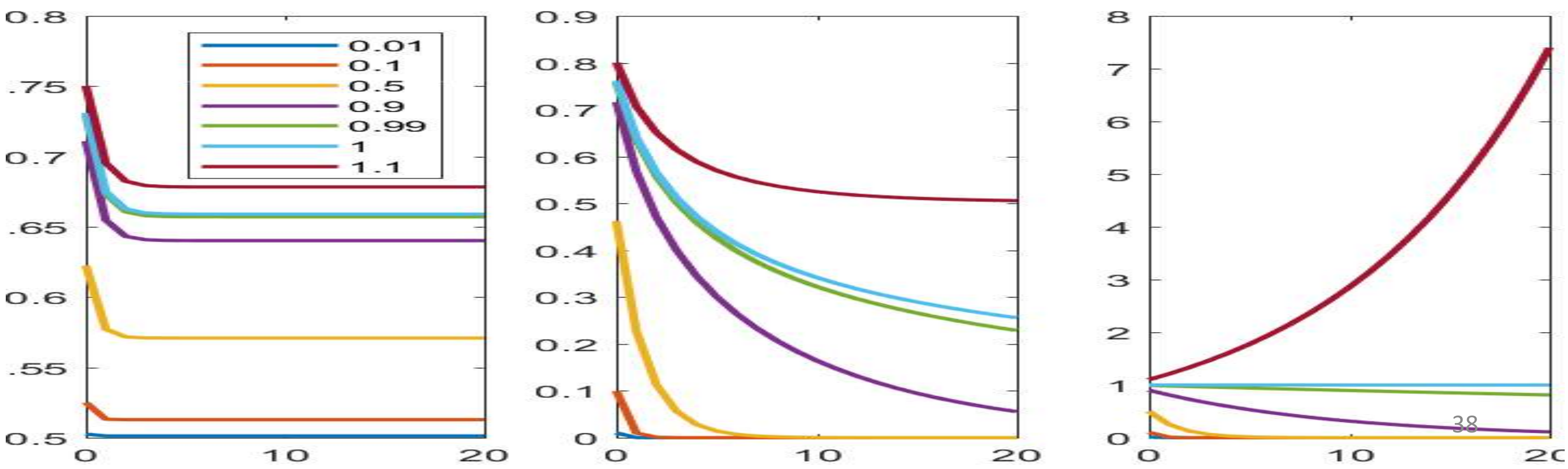
How about non-linearities (scalar)

$$h(t) = f(wh(t-1) + cx(t))$$

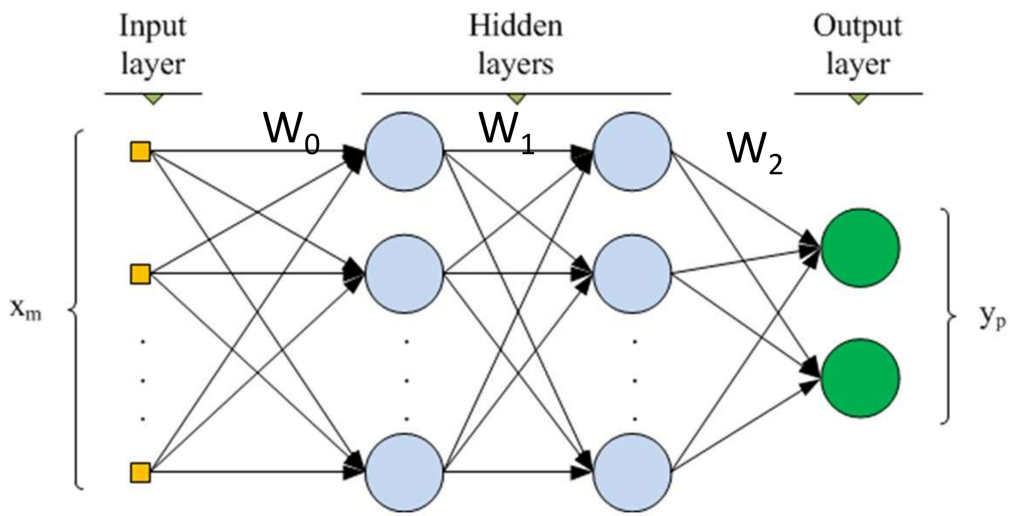
- The behavior of scalar non-linearities
- Left: Sigmoid, Middle: Tanh, Right: Relu
  - Sigmoid: Saturates in a limited number of steps, regardless of w
    - To a value dependent only on w (and bias, if any)
    - Rate of saturation depends on w
  - Tanh: Sensitive to w, but eventually saturates
    - "Prefers" weights close to 1.0
  - Relu: Sensitive to w, can blow up



