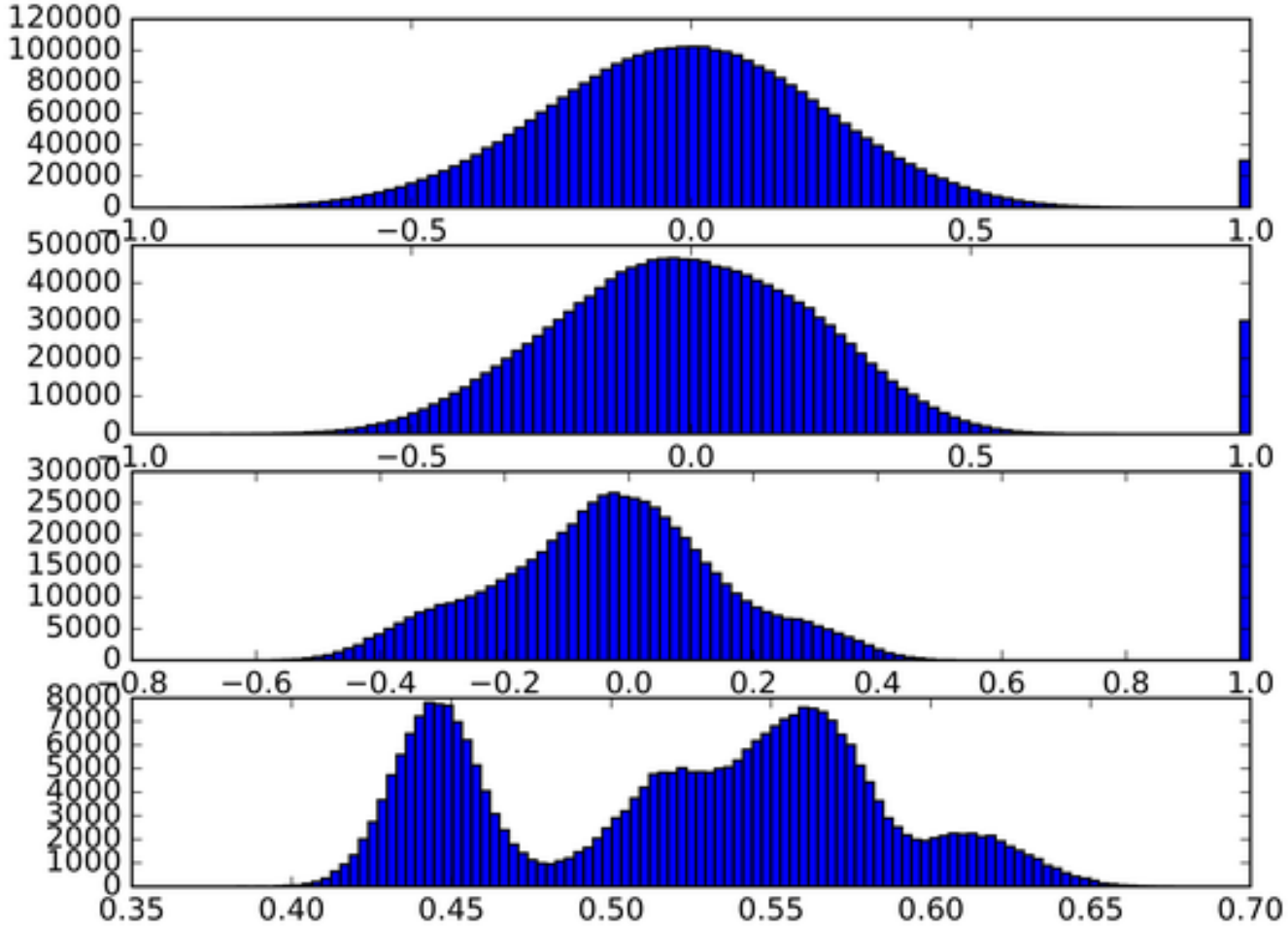
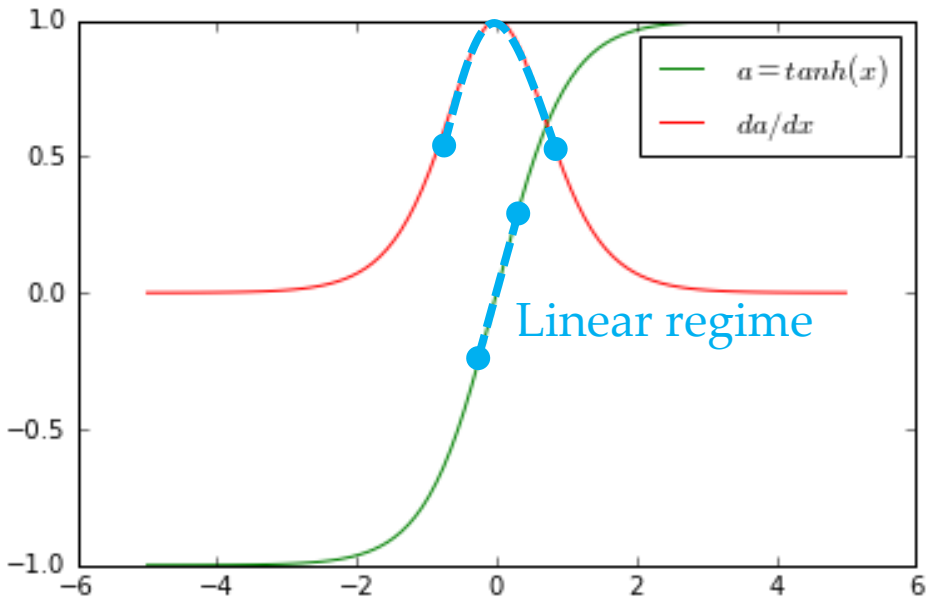
Initialization



