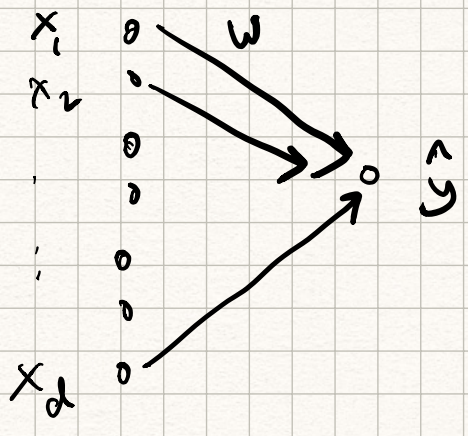
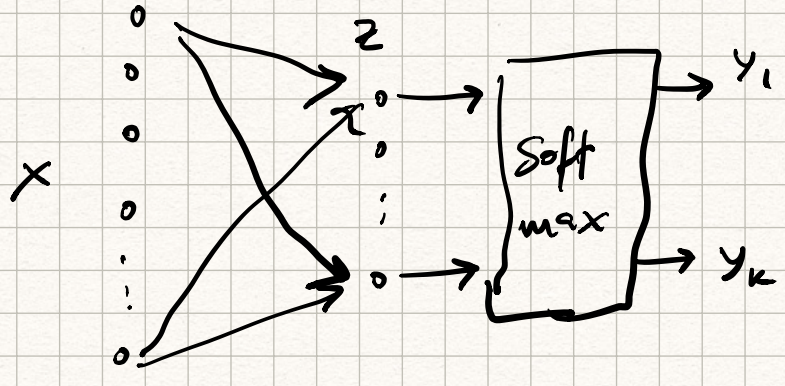
ECE-GY 6143
Lecture 9

- Recap: Linear models
- Neural nets
  - Architecture
  - Training via backpropagation.

- Reminder: Midterm exam due at 8pm ET.



Linear regression

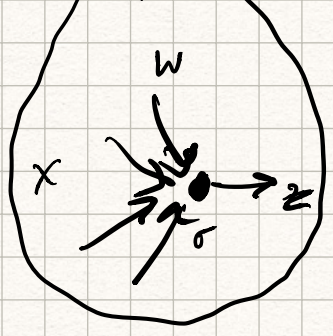


Logistic regression

- Feed forward
- Directed acyclic graphs

Neural network

Primitive: $z = \sigma(\langle w, x \rangle + b)$



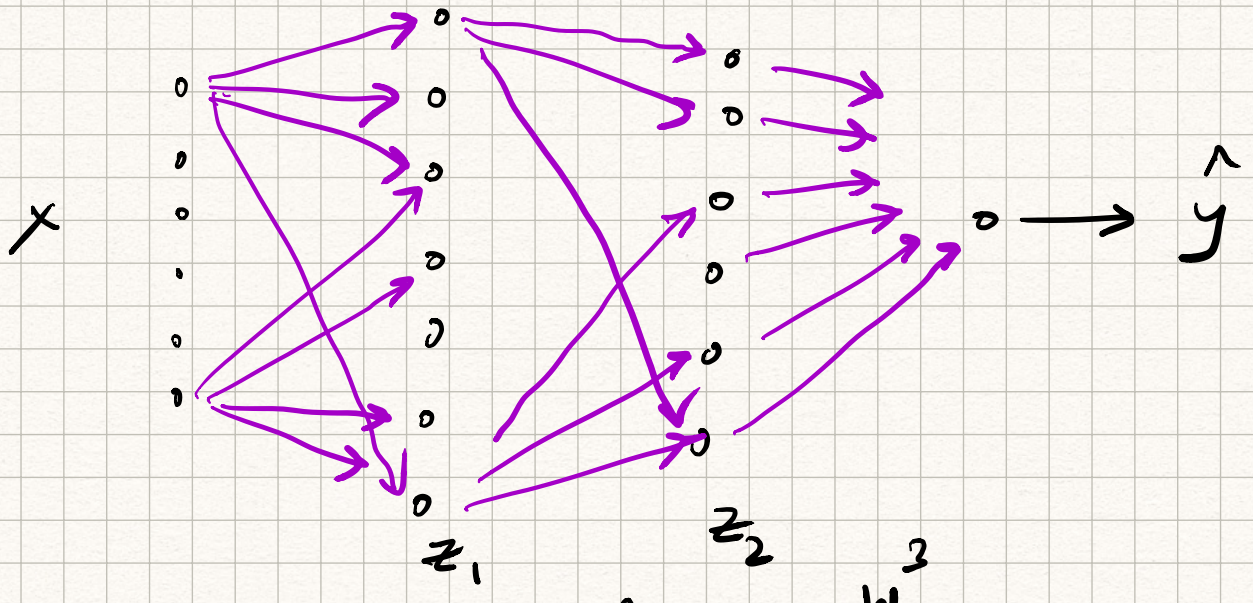
"Neuron".

σ: "Activation function".

Examples: $\sigma(z) = z$ → Linear regression

$\sigma(z) = \frac{1}{1+e^{-z}}$ → Logistic regression.

$\sigma(z) = \text{sign}(z)$ → Perceptron



W^1 W^2 W^3

Functional form

$$z_1 = \sigma^{(1)}(W^{(1)} x + b^{(1)})$$

$$z_2 = \sigma^{(2)}(W^{(2)} z_1 + b^{(2)})$$

$$\hat{y} = \sigma^{(3)}(W^{(3)} z_2 + b^{(3)}).$$

"3-layer network" / "2-hidden-layer network"

Why? - Historical / biological

- Mathematical.

Universal approximation theorem (Cybenko 1989)

:= Every continuous function can be approximated by a 1-hidden layer network of finite size (for most activation functions)

- Drawback
  - Existence theorem only
  - Practical issues

Representation
- What network size & shape?