The vanishing gradient problem for deep networks

- A particular problem with training deep networks..
  - (Any deep network, not just recurrent nets)
  - The gradient of the error with respect to weights is unstable..

Some useful preliminary math: The problem with training deep networks



- A multilayer perceptron is a nested function

$$Y = f_N\left(W_{N-1} f_{N-1}\left(W_{N-2} f_{N-2}\left(\ldots W_0 X\right)\right)\right)$$

- W_k is the weights *matrix* at the k[th] layer
- The *error* for X can be written as

$$Div(X) \ = D\left(f_N\left(W_{N-1} f_{N-1}\left(W_{N-2} f_{N-2}\left(\ldots W_0 X\right)\right)\right)\right)$$

Training deep networks

- Vector derivative chain rule: for any $f\big(Wg(X)\big)$:

$$\frac{df\big(Wg(X)\big)}{dX} = \frac{df\big(Wg(X)\big)}{dWg(X)}\frac{dWg(X)}{dg(X)}\frac{dg(X)}{dX}$$

Poor notation

$$\text{Let } Z = Wg(X)$$
$$\nabla_X f = \nabla_Z f . W . \nabla_X g$$

- Where
  - $\nabla_Z f$ is the *jacobian* ***matrix*** of $f(Z)$w.r.t Z
    - Using the notation $\nabla_Z f$ instead of $J_f(z)$ for consistency

Training deep networks

- For

$$Div(X) = D\left(f_N\left(W_{N-1}f_{N-1}\left(W_{N-2}f_{N-2}(\ldots W_0X)\right)\right)\right)$$

- We get:

$$\nabla_{f_k}Div = \nabla D.\nabla f_N.W_{N-1}.\nabla f_{N-1}.W_{N-2}\ldots\nabla f_{k+1}W_k$$

- Where
  - $\nabla_{f_k}Div$ is the gradient $Div(X)$ of the error w.r.t the output of the kth layer of the network
    - Needed to compute the gradient of the error w.r.t W_{k-1}
  - ∇f_n is *jacobian* of $f_N()$ w.r.t. to its current input
  - All blue terms are matrices
  - All function derivatives are w.r.t. the (entire, affine) argument of the function

Training deep networks

- For

$$Div(X) \ = D\left(f_N\left(W_{N-1}f_{N-1}\left(W_{N-2}f_{N-2}(\ldots W_0X)\right)\right)\right)$$

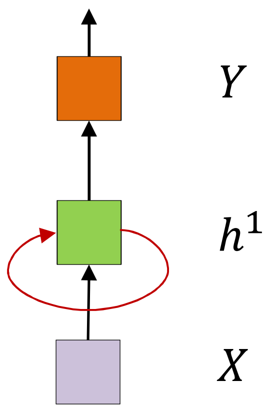
- We get:

$$\nabla_{f_k}Div = \nabla D.\nabla f_N.W_{N-1}.\nabla f_{N-1}.W_{N-2} \ldots \nabla f_{k+1}W_k$$

- Where
  - $\nabla_{f_k}Div$ is the gradient $Div(X)$ of the error w.r.t the output of the kth layer of the network
    - Needed to compute the gradient of the error w.r.t W_{k-1}
  - ∇f_n is *jacobian* of $f_N()$ w.r.t. to its current input
  - All blue terms are matrices

Lets consider these Jacobians for an RNN (or more generally for any network)

