6.036 Recitation Notes

(3/10/17)

Helen Zhou, Nathan Landman

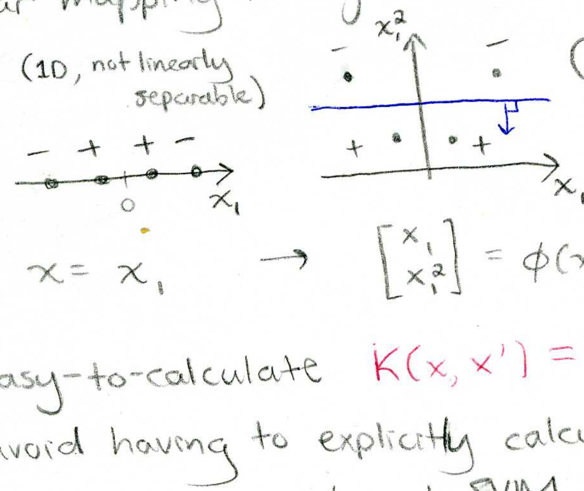
Agenda:

- Why neural nets
- Basics
- Activation Functions
- Forward Propagation
- Backpropagation

Context

- In this class, we started off with linear models
  - ↳ perceptron for classification
  - ↳ SVM + stochastic gradient descent for regression
- How have we introduced nonlinearity?
  - ↳ nonlinear mapping to higher dimensional feature space $x \rightarrow \phi(x)$

e.g: (1D, not linearly separable) (linearly separable)



$x = x_1 \rightarrow \begin{bmatrix} x_1 \\ x_1^2 \end{bmatrix} = \phi(x)$

  - ↳ for easy-to-calculate $K(x, x') = \phi(x) \cdot \phi(x')$, can avoid having to explicitly calculate $\phi(x)$ in kernel perceptron, kernel SVM

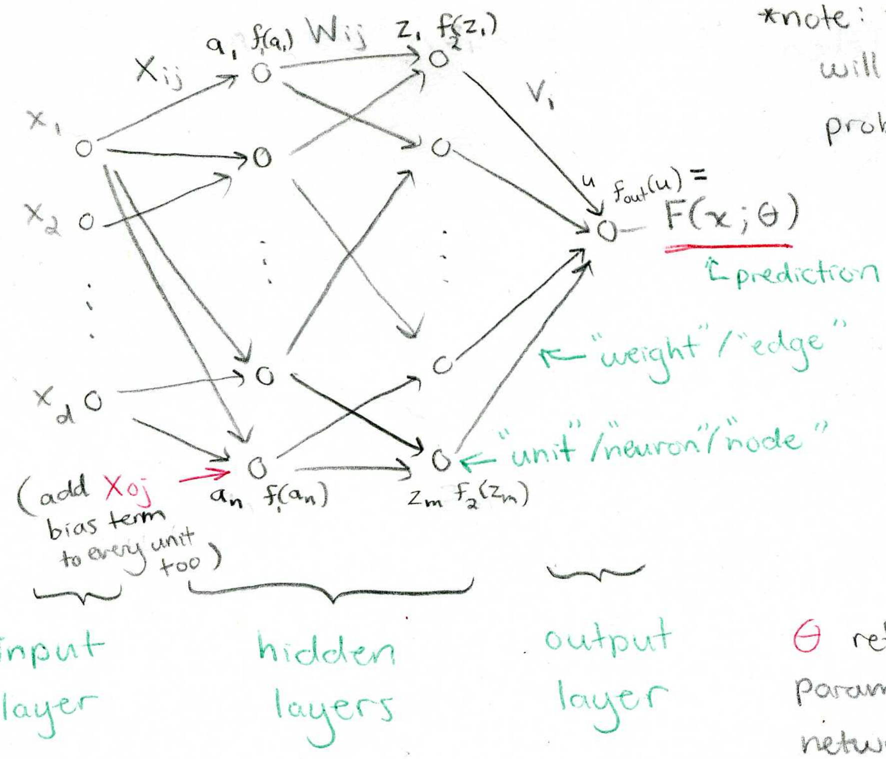
could even be ∞-dimensional

- but how do we choose ϕ?
  - ↳ manually / based on well-known kernels
  - ↳ Today: can implicitly learn it using neural nets!

