Neural Networks

Deep Learning Principles

CSCI 4850/5850

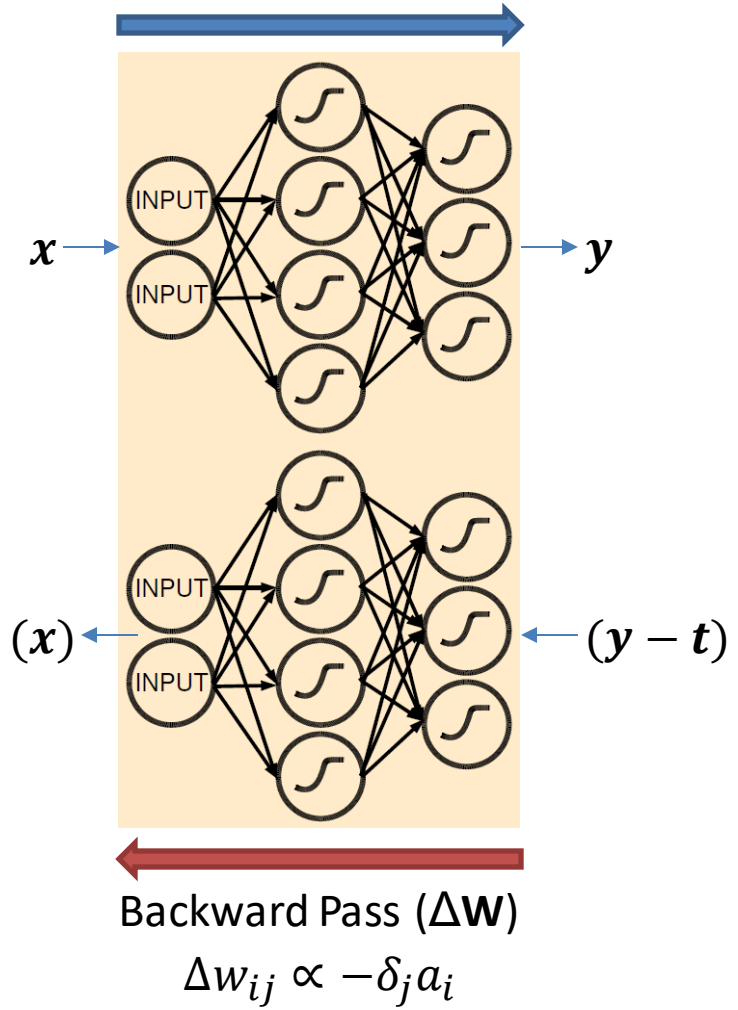
Generalized Delta Rule

- Also known as **error backpropagation** or "**backprop**"
- Two step process:
  - **Prediction**: the inputs are provided and information flows to calculate the output activations (forward pass)
  - **Fitting**: errors are calculated at the output layer and information flows in reverse to calculate the weight updates (backward pass)
- Issue: biological plausibility
  - Real neurons do not pass information backward across synapses…

Output unit: $\delta_j = g'(net_j)(a_j - t_j)$

Hidden unit: $\delta_j = g'(net_j)\sum_k w_{jk}\delta_k$

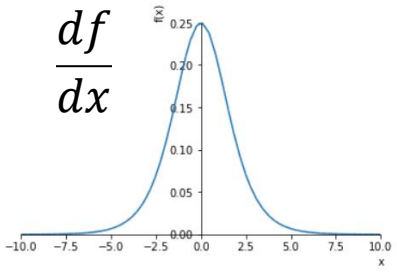
$a_i = g(net_i) \quad net_i = w_{i0} + \sum_j w_{ij}a_i$

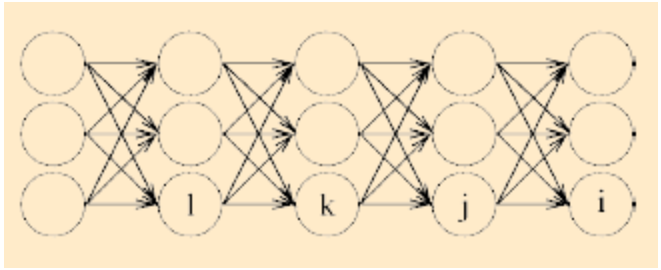


The Vanishing Gradient Problem

- Getting caught in the flat, planar regions of the error surface is problematic
- We **need** non-linear activation functions to make use of multiple layers, but typical non-linear activation functions (sigmoid, tanh) cause the gradient to **vanish**…