The Jacobian of the hidden layers for an RNN



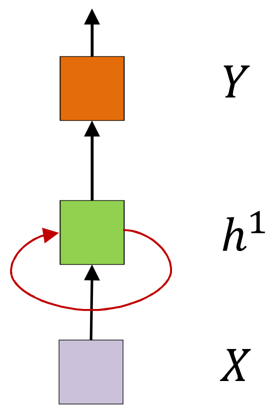
$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$

$$h_i^{(1)}(t) = f_1\left(z_i^{(1)}(t)\right)$$

- $\nabla f_t()$ is the derivative of the output of the (layer of) hidden recurrent neurons with respect to their input
  - For vector activations: A full matrix
  - For scalar activations: A matrix where the diagonal entries are the derivatives of the *activation* of the recurrent hidden layer

The Jacobian

$$h_i^{(1)}(t) = f_1\left(z_i^{(1)}(t)\right)$$

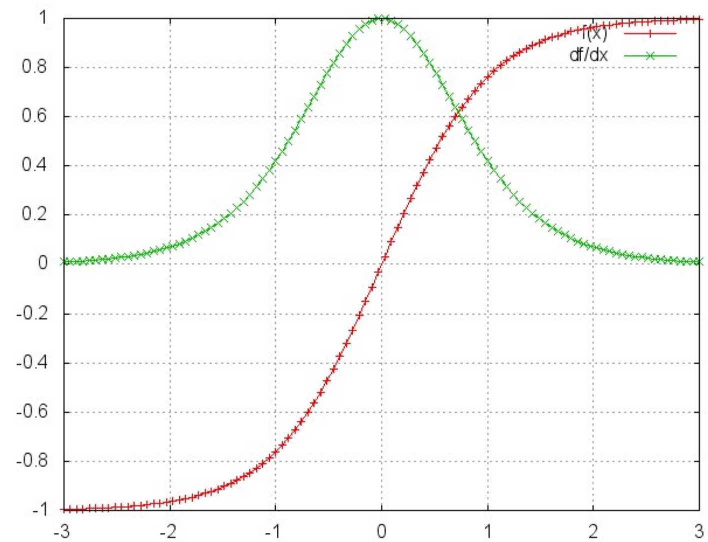


$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$

- The derivative (or subgradient) of the activation function is always bounded
  - The diagonals (or singular values) of the Jacobian are bounded
- There is a limit on how much multiplying a vector by the Jacobian will scale it

The derivative of the hidden state activation

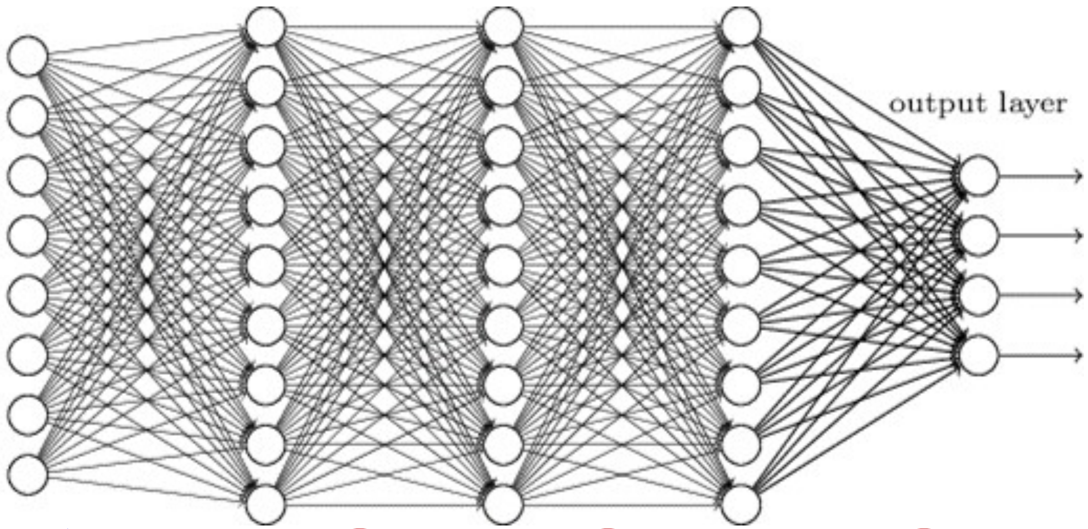
$$\nabla f_t(z_i) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \cdots & 0 \\ 0 & f'_{t,2}(z_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'_{t,N}(z_N) \end{bmatrix}$$



