The vanishing gradient problem revisited: Highway and residual connections

THE UNIVERSITY OF UTAH

Revisiting the vanishing gradient problem

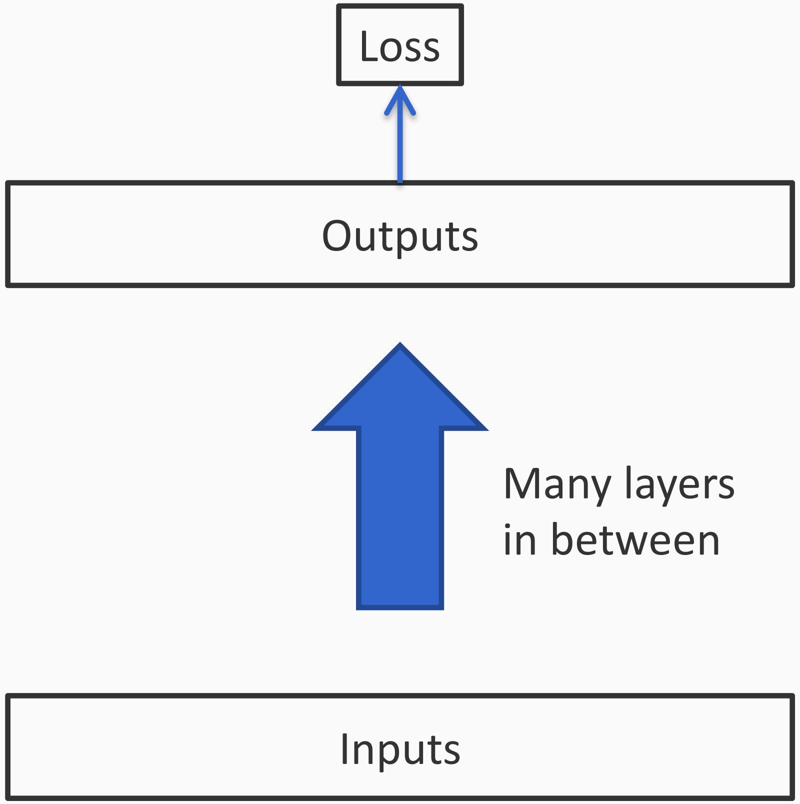
Stems from the fact that the derivative of the activation is between zero and one…