$$\frac{df}{dx}$$





$$\delta_j = g'(net_j)\sum_i w_{ij}\delta_i$$

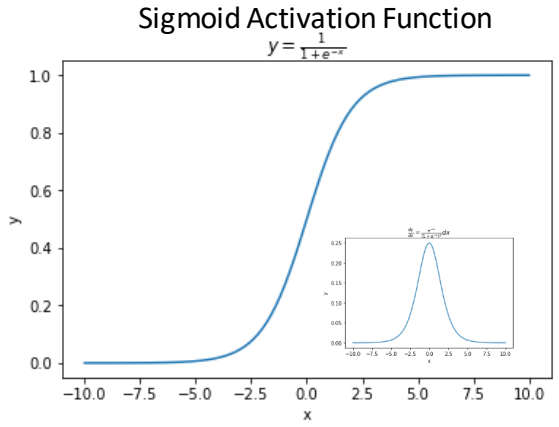
$$\delta_k = g'(net_k)\sum_j w_{jk}\delta_j$$

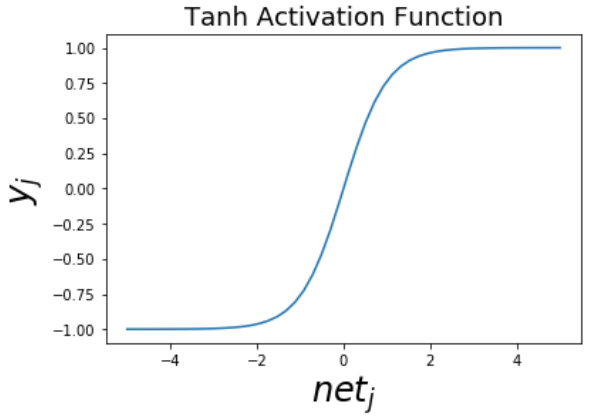
$$\delta_l = g'(net_l)\sum_k w_{kl}\delta_k$$

- We will see some workarounds to this *particular* issue soon, but even still, gradient optimization is tricky…

Hidden: Tanh over Sigmoid

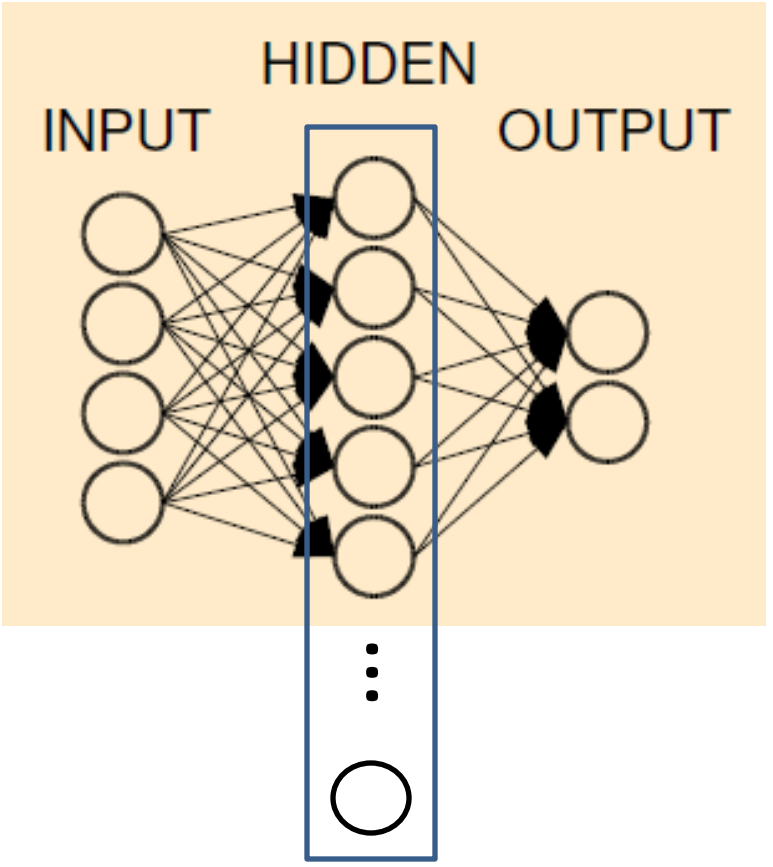
- When it comes to hidden unit activations, the sigmoid function disappoints in some ways…
- Gradient disappears at high-magnitude net inputs…
- Gradient highest at midpoint (zero net input), but activation is at 0.5!
- This will just make the gradient disappear even more at the next layer…
- Hyperbolic tangent is at least a little better here since tanh(0)=0 and is more similar to a linear unit near zero, but the gradient still disappears at high-magnitude net input values





So is wider better?