Training
- How to choose the network weights!

Generalization
- Does the learned generalize to unseen inputs?

Neural network architecture.

Design criteria: 1) Weights

2) Activation function.

Activation functions

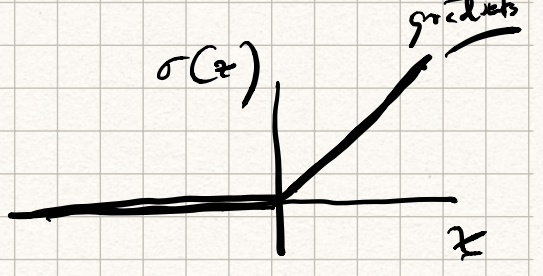
- $\sigma(z) = z$

[Not too useful for multi-layer networks]

$$y = W^{(3)} W^{(2)} W^{(1)} x$$
$$= \bar{W} x$$

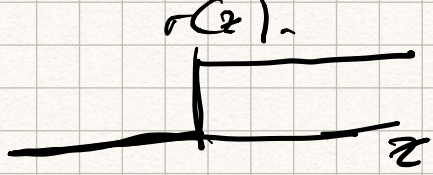
- $\sigma(z) = \frac{1}{1+e^{-z}}$ sigmoid

- $\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$ (tanh)



- $\sigma(z) = \underline{\text{ReLU}}(z)$
  $= \max(z, 0)$

- $\sigma(z) = \text{sgn}(z)$
- $\sigma(z) = \text{HT}(z)$



Weights.

- Dense layers → weights are arbitrary.
- Convolution layers → weights implement convolution.
- Pooling layers → Downsampling size of output.

- Batch normalization → Rescaling.
- Recurrent layers → Feedback.
- Attention layers → NLP (etc.)

Q: How do I mix & match?

A: No correct answer.
Thumb rule: Just use a good existing architecture.

Training

Given architecture, how do I train a network?

- Define loss function (+ optional regularization).
- Use some variant of gradient descent (GD/SGD/...).

$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, f_W(x_i)\right) + \lambda R(W)$$

$$W^{t+1} \leftarrow W^t - \alpha^t \nabla L(W)$$

by example

$$x \xrightarrow{\sigma} y$$

$$z = wx + b$$

$$f(z) = \sigma(z)$$

$$L(w,b) = \frac{1}{2}(y - f(z))^2 + \lambda w^2$$

$$= \frac{1}{2}(y - \sigma(wx+b))^2 + \lambda w^2$$

$$\nabla L(w,b) = \begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$

$$\frac{\partial L}{\partial w} = (\sigma(wx+b) - y) \cdot \sigma'(wx+b) \cdot x + 2\lambda w$$

$$\frac{\partial L}{\partial b} = (\sigma(wx+b) - y)\, \sigma'(wx+b)$$

Backpropagation / backprop.

- exploit the structure of the network

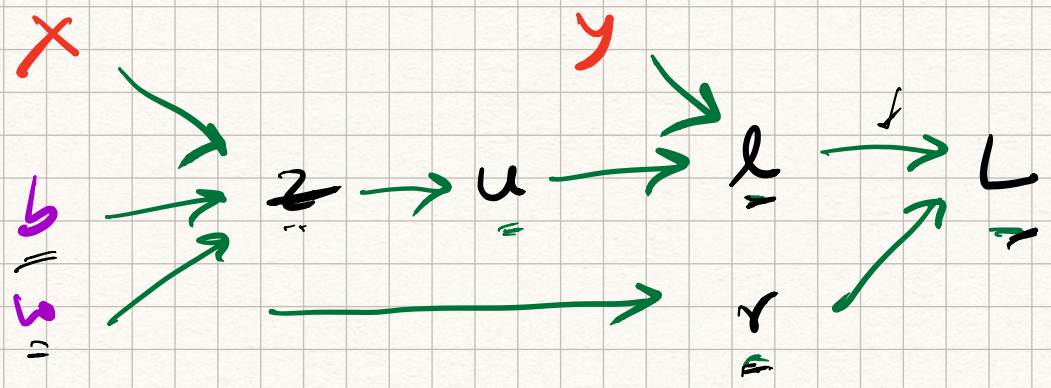
$$z = wx + b$$

$$u = \sigma(z)$$

$$\ell = \frac{1}{2}(u-y)^2$$

$$r = w^2 \qquad \text{(1)}$$

$$L = \ell + \lambda r$$



$i = 1, 2, \ldots, N$

N ↑ Number of nodes

FORWARD (graph):

For node $i = 1, 2, \ldots, N$:

Compute v_i as a function of parents of i.

BACKWARD (graph):

For node $i = N-1, \ldots, 1$:

Compute

$$\frac{\partial L}{\partial v_i} = \sum_{j \in \text{children}(i)} \frac{\partial L}{\partial v_j} \cdot \frac{\partial v_j}{\partial v_i}$$

$$\frac{\partial L}{\partial v} := \partial_v L$$

$$\partial_L L = 1$$

$$\partial_\ell L = 1$$

$$\partial_r L = \lambda$$

$$\partial_u L = \partial_e L \cdot \partial_u e$$

$$= 1 \cdot (u - y)$$

$$\partial_z L = \partial_u L \cdot \partial_z u$$

$$= \partial_u L \cdot \sigma'(z)$$

$$\partial_w L = \partial_z L \, \partial_w z + \partial_r L \, \partial_w r$$

$$= \partial_z L \cdot x + \lambda \cdot 2w$$

$$\partial_b L = \partial_z L \cdot \partial_b z$$

$$= \partial_z L \cdot$$

- Computationally efficient
- Modular
- Scalable.