- Most common activation functions, such as sigmoid, tanh() and RELU have derivatives that are always less than 1
- The most common activation for the hidden units in an RNN is the tanh()
  - The derivative of $\tanh()$is never greater than 1 (and mostly less than 1)

- **Multiplication by the Jacobian is always a *shrinking* operation**

Training deep networks



$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \cdots \nabla f_{k+1} W_k$$

- As we go back in layers, the Jacobians of the activations constantly *shrink* the derivative
  - After a few layers the derivative of the divergence at any time is totally "forgotten"

What about the weights

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \ldots \nabla f_{k+1} W_k$$

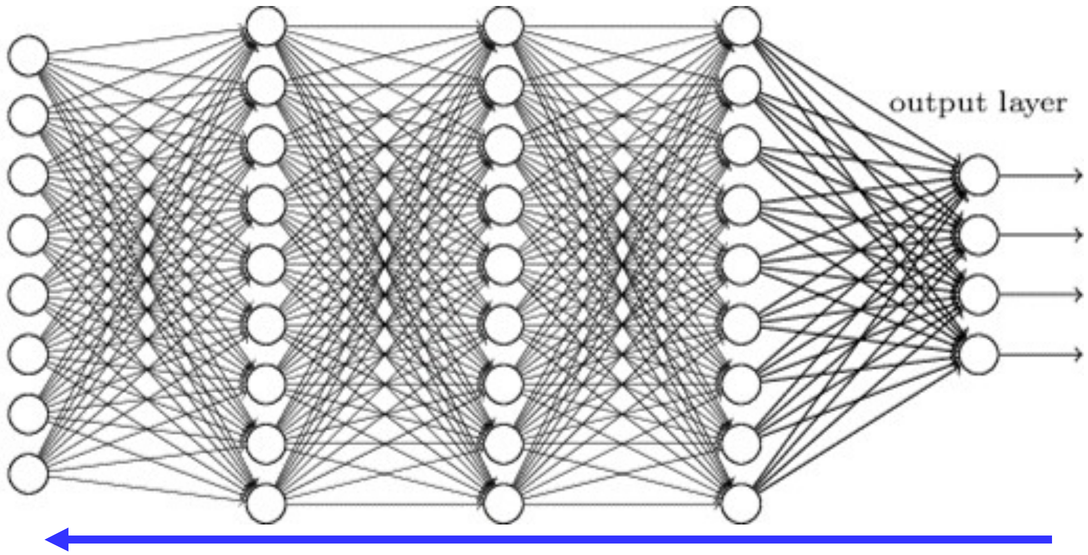
- In a single-layer RNN, the weight matrices are identical
  - The conclusion below holds for any deep network, though
- The chain product for $\nabla_{f_k} Div$ will
  - *Expand* ∇D along directions in which the singular values of the weight matrices are greater than 1
  - *Shrink* ∇D in directions where the singular values are less than 1
  - Repeated multiplication by the weights matrix will result in **Exploding** or **vanishing** gradients

Exploding/Vanishing gradients

$$\nabla_{f_k} Div = \nabla D . \underline{\nabla f_N . W_{N-1}} . \underline{\nabla f_{N-1} . W_{N-2}} \dots \underline{\nabla f_{k+1} W_k}$$

- Every blue term is a matrix
- ∇D is proportional to the actual error
  - Particularly for L_2 and KL divergence
- The chain product for $\nabla_{f_k} Div$ will
  - *Expand* ∇D in directions where each stage has singular values greater than 1
  - *Shrink* ∇D in directions where each stage has singular values less than 1