… and as the number steps of gradient computation grows, these get multiplied

Not just applicable for LSTMs

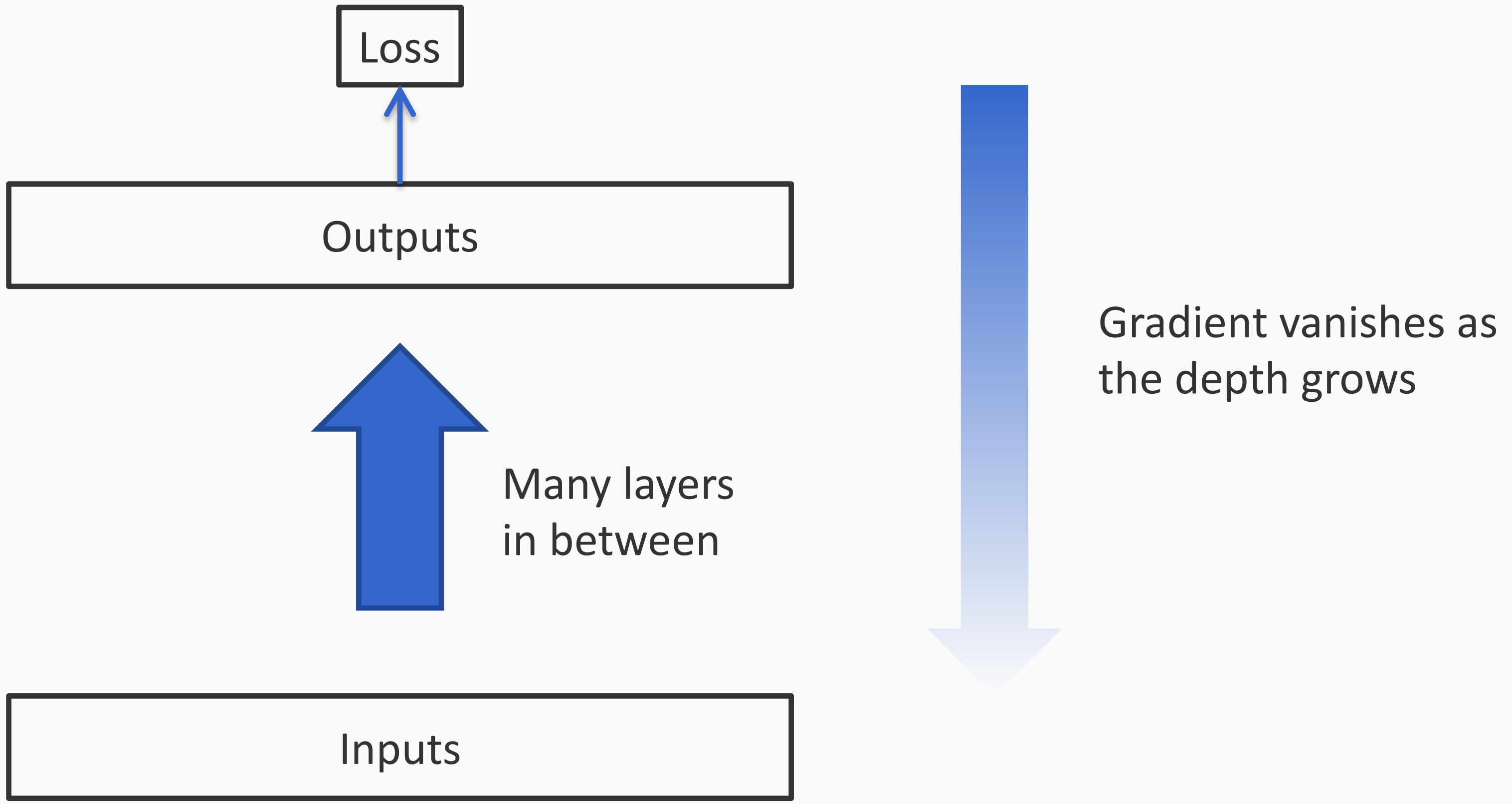
