the use of ReLU as the output activation in LSTMs, the result of which is a careful initialization of network weights to ensure that the network is stable prior to training. This is outlined in the 2015 paper titled “[A Simple Way to Initialize Recurrent Networks of Rectified Linear Units](#).”

“RNN은 매우 큰 output을 가지고있기에 ReLU함수는 부적절해보인다. 그래서 값이 튀는 현상이 나타날 것으로 기대된다”

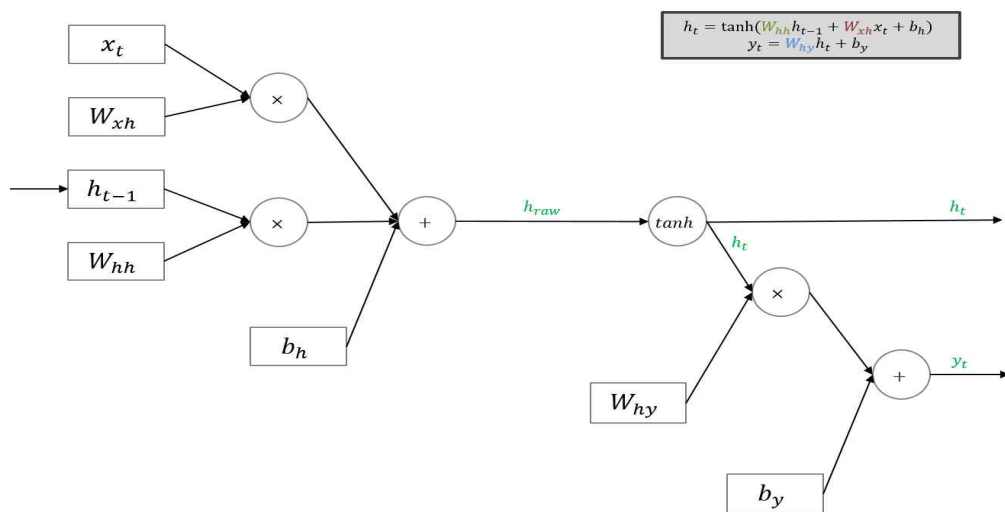
구조적인 접근이 필요한가? >> 기울기소실문제를 다루는 것은 feed forward deep net과는 다소 차이점이 존재한다. 이 문제를 특별히 다양한 게이트와 메모리셀이 사용되는 RNN계열 모델의 네트워크 구조를 통해 풀 수 있다. 기울기소실문제에 대해 ReLU를 사용함으로 다룬적이 있었다. ReLU는 좀더 일반적인 활성화함수인데 deep net에서 Tanh와 Sigmoid를 사용했을 때, saturation problem이 발생했다. 이는 앞의 pass가 포화상태일 때 backprop 기울기가 반드시 0으로 만들어지는 문제가 나타났다. ReLU는 이런 문제가 없고, 두 함수보다 덜 복잡한 계산식을 가지고 있으며 따라서 어떠한 경우에는 더 빠르고 잘 작동할 수 있다.

보통 sigmoid와 tanh가 특히 RNN모델에서 BPTT 알고리즘을 사용해 학습을 할 때, 문제가 많다고 한다. LSTM에선 게이트를 통해 output을 산출하기에 기울기소실문제가 나타나지 않는 것이다.

various, yet there is no comprehensive recipe for choosing which heuristic may be the best problem and data dependent. However, there are heuristics and some underlying theory that can help guide a practitioner to make better choices.

