This Lecture

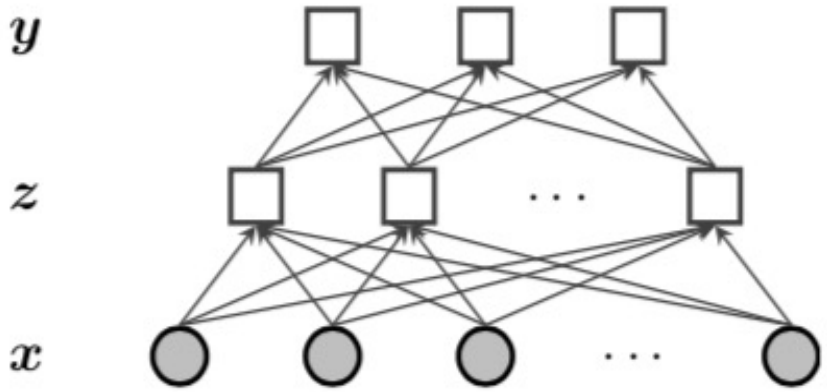
- Feedforward neural network
- Learning neural networks
- Text classification applications
- Evaluating text classifier

A Simple Feedforward Architecture

Suppose we want to label stories as $y \in \{Bad, Good, Okay\}$

- What makes a good story?
- Let's call this vector of features $\boldsymbol{z}$
- If $\boldsymbol{z}$ is well-chosen, it will be easy to predict from $\boldsymbol{x}$, and it will make it easy to predict y (the label)

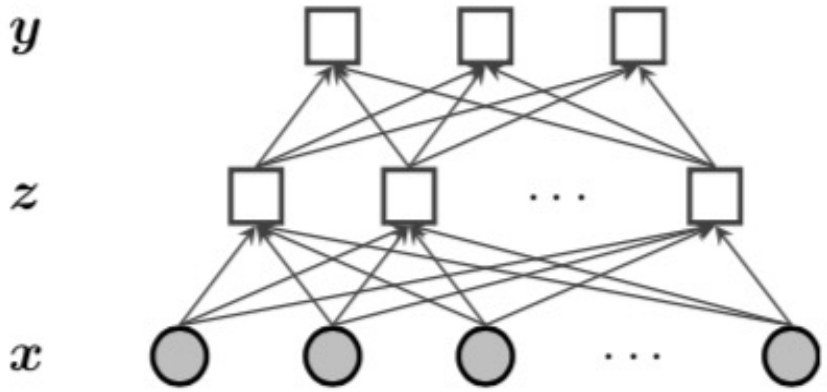
A Simple Feedforward Architecture



Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$
$$= (1 + \exp(-\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}))^{-1}$$

A Simple Feedforward Architecture

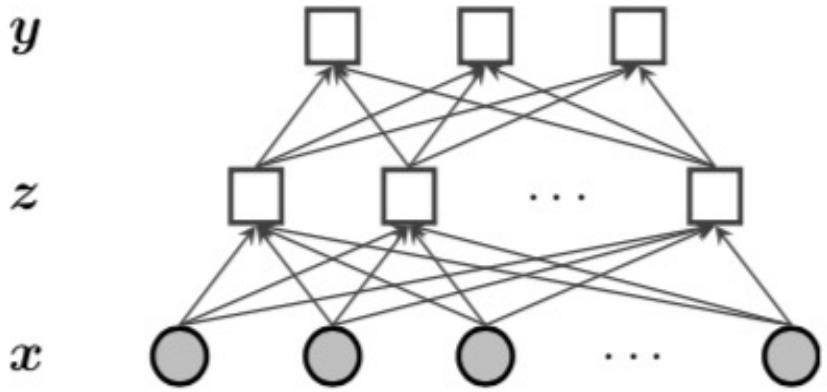


Let's predict each z_k from $\boldsymbol{x}$ by binary logistic regression:

$$\Pr(z_k = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})$$
$$= (1 + \exp(-\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}))^{-1}$$

$$\boldsymbol{\Theta}^{(x \to z)} = [\boldsymbol{\theta}_1^{(x \to z)}, \boldsymbol{\theta}_2^{(x \to z)}, \ldots, \boldsymbol{\theta}_{K_z}^{(x \to z)}]^\top$$

A Simple Feedforward Architecture



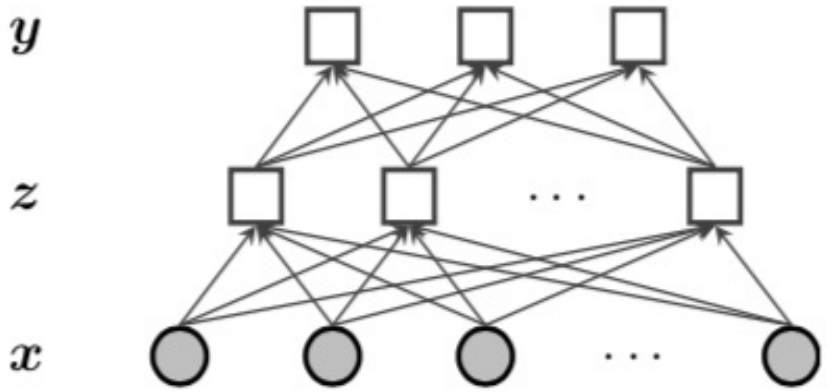
Next predict y from $\boldsymbol{z}$, again via logistic regression:

$$\Pr(y = j \mid z) = \frac{\exp(\boldsymbol{\theta}_j^{(z \to y)} \cdot \boldsymbol{z} + b_j)}{\sum_{j' \in \mathcal{Y}} \exp(\boldsymbol{\theta}_{j'}^{(z \to y)} \cdot \boldsymbol{z} + b_{j'})}$$

where each b_j is an offset. This is denoted:

$$p(\boldsymbol{y} \mid \boldsymbol{z}) = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

Feedforward Neural Network



To summarize:

$$\boldsymbol{z} = \sigma(\boldsymbol{\Theta}^{(x \to z)} \boldsymbol{x})$$

$$\mathrm{p}(\boldsymbol{y} \mid \boldsymbol{z}) = \mathrm{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