Revisiting the vanishing gradient problem

Not just applicable for LSTMs



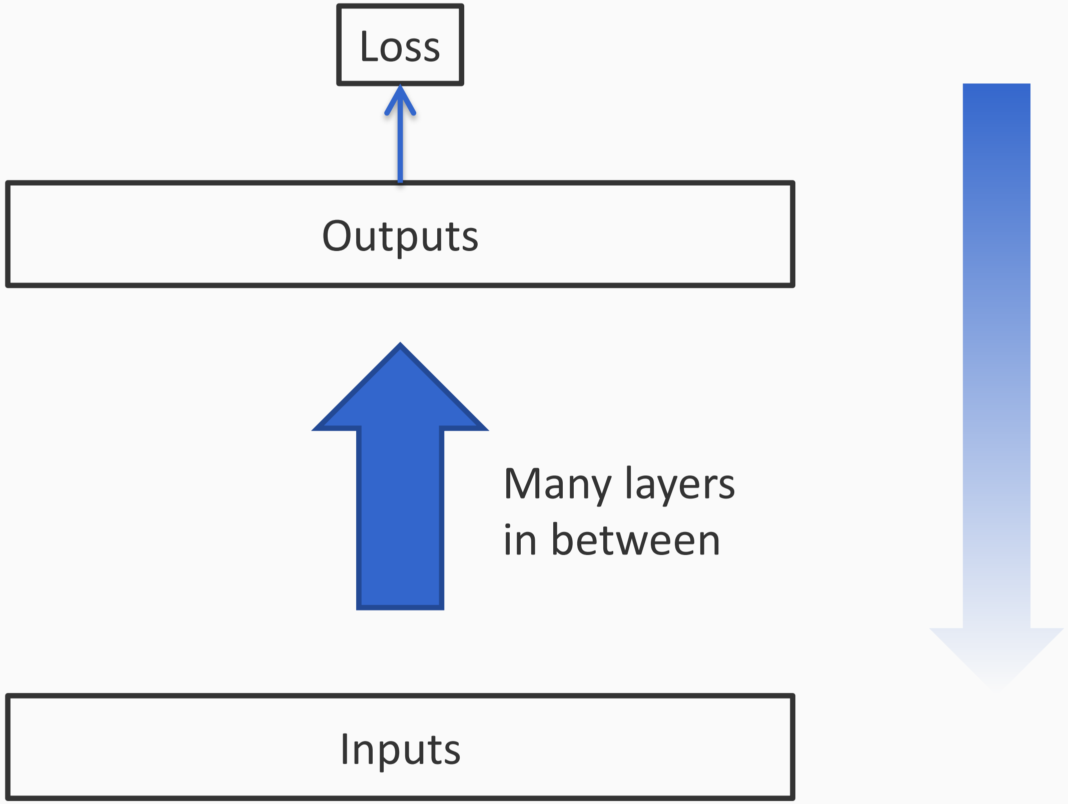
Revisiting the vanishing gradient problem