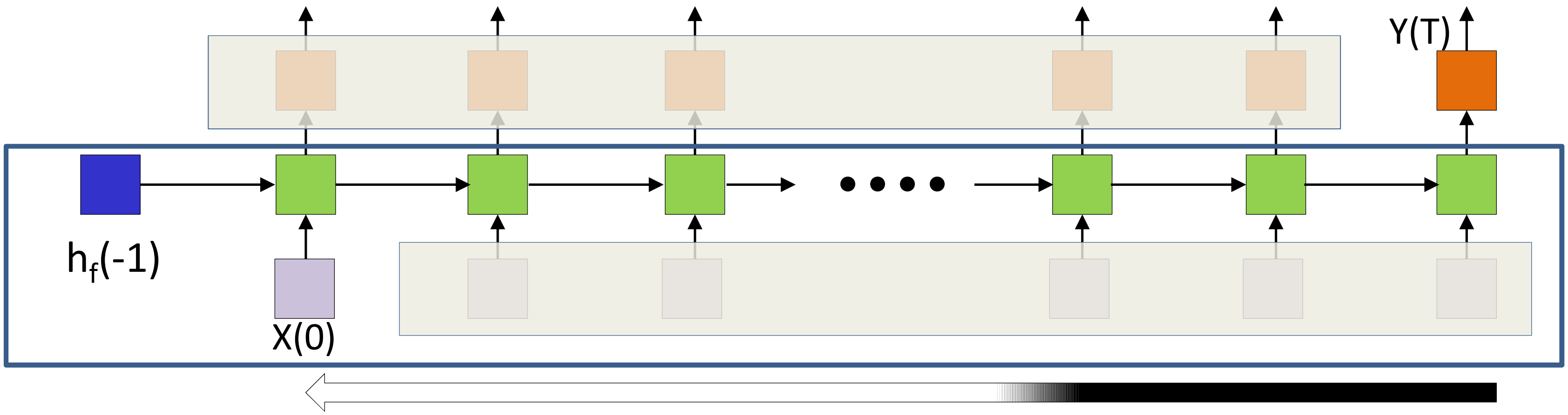
Gradient problems in deep networks



$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \dots \nabla f_{k+1} W_k$$

- The gradients in the lower/earlier layers can *explode* or *vanish*
  – Resulting in insignificant or unstable gradient descent updates
  – Problem gets worse as network depth increases

