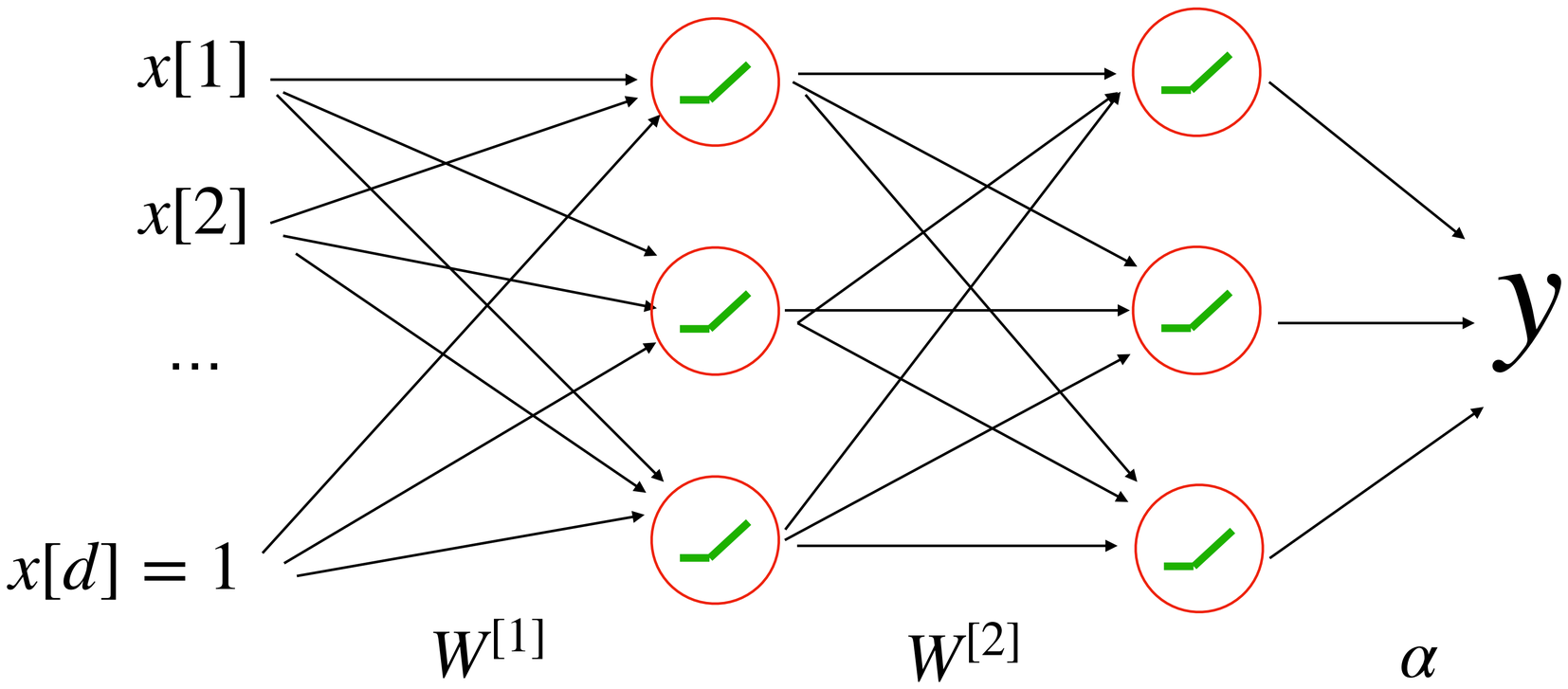
Let 22: Deep Learning

Backpropogation

CS3780/5780

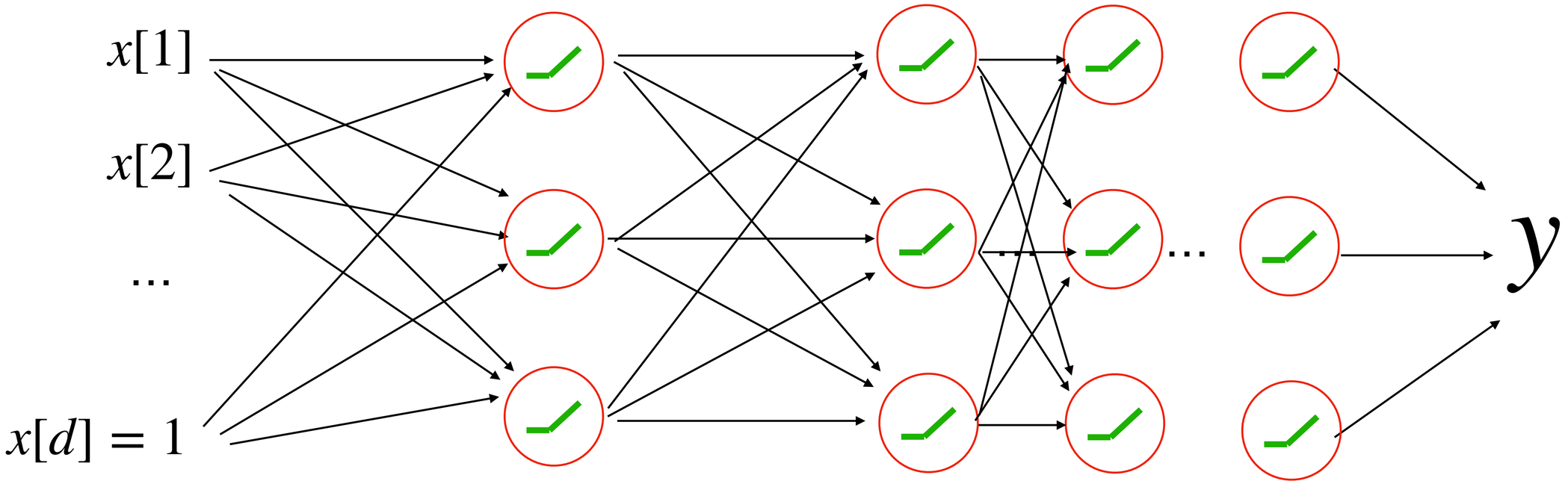
A multi-layer fully connected neural network