- In reality, we never observe $\boldsymbol{z}$, it is a hidden layer. We compute $\boldsymbol{z}$ directly from $\boldsymbol{x}$.
- This makes $\mathrm{p}(\mathrm{y}|\boldsymbol{x})$ a nonlinear function of $\boldsymbol{x}$

Designing Feedforward Neural Network

1. Activation Functions

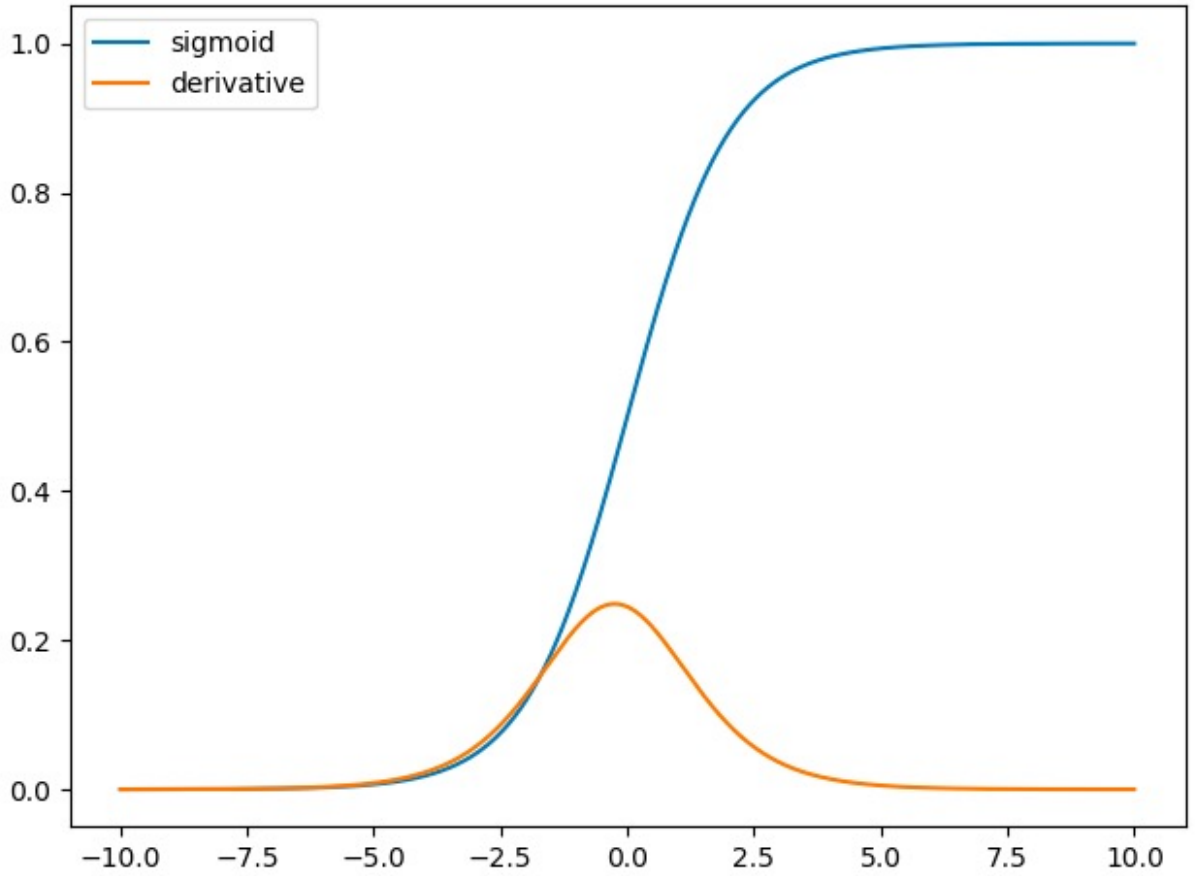
Sigmoid Function

The **sigmoid** in $z = \sigma(\Theta^{(x \to z)} x)$

is an **activation function**

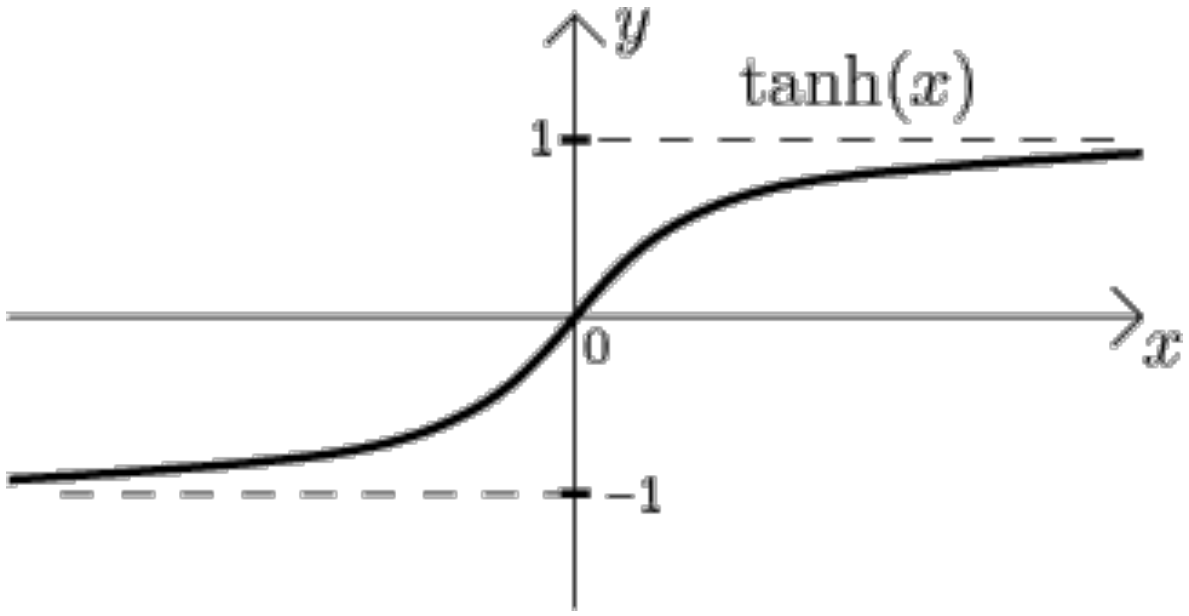
In general, we write $z = f(\Theta^{(x \to z)} x)$ to indicate an arbitrary activation function.

