CSDS 440: Machine Learning

Soumya Ray (he/him, sray@case.edu)

Olin 516

Office hours T, Th 11:15-11:45 or by appointment

Announcements

- Test 2 Thursday 10/17
  - Topics include everything up to and including 10/10 lecture
  - Bring a calculator (phone app is ok)

How to scale an ANN?

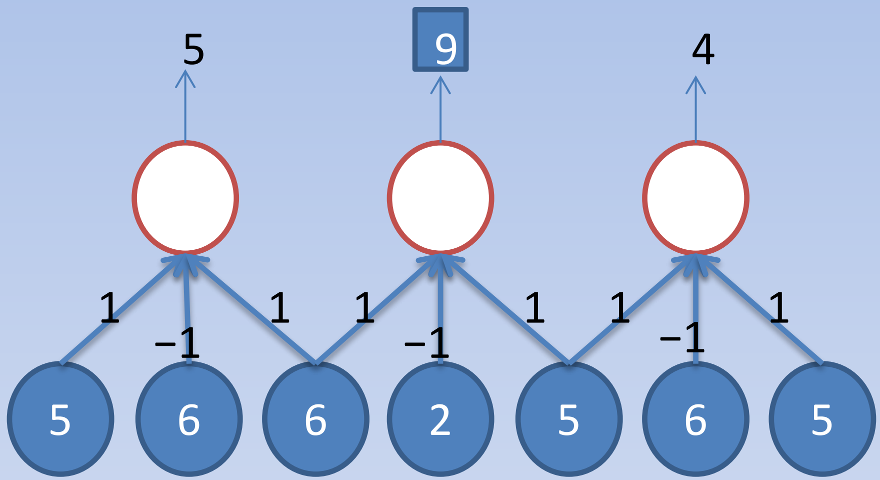
Suppose we create an ANN with LOTS of layers.

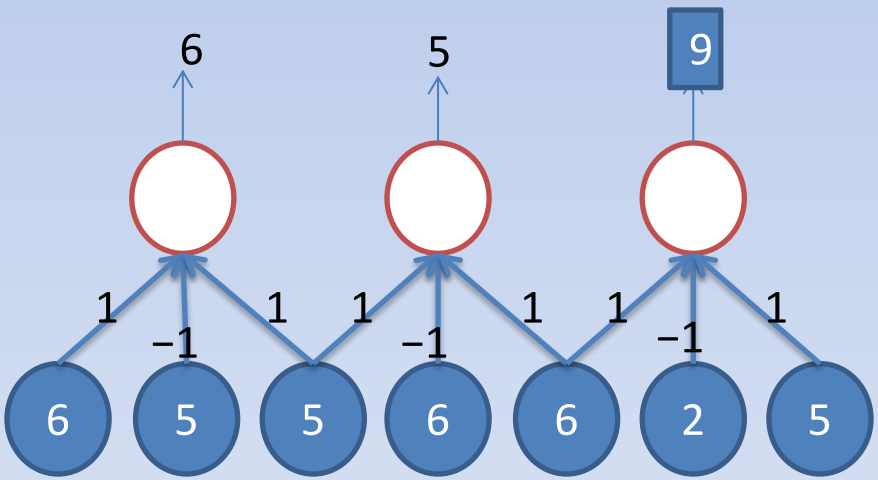
1. Why might we want to do that?
2. What will these layers *do*?
3. How can learning scale?
4. How to deal with vanishing gradients?
5. How to deal with overfitting?

Pooling Layers

- A *pooling* layer aggregates information from an adjacent layer
- Average pooling: $k=(1/l,\ 1/l, \ldots 1/l)$
- Max pooling: computes the maximum value of l inputs
  - For each feature detector, identifies whether that feature was found somewhere in the previous layer
- Downsamples input by factor of l

Max Pooling





Vanishing Gradients 1

- A key problem in ANNs is *vanishing gradients*
- To prevent vanishing gradients, we can use the "Rectified Linear Unit" (ReLU) activation function:

$$h(x) = \max(0, x)$$

Vanishing Gradients 2

- Each layer in an ANN *learns a completely new representation* from the previous layer
  - Can cause catastrophic failure due to one “bad” layer

- Instead, each layer can *add on to* the learned representation of the previous layer
  - Allows building much deeper structures robustly

Residual Networks

- Perturbing the representation is done through adding a "residual" function to each layer
- Replace

$$\mathbf{z}^{l+1} = h(\mathbf{W}^l \mathbf{z}^l)$$

- With

$$\mathbf{z}^{l+1} = h(\mathbf{z}^l + f(\mathbf{z}^l))$$

Residual function (learned from data; could be identity)

How do we prevent overfitting?