- Given the practical problems with the gradient disappearing the “wider is better” trend existed for about a decade without any clear resolution.
- Potential problems
  - (Major issue) Complex decision boundaries are probably better described by multilayered partitions instead of a single, very complex partition: might even be easier to learn this way if we could…
  - (Minor issue) Wider networks allow for poorer parallel training and performance since we can’t process more than two batches of patterns through a feed-forward network at any one time



Things get deep…

In the early 2000s, some clues start to emerge based on existing knowledge…

We **must** have nonlinearity in our hidden units to learn any non-linear discrimination boundary or non-linear regression function

But, typical non-linear activation functions **squash the gradient**!

Linear units produce strong gradients

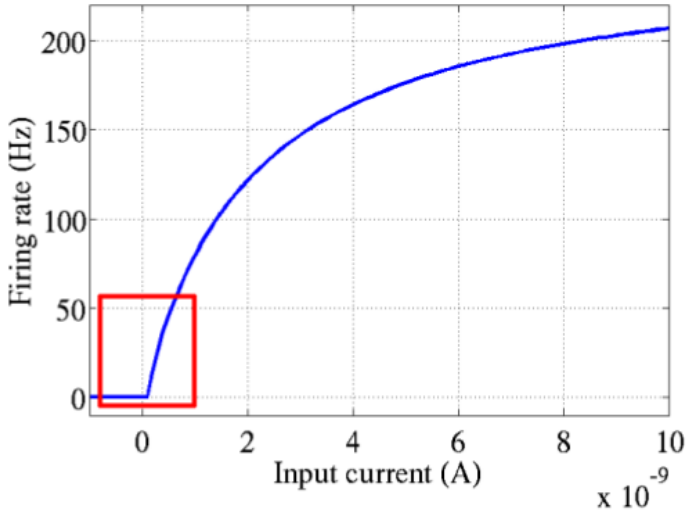
But, linear units can **only** learn any linearly separable discrimination boundary or linear regression function

What do we need?

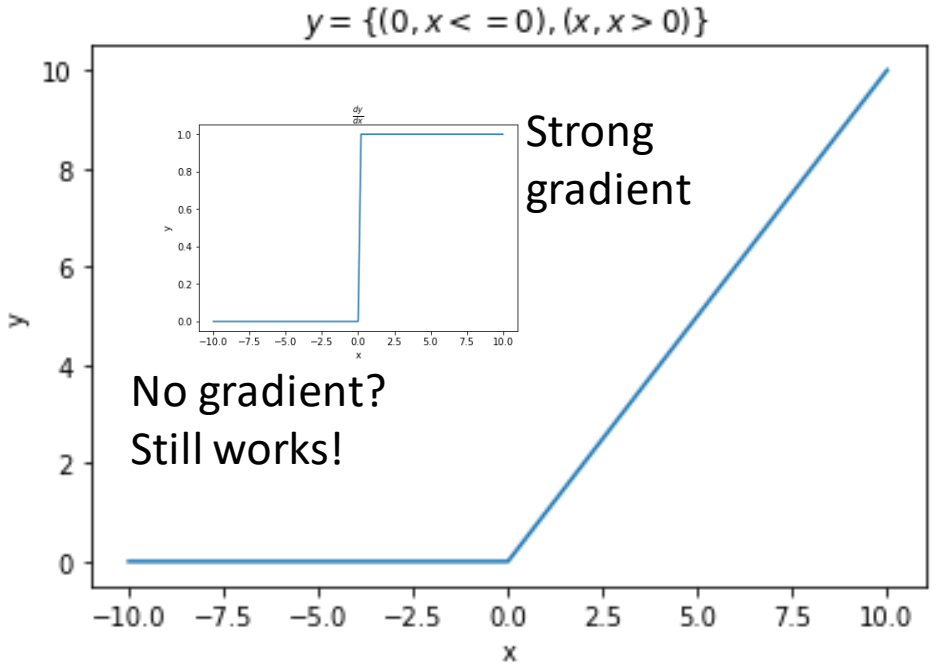
- The **gradient** of a linear unit
- The **non-linearity** of a tanh unit (sort-of)