Weight initialization

- There are few contradictory requirements:
- Weights need to be small enough
  - Otherwise output values explode
- Weights need to be large enough
  - Otherwise signal too weak to propagate



- Around origin (**0**) for symmetric functions (tanh, sigmoid)
  - In early training stimulate activation functions near their linear regime
  - Larger gradients → faster training

Weight initialization

- Weights initialized **to preserve the variance** of the activations/gradients
  - During the forward and backward computations
  - We want similar input and output variance because of modularity
- Weights must be initialized to be different from one another
  - Don't give same values to all weights (like all 0)
  - All neurons generate same gradient → no learning
- Initialization depends on
  - non-linearities
  - data normalization

Forward propagation

$h_0 = x$

$h_1 = \sigma(w_1 h_0)$ → Store h_1 . Remember that $\partial_x \sigma = \sigma \cdot (1 - \sigma)$

$h_2 = \sigma(w_2 h_1)$ → Store h_2

$\mathcal{L} = 0.5 \cdot \|l - h_2\|^2$

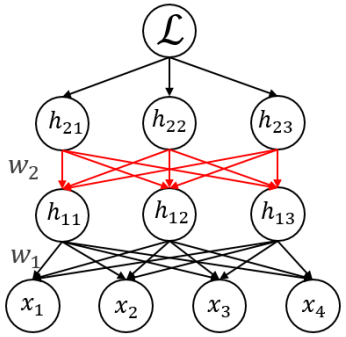
Backward propagation

$$\frac{d\mathcal{L}}{dh_2} = -(y^* - h_2)$$

$$\frac{d\mathcal{L}}{dw_2} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dw_2} = \frac{d\mathcal{L}}{dh_2} h_1 \sigma(w_2 h_1)(1 - \sigma(w_2 h_1)) = \frac{d\mathcal{L}}{da_2} h_1 h_2 (1 - h_2)$$

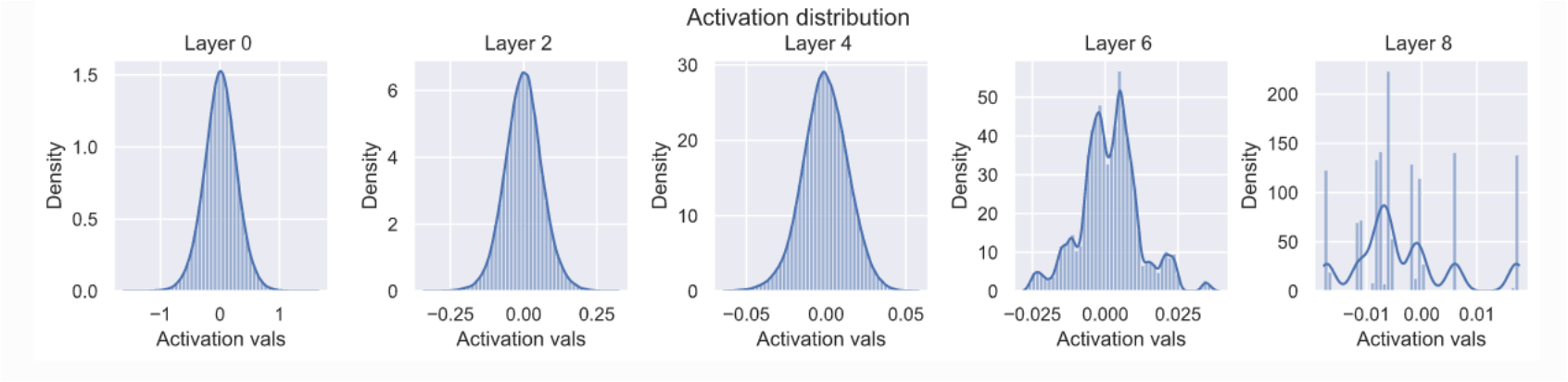
$$\frac{d\mathcal{L}}{dh_1} = \frac{d\mathcal{L}}{dh_2}\frac{dh_2}{dh_1} = \frac{d\mathcal{L}}{dh_2} w_2 \sigma(w_2 h_1)(1 - \sigma(w_2 h_1)) = \frac{d\mathcal{L}}{dh_2} w_2 h_2 (1 - h_2)$$