- ANNs are very prone to overfitting
  - Structure can be very complex, lots of parameters
  - Decision surface can be very nonlinear

Controlling Overfitting

- One strategy: add a “weight decay” term

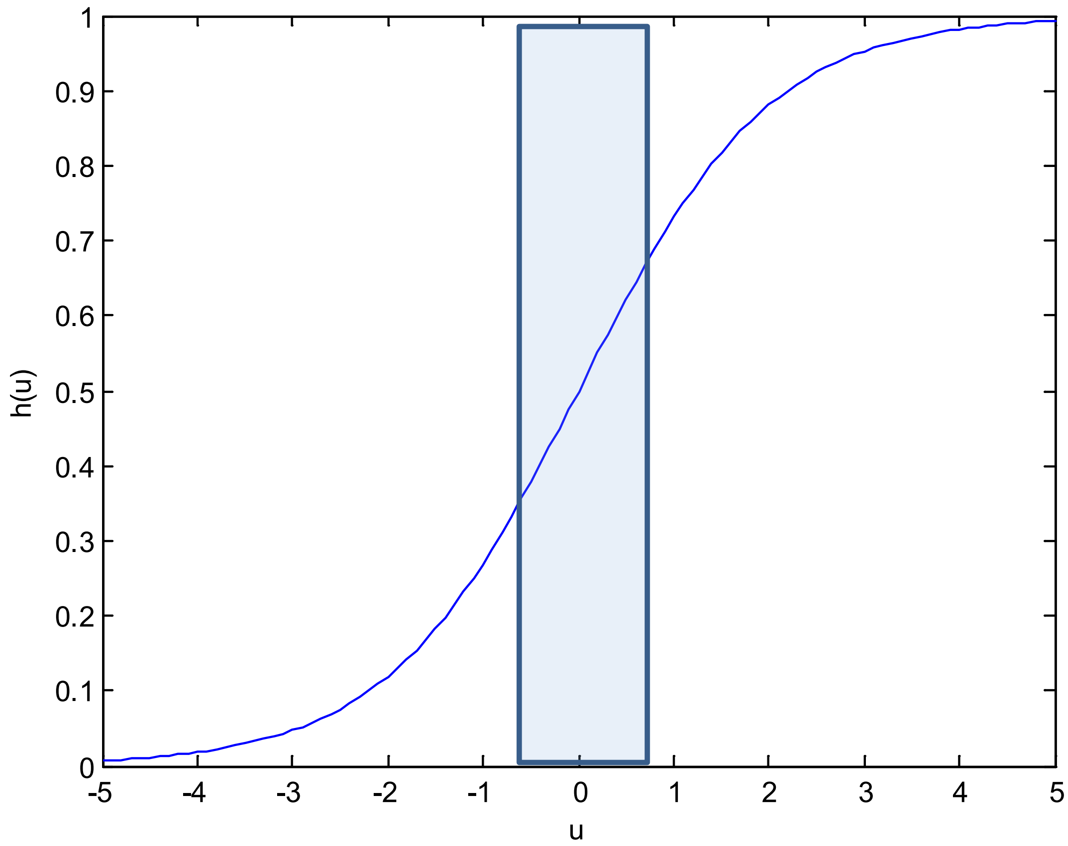
$$L_{OC}(\mathbf{w}) = L(\mathbf{w}) + \boxed{\gamma \sum_i \sum_j w_{ji}^2}$$

Weight Decay Term
“Complexity Penalty”

