# CSDS 440: Machine Learning

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

Olin 516

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

# Recap

- Convolution is a l___ operation. The k_____ can be l____ from d____.
- A tensor is a m____-d_____ m_____. It is used to preserve l_____ across layers.
- A pooling layer a_____ i_____ from adjacent layers.
- Deep NNs suffer from the v_____ g_____ problem.
- The ReLU activation is defined as h(x)=F(A,B).
- In a r___ network, the learned representation at each layer is a pe_____ of the previous.
- One way to control overfitting in ANNs is to use w____ d____. This adds a c____ p___ to the loss function.

# Today

- Artificial Neural Networks (Ch 4, Mitchell)
- Probabilistic Machine Learning

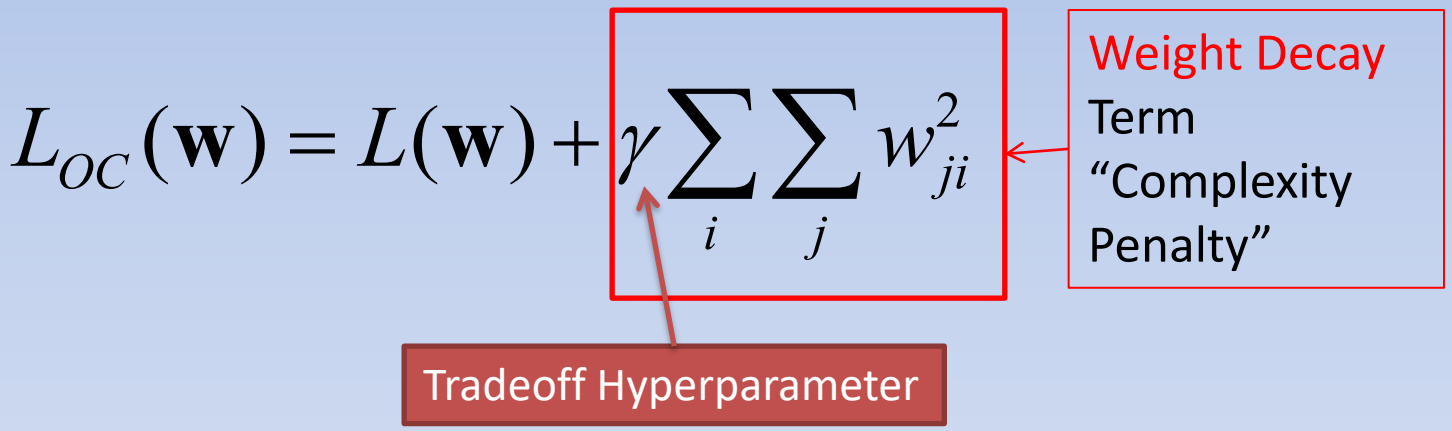
# How to scale an ANN?

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?