A little non-linearity goes a long way…

- Rectified Linear Unit (ReLU) $y = \max(0, x)$
- Glorot, Bordes, & Bengio (2011)

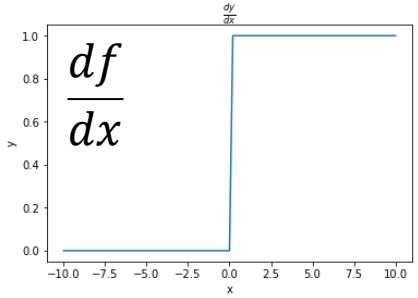


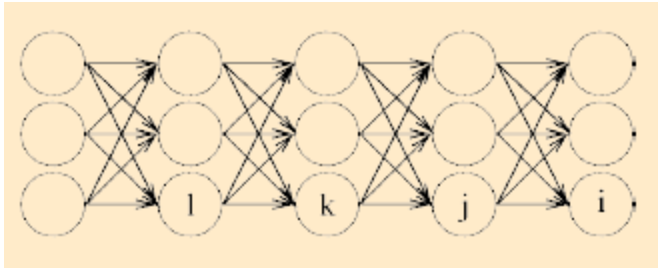
Real neurons don't usually operate near their maximal firing rate!



The Vanishing Gradient? No More

- Many ReLU-based hidden layers will now often show many hidden units with activations of zero (sparse activation patterns)
- However, for active units, the gradient is passed back at full strength





$$\delta_j = g'(net_j) \sum_i w_{ij}\delta_i$$

$$\delta_k = g'(net_k) \sum_j w_{jk}\delta_j$$

$$\delta_l = g'(net_l) \sum_k w_{kl}\delta_k$$

- ReLU is considered the de-facto standard hidden unit activation function: trains deep networks faster than tanh

Some additional variations…

- Leaky ReLU
- Parametric ReLU
- Exponential LU
- Scaled ELU

$$y = \max(0.1x, x)$$

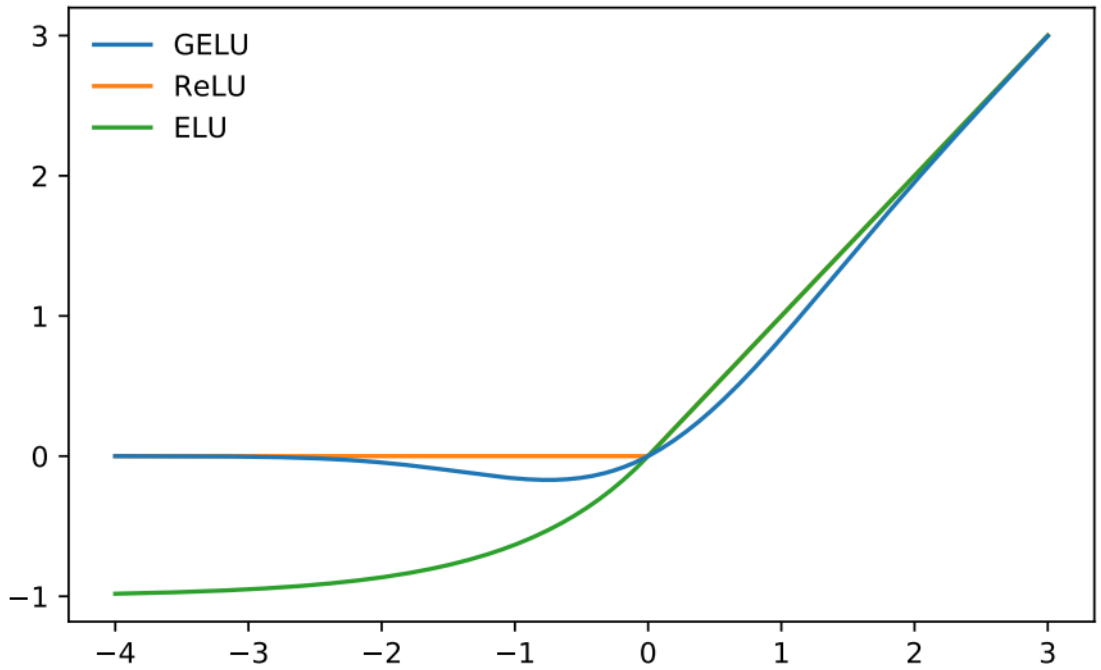
$$y = \max(\alpha x, x)$$

$$y = \begin{cases} x & if\ x > 0 \\ \alpha(\exp(x) - 1) & if\ x \leq 0 \end{cases}$$