$$\frac{\partial}{\partial a}\sigma(a) = \sigma(a)(1 - \sigma(a))$$



Tanh Function

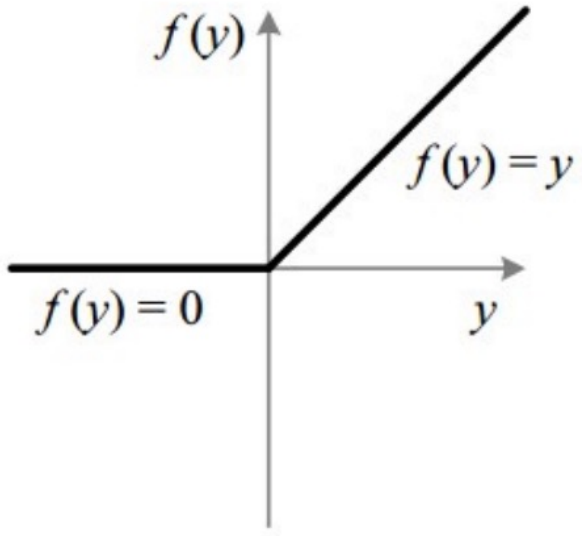
- Hyperbolic Tangent
- Range: (-1, 1)
- $\frac{\partial}{\partial a}\tanh(a) = 1 - \tanh(a)^2$

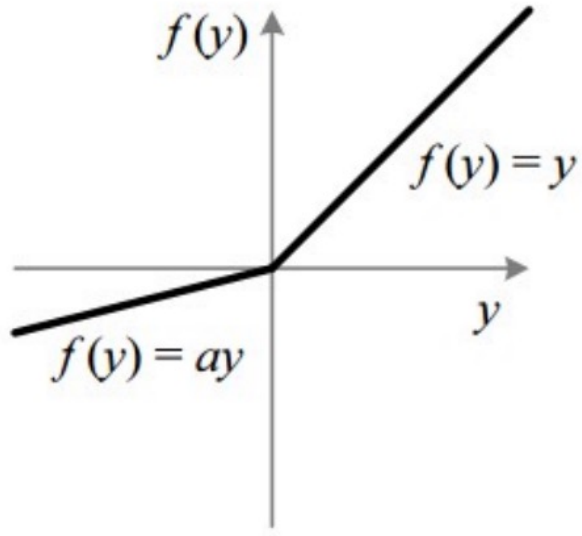


ReLU Function

- Rectified Linear Unit

$$\text{ReLU}(a) = \begin{cases} a, & a \geq 0 \\ 0, & \text{otherwise.} \end{cases}$$





- Leaky ReLU

Designing Feedforward Neural Network

2. Outputs and Loss Function

Outputs and Loss Functions

- The **softmax** output activation is used in combination with the negative log-likelihood loss, like logistic regression.
- In deep learning, this loss is called **the cross-entropy**:

$$\tilde{\boldsymbol{y}} = \text{SoftMax}(\boldsymbol{\Theta}^{(z \to y)} \boldsymbol{z} + \boldsymbol{b})$$

$$-\mathcal{L} = -\sum_{i=1}^{N} \boldsymbol{e}_{y(i)} \cdot \log \tilde{\boldsymbol{y}},$$

where $\boldsymbol{e}_{y(i)} = [0, 0, \ldots, 0, \underbrace{1}_{y^{(i)}}, 0, \ldots, 0]$, **a one-hot vector**

Designing Feedforward Neural Network

3. Learning Neural Networks

Gradient Descent in Neural Networks

Neural networks are often learned by gradient descent, typically with minibatches

$$\boldsymbol{\theta}_k^{(z\to y)} \leftarrow \boldsymbol{\theta}_k^{(z\to y)} - \eta^{(t)} \nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)}$$

$\eta^{(t)}$ is the learning rate at update t

$\ell^{(i)}$ is the loss on instance (minibatch) i

$\nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)}$ is the gradient of the loss wrt the column vector of output weigh $\boldsymbol{\theta}_k^{(z\to y)}$

$$\nabla_{\boldsymbol{\theta}_k^{(z\to y)}} \ell^{(i)} = \left[\frac{\partial \ell^{(i)}}{\partial \theta_{k,1}^{(z\to y)}}, \frac{\partial \ell^{(i)}}{\partial \theta_{k,2}^{(z\to y)}}, \ldots, \frac{\partial \ell^{(i)}}{\partial \theta_{k,K_y}^{(z\to y)}} \right]^\top$$

Backpropagation

If we don't observe z,

how can we learn the weight $\Theta^{(x \to z)}$?

Backpropagation

Compute loss on y & apply chain rule of calculus to compute gradient on all parameters

$$\frac{\partial \ell^{(i)}}{\partial \theta_{n,k}^{(x \to z)}} = \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial z_k}{\partial \theta_{n,k}^{(x \to z)}}$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \frac{\partial f(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x})}{\partial \theta_{n,k}^{(x \to z)}}$$

$$= \frac{\partial \ell^{(i)}}{\partial z_k} \times f'(\boldsymbol{\theta}_k^{(x \to z)} \cdot \boldsymbol{x}) \times x_n$$

Backpropagation as an algorithm

Forward propagation:

- Visit nodes in topological sort order
- Compute value of node given predecessors

Backward propagation:

- Visit nodes in reverse order
- Compute gradient wrt each node using gradient wrt successors

Backpropagation

Re-use derivatives computed for higher layers in computing derivatives for lower layers so as to minimize computation

Good news is that modern automatic differentiation tools did all for you!