Not just applicable for LSTMs



Revisiting the vanishing gradient problem

Not just applicable for LSTMs

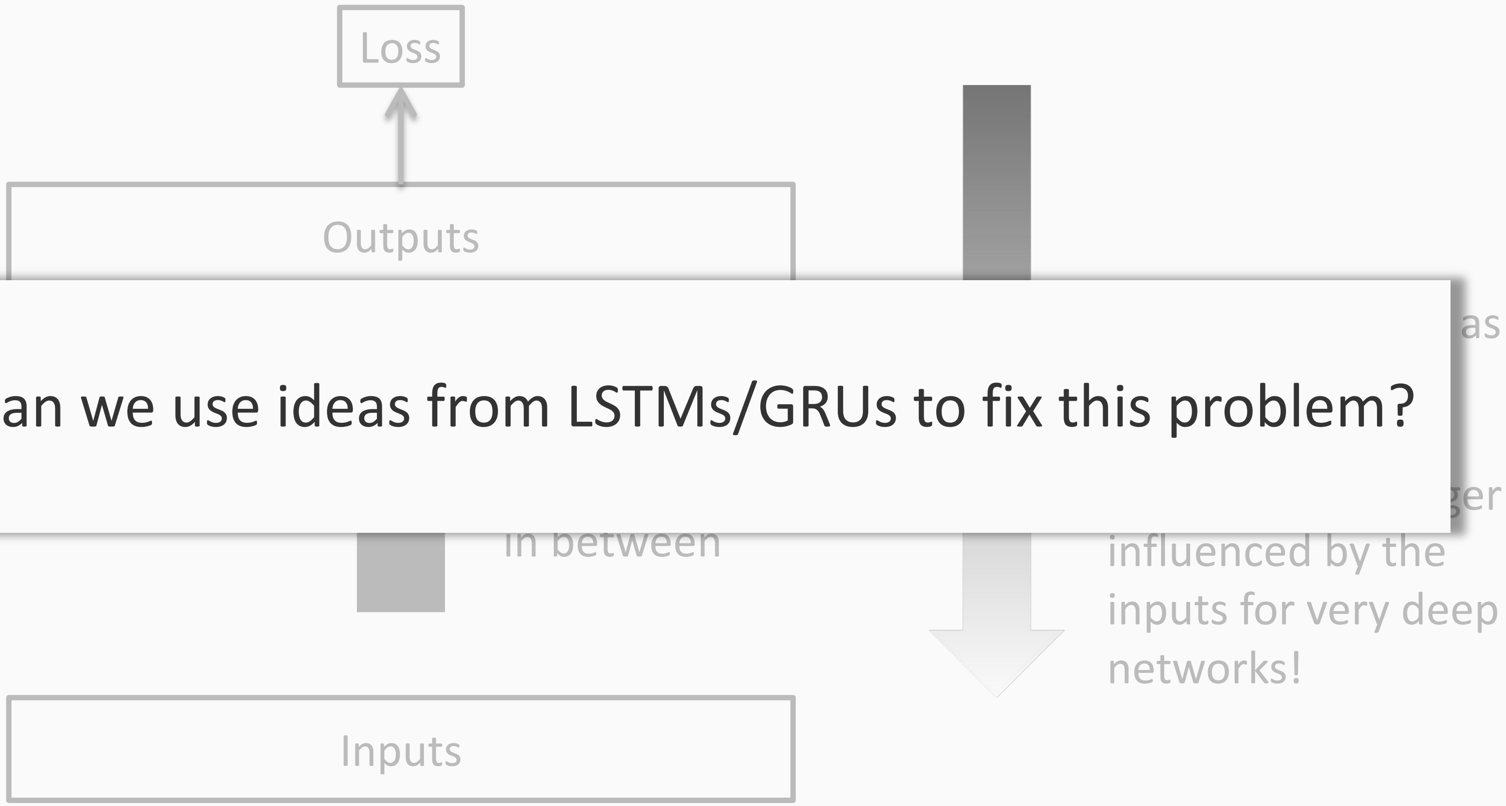


Gradient vanishes as the depth grows

The loss is no longer influenced by the inputs for very deep networks!

Revisiting the vanishing gradient problem

Not just applicable for LSTMs



Can we use ideas from LSTMs/GRUs to fix this problem?

Revisiting the vanishing gradient problem

Intuition: Consider a single layer

$$\boldsymbol{l}^t = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

The t-1th layer is used to calculate the value of the tth layer

Revisiting the vanishing gradient problem

Intuition: Consider a single layer

$$\boldsymbol{l}^t = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

Instead of a non-linear update that directly
calculates the next layer, let us try a linear update

$$\boldsymbol{l}^t = \boldsymbol{l}^{t-1} + g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

Revisiting the vanishing gradient problem

Intuition: Consider a single layer

$$\boldsymbol{l}^t = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

Instead of a non-linear update that directly
calculates the next layer, let us try a linear update

$$\boldsymbol{l}^t = \boldsymbol{l}^{t-1} + \ g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

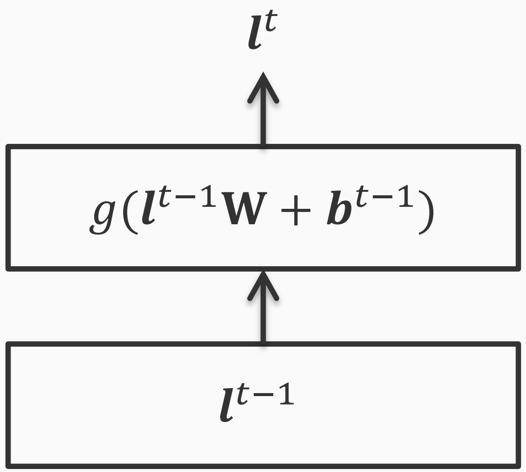
The gradients can be propagated all the way to the input without attenuation

Residual networks

[He et al 2015]

Each layer is reformulated as

$$\boldsymbol{l}^{t} = \boldsymbol{l}^{t-1} + g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$



