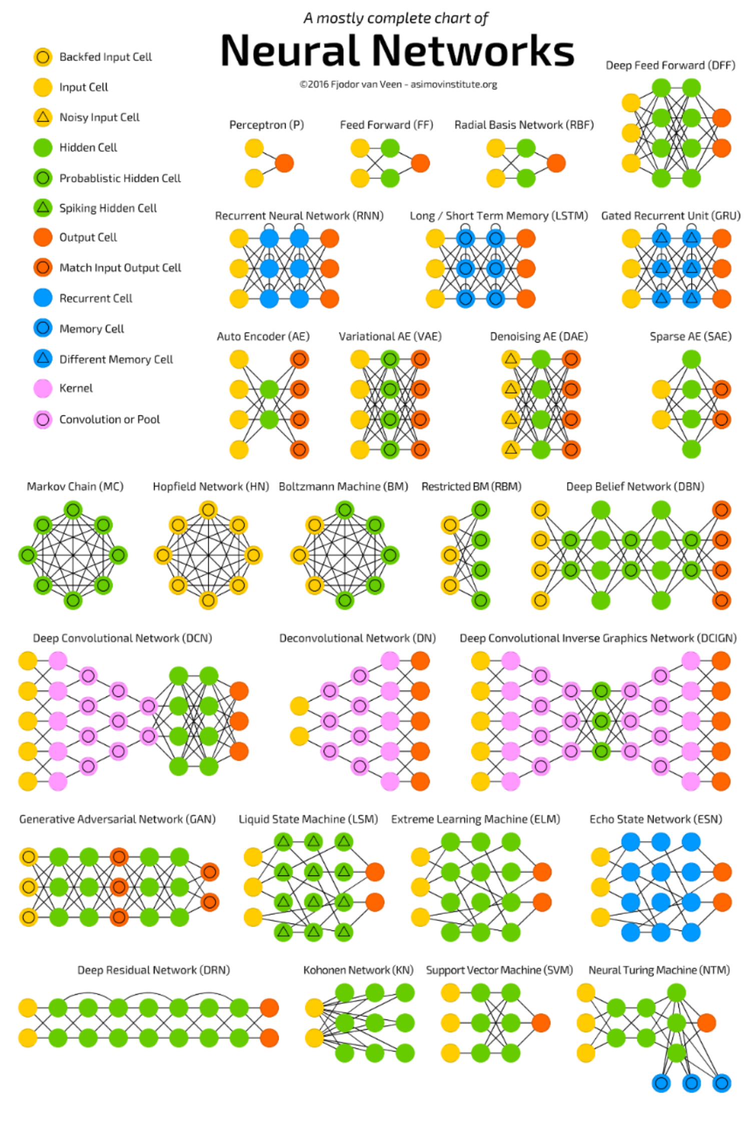
Deep learning modules

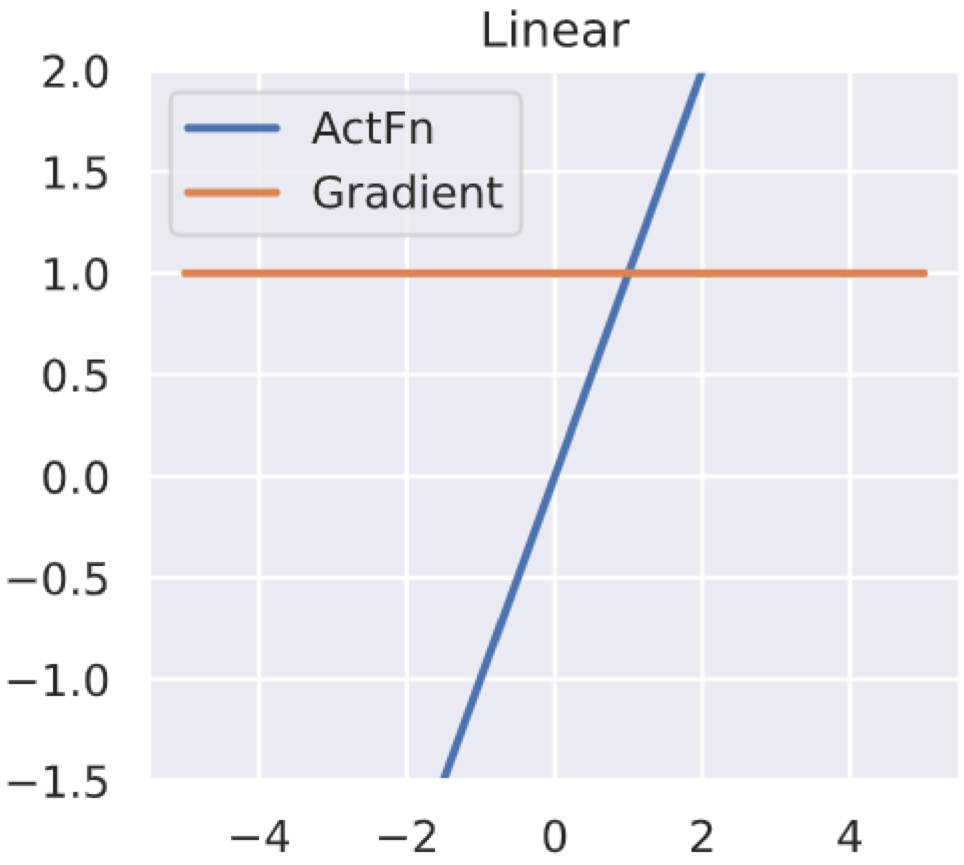
A neural network jungle

- Perceptrons, MLPs
- RNNs, LSTMs, GRUs
- Vanilla, Variational, Denoising Autoencoders
- Hopfield Nets, Restricted Boltzmann Machines
- Convolutional Nets, Deconvolutional Nets
- Generative Adversarial Nets
- Deep Residual Nets, Neural Turing Machines
- They all rely on <u>modules</u>



