

CS 489: Introduction to Natural Language Processing

Lecture 4: Supervised Learning Foundations

Instructor: Freda Shi

September 22, 2026



UNIVERSITY OF
WATERLOO

Outline

1. Text classification and input representations
2. Classification baselines: rules and Naïve Bayes
3. Logistic regression: learning, softmax, and regularization
4. From linear models to neural text classifiers

Recap: Where We Are

So far:

- **Lecture 1:** What is NLP? Language models. Probability and information theory.
- **Lecture 2:** Tokenization: BPE, WordPiece, Unigram LM.
- **Lecture 3:** Word embeddings: from co-occurrence matrices to Word2Vec.

We can now represent words as dense vectors in $\mathbb{R}^d$.

Recap: Where We Are

So far:

- **Lecture 1:** What is NLP? Language models. Probability and information theory.
- **Lecture 2:** Tokenization: BPE, WordPiece, Unigram LM.
- **Lecture 3:** Word embeddings: from co-occurrence matrices to Word2Vec.

We can now represent words as dense vectors in $\mathbb{R}^d$.

Today's question: How do we go from word representations to **predictions**?

Classification: The Simplest NLP Application

Given a piece of text, assign it to one of C predefined **categories**.



Classification: The Simplest NLP Application

Given a piece of text, assign it to one of C predefined **categories**.



Examples:

- **Sentiment analysis:** “Great movie!” → **positive**
- **Spam detection:** “You’ve won \$1M!” → **spam**
- **Topic classification:** news article → {sports, politics, tech, ...}
- **Language identification:** text → {English, French, Chinese, ...}

General Formulation

Data: A set of labeled examples $\mathcal{D} = \{(s_1, y_1), (s_2, y_2), \dots, (s_n, y_n)\}$.

- s_i : a sentence (or a piece of text).
- $y_i \in \{1, \dots, C\}$: the label (category).

Goal: Build a classifier that takes an arbitrary sentence s as input and predicts the label y .

$$\text{classify}(s) = \arg \max_y \text{score}(s, y; \Theta)$$

General Formulation

Data: A set of labeled examples $\mathcal{D} = \{(s_1, y_1), (s_2, y_2), \dots, (s_n, y_n)\}$.

- s_i : a sentence (or a piece of text).
- $y_i \in \{1, \dots, C\}$: the label (category).

Goal: Build a classifier that takes an arbitrary sentence s as input and predicts the label y .

$$\text{classify}(s) = \arg \max_y \text{score}(s, y; \Theta)$$

Three key design decisions:

1. **Modeling:** How do we define the score function?
2. **Learning:** How do we choose the parameters Θ ?
3. **Inference:** How do we solve the argmax?

How Do We Represent Text for a Classifier?

The classifier needs a **fixed-size numerical input**. But text is variable-length!

Two paradigms:

Feature engineering:

- Manually design features.
- Bag-of-words counts.
- Hand-crafted rules (e.g., “contains exclamation mark”).
- Requires domain expertise.

Learned representations:

- Use word embeddings (Lecture 3).
- Average word vectors to get a sentence vector.
- Let the model learn useful features.
- Less manual effort, more data-driven.

Bag-of-Words Features

Idea: Represent a document as a vector of word counts, ignoring order.

Example: “I love this movie. I love the acting.”

	I	love	this	movie	the	acting	...
x	2	2	1	1	1	1	...

Advantages:

- Simple, fast, interpretable.
- A strong baseline for many tasks.

Bag-of-Words Features

Idea: Represent a document as a vector of word counts, ignoring order.

Example: “I love this movie. I love the acting.”

	I	love	this	movie	the	acting	...
x	2	2	1	1	1	1	...

Advantages:

- Simple, fast, interpretable.
- A strong baseline for many tasks.

Disadvantages:

- Ignores word order: “the cat sat on the mat” = “the mat sat on the cat”.
- High-dimensional, sparse vectors ($|V|$ can be 10^4 – 10^6).
- Cannot capture compositionality or negation.

