Terminology

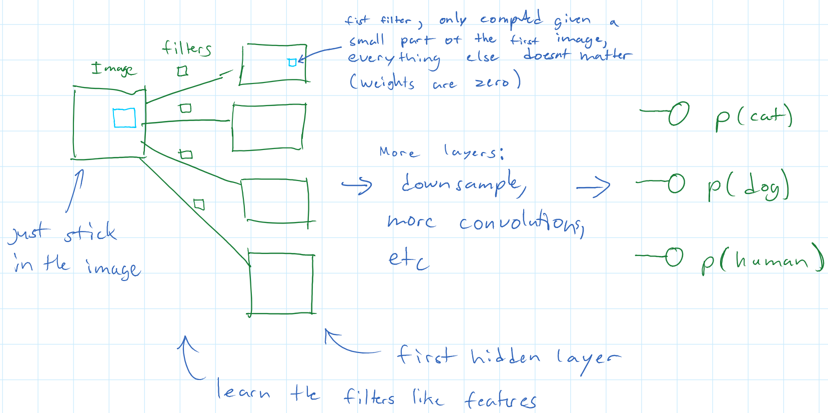
Regression	modelling continuous function
Classification	modelling a discrete categorical function
Supervised Learning	learning a function from (x_i, y_i) pairs
Unsupervised Learning	learning a structure from a data set $\{X\}$
Overfitting	model learns peculiarities of training set
Underfitting	model insufficiently complex
Sigmoid	function that looks like an 'S'
Logistic Function	$f(x) = 1/(1+e^{-x})$
Early Stopping	stop training when validation error starts to increase
Momentum	$V_{t+1} = \mu V_t - \varepsilon \nabla L(\theta_t)$
Weight Decay	L2 regularization on weights

Announcements

- No class Monday (Thanksgiving)
- Assignment 4 & Marked Assignment 2 due Wednesday (October 15)

Convolutional Neural Networks (CNNs)

- used for image/vision tasks



- exactly a NN but with a lot of weights set to 0, i.e. only looking at locality with filters

Refresher On Convolutions

Image (1000 x 1000)

$$\begin{matrix} 1 & 2 & 1 & 0 & 3 & \cdots \\ 4 & 5 & -2 & 0 & 2 & \\ 1 & 3 & 4 & 5 & 3 & \\ \vdots & & & & & \ddots \end{matrix}$$

Filter (2 x 2)

$$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Step 1: 1+4 = 5

Step 2: 2+5 = 7

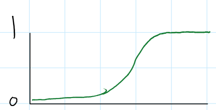
Step 3: 1−2 = −1

"responds" to situations similar to the filter (dot product is large)

Activation Functions

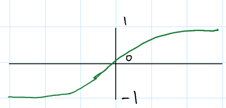
Logistic Function:

$$\theta(x) = \frac{1}{1+e^{-x}}$$



Hyperbolic Tangent:

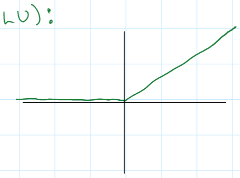
$$\theta(x) = \tanh(x)$$



These are like Sigmoid functions

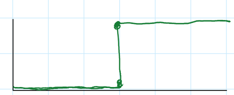
Rectified Linear Units (RELU):

$$\theta(x) = \max\{0, x\}$$



- this one is better

These are related to / emulate: Threshold function:



aka heavyside step function (non differentiable)

• initialization is important

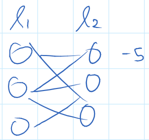
say Rand (−100, 100)

: we would rather near "scale" of sigmoid (−1, 1)

("saturated" if you're far away)

vanishing gradient : gradients less than 1, going back layers, learning will be slow because steps will be small

relu : not "scaled" with size



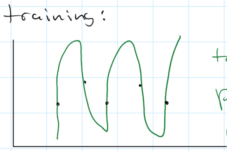
derivative can flow through another path

Overfitting and Regularization

"getting good at peculiarities" in the training set

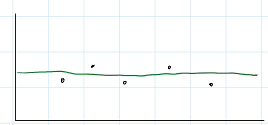
something like $-250x^3 - 922x^2 + 2594x + 1000$
↓ cancels perfectly at points

training:



too high polynomial (minimize error)

actual:



to try and fix this:

Regularization:

$$L = \sum_i \|y_i - \hat{y}_i\|_2^2 + \lambda \sum_{\text{all weights}} |w_j|^2$$

(predicted → y_i, actual → $\hat{y}_i$)

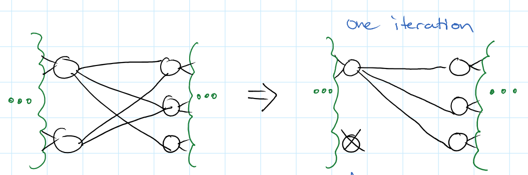
⇒ "Weight Decay"

- make weights small
- helps prevent overfitting

- could also use L1

Dropout (a regularization method) (2012)

- During training, at every iteration, "Drop out" (set to zero) each unit with probability P.
- During prediction, multiply weights by (1 − P)



↑ maybe this value was over fitting with a huge value

- kind of like taking the geometric mean of a bunch of different neural networks

Optimization

- Use Stochastic Gradient Descent

  $W_{t+1} = W_t - \alpha_t \nabla L(W_t)$ ← gradient of loss

  ↑ learning rate

  kinda like velocity

- Momentum

  $W_{t+1} = W_t + V_t + 1$

  $W_{t+1} = W_t - \alpha_t L(W_t) + \varepsilon V_t$

  ↖ retain some momentum from the previous step

"Second order method"

Example with a ball