Linear module

$$\boldsymbol{x} \in \mathbb{R}^{1\times M}, \boldsymbol{w} \in \mathbb{R}^{N\times M}$$
$$h(x; w) = \boldsymbol{x} \cdot \boldsymbol{w}^T + b$$
$$\frac{dh}{d\boldsymbol{x}} = \boldsymbol{w}$$



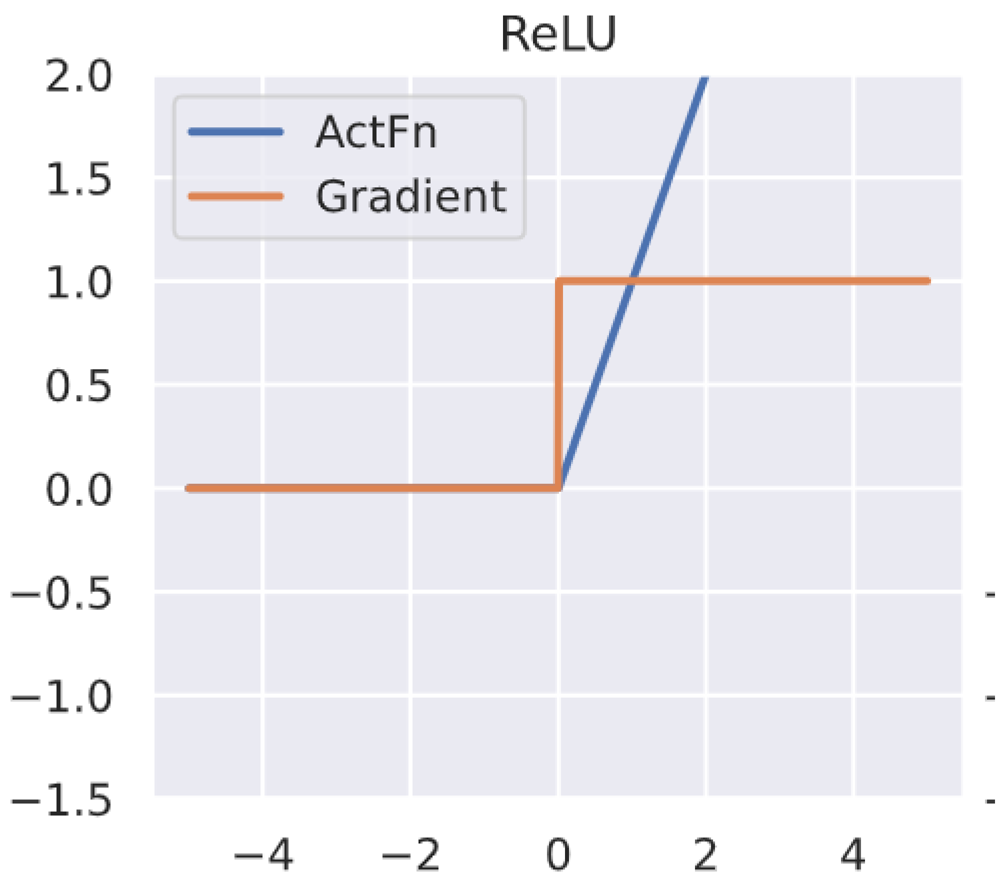
- No activation saturation
- Hence, strong & stable gradients
  - Reliable learning with linear modules

Rectified Linear Unit (ReLU)

ReLU

$$h(x) = \max(0, x)$$

$$\frac{\partial h}{\partial w} = \begin{cases} 1 \text{ when } x > 0 \\ 0, \text{when } x \leq 0 \end{cases}$$

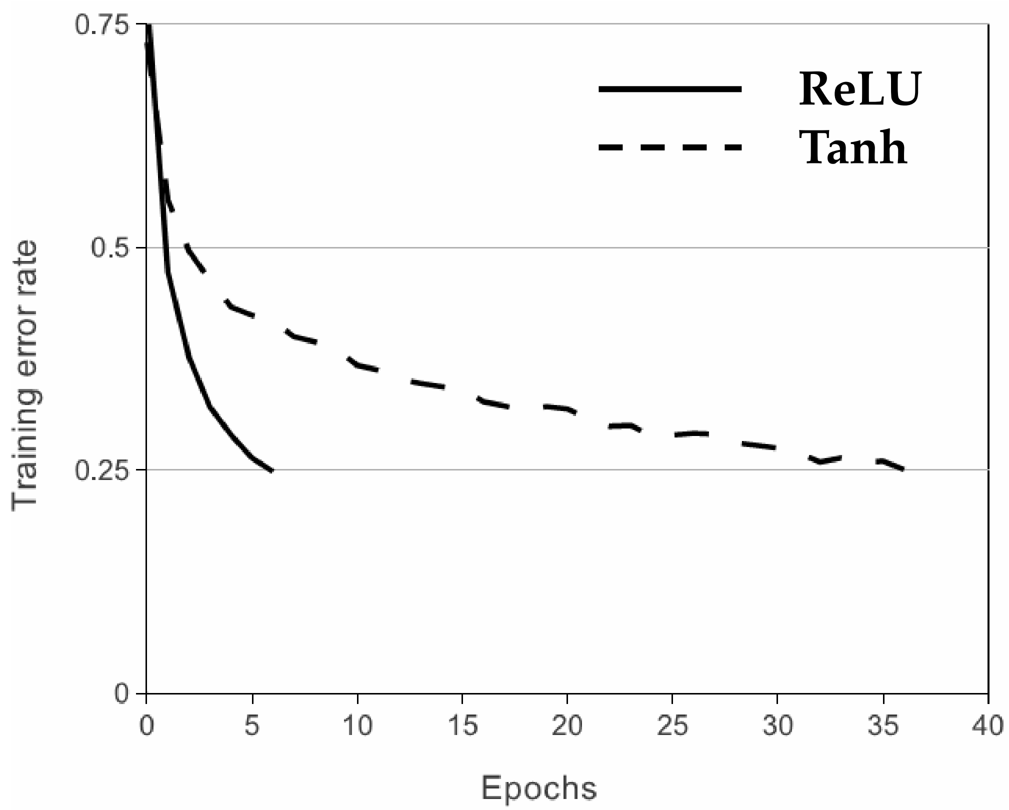


Rectified Linear Unit (ReLU)