$$\frac{d\mathcal{L}}{dw_1} = \frac{d\mathcal{L}}{dh_1}\frac{dh_1}{dw_1} = \frac{d\mathcal{L}}{dh_1} h_0 \sigma(w_1 h_0)(1 - \sigma(w_1 h_0)) = \frac{d\mathcal{L}}{dh_1} h_0 h_1 (1 - h_1)$$

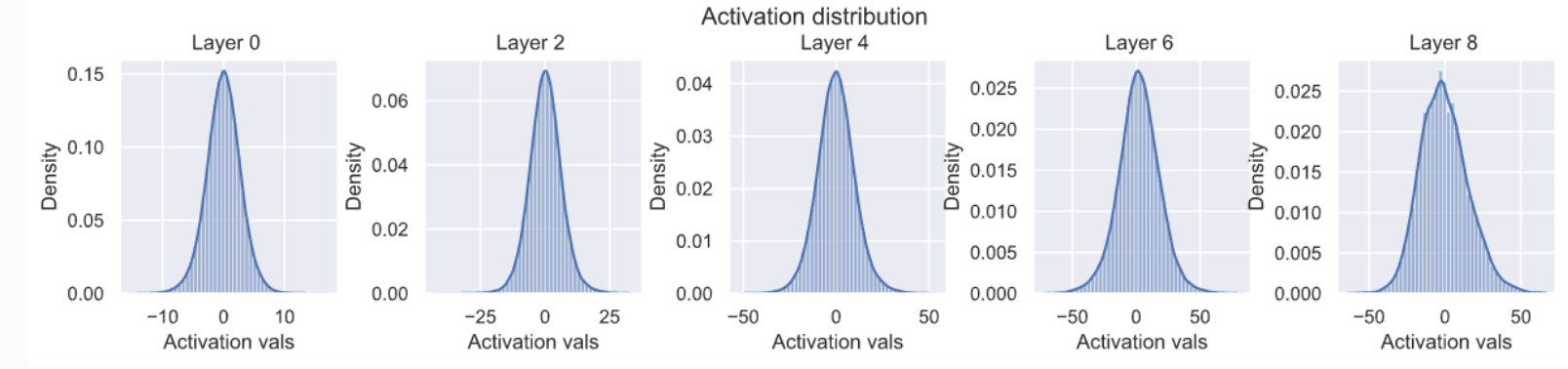


Bad initialization can cause problems

Initializing weights in every layer with same constant variance → can diminish variance in activations



Initializing weights in every layer with increasing variance → can explode the variance in activations



Initializing weights by preserving variance

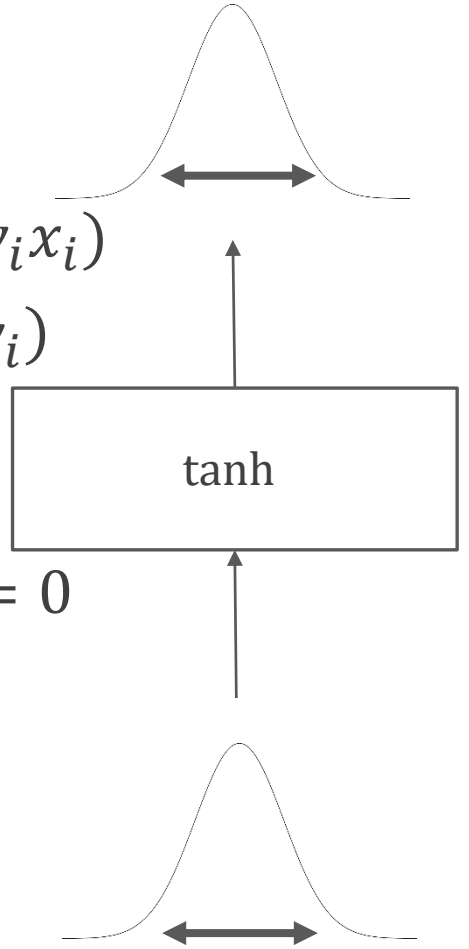
- For x and y independent
  - $var(xy) = \mathbb{E}[x]^2 var(y) + \mathbb{E}[y]^2 var(x) + var(x)var(y)$
- For $a = wx \Rightarrow var(a) = var(\sum_i w_i x_i) = \sum_i var(w_i x_i) \approx d \cdot var(w_i x_i)$