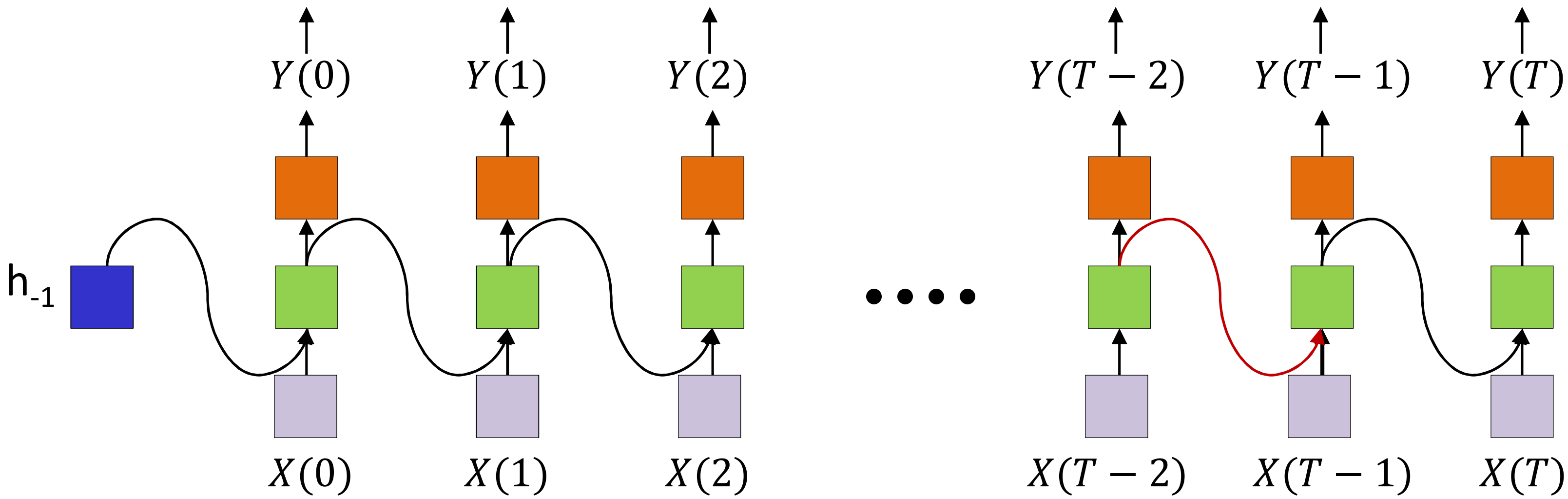
Recurrent nets are very deep nets

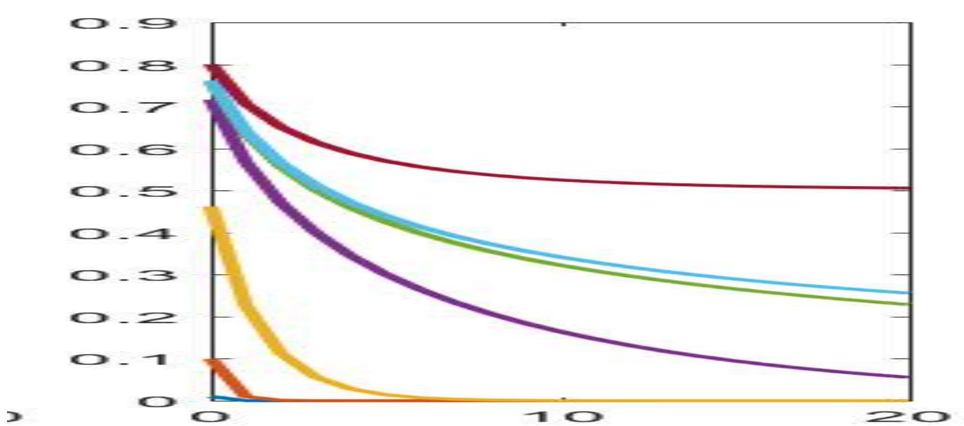


$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \dots \nabla f_{k+1} W_k$$

- The relation between $X(0)$ and $Y(T)$ is one of a very deep network
  - Gradients from errors at t $= T$ will vanish by the time they're propagated to $t = 0$

Recall: Vanishing stuff..





- Stuff gets forgotten in the forward pass too
  - Each weights matrix and activation can shrink components of the input

1. We know that $\mathbf{h}(t) = f(W^{(11)}\mathbf{h}(t-1) + W^{(1)}X(t) + \mathbf{b})$
2. As discussed earlier $DIV\left(\{Y(0),\ldots,Y(T)\},\{D(0),\ldots,D(T)\}\right) = \sum_{t=0}^{T} Div\left(Y(t), D(t)\right)$
3. Let $\Theta = \{W^{(1)}, W^{(11)}, \mathbf{b}\}$. Then

$$\frac{\partial DIV}{\partial \Theta} = \sum_{t=0}^{T} \frac{\partial Div(Y(t), D(t))}{\partial \Theta}$$

$$\frac{\partial Div(Y(t), D(t))}{\partial \Theta} = \sum_{k=0}^{t} \frac{\partial Div(Y(t), D(t))}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} \frac{\partial \mathbf{h}^{+}(k)}{\partial \Theta}$$

4. We do this because $Y(t)$ depends on $\mathbf{h}(t)$. $\mathbf{h}(t)$ depends on $\mathbf{h}(t-1)$, thus $Y(t)$ depends on $\mathbf{h}(t-1)$. Continuing this argument, $Y(t)$ depends on $\mathbf{h}(0)$.

1. $\frac{\partial \mathbf{h}^+(k)}{\partial \Theta}$ refers to the **immediate** partial derivative of $\mathbf{h}(k)$ with respect to Θ, where $\mathbf{h}(k-1)$ is assumed to be constant with respect to Θ.
2. Each temporal contribution $\frac{\partial Div(Y(t),D(t))}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} \frac{\partial \mathbf{h}^+(k)}{\partial \Theta}$ measures how Θ at step k affects the cost at step $t > k$. The factors $\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)}$ transport the error "in time" from step t back to step k.
3. But, we see that

$$
\begin{aligned}
\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} &= \Pi_{i=k+1}^{t} \frac{\partial \mathbf{h}(i)}{\partial \mathbf{h}(i-1)} \\
&= \Pi_{i=k+1}^{t} \begin{bmatrix} \frac{\partial h_1(i)}{\partial h_1(i-1)} & \frac{\partial h_1(i)}{\partial h_2(i-1)} & \cdots & \frac{\partial h_1(i)}{\partial h_{N_1}(i-1)} \\ \frac{\partial h_2(i)}{\partial h_1(i-1)} & \frac{\partial h_2(i)}{\partial h_2(i-1)} & \cdots & \frac{\partial h_2(i)}{\partial h_{N_1}(i-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_{N_1}(i)}{\partial h_{N_1}(i-1)} & \frac{\partial h_{N_1}(i)}{\partial h_2(i-1)} & \cdots & \frac{\partial h_{N_1}(i)}{\partial h_{N_1}(i-1)} \end{bmatrix}
\end{aligned}
$$