- Strong gradients: either 0 or 1
- Fast gradients: just a binary comparison
- Not differentiable at 0, no biggie
  - Rare to have 0 activation anyways
- Dead neurons is an issue
  - Large gradients might cause a neuron to die
  - Higher learning rates might help
  - Assuming a linear layer before ReLU $h(x) = \max(0, wx + b)$, set initial b to a small initial value, $e.g. 0.1$ → more likely the ReLU is positive and therefore there is non zero gradient
- Nowadays ReLU is the default non-linearity

Rectified Linear Unit (ReLU)

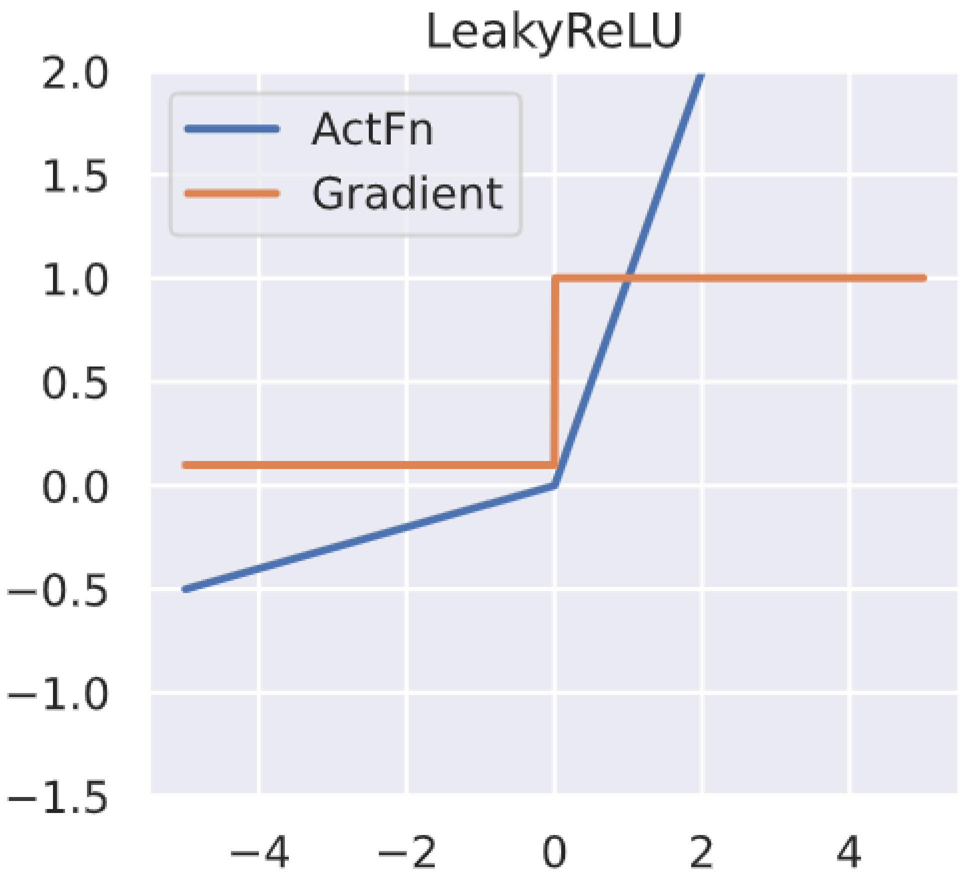
- ReLUs are very data efficient



Leaky ReLU

$$h(x) = \begin{cases} x, \text{when } x > 0 \\ ax, \text{when x} \leq 0 \end{cases}$$

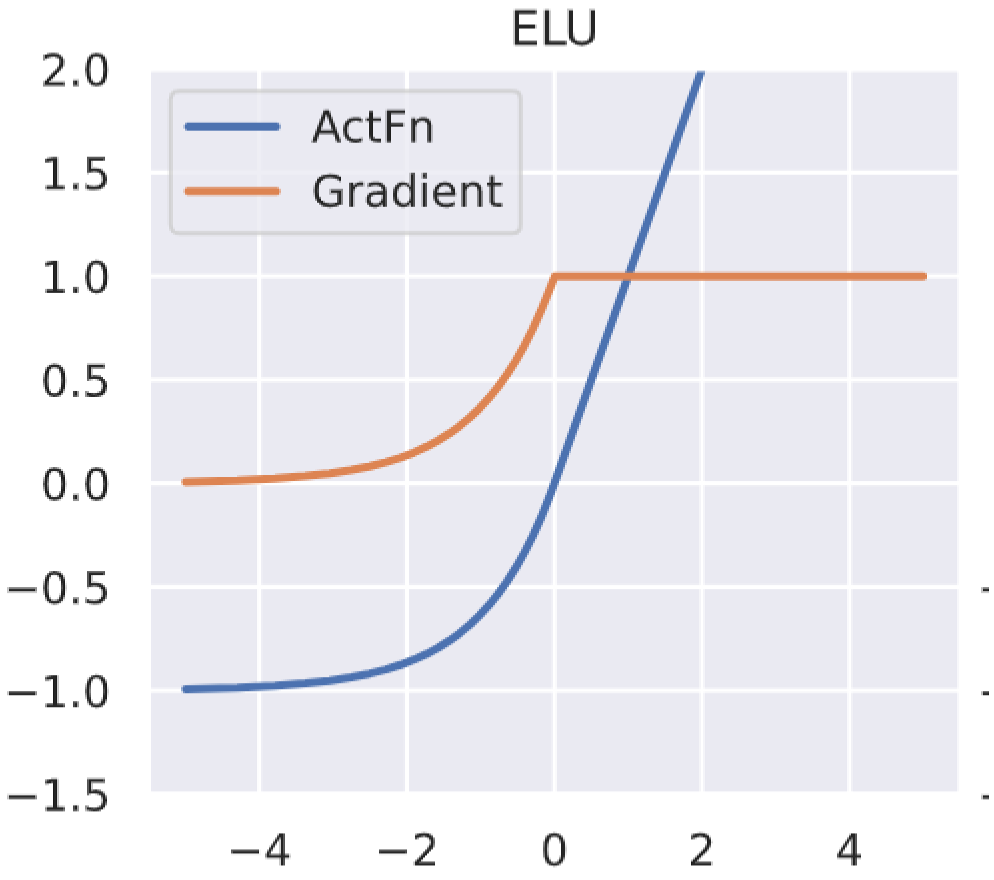
$$\frac{\partial h}{\partial x} = \begin{cases} 1, \text{when } x > 0 \\ a, \text{when x} \leq 0 \end{cases}$$