Implementing backprop by hand is like programming in assembly language.

“Tricks” for Better Performance

- Preventing overfitting with regularization and dropout
- Smart initialization
- Online learning

“Tricks”: Regularization and Dropout

Because neural networks are powerful, overfitting is a potential problem.

- Regularization works similarly for neural nets as it does in linear classifiers: penalize the weights by $\lambda||\boldsymbol{\theta}||_2^2$.
- Dropout prevents overfitting by randomly deleting weights or nodes during training. This prevents the model from relying too much on individual features or connections.
- Dropout rates are usually between 0.1 and 0.5, tuned on validate data.

“Tricks”: Initialization

Unlike linear classifiers, initialization in neural networks can affect the outcome.

- If the initial weights are too large, activations may saturate the activation function (for sigmoid or tanh, small gradients) or overflow (for ReLU activation).
- If they are too small, learning may take too many iterations to converge.

Other “Tricks”

Stochastic gradient descent is the simplest learning algorithm for neutral networks, but there are many other choices:

- Use adaptive learning rates for each parameter
- In practice, most implementations clip gradient to some maximum magnitude before making updates

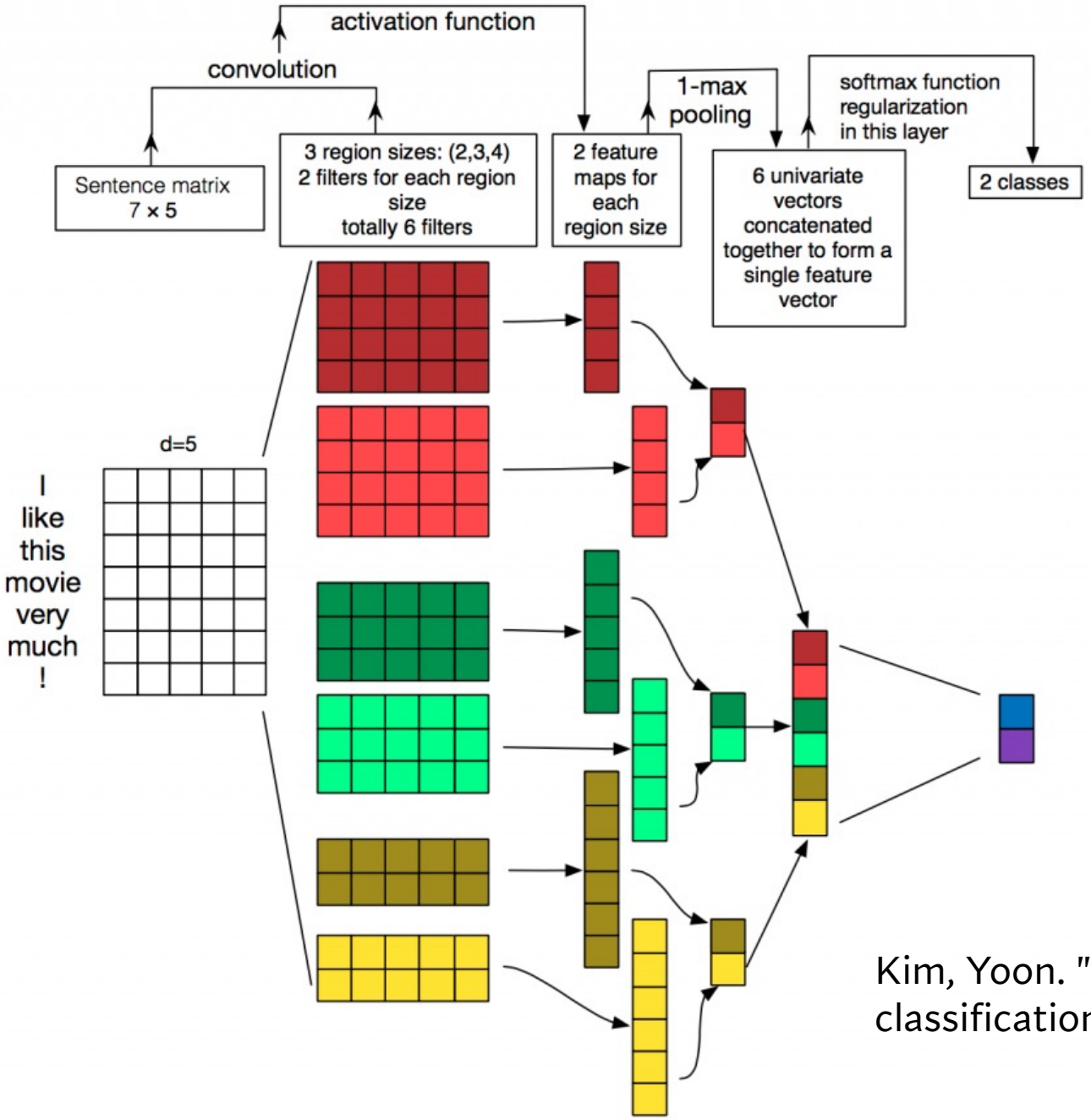
Early stopping: check performance on a development set, and stop training when performances starts to get worst.

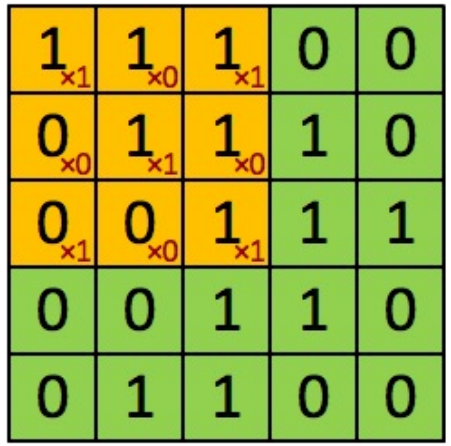
Neural Architectures for Sequence Data

Text is naturally viewed as a sequence of tokens $w_1, w_2, \ldots, w_M$

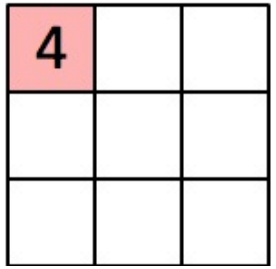
- Context is lost when this sequence is converted to a bag-of-words
- Instead, a lookup layer can compute embeddings for each token, resulting in a matrix $\mathbf{X}^{(0)} = \mathbf{\Theta}^{(x \to z)}[\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \ldots, \boldsymbol{e}_{w_M}]$, where $\mathbf{X}^{(0)} \in \mathbb{R}^{K_e \times M}$
- Higher-order representations $\mathbf{X}^{(d>0)}$ can then be computed from $\mathbf{X}^{(0)}$
- For classification, the top layer $\mathbf{X}^{(D)}$ must be converted into a vector, using a pooling operation, such as max-pooling

$$z_k = \max\left(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \ldots, x_{k,M}^{(D)}\right)$$