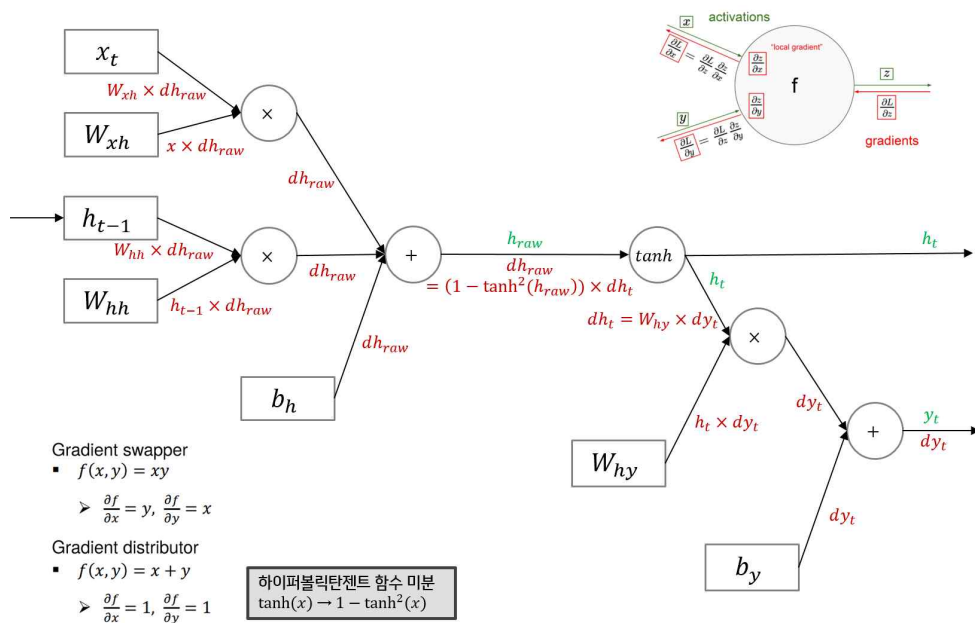
In the first section below we introduce standard backpropagation and discuss

## RNN의 순전파



## RNN의 역전파





여기서 RNN의 단점이 드러난다. RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하되는 것으로 알려져 있다, 이를 **vanishing gradient problem**이라고 한다. 이 문제를 극복하기 위해 고안된 것이 LSTM이고, LSTM은 RNN의 hidden state에 cell-state를 추가한 구조.

## 문제점

- 시간을 많이 거슬러 올라가면(long-term) 경사를 소실하는 문제가 발생(Gradient Vanishing)
  - 선형함수가 아닌 비선형함수를 Activation function으로 쓰는 것과 비슷한 이유로, 초기값에 따라서 과거데이터를 계속 곱할수록 작아지는 문제가 발생하게 된다, LSTM은 구조를 개선하여 이 문제를 해결했다<sup>5)</sup>
  - Backward Flow가 안좋기에 Backpropagation을 진행할 때, vanishing gradient문제가 발생
  - Whh에 동일한 수를 계속해서 곱하면 두가지 발생 : Exploding, Vanishing
    - Case 1) 고유값(eigenvalue) > 1 : gradient will explode
    - Case 2) 고유값(eigenvalue) < 1 : gradient will vanish
- 예방법
  - 1) Gradient Clipping (ex> -5~5의 범위를 설정해 놓은 후, 범위 초과시 모두 제거(clip))

5) Quoc V. Le, Navdeep Jaitly, Geoffrey E. Hinton의 'A Simple Way to Initialize Recurrent Networks of Rectified Linear Units'에 나와있는 연구에 의하면, **활성함수를 ReLU로 사용하고 가중치를 단위행렬로 초기화**하면 long-term을 학습시킬 수 있다고 제시.



## 2) LSTM

- 장기 의존성(Long-Term Dependency) 문제 ( 많은 메모리가 사용되는 것이 필연적 )
  - Gap(time-step에 따라)이 점점 늘면 늘수록 RNN은 정보를 연결시켜 학습할 수 없게 된다
  - 수치로 설명해보면 RNN은 1보다 큰 값들이 계속 발생하면 지속적으로 곱해져 발산하게 되고 1보다 계속 작은 값을 곱하면 0으로 수렴해 사라지게 된다. 이렇게 되면 RNN은 짧은 기억은 가능하지만 긴 기억은 기억하지 못하는 문제가 발생한다.
  - 이론적으로, RNN모델은 파라미터값을 조정해 장기 의존성 문제를 당연히 다룰 수 있으나, 안타깝게도 실전에선 전혀 그렇지 못하다
  - 이전 정보만 참고하는 것이 아니라, 그 전 정보를 고려해야 하는 경우(longer-term)가 있는데 시퀀스가 있는 문장에서 문장간의 간격이 커질수록, RNN은 두 정보의 맥락을 파악하기 어렵다. 즉, 길어진 데이터를 처리하면서 Input\_data의 초기 타임스텝을 점점 잊어버리게 되는 것이다.
- RNN은 Time Sequential하게 과거의 데이터를 지속적으로 꺼내서 쓴다. 즉, 과거의 값들이 끊임없이 재귀적으로 사용되기 때문에 -1 ~ 1 사이의 값으로 Normalization이 반드시 필요. 여기서 ReLU함수를 사용하게 되면 0이상의 값에선 input값을 그대로 가져가게 되고 time step이 쌓일수록 발산할 확률이 높아지게 된다. 그렇기에, output의 값을 Normalization 해줄 수 있는 함수인 sigmoid, tanh가 주로 사용된다. Vanishing Gradient 문제를 잡기 위해 사용했던 ReLU는 RNN에서는 LSTM 및 tanh로 해결했으므로, 굳이 발산할 가능성이 있는 ReLU를 쓸 필요가 없고, 쓰면 발산한다.