ELU

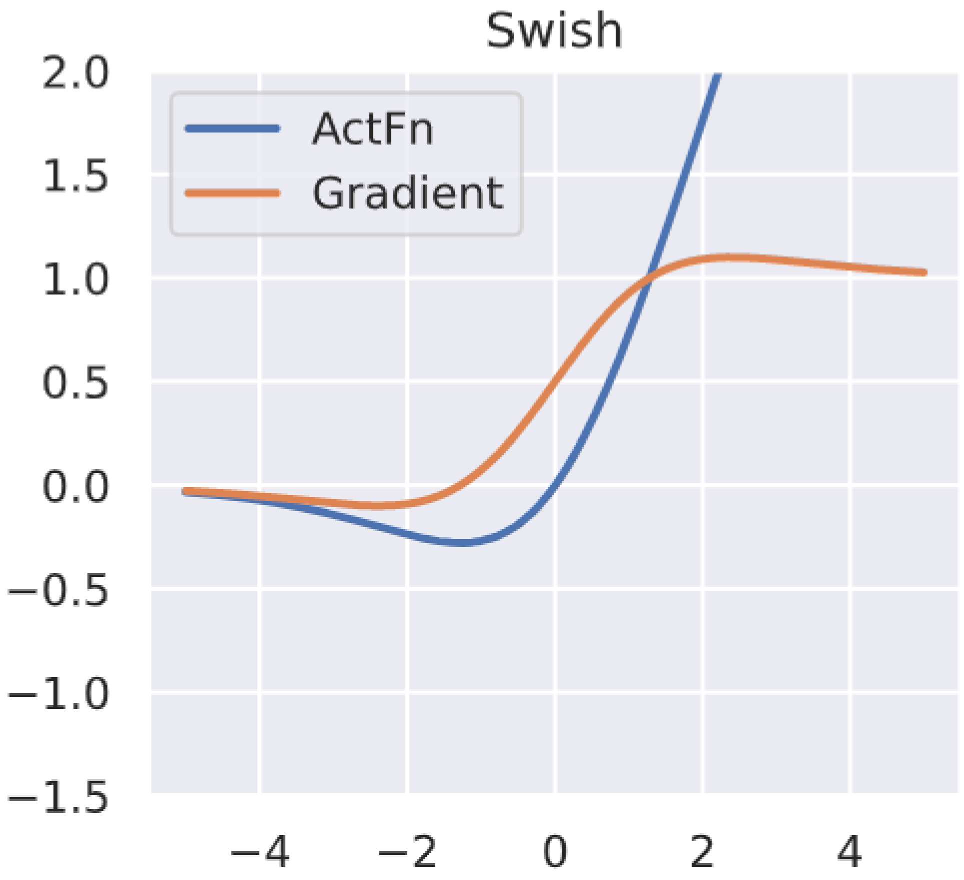
$$h(x) = \begin{cases} x, \text{when } x > 0 \\ \exp(x) - 1\,, \mathrm{x} \leq 0 \end{cases}$$

$$\frac{\partial h}{\partial x} = \begin{cases} 1, \text{when } x > 0 \\ \exp(x)\,, \mathrm{x} \leq 0 \end{cases}$$