From Bag-of-Words to Embeddings

Alternative: Use the word embeddings from Lecture 3!

Given sentence $s = w_1 w_2 \cdots w_k$ with word embeddings $\text{emb}(w_i) \in \mathbb{R}^d$:

Average pooling:

$$\mathbf{x} = \frac{1}{k} \sum_{i=1}^k \text{emb}(w_i) \in \mathbb{R}^d$$

Advantages over bag-of-words:

- Dense and low-dimensional ($d \approx 100\text{--}300$ vs. $|V| \approx 50,000$).
- Captures semantic similarity (synonyms map to similar vectors).
- Can handle unseen words (if subword embeddings are used).

Disadvantage: Still ignores word order (just like bag-of-words). More on this later in the course (RNNs, Transformers).

Approach 1: Rule-Based Classification

Example: Sentiment classification of sentence s .

- If s contains words in [*good, excellent, extraordinary, ...*] → return **positive**.
- If s contains words in [*bad, terrible, awful, ...*] → return **negative**.

Approach 1: Rule-Based Classification

Example: Sentiment classification of sentence *s*.

- If *s* contains words in [*good, excellent, extraordinary, ...*] → return **positive**.
- If *s* contains words in [*bad, terrible, awful, ...*] → return **negative**.

Pros:

- Interpretable.
- Can be very accurate with carefully refined rules.
- No training data needed.

Cons:

- Rules are **difficult to define**.
- Systems become very complicated.
- **Hardly generalizable** to new domains.
- Cannot handle nuance (“not bad” = positive?).

Example tool: VADER (Valence Aware Dictionary and sEntiment Reasoner), a lexicon and rule-based sentiment analyzer for social media text.

Approach 2: Naïve Bayes

A **generative** classifier: model $P(y \mid s)$ via Bayes' rule.

Modeling: Set $\text{score}(s, y; \Theta) := P(y \mid s)$, estimated from data.

$$P(y \mid s) = \frac{P(s \mid y) P(y)}{P(s)} \propto P(s \mid y) P(y)$$

Approach 2: Naïve Bayes

A **generative** classifier: model $P(y | s)$ via Bayes' rule.

Modeling: Set $\text{score}(s, y; \Theta) := P(y | s)$, estimated from data.

$$P(y | s) = \frac{P(s | y) P(y)}{P(s)} \propto P(s | y) P(y)$$

Naïve (unigram) assumption: Words are conditionally independent given the class.

$$P(s | y) = P(w_1 | y) P(w_2 | y) \cdots P(w_k | y) \quad \text{if } s = w_1 w_2 \cdots w_k$$

Θ : a look-up table storing $P(w | y)$ and $P(y)$.

Naïve Bayes: Learning

Each $P(w \mid y)$ and $P(y)$ can be estimated by **counting** from the training corpus:

$$P(w \mid y) = \frac{\text{count}(w, y)}{\sum_{w' \in V} \text{count}(w', y)} \qquad P(y) = \frac{\text{count}(y)}{n}$$

This is the **maximum likelihood estimate** (MLE).

Naïve Bayes: Learning

Each $P(w \mid y)$ and $P(y)$ can be estimated by **counting** from the training corpus:

$$P(w \mid y) = \frac{\text{count}(w, y)}{\sum_{w' \in V} \text{count}(w', y)} \qquad P(y) = \frac{\text{count}(y)}{n}$$

This is the **maximum likelihood estimate** (MLE).

Problem: An unseen word in a class leads to $P(w \mid y) = 0$, which zeros out the entire product!

Naïve Bayes: Learning

Each $P(w \mid y)$ and $P(y)$ can be estimated by **counting** from the training corpus:

$$P(w \mid y) = \frac{\text{count}(w, y)}{\sum_{w' \in V} \text{count}(w', y)} \qquad P(y) = \frac{\text{count}(y)}{n}$$

This is the **maximum likelihood estimate** (MLE).

Problem: An unseen word in a class leads to $P(w \mid y) = 0$, which zeros out the entire product!