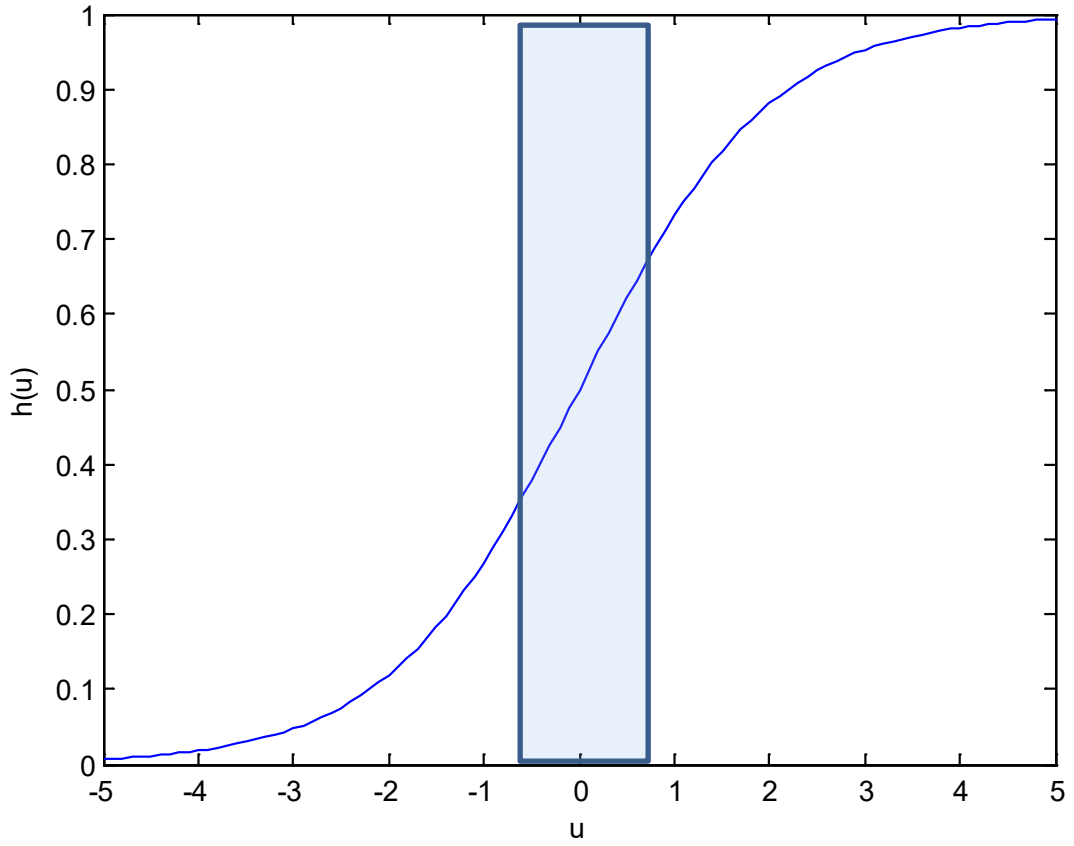
# Controlling Overfitting

- One strategy: add a "weight decay" term



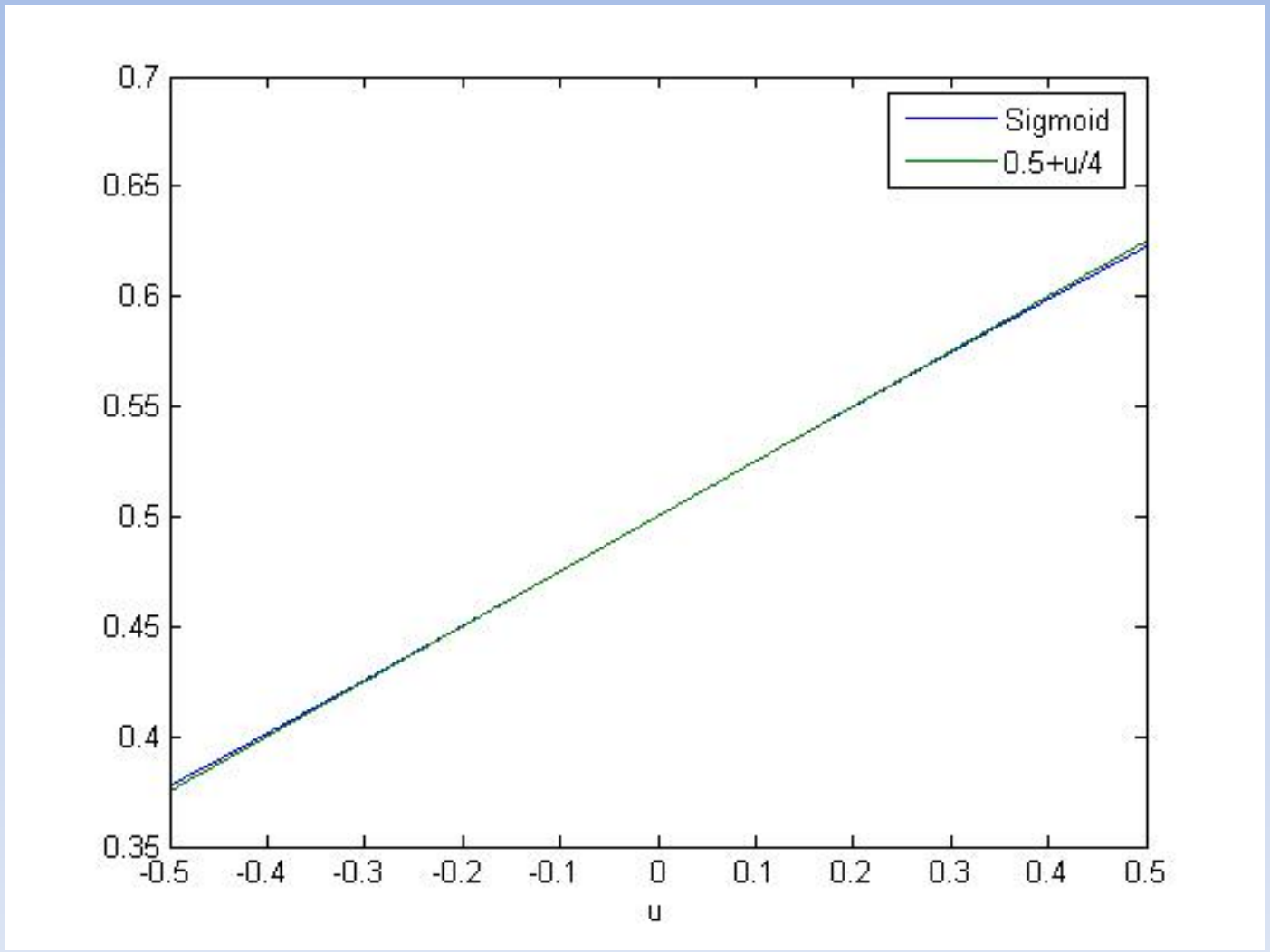

- This will prevent weights from growing too large

# 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

# Implementation: Initialization

- When initializing, generally set weights to small random values
- But some random choices work better than others
- One choice is “Xavier initialization”: choose weights from a normal distribution with mean 0 and variance $\frac{2}{n_i + n_0}$

# Other architectures

- People have also investigated networks with loops, called “recurrent neural networks”
  - This gives ANNs a “memory” (can “remember” previous inputs)
  - Different architectures proposed (Jordan networks, Elman networks, Hopfield networks, LSTMs etc)

# Probabilistic Learning (Ch 6, Mitchell)

- So far, focused on classifiers: functions mapping examples to classes
- Now, look at algorithms that explicitly estimate *probabilities* of class membership
  - $p(\mathbf{x},y)$
  - $p(y|\mathbf{x})$
  - $p(\mathbf{x}|y)$ ("Class conditional distribution")

# Why?

- Bayesian Decision Theory: optimal thing to do is to choose hypothesis to minimize *expected risk*

Risk functional

Expected Risk

$$\hat{h} = \arg\min_{h} \int_{D} R(h \mid \mathbf{x}, y)\, p(\mathbf{x})\, d\mathbf{x}$$

$$R(h \mid \mathbf{x}, y) = \sum_{\hat{y}_j} L(h(\mathbf{x}) = \hat{y}_j)\, p(h(\mathbf{x}) = \hat{y}_j)$$