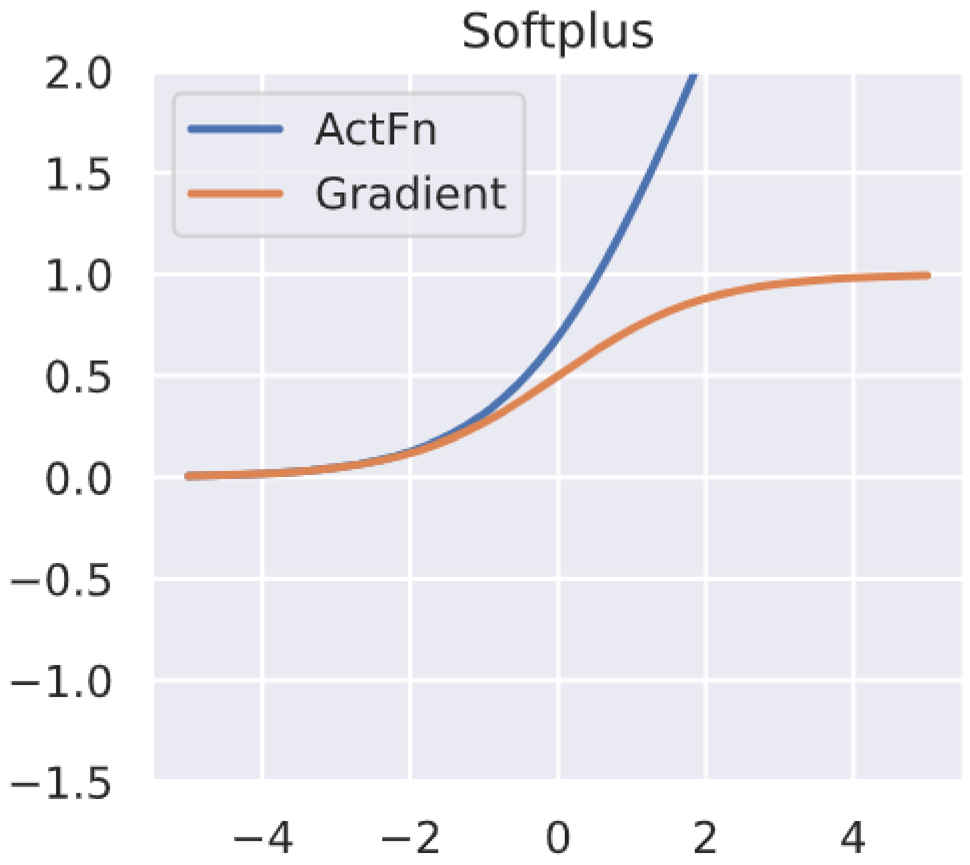
Swish

$$h(x) = x \cdot \sigma(x)$$
$$\frac{\partial h}{\partial x} = \sigma(x)(1 + x - x\sigma(x))$$



Softplus

$$h(x) = \ln(1 + e^x)$$
$$\frac{\partial h}{\partial x} = \frac{1}{1+e^{-x}}$$



Sigmoid and Tanh

Sigmoid

$$h(x) = \frac{1}{1 + e^{-x}}$$

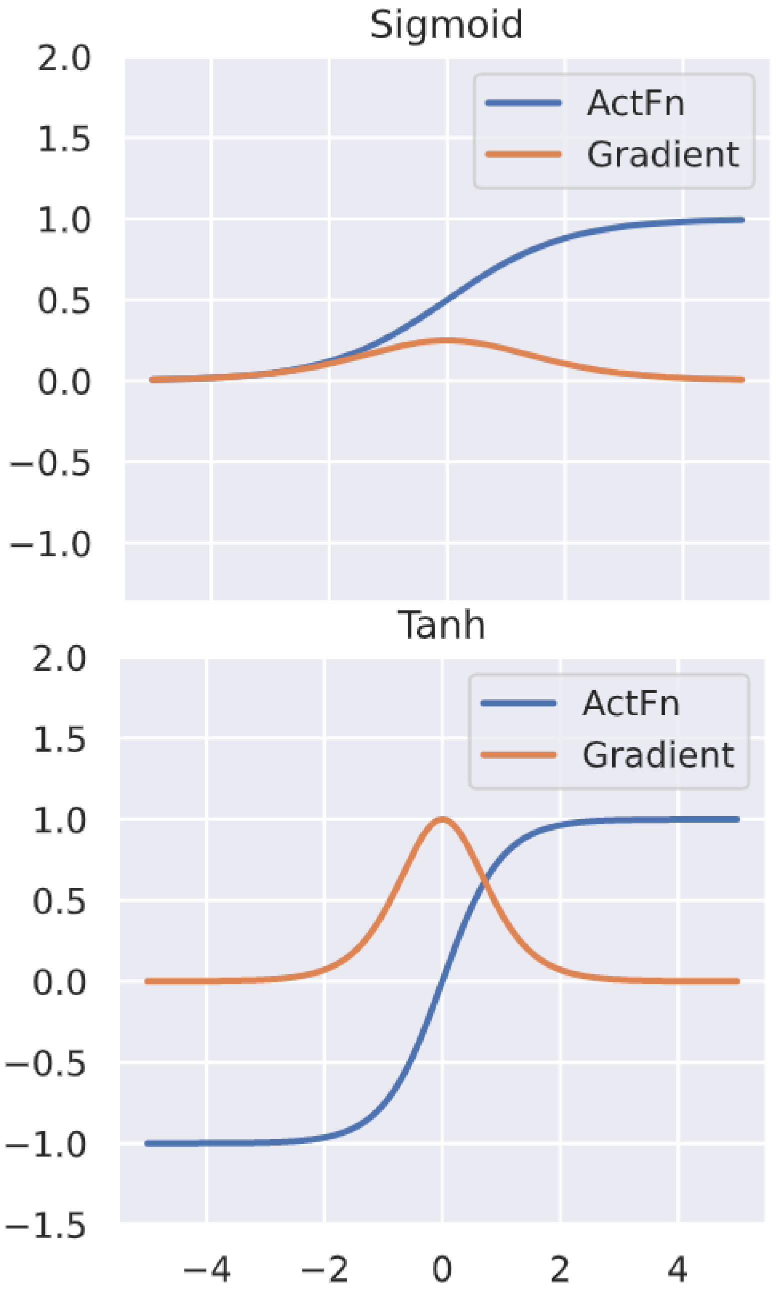
$$\frac{\partial h}{\partial x} = \sigma(x)(1 - \sigma(x))$$

Tanh

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{\partial h}{\partial x} = 1 - tanh^2(x)$$