Input Feature



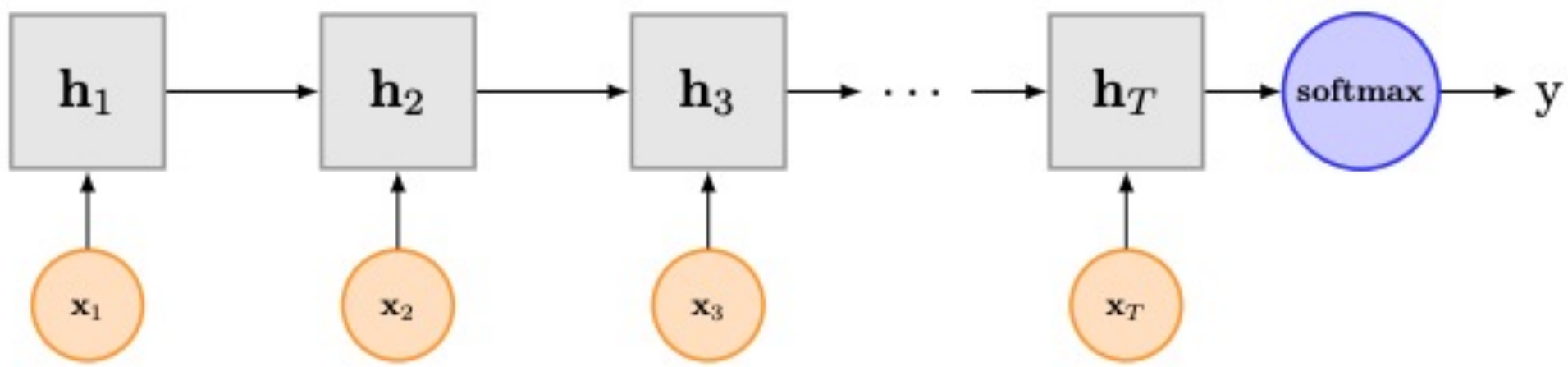
Convoluted Feature

Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).

Other Neural Architecture

- CNN are sensitive to local dependencies between words
- In **Recurrent Neural Networks**
  - A model of context is constructed while processing the text from left-to-right. Those networks are theoretically sensitive to global dependencies
  - LSTM, Bi-LSTM, GRU, Bi-GRU, ...

Example Recurrent Neural Network for Classification



Liu, Pengfei, Xipeng Qiu, and Xuanjing Huang. "Recurrent neural network for text classification with multi-task learning." arXiv preprint arXiv:1605.05101 (2016).