## 요약

- RNN모델은 높은 유연성을 가진 모델이다
- Vanilla RNN은 간단하긴하나 잘 작동하지않는다(현업에서 사용x), 상가작용(additive interaction)이 기울기 흐름을 향상시켜주기에 보통 LSTM이나 GRU모델을 많이 사용한다
- RNN에서의 기울기 오차역전파는 폭발이나 손실을 일으킨다

## 다양한 RNN 모델

### IRNN

『A SimpleWay to Initialize Recurrent Networks of Rectified Linear Units』은 ReLU 함수를 사용함으로써 tanh activation function을 사용하면 나타날 수 있는 Vanishing and Exploding Gradient Problem에 대한 해결책을 제시하였다.

기존 RNN 모델은 BPTT를 사용함으로써 에러 미분을 계산하는과정에서 어려움이 나타났었고 V&E Gradient Problem은 장기의존적인 학습에 무리가 많았다. 그러면서 이 문제를 해결하기 위해 LSTM으로 변화가 나타났고, 이는 구조적으로 Stochastic gradient descent >> BPTT를 통한 hidden unit 생성으로 바뀌었다.

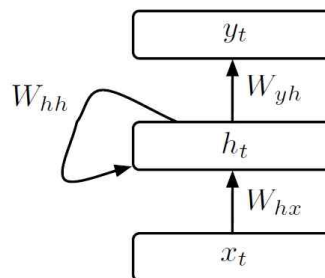


Figure 1: Schematic diagram of a simple RNN network

첫 번째, 비선형함수인 ReLU함수 사용

두 번째, 은닉레이어 기울기값인 Whh(Weight Matrix)를 단위행렬로 초기화해주고 bias를 0으로 세팅해준다.