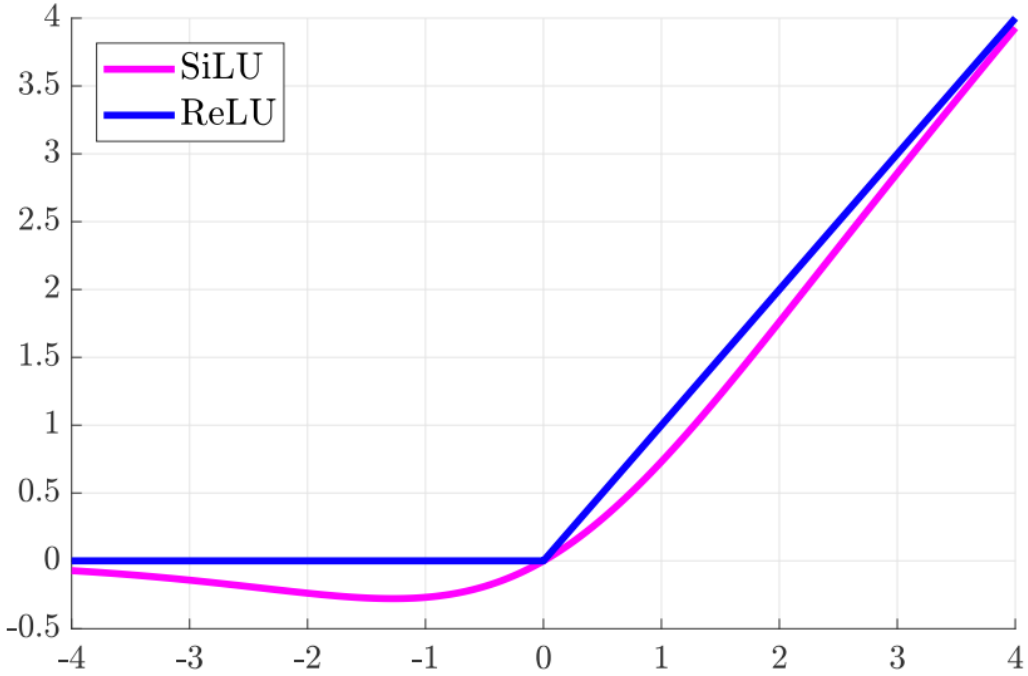
$$y = \lambda \begin{cases} x & if\ x > 0 \\ \alpha \exp(x) - \alpha & if\ x \leq 0 \end{cases}$$

- SELU requires some additional changes to weight initialization as well (Klambauer, et al. 2017) [Use ReLU/LReLU for now…]

GELU and SiLU (a.k.a. Swish)