$$y = \alpha^\top \sigma\left(W^{[2]}\sigma\left(W^{[1]}x\right)\right) + b$$

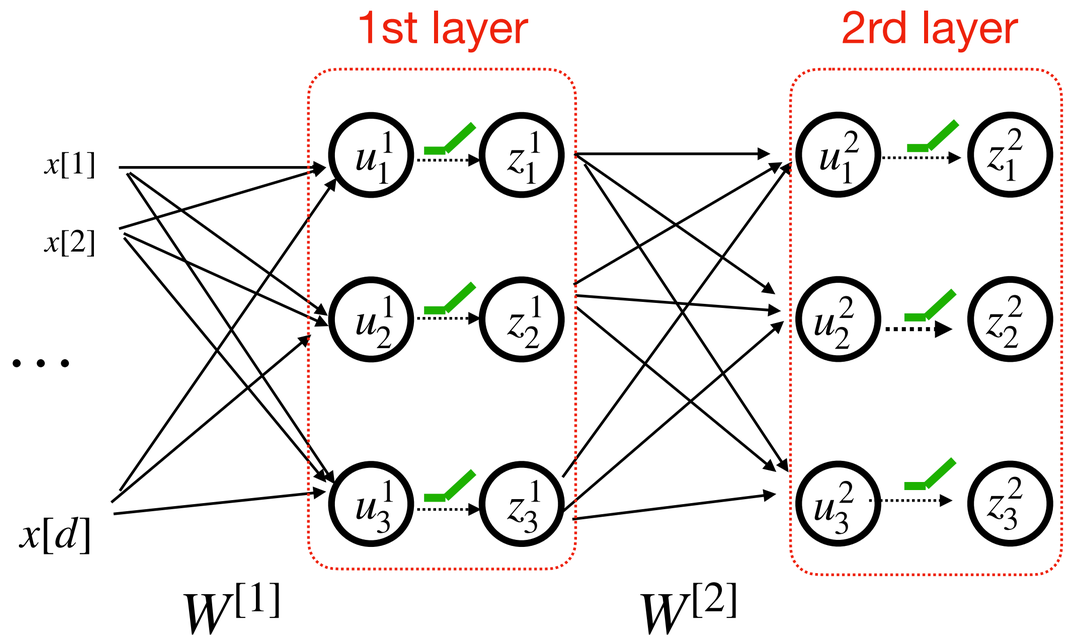
Relu activation models piecewise linear functions.