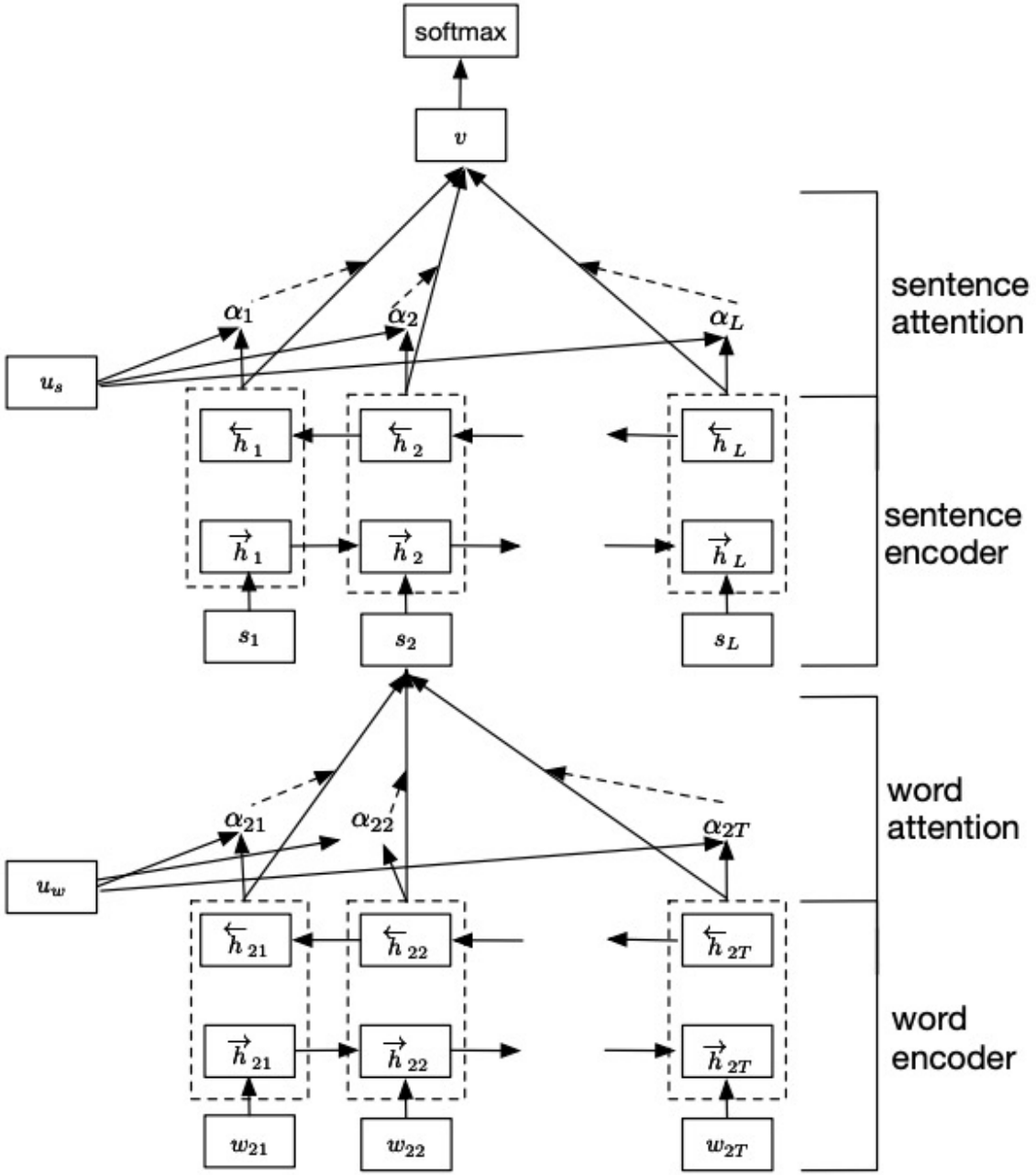


Figure 2: Hierarchical Attention Network.

Yang, Zichao, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. "Hierarchical attention networks for document classification." NAACL 2016.

Text Classification Applications

- Classical Applications of Text Classification
  - Sentiment and opinion analysis
  - Word sense disambiguation
- Design decisions in text classification
- Evaluation

Sentiment Analysis

- The **sentiment** expressed in a text refers to the author's subjective or emotional attitude towards the central topic of the text.
- Sentiment analysis is a classical application of text classification, and is typically approached with a bag-of-words classifier.

Beyond the Bag-of-words

Some linguistic phenomena require going beyond the bag-of-words:

- *That's not bad for the first day*
- *This is not the worst thing that can happen*
- *It would be nice if you acted like you understood*
- *This film should be brilliant. The actors are first grade. Stallone plays a happy, wonderful man. His sweet wife is beautiful and adores him. He has a fascinating gift for living life fully. It sounds like a great plot, however, the film is a failure.*

Word Sense Disambiguation

Many words have multiple senses, or meanings.

- Word sense disambiguation is the problem of identifying the intended word sense in a given context.
- More formally, senses are properties of lemmas (uninflected word forms), and are grouped into synsets (synonym sets). Those synsets are collected in WORDNET.

Word Sense Disambiguation as Classification

How can we tell living *plants* from manufacturing plants? Context.

- *Town officials are hoping to attract new manufacturing plants through weakened environmental regulations.*
- *The endangered plants play an important role in the local ecosystem.*

Applying Text Classification

- The “raw” form of text is usually a sequence of characters
- Converting this into a meaningful feature vector x requires a series of design decisions, such as tokenization, normalization, and filtering

Tokenization

- Tokenization is the task of splitting the input into discrete tokens
- This may seem easy for Latin script languages like English, but there are some tricky parts. How many tokens do you see in this example?
- *O'Neill's prize-winning pit bull isn't really a "bull".*

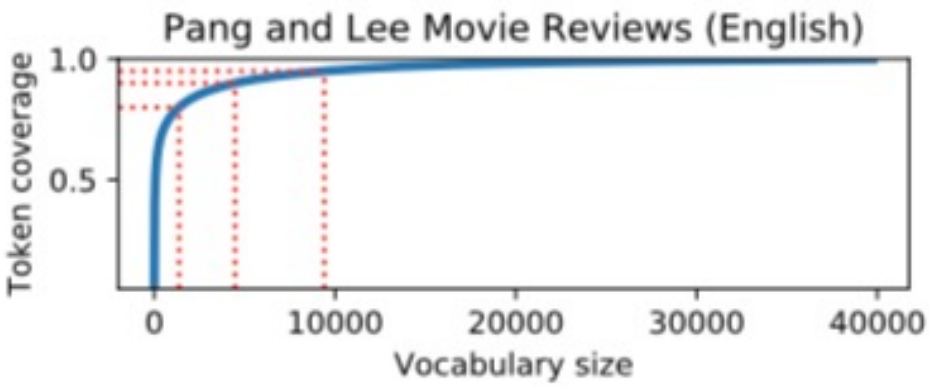
Four English Tokenizers

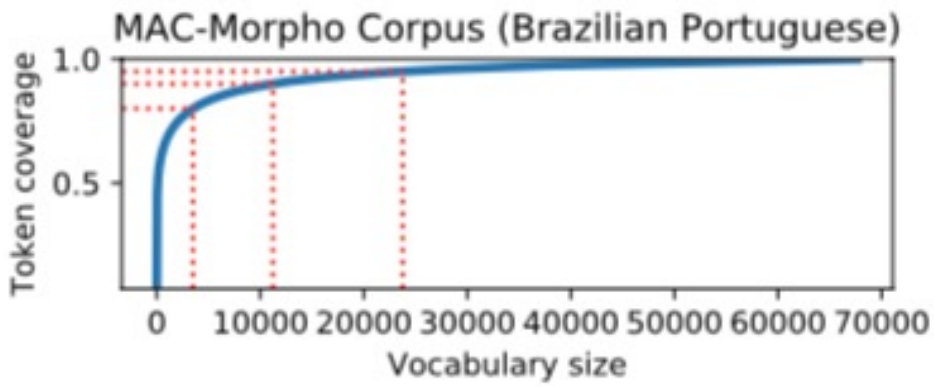
- Input: *Isn't Ahab, Ahab? ;)*

Whitespace	Isn't	Ahab,	Ahab?	;)				
Treebank	Is	n't	Ahab	,	Ahab	?	;	)
Tweet	Isn't	Ahab	,	Ahab	?	;)		
TokTok	Isn	'	t	Ahab	,	Ahab	?	;	)

Vocabulary Size Filtering

A small number of word **types** accounts for the majority of word **tokens**.