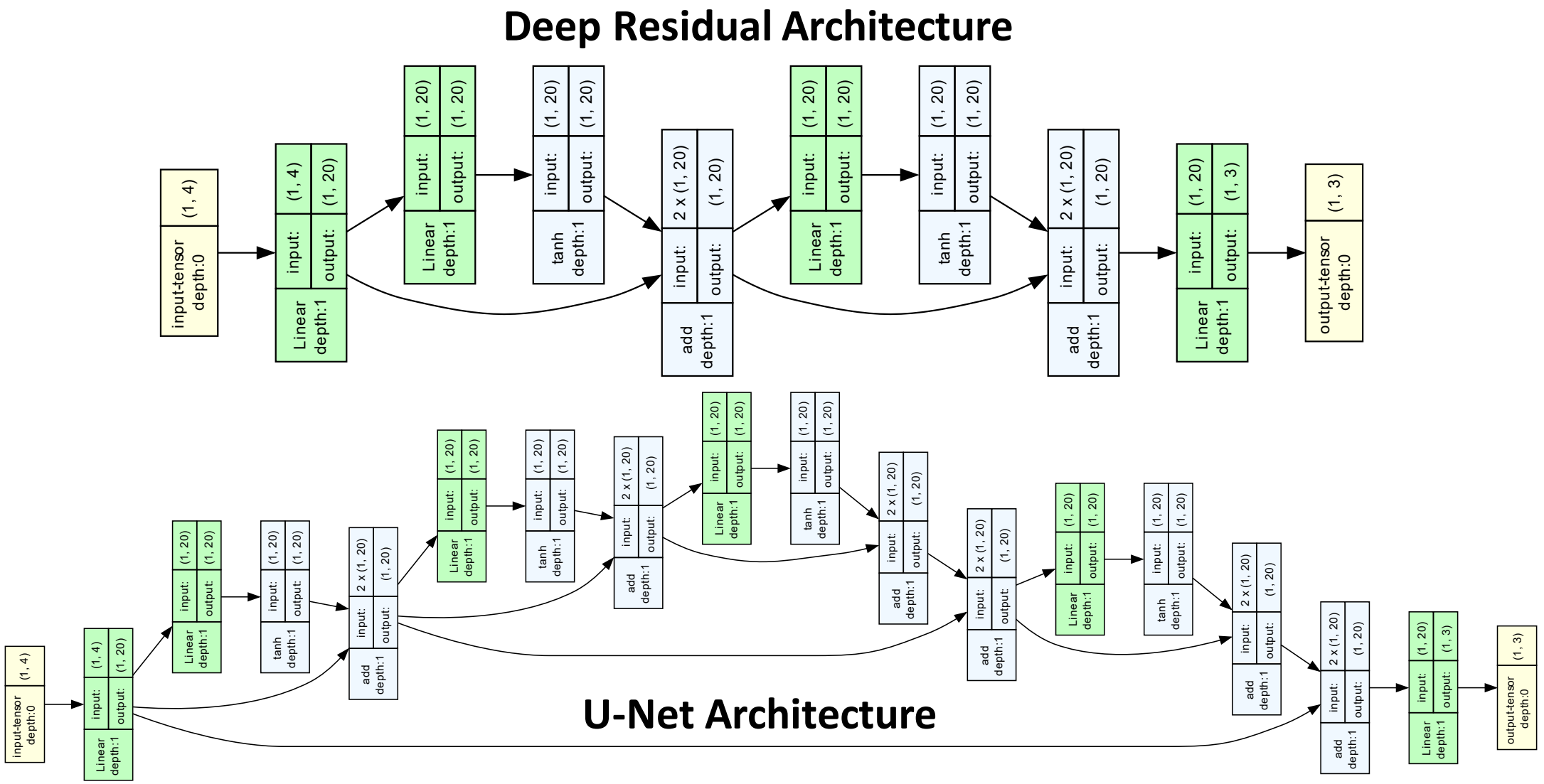


$$\mathrm{GELU}(x) = xP(X \leq x) = x\Phi(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}(x/\sqrt{2})\right]$$



