Feed-forward Neural Networks

$$\underline{f} = g^{(3)}\big(\underbrace{W^{(3)}\underline{h}^{(2)} + \underline{b}^{(3)}}_{\underline{a}^{(3)}}\big)$$

$$\uparrow$$

$$\underline{h}^{(2)} = g^{(2)}\big(W^{(2)}\underline{h}^{(1)} + \underline{b}^{(2)}\big)$$

$$\uparrow$$

$$\underline{h}^{(1)} = g^{(1)}\big(W^{(1)}\underline{x} + \underline{b}^{(1)}\big)$$

$$\uparrow$$

$\underline{x}$, inputs

$$h_k^{(2)} = g^{(2)}\Big(\sum_l W_{kl}^{(2)} h_l^{(1)} + b_k^{(2)}\Big)$$

Homework: Try plotting for random weights.

Change: scale W, # layers, g, without $\underline{b}$

Fitting params $\underline{\theta}$

$\underline{\theta}$ — Vector containing $\{W^{(\ell)}, b^{(\ell)}\}_{\ell=1}^{L}$ and any other params.

L — # layer

Stochastic Gradient Descent on cost c

$$\underline{\theta} \leftarrow \underline{\theta} - \eta \nabla_{\theta} c(\underline{\theta})$$

η — learning rate

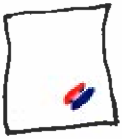
Tomorrow: how to compute $\nabla_{\theta} c(\theta$

Cost c describes task

- Match $\underline{f}$ to one-hot class label
- Match f to strength of concrete
- Match $\underline{f}$ to pixels of next image

...

Side-effect: learn "hidden" representations $\underline{h}^{(\ell)}$, and θ

$W^{(1)}_{k,:}$ as image:  this "filter" looks for a type of edge.

$\underline{h}^{(1)}$ indicates which edges present.

Embeddings:

If $\underline{x}$ one-hot, $W^{(1)}_{:,d}$ is vector for choice d

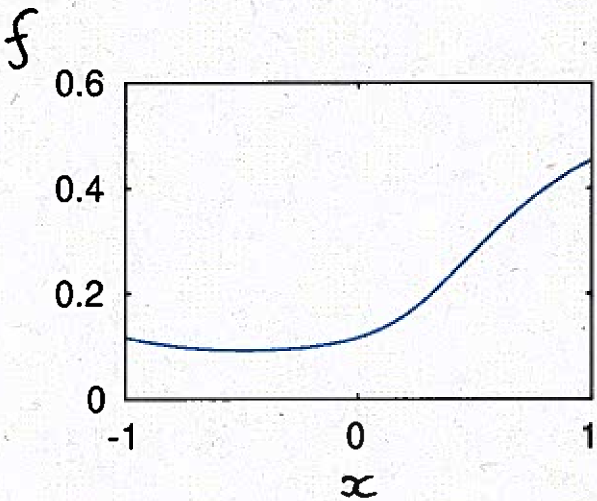
[Often set $\underline{h}^{(1)}$ directly to $W^{(1)}_{:,d}$

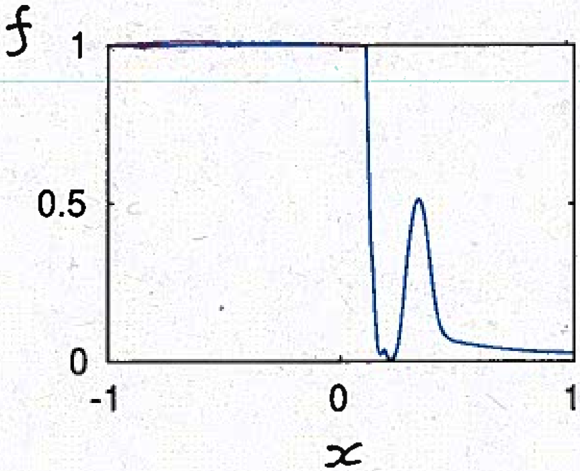
no point doing $W\underline{x}^{(1)}$ and applying function]

```
gg = @(a) 1./(1 + exp(-a));
X = (-1:0.01:1)'; % Nx1
H1 = gg(X * randn(1, 100)); % Nx100
H2 = gg(H1 * randn(100, 50)); % Nx50
F = gg(H2 * randn(50, 1)); % Nx1
plot(X, F);
```

```
gg = @(a) 1./(1 + exp(-a));
X = (-1:0.01:1)'; % Nx1
H1 = gg(X * randn(1, 100) * 10); % Nx100
H2 = gg(H1 * randn(100, 50) * 10); % Nx50
F = gg(H2 * randn(50, 1) * 10); % Nx1
plot(X, F);
```