- Quite similar: $tanh(x) = 2\sigma(2x) - 1$



Centered non-linearities

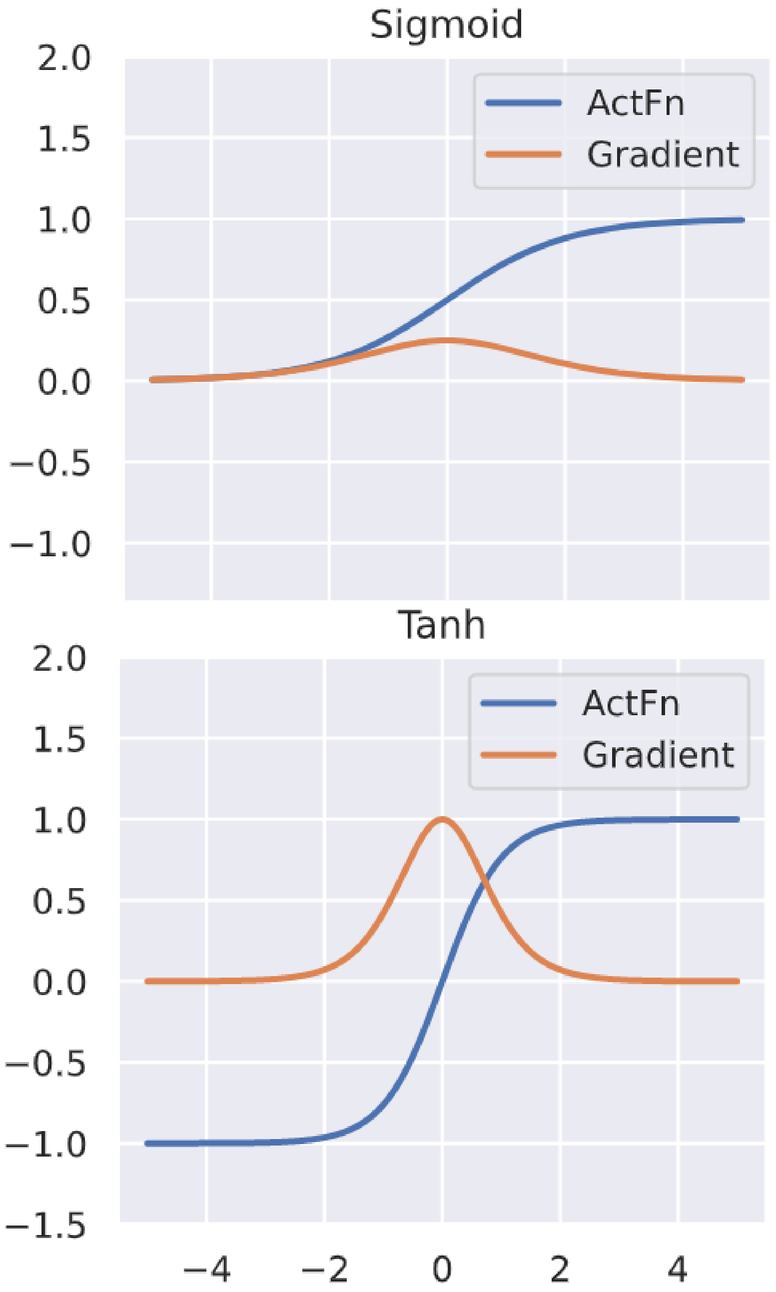
- A good rule of thumb: pick centered non-linearities

- Remember: A deep network is a hierarchy of similar modules
  - The output of one module is the input of the next
- Easier to guarantee consistent behavior
- Example: ReLU-1 returns highly positive numbers, e.g. ~10,000
- ReLU-2 "biased" towards highly positive or negative inputs → dead neurons

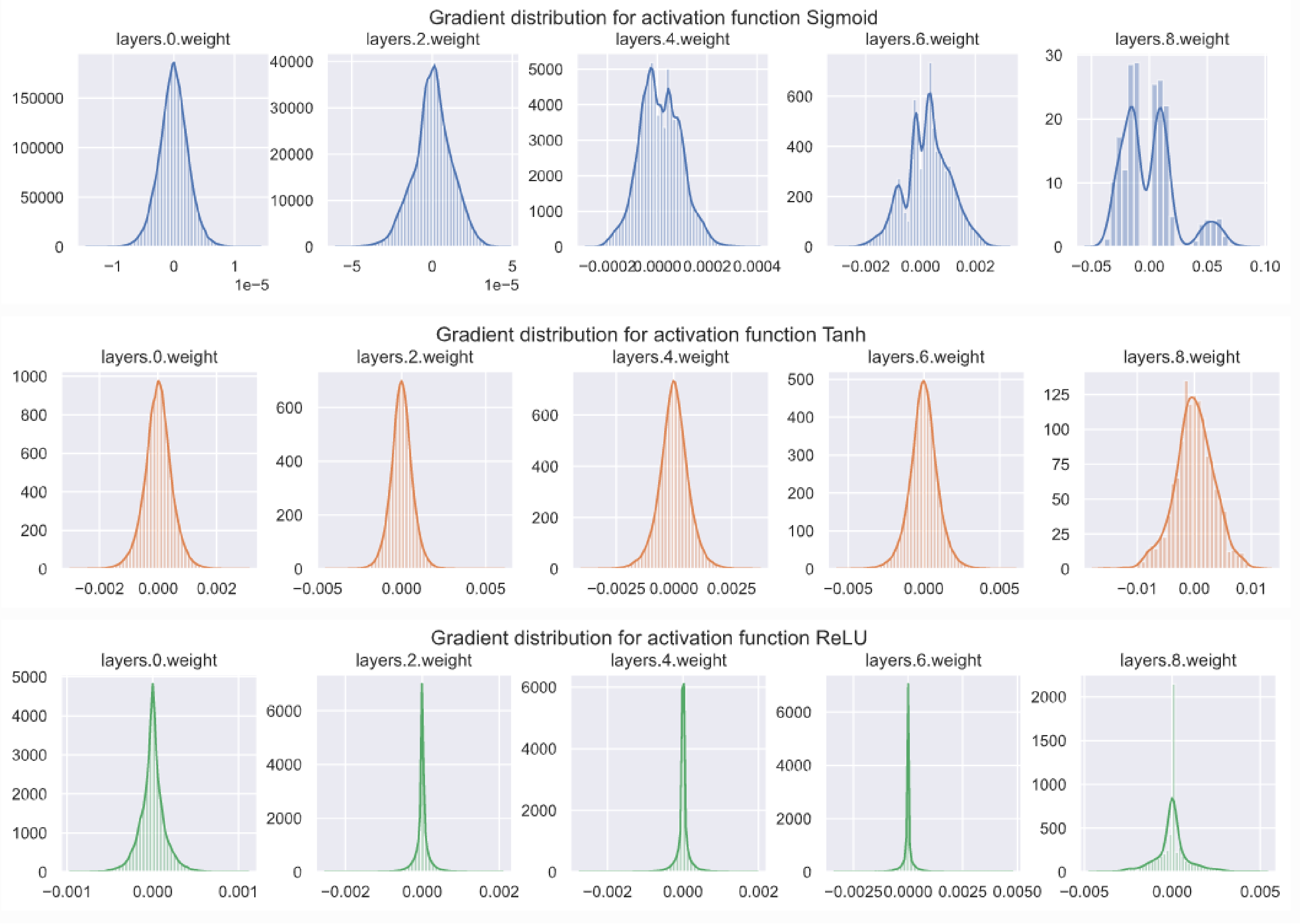
ReLU-2 becomes effectively a linear function