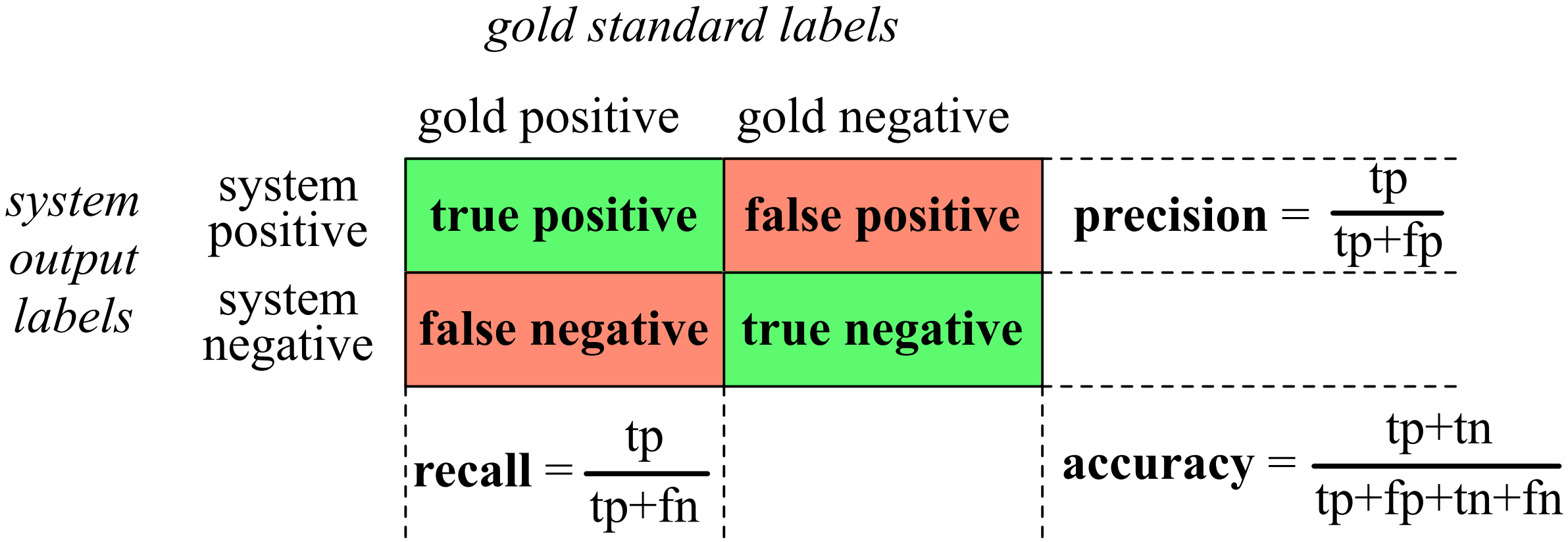


- The number of parameters in a classifier usually grows linearly with the size of the vocabulary
- It can be useful to limit the vocabulary, e.g., to word types appearing at least x times, or in at least y% of documents.

Evaluating Your Classifier

- Let's consider just binary text classification tasks
- Imagine you're the CEO of Delicious Pie Company
- You want to know what people are saying about your pies
- So you build a "Delicious Pie" tweet detector
  - Positive class: tweets about Delicious Pie Co
  - Negative class: all other tweets

The 2-by-2 confusion matrix



Evaluation: Accuracy

- Why don't we use **accuracy** as our metric?
- Imagine we saw 1 million tweets
  - 100 of them talked about Delicious Pie Co.
  - 999,900 talked about something else
- We could build a dumb classifier that just labels every tweet "not about pie"
  - It would get 99.99% accuracy!!! Wow!!!!
  - But useless! Doesn't return the comments we are looking for!
  - That's why we use **precision** and **recall** instead

Evaluation: Precision

% of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels)

$$\textbf{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Evaluation: Recall

% of items actually present in the input that were correctly identified by the system.

$$\textbf{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Why Precision and Recall

- Our dumb pie-classifier
  - Just label nothing as "about pie"

Accuracy=99.99%

but

Recall = 0

- (it doesn't get any of the 100 Pie tweets)

Precision and recall, unlike accuracy, emphasize true positives:

- finding the things that we are supposed to be looking for.

Evaluation: F Measure

- F measure: a single number that combines P and R:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

- We almost always use balanced F_1 (i.e., $\beta = 1$)

$$\mathrm{F}_1 = \frac{2PR}{P + R}$$

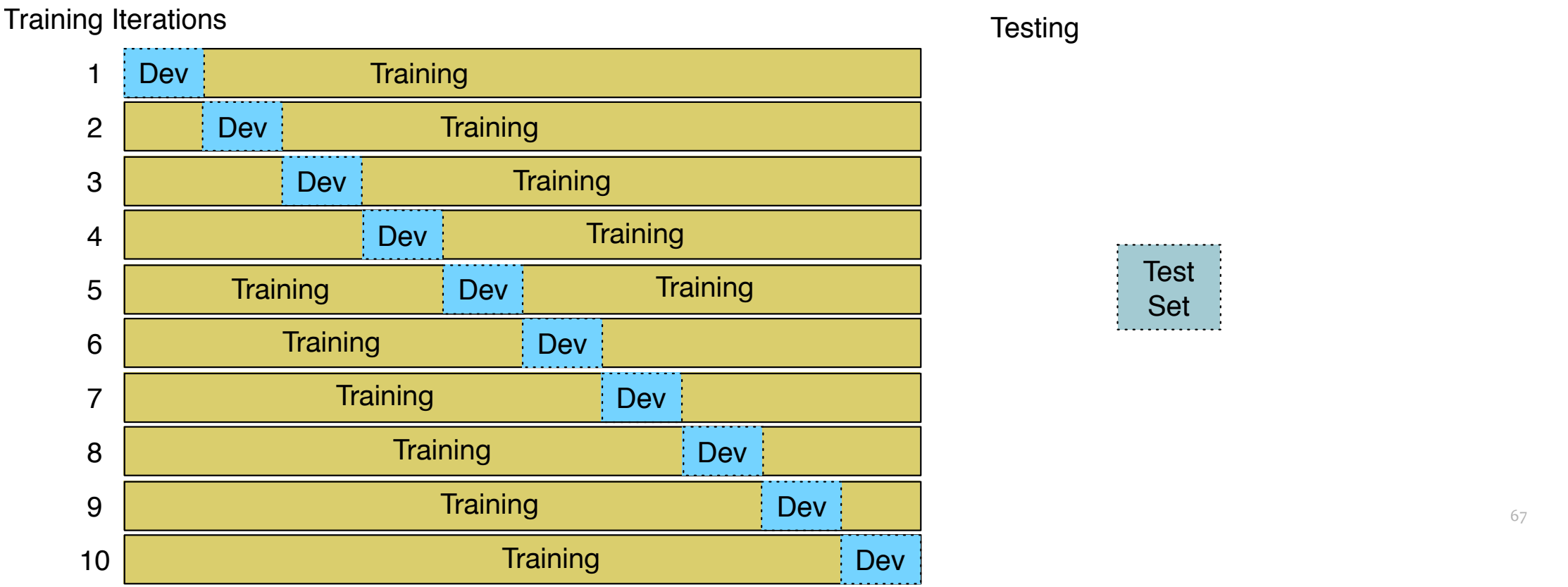
Development Test Sets and Cross-validation