Solution: Laplace smoothing.

$$P(w \mid y) = \frac{\text{count}(w, y) + \alpha}{\sum_{w' \in V} \text{count}(w', y) + \alpha|V|}$$

Typically $\alpha = 1$ (add-one smoothing).

Naïve Bayes: Worked Example

Type	Label	Sentence
Train	-	just plain boring
Train	-	entirely predictable
Train	-	no surprises and very few laughs
Train	+	very powerful
Train	+	the most fun film of the summer
Test	?	predictable with no fun

Prior: $P(+) = \frac{2}{5}$, $P(-) = \frac{3}{5}$

$|V| = 18$ from training set. Smoothing $\alpha = 1$.

$$P(\text{predictable} \mid +) = \frac{0+1}{9+18} = \frac{1}{27}$$

$$P(\text{no} \mid +) = \frac{0+1}{9+18} = \frac{1}{27}$$

$$P(\text{fun} \mid +) = \frac{1+1}{9+18} = \frac{2}{27}$$

$$P(\text{predictable} \mid -) = \frac{1+1}{11+18} = \frac{2}{29}$$

$$P(\text{no} \mid -) = \frac{1+1}{11+18} = \frac{2}{29}$$

$$P(\text{fun} \mid -) = \frac{0+1}{11+18} = \frac{1}{29}$$

Naïve Bayes: Worked Example

Type	Label	Sentence
Train	-	just plain boring
Train	-	entirely predictable
Train	-	no surprises and very few laughs
Train	+	very powerful
Train	+	the most fun film of the summer
Test	?	predictable with no fun

Prior: $P(+) = \frac{2}{5}$, $P(-) = \frac{3}{5}$

$|V| = 18$ from training set. Smoothing $\alpha = 1$.

$$P(\text{predictable} \mid +) = \frac{0+1}{9+18} = \frac{1}{27}$$

$$P(\text{no} \mid +) = \frac{0+1}{9+18} = \frac{1}{27}$$

$$P(\text{fun} \mid +) = \frac{1+1}{9+18} = \frac{2}{27}$$

$$P(\text{predictable} \mid -) = \frac{1+1}{11+18} = \frac{2}{29}$$

$$P(\text{no} \mid -) = \frac{1+1}{11+18} = \frac{2}{29}$$

$$P(\text{fun} \mid -) = \frac{0+1}{11+18} = \frac{1}{29}$$

Scoring: (with is out of vocabulary and is ignored.)

$$P(+) \cdot P(\text{pred} \mid +) \cdot P(\text{no} \mid +) \cdot P(\text{fun} \mid +) = \frac{2}{5} \cdot \frac{1}{27} \cdot \frac{1}{27} \cdot \frac{2}{27} = 4.1 \times 10^{-5}$$

$$P(-) \cdot P(\text{pred} \mid -) \cdot P(\text{no} \mid -) \cdot P(\text{fun} \mid -) = \frac{3}{5} \cdot \frac{2}{29} \cdot \frac{2}{29} \cdot \frac{1}{29} = 9.8 \times 10^{-5}$$

⇒ Classify as **negative** (higher score). Correct!

Naïve Bayes: Inference

$$\text{classify}(s) = \arg \max_y \text{score}(s, y; \Theta) = \arg \max_y P(y \mid s)$$

$$P(y \mid s) \propto P(y) P(s \mid y) = P(y) P(w_1 \mid y) P(w_2 \mid y) \cdots P(w_k \mid y)$$

In practice, work in **log space** to avoid numerical underflow:

$$\log P(y \mid s) = \text{const.} + \log P(y) + \sum_{i=1}^k \log P(w_i \mid y)$$

Inference is just a **table lookup** + summation: very fast!

Generative vs. Discriminative Classifiers

Generative (e.g., Naïve Bayes):

- Models $P(s, y) = P(s | y) P(y)$.
- Can generate new data examples.
- Makes strong independence assumptions.
- Parameters estimated by counting.

Discriminative (e.g., logistic regression):

- Models $P(y | s)$ directly.
- Only learns the decision boundary.
- Fewer assumptions about data.
- Parameters learned by optimization.