현재의 은닉벡터는 이전의 은닉벡터 값을 단순히 취하게 한 후에 현재의 Input에 추가해주고, 모든 음수값은 0으로 치환된다. 이러한 Input의 부재속에, ReLU함수를 활성화하고 단위행렬을 초기값으로 지정해준다. 이를통해, 근본적으로 RNN이 가지는 long-term dependency의 문제를 극복 할 수 있을 것이며 LSTM이 가지는 forget gate의 작동 메커니즘과 비슷한 원리.

```

class IRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(IRNN, self).__init__()
        self.hidden_size = hidden_size
        self.rnn = nn.RNN(input_size, hidden_size,
                           nonlinearity='relu', batch_first=True, bias=True)
        self.output_weights = nn.Linear(hidden_size, output_size)

        # Parameters initialization
        self.rnn.state_dict()['weight_hh_l0'].copy_(torch.eye(hidden_size))
        self.rnn.bias_ih_l0.data.fill_(0)
        self.rnn.bias_hh_l0.data.fill_(0)
        self.rnn.state_dict()['weight_ih_l0'].copy_(
            torch.randn(hidden_size, input_size) / hidden_size)

    def forward(self, inp):
        _, hnn = self.rnn(inp)
        out = self.output_weights(hnn[0])
        return out

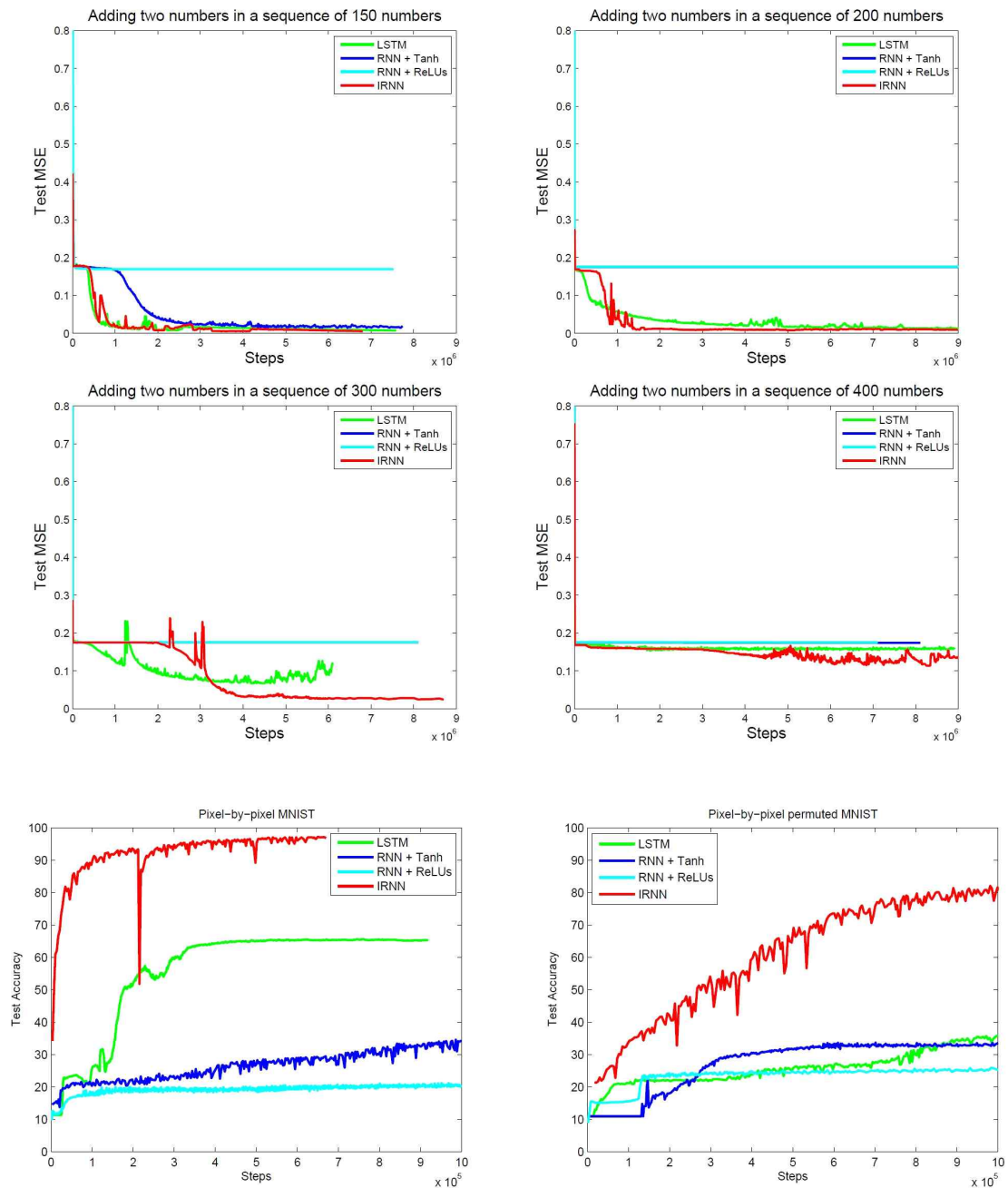
```

이러한 일련의 과정을 해당 논문에서 **IRNN**으로 칭하였고 이는 상당한 performance를 뽑아낸다고 주장한다. 또한 더 적은 장기 의존성을 가지는 경우, 적은 스칼라값으로 조정된 단위행렬은 더 효율적으로 작동한다는 사실도 밝혔다. 이러한 메커니즘은 LSTM의 forget gate가 메모리를 좀 더 빠르게 손실하기 위한 방법과 같은 이치임을 밝혔다.