Sigmoid and Tanh

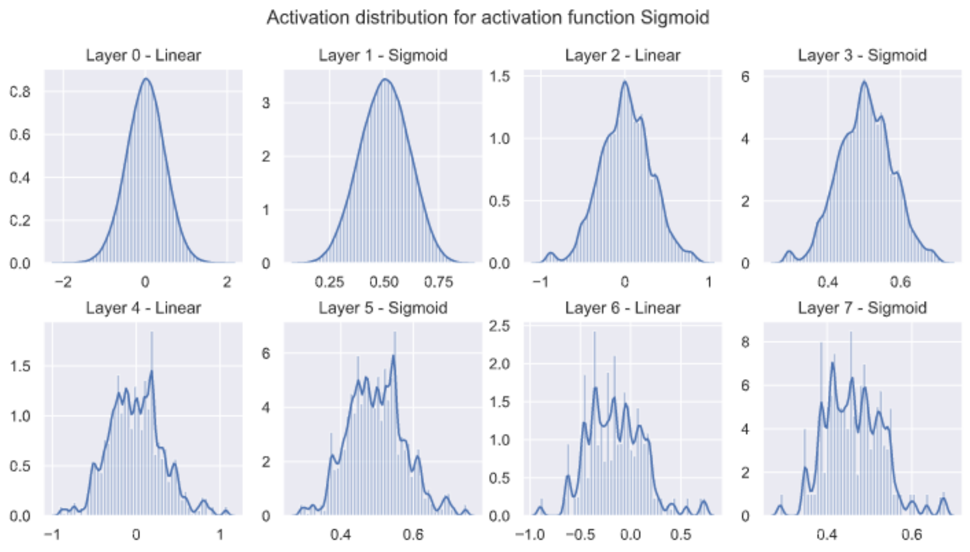
- $tanh(x)$ has better output range $[-1, +1]$
  - Data centered around 0 (not 0.5) → stronger gradients
  - Less "positive" bias for next layers (mean 0, not 0.5)
- Both saturate at the extreme → 0 gradients
  - Easily become "overconfident" (0 or 1 decisions)
  - Undesirable for middle layers
  - Gradients ≪ 1 with chain multiplication
- $tanh(x)$ better for middle layers
- Sigmoids for outputs to emulate probabilities
  - Still tend to be overcofident