2019 L20 3





2019 L20 4

Specialized Architectures

ConvNets: learned filters move over whole image

don't relearn in every location

$\Rightarrow$ share parameters

...

Simple pooling (example)

"Bag of words" text classification

$\underline{e}^{(t)} = W \underline{x}^{(t)}$ = "embedding vector for t^{th} word"

$\quad$ ↳ onehot

$\underline{h} = \left(\frac{1}{T}\right) \sum_{t=1}^{T} \underline{e}^{(t)}$ "document vector"

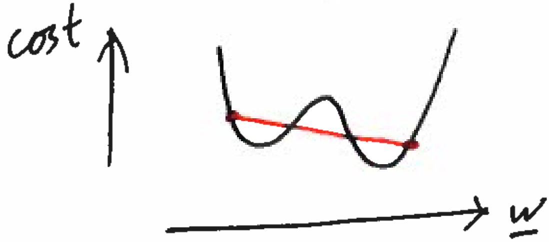
$\sum$?

$\underline{f} = \text{Softmax}(V \underline{h})$ embedding dim × 1

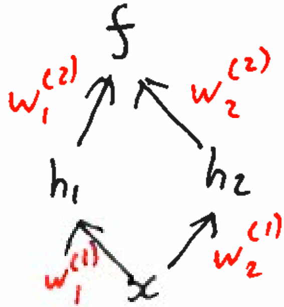
prob. over classes

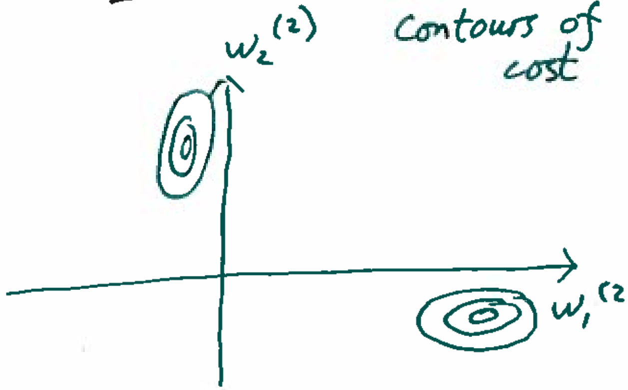
\# classes × embedding dim

NN don't have a convex cost



No unique optimum
$\Rightarrow$ not convex





Set $\underline{w}^{(1)}$ to best values for $\underline{w}^{(2)}$

- These local optima, don't matter
  → because the functions (prediction) are same.

- Not all optima are equivalent.

Initialization

Must not set all $W^{(\ell)}$ to zero

→ all hidden extract same feature

→ SGD applies same update to all params

→ all hidden stay same ...

Set randomly $W_{ij}^{(\ell)} \sim N(0,1)$?

[Cf WOf note]

- activation $W^{(1)}\underline{x}$ sum of D random values for each hidden

- typically each term in sum $\sim \pm 1$ (if data standardized)

- $\underline{w}^T\underline{x}$ is $\sim \pm\sqrt{D}$

- instead $w \sim N\left(0, \left(\frac{1}{\sqrt{D}}\right)^2\right)$

MLP course notes have more ideas.