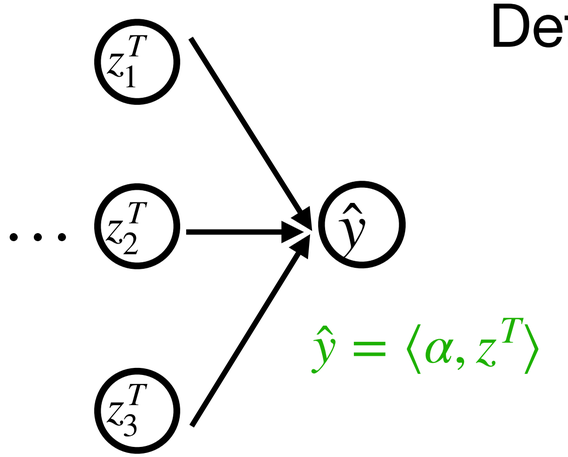
The benefits of going deep



- Wide enough (single hidden layer) networks can capture any piecewise linear function
- Depth can exponentially reduce width of neural netowrk

A multi-layer fully connected neural network





$z^0 = x$ $u^1 = W^{[1]} z^0 + b^1$ $z^1 = \sigma(u^1)$ $z^T = \sigma(u^T)$

Define it by a forward pass:

$$z^0 = x$$

For t = 1 to T:

$$u^t = W^{[t]} z^{t-1} + b^t$$

$$z^t = \sigma\left(u^t\right)$$

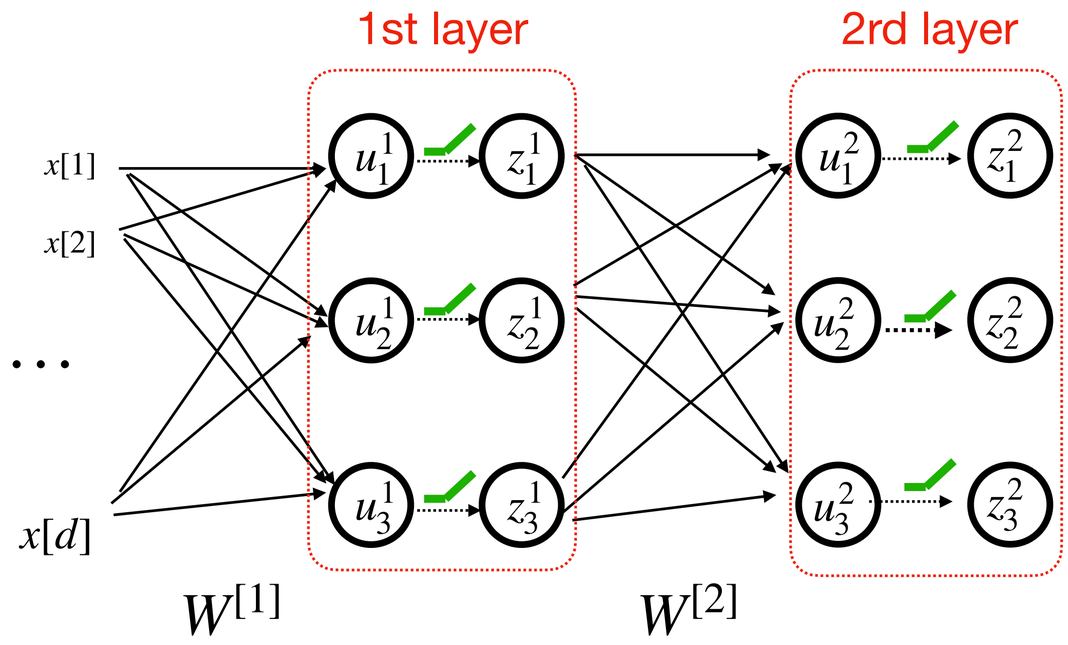
End For

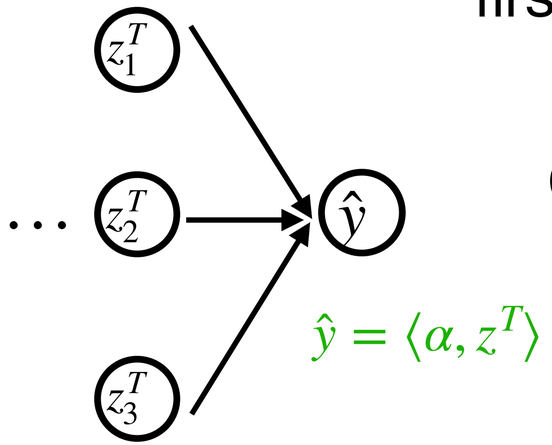
$$y = \alpha^\top z^{[T]} + b$$

Q: what is the computation complexity of the forward pass?

A: linear in # of Edges + # of nodes

Forward Pass





Forward Pass: Starting from first layer moving to the last

Computes output of each neuron

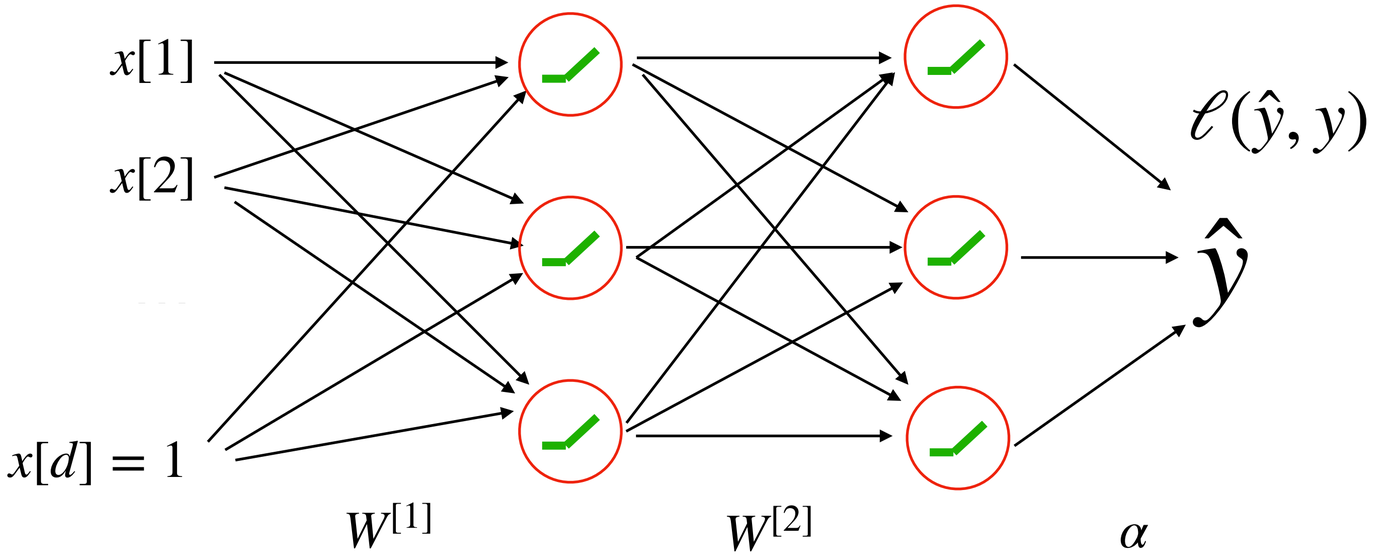
+

Final Output

$z^0 = x$ $u^1 = W^{[1]} z^0$ $z^1 = \sigma(u^1)$ $z^T = \sigma(u^T)$

Training neural network via SGD

$$h(x) := \alpha^\top \sigma\left(W^{[2]}\sigma\left(W^{[1]}x\right)\right) + b$$