Basics / Notation:

Today we're discussing feed-forward neural nets

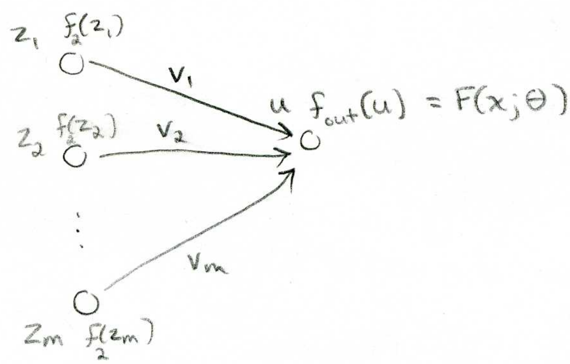
*note: exact notation / letters used will vary depending on the problem



This is a 3-layer model
↑
layers = # hidden layers + output

θ refers to all the parameters in the network ($W_{11}, \dots, W_{nm}, W_{01}, \dots, W_{0m}, V_1, \dots, V_m, V_0$)

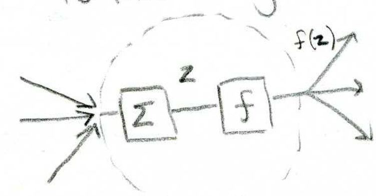
Let's zoom in on the output unit:



$$F(x;\theta) = f_{out}(u) = f_{out}\left(\sum_{j=1}^{m} f(z_j) V_j + V_0\right)$$

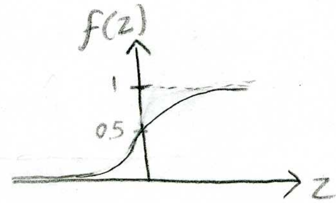
Every unit (in a hidden layer / the output layer) does the following:

1) weighted sum of units in previous layer
2) activation function applied to this weighted sum



Activation Functions

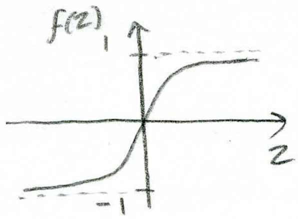
- Linear: $f(z) = z$
- Sigmoid: $f(z) = \frac{1}{1+e^{-z}}$



- nice since it pushes positive values to 1, negative values to 0.

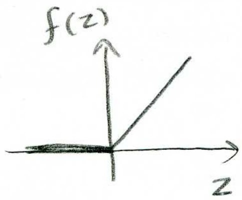
- tanh: $f(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

  ↳ preferred to sigmoid



- outputs centered @ 0
- pushes pos./neg. values to +1/-1.

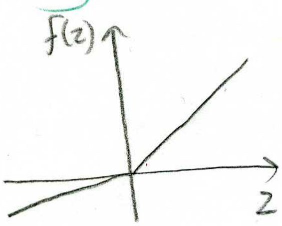
- ReLU: $f(z) = \max\{z, 0\}$



- in practice helps w/convergence
- widely used
- dead unit problem
- derivative is nice

- Leaky ReLU: $f(z) = \begin{cases} \max(0,z) & \text{if } z > 0 \\ \alpha \min(0,z) & \text{if } z < 0 \end{cases}$

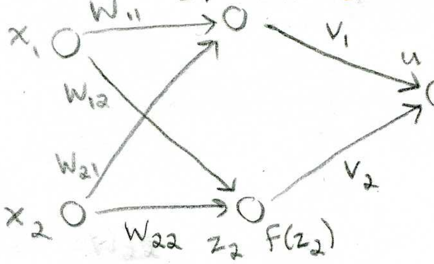
  ↑ small



- helps avoid dead unit problem
- derivative is nice

Forward & Back propagation

We'll be doing forward & back prop. on the following network:



$f_{out}(u) = F(x;\theta)$

★ Let $f_{out}(u) = u$

$\Rightarrow F(x;\theta) = u$

★ Let $f(z) = \text{ReLU}(z)$

$= \max\{z, 0\}$

Applying what we learned about units, we get a general expression:

$$f(z_j) = f\left(\sum_{i=1}^{d} x_i W_{ij} + W_{0j}\right)$$

↳ this idea is all you need for forward propagation

Forward Propagation

The idea behind forward prop. is that we just keep taking weighted sums & applying activation functions, feeding values forward through the network until we have a prediction $F(x;\theta)$.