$$var(w_i x_i) = \mathbb{E}[x_i]^2 var(w_i) + \mathbb{E}[w_i]^2 var(x_i) + var(x_i)var(w_i)$$

$$= var(x_i)var(w_i)$$

because we assume that x_i, w_i are unit gaussians $\rightarrow \mathbb{E}[x_i] = \mathbb{E}[w_i] = 0$

- So, the variance in our activation $var(a) \approx d \cdot var(x_i)var(w_i)$



Understanding the difficulty of training deep feedforward neural networks, Glorot, Bengio, 2010

Initializing weights by preserving variance

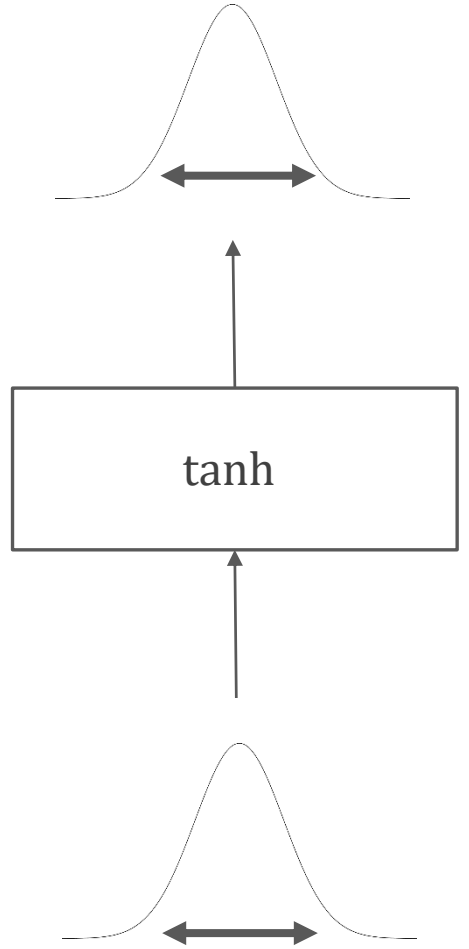
- Since we want the same input and output variance

$$var(a) = d \cdot var(x_i)var(w_i) \Rightarrow var(w_i) = \frac{1}{d}$$

- Draw random weights from

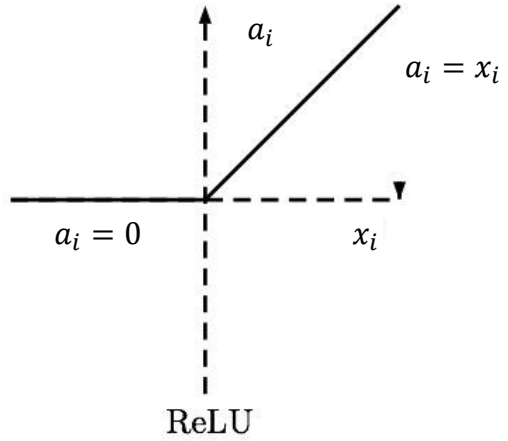
$$w \sim N(0, 1/d)$$

where d is the number of input variables to the layer



Understanding the difficulty of training deep feedforward neural networks, Glorot, Bengio, 2010

Initialization for ReLUs



- Unlike sigmoidals, ReLUs return 0 half of the time
  - $\mathbb{E}[w_i] = 0$ but $\mathbb{E}[x_i] \neq 0$
- Redoing the computations

$$var(w_i x_i) = var(w_i)(\mathbb{E}[x_i]^2 + var(x_i)) = var(w_i)\mathbb{E}\left[x_i^2\right] \ (var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2)$$

- $\mathbb{E}\left[x_i^2\right] = \int_{-\infty}^{\infty} x_i^2 p(x_i) dx_i = \int_{-\infty}^{\infty} \max(0, a_i)^2 \, p(a_i) da_i = \int_0^{\infty} a_i^2 p(a_i) da_i$

$$= 0.5 \int_{-\infty}^{\infty} a_i^2 p(a_i) dy_i = 0.5 \cdot \mathbb{E}\left[a_i^2\right] = 0.5 \cdot var(a_i)$$

- Draw random weights from $w \sim N(0, 2/d)$

Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, He, Zhang, Ren, Sun, 2015 Link