Train on training set, tune on dev set, report on tes tset

- This avoids overfitting (‘tuning to the test set’)
- More conservative estimate of performance
- But paradox: want as much data as possible for training, and as much for dev; how to split?

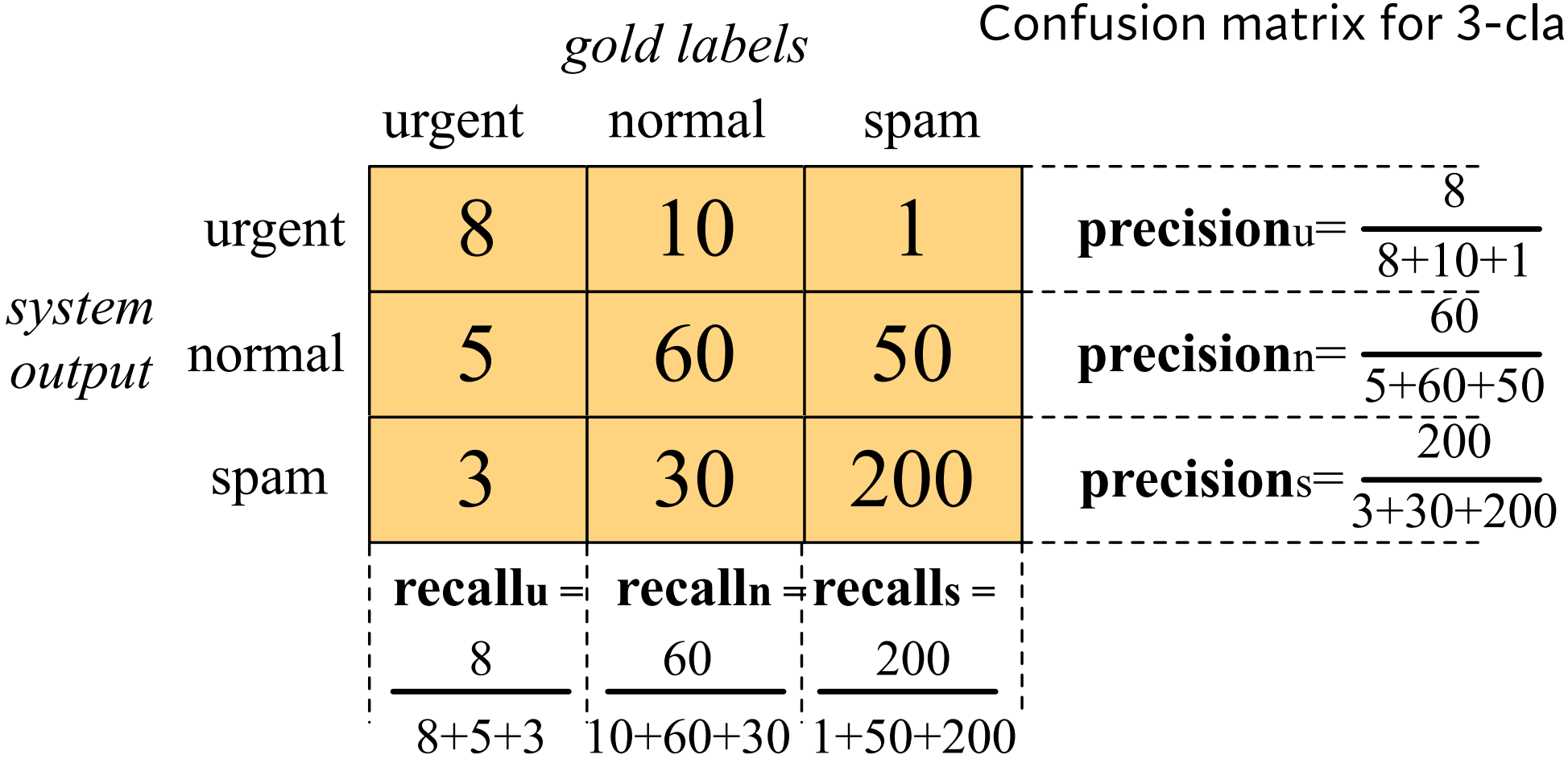
Cross-validation: multiple splits

- Pool results over splits, Compute pooled dev performance



Evaluating Multi-Class Classification

Confusion matrix for 3-class classification



Evaluating Multi-Class Classification

- Two ways to combine performance across classes:
  - **Macro F-measure:** compute the F-measure per class, and average across all classes. This treats all classes equally, regardless of their frequency.
  - **Micro F-measure:** compute the total number of true positives, false positives, and false negatives across all classes, and compute a single F-measure. This emphasizes performance on high-frequency classes.

Micro- and Macro- Averaging

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$