Tradeoff Hyperparameter

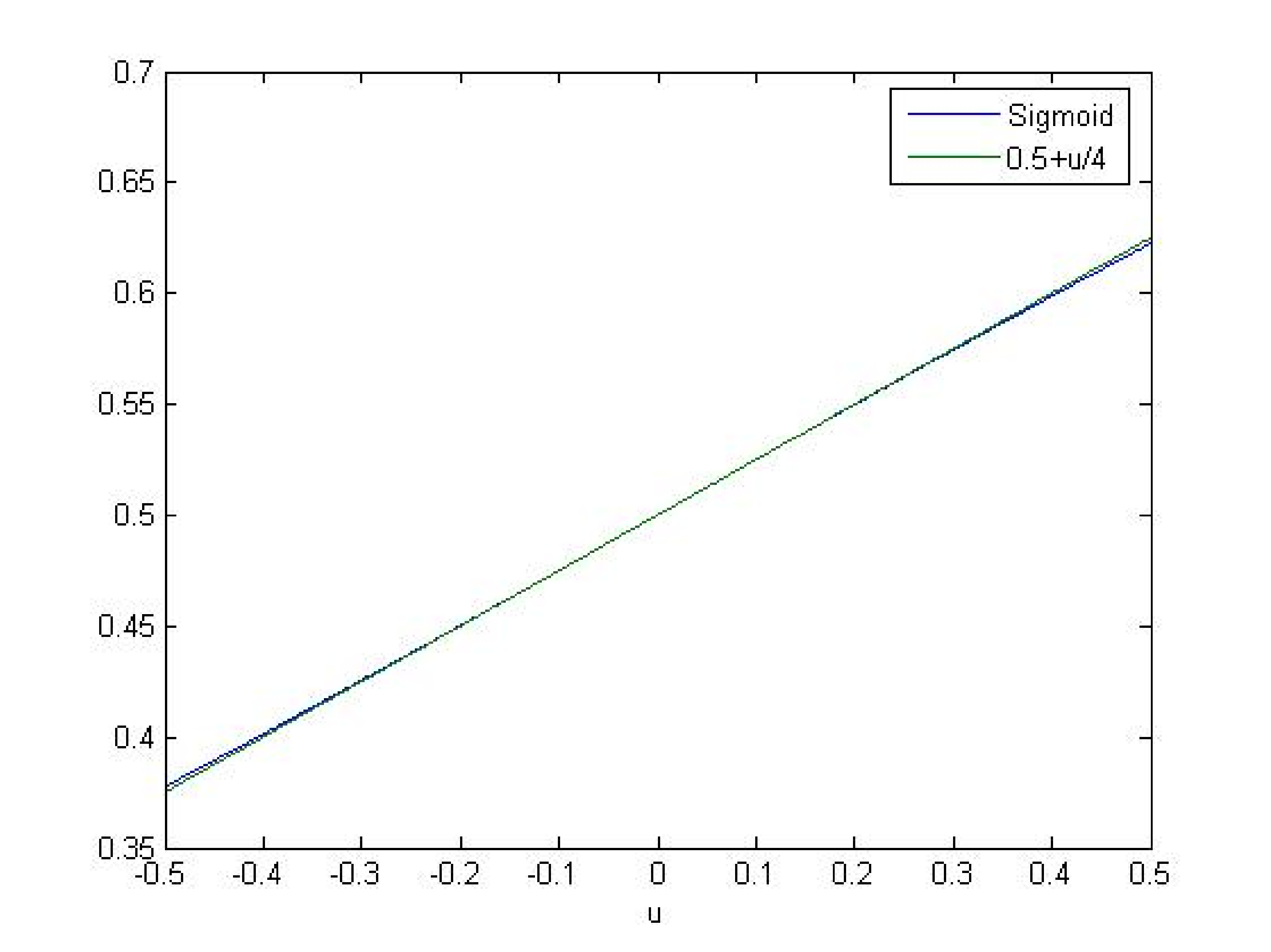
- This will prevent weights from growing too large

Controlling Overfitting



If the weights are not too large (and assuming the input is suitably scaled, see later), the sigmoid operates in the "nearly-linear" region. This makes the decision surface of the ANN less nonlinear and reduces the complexity of the concepts it can learn.

Controlling Overfitting



Dropout Regularization

- Each backprop step, randomly sample a set of hidden units *to leave out of the update*
- Why?
  - Forces different feature detectors to do useful work in the final classifier
    - Classifier produced is more robust
  - Approximates training an *ensemble* of networks (later)

Implementation: Input Standardization

- Since ANNs use linear functions, if inputs are badly scaled, can lead to problems at runtime
  – Average human weight=6e+10 μg, height=1.7e-18 light years
- To avoid this, often standardize the input to zero mean, unit variance

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$$

Batch Normalization

- This kind of standardization can also be done at the node level
- Suppose for a node z, the values of z for each example i are z_i
- Replace z_i with:

$$\hat{z}_i = \beta + \gamma \frac{z_i - \mu}{\sqrt{\varepsilon + \sigma^2}}$$

- Empirically improves performance

Implementation: Nominal Features

- If data is described by nominal features, we will need to re-encode it
- 1 of N
  - N input units for each nominal attribute with N values, only 1 is active for each example
- Logarithmic
  - $\log(N)$ input units for each nominal attribute with N values
  - Each input is represented as a binary code