SiLU(x) = x σ(x)
where σ(x) is the sigmoid activation function

Other Delta-Preserving Changes

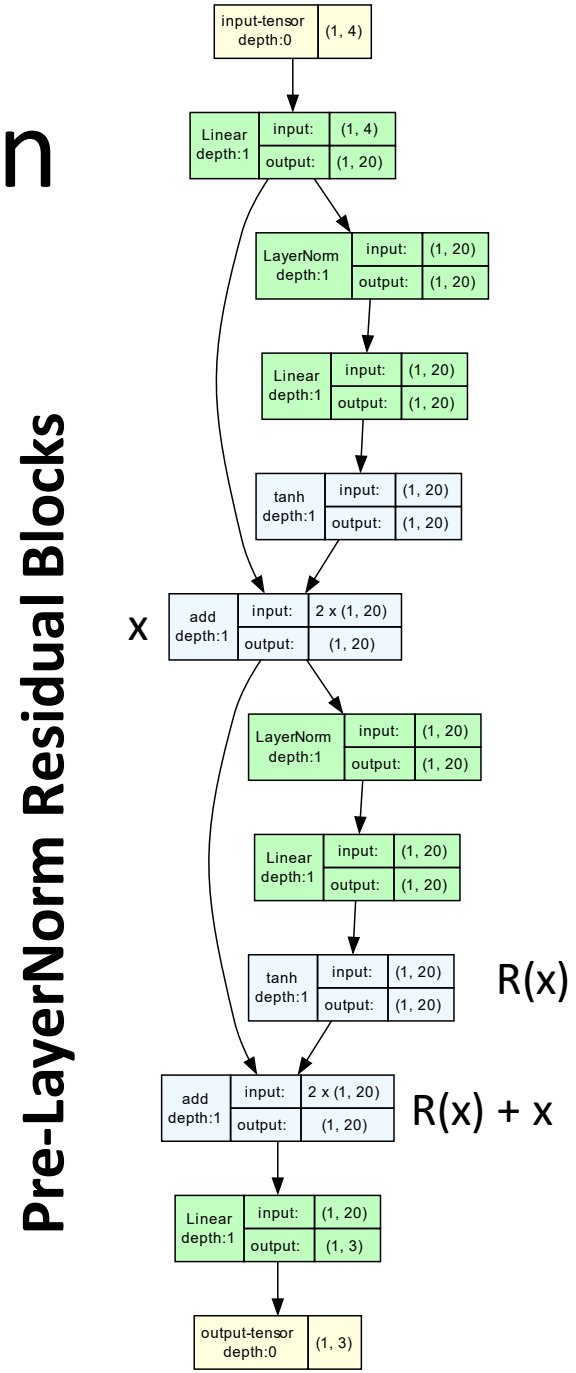


Common Residual Pattern

Residual Function Mappings

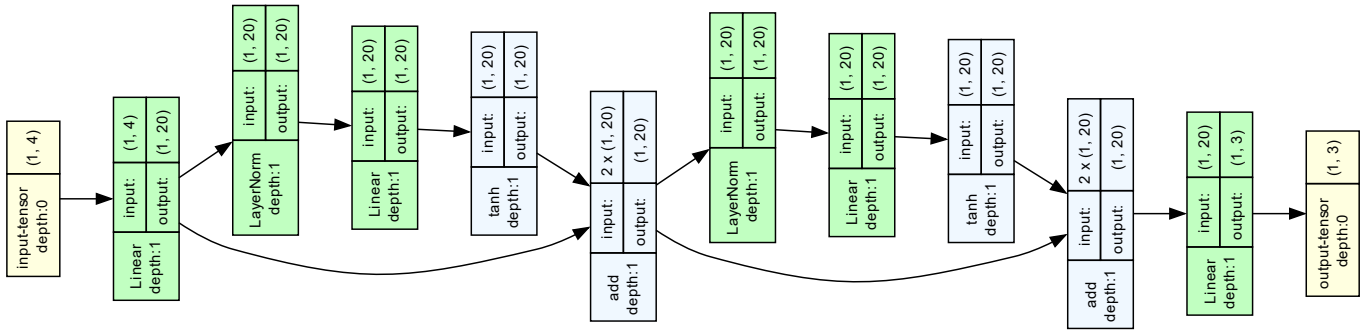
- F(x) := R(x) – x
- Residual Connections Learn: R(x) + x

- We already know (it's proven!) that a single hidden layer is **sufficient** for learning any functionEmpirically, but deeper nets take **less time** (fewer epochs) to train and are **more accurate** (in terms of testing generalization) than shallow networks…
- Intuitively, this is because the layers provide **nested structure** in the obtained solution
  - Analogous to breaking a task into **subtasks** which can be reused to perform other tasks later on
  - Many complex functions have nested substructures
  - A wide network must find the "whole" function in the hidden unit transformation instead of substructures… (more complicated?)
- Recent work has shown that shallow nets can be trained using **generated data** from a deep net, but rarely perform as well when trained on the *original* training data (Ba and Caruana, 2014)