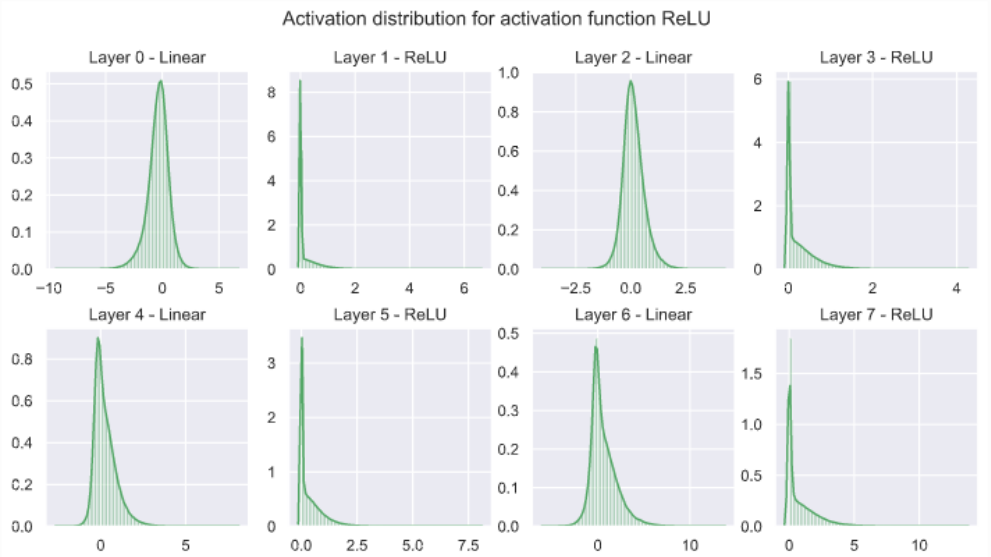


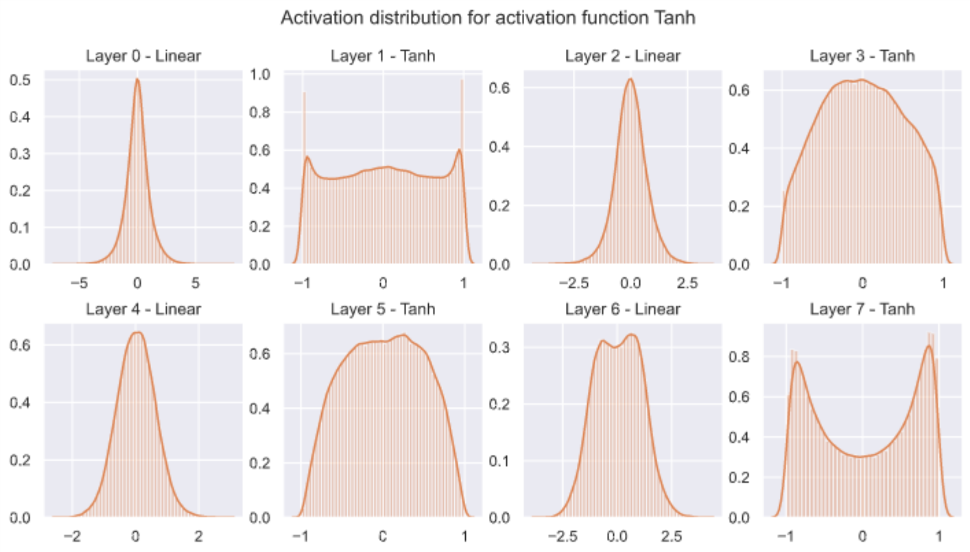
Comparing gradient behavior

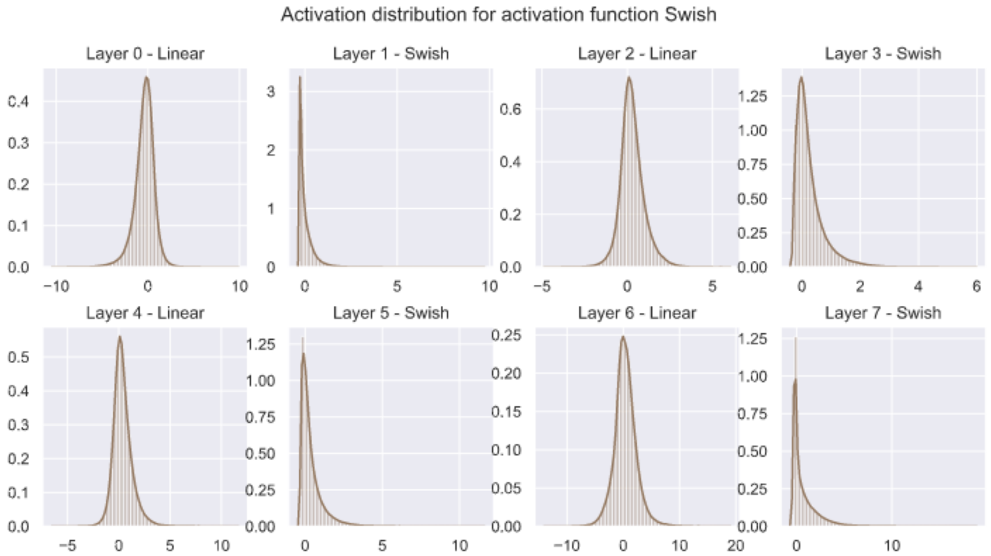


Comparing activation behavior





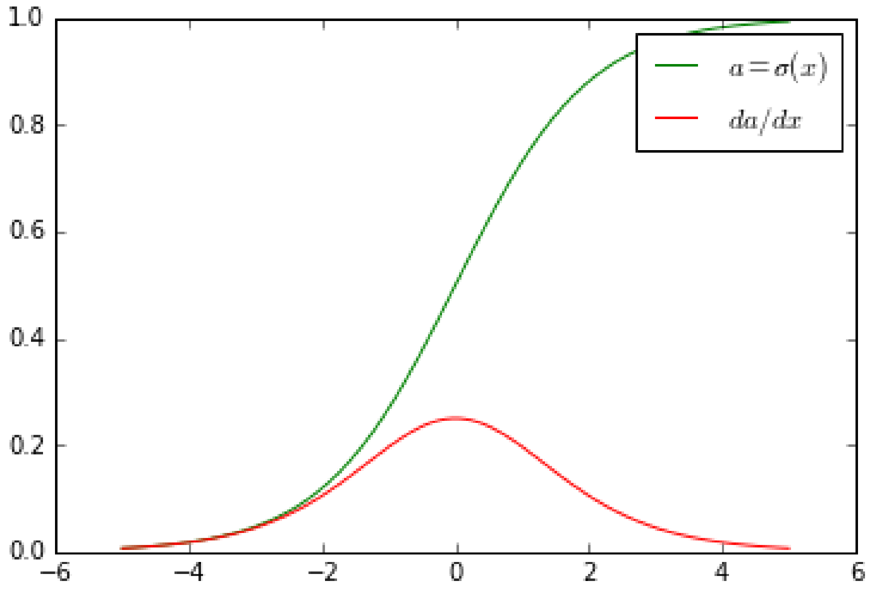




Softmax

Sigmoid

$$h(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$



- Outputs probability distribution, $\sum_{i=1}^{K} h(x_i) = 1$ for K classes
- Avoid exponentianting too large/small numbers for better stability

$$h(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - \mu}}{\sum_j e^{x_j - \mu}} \quad , \mu = \max_i x_i$$

Losses (also modules)

Euclidean loss

$$h(x, y) = 0.5 \, \|y - x\|^2$$

- Suitable for regression problems
- Sensitive to outliers
  - Magnifies errors quadratically

Cross Entropy loss

$$h(x, y) = -\sum_{j=1}^{K} y_j \log x_j, y_j \in \{0, 1\}$$

- Suitable for classification problems
- Derived from max likelihood learning
- Couples well with softmax/sigmoid

New modules

- Any function that is differentiable (almost everywhere), that is

$$\frac{\partial h}{\partial x} \text{ and } \frac{\partial h}{\partial w}$$

for all but a zero-measure set

- Also, modules of modules are just as easy

One module

$$h_1 = \tanh(ReLU(x))$$

Two modules

$$h_1 = ReLU(x)$$
$$h_2 = \tanh(h_1)$$

- Better write them as cascades of simple modules, easier to debug

Many, many more modules out there …

- Many will work comparably to existing ones
  - Not interesting, unless they work consistently better and there is a reason
- Regularization modules: dropout, weight decay
- Normalization modules: batch ℓ_2, ℓ_1-normalization
- Other loss modules: contrastive loss, hinge loss, KL divergence, …
- …