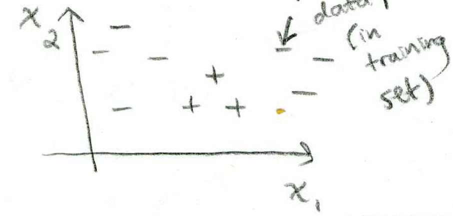
Let's write everything out to gain an intuition for how neural nets can be nonlinear predictors:

(in fact, a 2-layer feed-forward neural net w/ enough hidden units is a universal approximator!)

$$\left.\begin{aligned} z_1 &= W_{11}x_1 + W_{21}x_2 + W_{01} \\ z_2 &= W_{12}x_1 + W_{22}x_2 + W_{02} \end{aligned}\right\} \quad (1)$$

$$\left.\begin{aligned} f(z_1) &= \max\{0, z_1\} \\ f(z_2) &= \max\{0, z_2\} \\ F(x;\theta) &= f_{out}(u) = u \\ &= v_1 \max\{0, W_{11}x_1 + W_{21}x_2 + W_{01}\} \\ &\quad + v_2 \max\{0, W_{12}x_1 + W_{22}x_2 + W_{02}\} + V_0 \end{aligned}\right\} \quad (2)$$

Say we start w/ data that isn't linearly separable:

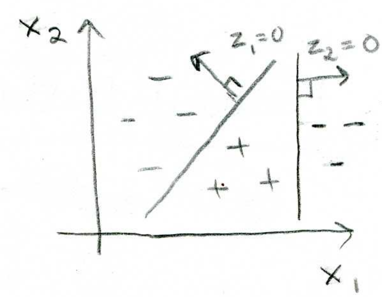


In (1), we can essentially learn 2 decision boundaries

$$z_1 = \begin{pmatrix} W_{11} \\ W_{21} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + W_{01} = 0$$

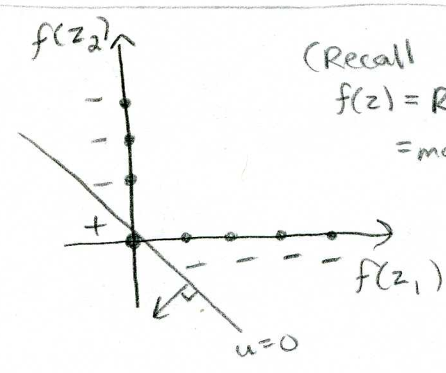
and

$$z_2 = \begin{pmatrix} W_{12} \\ W_{22} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + W_{02} = 0$$



In (2), we can think of the data points as transformed into the space $f(z_2)$ vs. $f(z_1)$, within which we can learn another linear dec. boundary:

$$u = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \cdot \begin{pmatrix} f(z_1) \\ f(z_2) \end{pmatrix} = 0$$



(Recall $f(z) = \mathrm{ReLU}(z) = \max\{0, z\}$)

Backpropagation

(still w/ the example we used in forward propagation)

While forward propagation tells us what the NN predicts, the goal of backpropagation is to learn/ update the weights in the network.

How do we evaluate our prediction $F(x^{(t)}; \theta)$?

↳ average loss:

for our network in our example, $F(x^{(t)}; \theta) = u^{(t)}$

$$J(\theta) = \frac{1}{n} \sum_{t=1}^{n} \text{Loss}(y^{(t)} \underbrace{F(x^{(t)}; \theta)})$$

(we use Hinge loss in our example)

$$\text{Loss}_h(z) = \max\{0, 1-z\}$$

$$\text{Loss}_h(y^{(t)}u^{(t)}) = \max\{0, 1-y^{(t)}u^{(t)}\}$$

$$= \begin{cases} 1-y^{(t)} \cdot u^{(t)} & \text{if } 1-y^{(t)} \cdot u^{(t)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

How do we optimize / minimize our objective?