Key difference: A generative model lets you *generate* new (s, y) pairs. A discriminative model only classifies.

In practice, discriminative models often achieve **higher accuracy** for classification tasks.

Approach 3: Logistic Regression

A **discriminative** linear classifier.

Suppose we can represent sentence s with a feature vector $\mathbf{x} \in \mathbb{R}^d$ (e.g., bag-of-words or averaged word embeddings).

Approach 3: Logistic Regression

A **discriminative** linear classifier.

Suppose we can represent sentence s with a feature vector $\mathbf{x} \in \mathbb{R}^d$ (e.g., bag-of-words or averaged word embeddings).

Binary classification ($y \in \{0, 1\}$):

$$P(y = 1 \mid s) = \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b)$$

$$P(y = 0 \mid s) = 1 - \sigma(\boldsymbol{\theta}^\top \mathbf{x} + b)$$

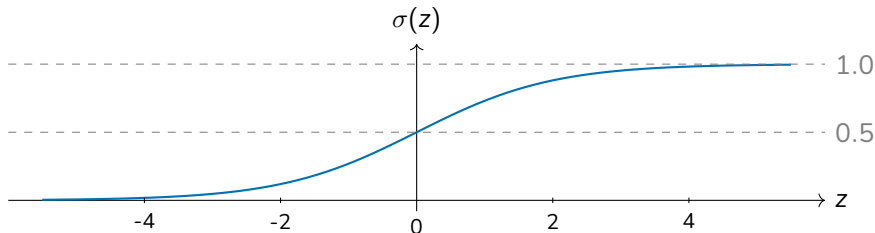
where σ is the **logistic (sigmoid) function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \quad \sigma: \mathbb{R} \rightarrow (0, 1)$$

$\boldsymbol{\theta}^\top \mathbf{x} + b$ can be written as $\boldsymbol{\theta}^\top \mathbf{x}$ if we append a constant 1 to $\mathbf{x}$.

The Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

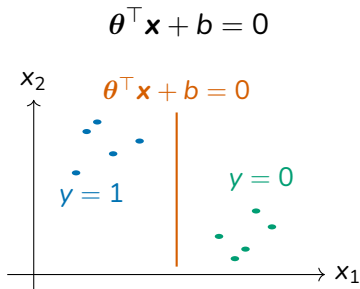


Properties:

- Maps any real number to $(0, 1)$, interpretable as a probability.
- $\sigma(0) = 0.5$; $\sigma(z) \rightarrow 1$ as $z \rightarrow \infty$; $\sigma(z) \rightarrow 0$ as $z \rightarrow -\infty$.
- Derivative: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

What Does Logistic Regression Learn?

The model learns a **linear decision boundary** in feature space:



- $\theta^\top \mathbf{x} + b > 0 \Rightarrow P(y = 1) > 0.5 \Rightarrow$ predict class 1.
- $\theta^\top \mathbf{x} + b < 0 \Rightarrow P(y = 1) < 0.5 \Rightarrow$ predict class 0.
- The model is **linear**: it cannot capture complex, nonlinear patterns.

Logistic Regression: Learning

Objective: Find θ that maximizes the probability of the training data.

Assuming each example is sampled independently:

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^n \sigma(\theta^{\top} \mathbf{x}_i)^{y_i} (1 - \sigma(\theta^{\top} \mathbf{x}_i))^{1-y_i}$$

Logistic Regression: Learning

Objective: Find θ that maximizes the probability of the training data.

Assuming each example is sampled independently:

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^n \sigma(\theta^{\top} \mathbf{x}_i)^{y_i} (1 - \sigma(\theta^{\top} \mathbf{x}_i))^{1-y_i}$$

Equivalently, minimize the **negative log-likelihood** (cross-entropy loss):

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \left[y_i \log \sigma(\theta^{\top} \mathbf{x}_i) + (1 - y_i) \log (1 - \sigma(\theta^{\top} \mathbf{x}_i)) \right]$$

Logistic Regression: Learning

Objective: Find θ that maximizes the probability of the training data.