더불어, LSTM과 비교해서 비교적 적은 튜닝, 그리고 더 단순한 메커니즘을 통해 더 좋은 성능을 뽑아낼 수 있다

## Experiment

1. LSTM ( Setting to a higher initial forget gate bias for long-term dependency )
2. RNN ( tanh )
3. RNN ( ReLU + Gaussian Initialization )
4. IRNN



두 번째 결과지는 MNIST 데이터를 활용한 benchmark인데 MNIST는 pixel to pixel이기 때문에 784(28 \* 28) 번 time-step 하기 때문에 엄청난 장기 의존성 문제가 나타날것. (This is therefore a huge long range dependency problem because each recurrent network has 784 time steps)

해당 문제를 더 심화시키기위해, 본 연구에선 pixel을 무작위하게 permutation하도록 조정한 상태로 학습에 들어갔다. 그럼에도 불구하고, IRNN은 LSTM보다 월등한 성능을 뽑아낼 수 있었다.

### CW-RNN (clockwork RNN)

Koutník, J., Greff, K., Gomez, F., and Schmidhuber, J.의 논문 『A Clockwork RNN』에서 제안하는 CW-RNN을 통해 기존 RNN에서 발생하는 장기기억 의존성의 V&E Gradient 문제를 해결했다. CW-RNN의 은닉레이어를 분리된 모듈에 나눔으로써, 다른 clock speed를 적용해 작동시킨다. 이 방법으로 CW-RNN은 좀 더 효율적으로 학습할 수 있다.

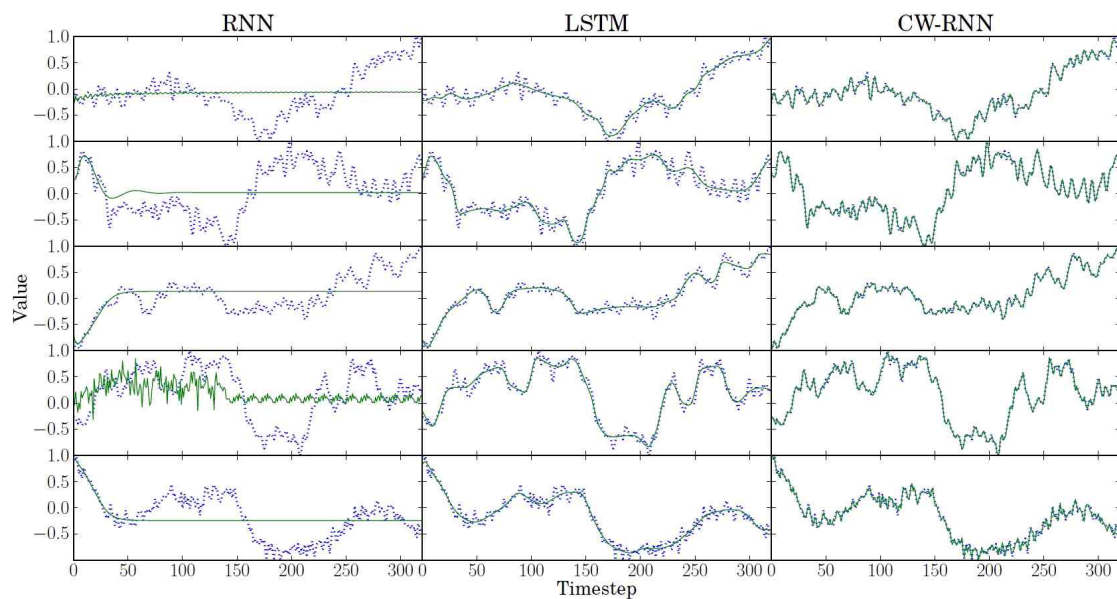


Figure 5. Output of the best performing network (solid, green) compared to the target signal (dotted, blue) for each method (column) and each training sequence (row). RNN tends to learn the first few steps of the sequence and then generates the mean of the remaining portion, while the output of LSTM resembles a sliding average, and CW-RNN approximates the sequence much more accurately.