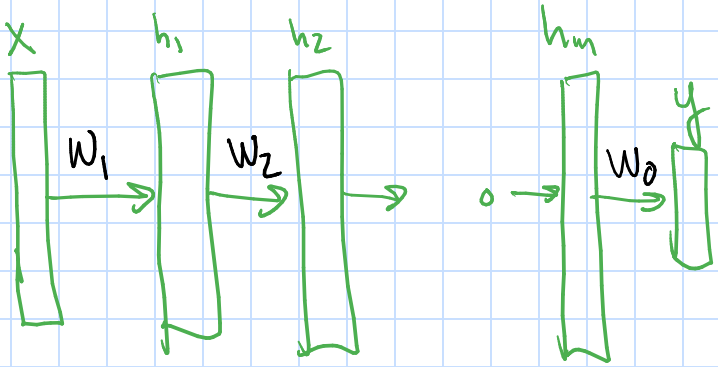
Deep Learning 03/22/2021

$\rightarrow \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_0 \end{bmatrix}$ (raw input vector) $\rightarrow$ (slightly higher representation) $\rightarrow \cdots$ (higher representation) $\rightarrow$ (people waiting ...)



$h_1 = g(W_1 x + b_1)$

$h_2 = g(W_2 h_1 + b_2)$

$\downarrow$

(even higher representation)

Local gradient: $\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$

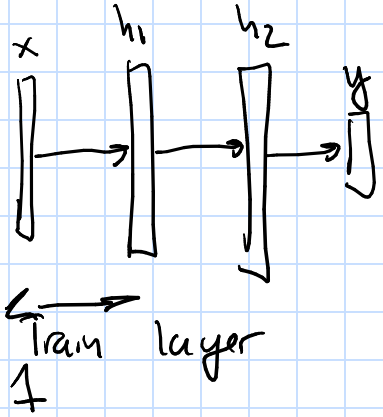
↑ from the previous layer

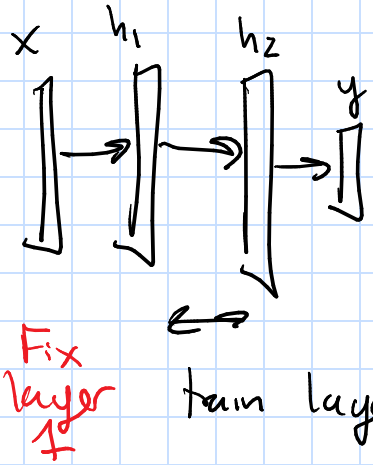
The $\delta_j(n)$'s may vanish due to the repeated multiplications as we propagate the error signal back through the network. This is a problem if we want to train a deep net. See Erhan et al (2009) on deep nets performance on MNIST as a function of hidden layers 99% +

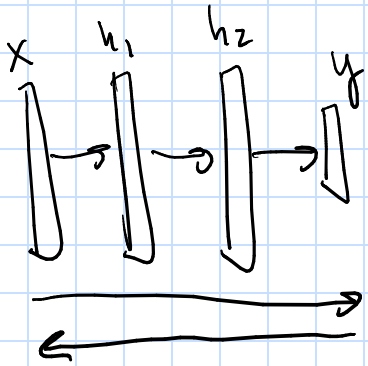
More expensive + Worse Error!

- Optimization of neural net is non-convex and will lead us to a local minima. A deep network trained w/ back-prop alone leads to a worse local minima

Layer-wise Pretraining

① 

② 

③ 

Apply backprop to the network

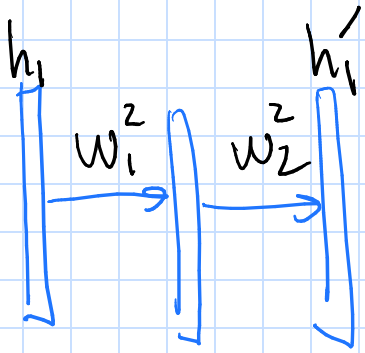
Core Idea

We are focusing on modelling $p(x)$ at each successive layer. Worry about modelling $p(y|x)$ later.

Traditional ML: $f: \mathcal{X} \mapsto \mathcal{Y}$

New Idea: $\mathcal{X} \to \mathcal{H} \to \mathcal{Y}$

↳ hidden (latent) features



new features

Encoder

$$h = \sigma(W_1 x + b_1)$$

Decoder

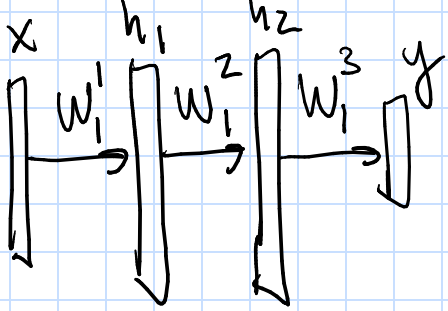
$$x' = \sigma(W_2 h + b_2)$$

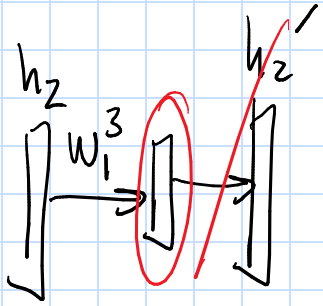
Autoencoder

$N = 2500$

$N_e = 500$

$N_d = 2000$





Transfer learning

$$h_1 = \sigma(W_1^1 x + b_1^1)$$

$$h_2 = \sigma(W_1^2 h_1 + b_1^2)$$

$$h_3 = \sigma(W_1^3 h_2 + b_1^3)$$

$$y = f(W_{out} h_3 + b_{out})$$