Let $\ell(h(x), y)$ be any differentiable loss function

Compute gradients:

$$\frac{\partial \ell(h(x), y)}{\partial W^{[1]}} \qquad \frac{\partial \ell(h(x), y)}{\partial W^{[2]}}$$

$$\frac{\partial \ell(h(x), y)}{\alpha} \qquad \frac{\partial \ell(h(x), y)}{b}$$

Training neural network via SGD

Mini-batch Stochastic gradient descent

$\theta = [W^{[1]}, W^{[2]}, \alpha, b]$

go through dataset multiple times

For epoc $e = 1$ to E:

important (unbiased estimate of the true gradient)

Randomly shuffle the dataset

Split the data into n/B many batches $\mathcal{D}_i$, each with size B

For j = 1 to n/B

Mini-batch gradient $g = \sum_{x,y \in \mathcal{D}_i} \nabla_\theta \ell(h_\theta(x), y)/B$

$\theta = \theta - \eta g$

Chain Rule of Calculus

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

Lets first look at the one dimensional example

Backpropagation

Backward Pass: Computes gradients given variables from Forward Pass

$$\nabla_\alpha \ell(\hat{y}, y) = \ell'(\hat{y}, y)\ z^T$$

$$\nabla_b \ell(\hat{y}, y) = \ell'(\hat{y}, y)$$

Set $\delta^T = \ell'(\hat{y}, y)\ \alpha \odot \sigma'(u^T)$

For $t = T$ to 1

$$\nabla_{W^{[t]}} \ell(\hat{y}, y) = \delta^t\ (z^{t-1})^\top$$

$$\nabla_{b^t} \ell(\hat{y}, y) = \delta^t$$

$$\delta_{t-1} = \sigma'(u^{t-1}) \odot W^{[t]} \delta_t$$

End For

Optimization Tricks

- Use momentum
- Reduce stepwise as we proceed
- Use minibatch of small size
- Scale features to within [0,1]
- Randomly initialize weights from a normal distribution (scaled)

Batch Normalization

- For each layer, over compute:

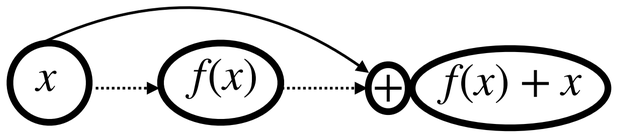
$$\mu^t = \frac{1}{B} \sum_{(x,y) \in D_i} u^t(x, y) =$$ average u^t over minibatch

$\sigma^t =$ variance of u^t over minibatch

- Renormalize as: $\mathrm{BN}(u^t) = \gamma_t \frac{(u^t - \mu_t)}{\sqrt{\sigma^t + \epsilon}} + \beta_t$
- Compute $z^t = \sigma(BN(u^t))$
- γ_t and β_t are two parameter that are learnt
- Quickens training, reduces internal covariate shift

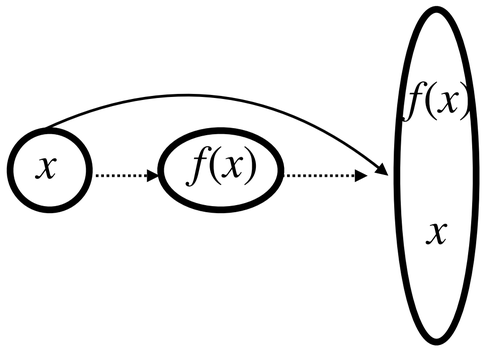
Residual Connections

- Add input to output activations after each layer
- $\phi(x) = x$ is hard to learn
- Helps with the problem of vanishing gradients



Dense Connections

- Concatenate input to output activations after each layer



Next Lecture(s)

- Convolutional Neural Networks to handle images as input
- Word-2-vec and Transformers for large language models