In order to minimize risk, we need good probability estimates.

# Why?

- Naturally produce “confidence estimates”
- Naturally incorporate “prior knowledge”
- Can also give us tools to analyze some of our algorithms
- Can generate data
  - Basis for modern generative ML algorithms

# Probabilistic Classification

- Learning Task: Given data, learn probabilistic model to predict probability of class membership (*Estimation*)
  - Parameters?
  - "Structure"? (CSDS 491)

- Prediction Task: Find *most probable* class of a new example
  - *Probabilistic Inference* (classification)

# Two Approaches to Probabilistic Classification

- Generative approaches model the joint distribution $p(\mathbf{x}, y)$

- Discriminative approaches model the conditional distribution $p(y|\mathbf{x})$

# Generative Approaches

- Note: $p(\mathbf{x}, y) = p(\mathbf{x} \mid y)p(y)$
- "Sample a class, then sample an instance from it"---can "generate" the observed data
- Nicely handles missing values, unlabeled examples etc
  - If modeling assumptions met, work very well
- But do "more work" than necessary

# Discriminative Approaches

- In classification, only interested in $p(y|\mathbf{x})$ anyway
- Discriminative approaches directly model this
- Robust to modeling errors
- No way to recover/ "generate" the data
  - Does not easily handle missing data
- Usually tend to be more accurate than generative counterparts, but some evidence that convergence is slower