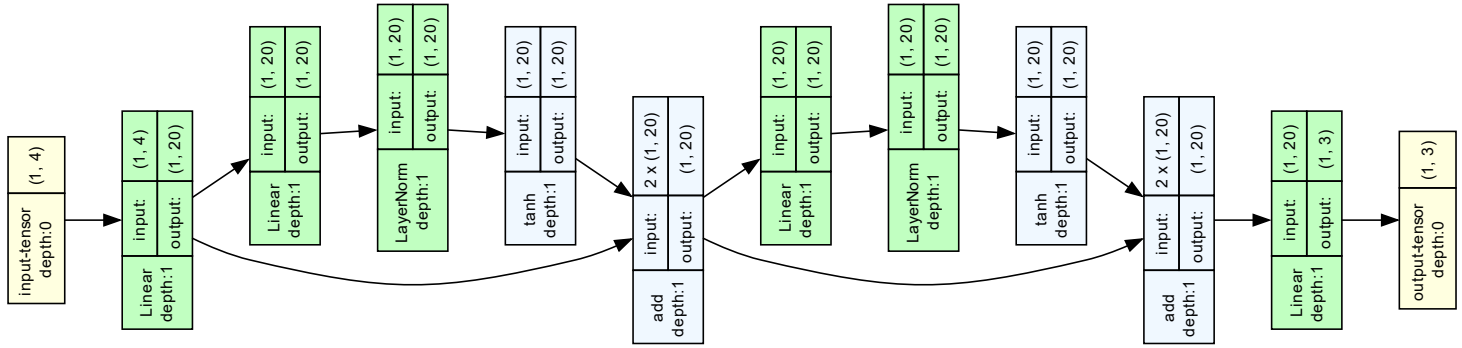


Batch/LayerNorm Placement

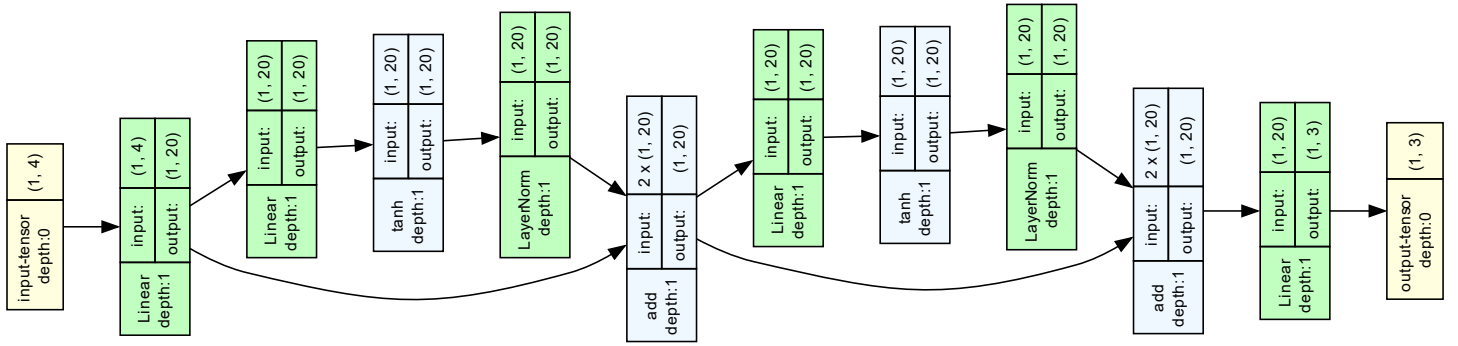
Pre-LayerNorm
(Modern Transformers)

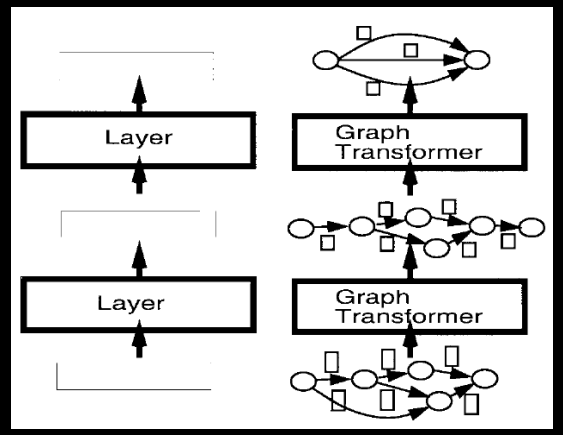


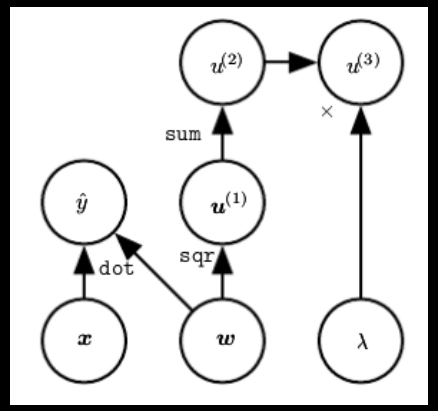
Intra-LayerNorm
(Modern ConvNets)



Post-LayerNorm
(Older Transformers
And ConvNets)







Computational Graphs LeCun, Bottou, Bengio and Haffner, 1998

- Some responsibility for the advance of deep nets into the spotlight is more due to **software** and **hardware** advances

- Traditional linear-algebra formulation is useful for understanding how a neural net works, but…
  - Computational graphs provide a more general framework for constructing components used in neural networks
  - Optimization of graph processing can now be performed
  - Smaller subcalculations map nicely onto GPU and other many-core technologies