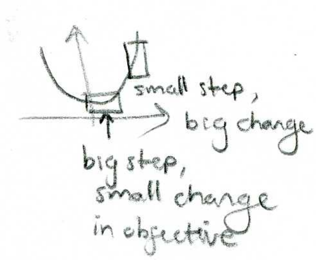
↳ Stochastic gradient descent:

1) initialize θ to small random vals
↳ all weights in the network

2) select $i \in \{1, \ldots, n\}$ @ random

3) $\theta \leftarrow \theta - \underbrace{\eta_k} \nabla_\theta \text{Loss}(y^{(t)} F(x^{(t)}; \theta))$

- we've been decreasing this w/ # of updates k
- can also use more sophisticated choices such as Adagrad (based on idea that smaller steps w/ steeper gradients can result in bigger change in objective fn. value than bigger steps w/ more gentle gradients).



- So, we need to find the gradient of the Loss with respect to each of our weights in order to know how to update them.

updating V_i's:

$$\frac{d}{dV_i} \text{Loss}(y^{(t)} u^{(t)}) = \frac{d}{dV_i} \text{Loss}(y^{(t)} u^{(t)})$$

(CHAIN RULE)

$$= \left[\frac{d\text{Loss}(y^{(t)} u^{(t)})}{du^{(t)}}\right]\left[\frac{du^{(t)}}{dV_i}\right]$$

↳ make sure you take into account what depends on what else

$$= \left[\frac{d\text{Loss}(y^{(t)} u^{(t)})}{du^{(t)}}\right]\left[\frac{d(f(z_1^{(t)}) V_1 + f(z_2^{(t)}) V_2 + V_0)}{dV_i}\right]$$

↳ helps us since we know $\text{Loss}(y^{(t)} u^{(t)})$ in terms of $u^{(t)}$, & $u^{(t)}$ in terms of V_i

$$= \begin{bmatrix} -y \text{ if } \text{Loss}(y^{(t)} \cdot u^{(t)}) > 0 \\ 0 \text{ otherwise} \end{bmatrix}\left[f(z_i^{(t)}) \right]$$

$$\Rightarrow V_i \leftarrow V_i - \eta_k \frac{d\text{Loss}(y^{(t)} u^{(t)})}{dV_i}$$

$$= V_i + \eta_k f(z_i^{(t)}) y^{(t)}$$

updating W_{ij}'s:

$$\frac{d}{W_{ij}} \text{Loss}(y^{(t)} \cdot u^{(t)}) = \left[\frac{d\text{Loss}(y^{(t)} u^{(t)})}{du^{(t)}}\right]\left[\frac{du^{(t)}}{df(z_j^{(t)})}\right]\left[\frac{df(z_j^{(t)})}{dz_j^{(t)}}\right]\left[\frac{dz_j^{(t)}}{dW_{ij}}\right]$$

$$= \begin{bmatrix} -y^{(t)} \text{ if } \text{Loss}(y^{(t)} u^{(t)}) > 0 \\ 0 \text{ otherwise} \end{bmatrix}\left[\frac{d(V_1 f(z_1^{(t)}) + V_2 f(z_2^{(t)}))}{df(z_j^{(t)})}\right]\left[\frac{d(\max\{0, z_j^{(t)}\})}{dz_j^{(t)}}\right] \cdot \left[\frac{d(W_{1j} x_1^{(t)} + W_{2j} x_2^{(t)} + W_{0j})}{dW_{ij}}\right]$$

indicator function; 1 if true, 0 if false

$$= \begin{bmatrix} -y^{(t)} \text{ if } \text{Loss}(y^{(t)} u^{(t)}) > 0 \\ 0 \text{ otherwise} \end{bmatrix}[V_j]\big[[z_j^{(t)} > 0]\big]\left[x_i^{(t)}\right]$$

$$\Rightarrow W_{ij} \leftarrow W_{ij} + \eta_k x_i^{(t)} [[z_j^{(t)} > 0]] V_j y^{(t)}$$

· can see why initializing to 0 is bad by looking @ these updates