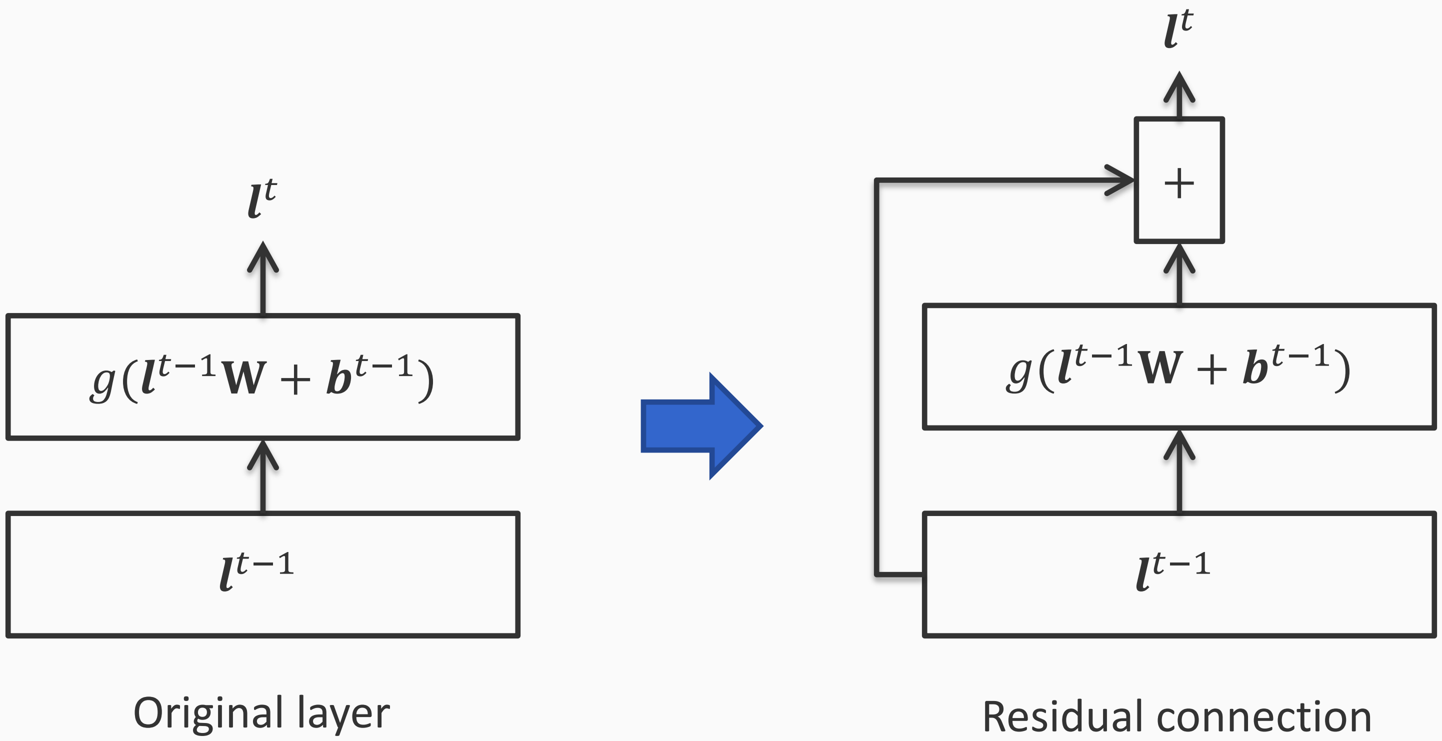
Original layer

Residual networks

[He et al 2015]

Each layer is reformulated as

$$\boldsymbol{l}^t = \boldsymbol{l}^{t-1} + g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$



Residual networks

[He et al 2015]

Each layer is reformulated as

$$\boldsymbol{l}^t = \boldsymbol{l}^{t-1} + g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

The computation graph g is not trained to predict the next layer

It predicts an *update to* the current layer value instead

That is, it can be seen as a residual function (that is the difference between the layers)

Highway connections

[Srivastava et al 2015]

Extend the idea, using gates to stabilize learning

Highway connections

[Srivastava et al 2015]

Extend the idea, using gates to stabilize learning

- First, compute a proposed update

$$\mathbf{C} = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

Highway connections

[Srivastava et al 2015]

Extend the idea, using gates to stabilize learning

- First, compute a proposed update

$$\mathbf{C} = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

- Next, compute how much of the proposed update should be retained

$$\mathbf{T} = \sigma(\mathbf{l}^{t-1}\mathbf{W}_T + \mathbf{b}_T)$$

Highway connections

[Srivastava et al 2015]

Extend the idea, using gates to stabilize learning

- First, compute a proposed update

$$\mathbf{C} = g(\boldsymbol{l}^{t-1}\mathbf{W} + \boldsymbol{b}^{t-1})$$

- Next, compute how much of the proposed update should be retained

$$\mathbf{T} = \sigma(\mathbf{l}^{t-1}\mathbf{W}_T + \mathbf{b}_T)$$

- Finally, compute the actual value of the next layer

$$\boldsymbol{l}^t = (1 - \mathbf{T}) \odot \boldsymbol{l}^{t-1} + \mathbf{T} \odot \mathbf{C}$$

Why residual/highway connections?

- As networks become deeper, or as sequences get larger, we can no longer hope for gradients to be carried through the network

- If we want to capture long-range dependencies with the input, we need this mechanism

- More generally, a blueprint of an idea that can be combined with your neural network model if it gets too deep