Noting that $\frac{\partial h_p(i)}{\partial h_j(i-1)} = \frac{\partial h_p(i)}{\partial Z_p^{(1)}(i)} \frac{\partial Z_p^{(1)}(i)}{\partial h_j(i-1)} = w_{jp}^{(11)} \frac{\partial h_p(i)}{\partial Z_p^{(1)}(i)}$. Thus,

$$\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} = \Pi_{i=k+1}^{t} \begin{bmatrix} w_{11}^{(11)} \frac{\partial h_1(i)}{\partial Z_1^{(1)}(i)} & w_{21}^{(11)} \frac{\partial h_1(i)}{\partial Z_1^{(1)}(i)} & \cdots & w_{N_1 1}^{(11)} \frac{\partial h_1(i)}{\partial Z_1^{(1)}(i)} \\ w_{12}^{(11)} \frac{\partial h_2(i)}{\partial Z_2^{(1)}(i)} & w_{22}^{(11)} \frac{\partial h_2(i)}{\partial Z_2^{(1)}(i)} & \cdots & w_{N_1 2}^{(11)} \frac{\partial h_2(i)}{\partial Z_2^{(1)}(i)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1N_1}^{(11)} \frac{\partial h_{N_1}(i)}{\partial Z_{N_1}^{(1)}(i)} & w_{2N_1}^{(11)} \frac{\partial h_{N_1}(i)}{\partial Z_{N_1}^{(1)}(i)} & \cdots & w_{N_1 N_1}^{(11)} \frac{\partial h_{N_1}(i)}{\partial Z_{N_1}^{(1)}(i)} \end{bmatrix}$$

$$= \Pi_{i=k+1}^{t} W^{(11)} \begin{bmatrix} \frac{\partial h_1(i)}{\partial Z_1^{(1)}(i)} & 0 & \cdots & 0 \\ 0 & \frac{\partial h_2(i)}{\partial Z_2^{(1)}(i)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial h_{N_1}(i)}{\partial Z_{N_1}^{(1)}(i)} \end{bmatrix} = \Pi_{i=k+1}^{t} \left(W^{(11)} \;\; \nabla_{Z^{(1)}(i)} \mathbf{h}(i) \right)$$

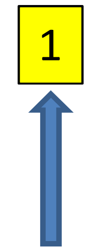
Vanishing/Exploding Gradient Problem

1. We know that $h_i(t) = f\left(Z_i^{(1)}(t)\right)$ where $Z_i^{(1)}(t) = \sum_{j=1}^{N_1} w_{ji}^{(11)} h_j(t-1) + \sum_{j=1}^{N} w_{ji}^{(1)} X_j(t) + b_i^{(1)}$, where $f(.)$ is the activation function used.
2. Thus, $\frac{\partial h_i(t)}{\partial Z_i^{(1)}(t)} = f'(Z_i^{(1)}(t))$.
3. Let f be such that $|f'(Z_i^{(1)}(t))| \leq \gamma$ all $i = 1 \dots N_1$. For example, sigmoid $(\gamma = 0.25)$, tanh $(\gamma = 1)$.
4. Thus, $\|\nabla_{Z^{(1)}(t)} \mathbf{h}(t)\| \leq \max_{i \in \{1,\dots,N_1\}} |f'(Z_i^{(1)}(t))| \leq \gamma$.
5. Thus, if $\|W^{(11)}\| < \frac{1}{\gamma}$, then it is easy to show that vanishing gradient problem can happen.
6. For all k, we see that $\left\|\frac{\partial \mathbf{h}(k+1)}{\partial \mathbf{h}(k)}\right\| = \|W^{(11)} \ \nabla_{Z^{(1)}(k+1)} \mathbf{h}(k+1)\| \leq \|W^{(11)}\| \ \|\nabla_{Z^{(1)}(k+1)} \mathbf{h}(k+1)\| < \frac{1}{\gamma}\gamma = 1$

Vanishing/Exploding Gradient Problem

1. Let $\eta \in \mathbb{R}$ be such that for all k, $\left\| \frac{\partial \mathbf{h}(k+1)}{\partial \mathbf{h}(k)} \right\| \leq \eta < 1$. Such an η exists.
2. Thus, it can be shown that
$\left\| \frac{\partial Div(Y(t),D(t))}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(k)} \right\| = \left\| \frac{\partial Div(Y(t),D(t))}{\partial \mathbf{h}(t)} \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}(i)}{\partial \mathbf{h}(i-1)} \right\| \leq \eta^{t-k} \left\| \frac{\partial Div(Y(t),D(t))}{\partial \mathbf{h}(t)} \right\|$
3. As $\eta < 1$, it follows that, long term contributions (for which $t - k$ is large) go to 0 exponentially fast with $t - k$.
4. By inverting this proof we get the necessary condition. for exploding gradients.

The long-term dependency problem



PATTERN1 [……………………….....] PATTERN 2

Jane had a quick lunch in the bistro. Then *she..*

- Any other pattern of any length can happen between pattern 1 and pattern 2
  - RNN will "forget" pattern 1 if intermediate stuff is too long
  - "Jane" → the next pronoun referring to her will be "she"
- Must know to "remember" for extended periods of time and "recall" when necessary
  - Can be performed with a multi-tap recursion, but how many taps?
  - Need an alternate way to "remember" stuff

Exploding/Vanishing gradients

$$Y = f_N\left(\underline{W_{N-1}f_{N-1}}\left(\underline{W_{N-2}f_{N-2}}\left(\ldots \underline{W_0}X\right)\right)\right)$$

$$\nabla_{f_k} Div = \nabla D.\nabla f_N.W_{N-1}.\nabla f_{N-1}.W_{N-2} \ldots \underline{\nabla f_{k+1} W_k}$$

- The memory retention of the network depends on the behavior of the underlined terms
  – Which in turn depends on the parameters W rather than what it is trying to "remember"

- Can we have a network that just "remembers" arbitrarily long, to be recalled on demand?
  – Not be directly dependent on vagaries of network parameters, but rather on input-based determination of *whether it must be remembered*

Exploding/Vanishing gradients

$$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \dots \underline{\nabla f_{k+1} W_k}$$

- Replace this with something that doesn't fade or blow up?

- Network that "retains" *useful* memory arbitrarily long, to be recalled on demand?
  - Input-based determination of *whether it must be remembered*
  - **Retain memories until a switch *based on the input* flags them as ok to forget**
    - Or remember less
  - $Memory(k) \approx C(x).\sigma_k(x).\sigma_{k-1}(x).\dots\sigma_1(x)$
  - $\nabla_{f_k} Div \approx \nabla D C \sigma'_N C \sigma'_{N-1} C \dots \sigma'_k$

1. Suppose that instead of a matrix multiplication, we had an identity relationship between the hidden states! $\mathbf{h}(t) = \mathbf{h}(t-1) + F(X(t))$
2. Then, $\frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-1)} = \mathbf{I}$,which is identity matrix.
3. Thus, $\frac{\partial Div(Y(t),D(t))}{\partial \mathbf{h}(1)} = \frac{\partial Div(Y(t),D(t))}{\partial Y(t)} \frac{\partial Y(t)}{\partial \mathbf{h}(t)} \frac{\partial \mathbf{h}(t)}{\partial \mathbf{h}(t-1)} \cdots \frac{\partial \mathbf{h}(2)}{\partial \mathbf{h}(1)} = \frac{\partial Div(Y(t),D(t))}{\partial Y(t)} \frac{\partial Y(t)}{\partial \mathbf{h}(t)}$
4. The gradient does not decay as the error is propagated all the way back also called as "Constant Error Flow"!

1. The LSTM uses this idea of "Constant Error Flow" for RNNs to create a "Constant Error Carousel" (CEC) which ensures that gradients don't decay!
2. The key component is a memory cell that acts like an accumulator (contains the identity relationship) over time!
3. Instead of computing new state as a matrix product with the old state, it rather computes the difference between them. Expressivity is the same, but gradients are better behaved!