Assuming each example is sampled independently:

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^n \sigma(\theta^{\top} \mathbf{x}_i)^{y_i} (1 - \sigma(\theta^{\top} \mathbf{x}_i))^{1-y_i}$$

Equivalently, minimize the **negative log-likelihood** (cross-entropy loss):

$$\mathcal{L}(\theta) = - \sum_{i=1}^n \left[y_i \log \sigma(\theta^{\top} \mathbf{x}_i) + (1 - y_i) \log (1 - \sigma(\theta^{\top} \mathbf{x}_i)) \right]$$

No closed-form solution! Must optimize iteratively.

Binary Cross-Entropy Loss: Intuition

For a single example $(\mathbf{x}, y)$ with prediction $\hat{y} = \sigma(\boldsymbol{\theta}^\top \mathbf{x})$:

$$\ell(\boldsymbol{\theta}) = -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right]$$

Binary Cross-Entropy Loss: Intuition

For a single example $(\mathbf{x}, y)$ with prediction $\hat{y} = \sigma(\boldsymbol{\theta}^\top \mathbf{x})$:

$$\ell(\boldsymbol{\theta}) = -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right]$$

If $y = 1$ (true label is positive):

$$\ell = -\log \hat{y}$$

- $\hat{y} \rightarrow 1$: loss $\rightarrow 0$ (correct, confident).
- $\hat{y} \rightarrow 0$: loss $\rightarrow \infty$ (wrong, penalize heavily).

If $y = 0$ (true label is negative):

$$\ell = -\log(1 - \hat{y})$$

- $\hat{y} \rightarrow 0$: loss $\rightarrow 0$ (correct, confident).
- $\hat{y} \rightarrow 1$: loss $\rightarrow \infty$ (wrong, penalize heavily).

The loss **penalizes confident wrong predictions** severely.

Gradient Descent

To minimize $\mathcal{L}(\theta)$, take steps in the direction of the **negative gradient**:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta)$$

where $\eta > 0$ is the **learning rate**.

Gradient Descent

To minimize $\mathcal{L}(\theta)$, take steps in the direction of the **negative gradient**:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta)$$

where $\eta > 0$ is the **learning rate**.

For logistic regression:

$$\nabla_{\theta} \mathcal{L} = - \sum_{i=1}^n (y_i - \sigma(\theta^{\top} \mathbf{x}_i)) \mathbf{x}_i$$

Intuition: If $y_i = 1$ but $\sigma(\theta^{\top} \mathbf{x}_i)$ is small (wrong prediction), the gradient pushes θ in the direction of $\mathbf{x}_i$ to increase the score.

Stochastic Gradient Descent (SGD)

Problem: Computing the full gradient over all n examples is expensive.

Stochastic Gradient Descent (SGD)

Problem: Computing the full gradient over all n examples is expensive.

Solution: Stochastic Gradient Descent.

1. Randomly shuffle the training data.
2. For each example (or small **mini-batch**) $(\mathbf{x}_i, y_i)$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ell_i(\boldsymbol{\theta})$$

3. Repeat for multiple **epochs** (passes over the data).

Stochastic Gradient Descent (SGD)

Problem: Computing the full gradient over all n examples is expensive.

Solution: Stochastic Gradient Descent.

1. Randomly shuffle the training data.
2. For each example (or small **mini-batch**) $(\mathbf{x}_i, y_i)$:

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \ell_i(\theta)$$

3. Repeat for multiple **epochs** (passes over the data).

Key properties:

- Each update is an **unbiased estimate** of the full gradient.
- Much faster per update: $\mathcal{O}(1)$ vs. $\mathcal{O}(n)$.
- In non-convex problems, SGD noise can help escape local minima.
- **Mini-batch SGD** (batch size B): balance between speed and variance.

Regularization

Problem: With many features, logistic regression can **overfit**: it memorizes training data but generalizes poorly.

Regularization

Problem: With many features, logistic regression can **overfit**: it memorizes training data but generalizes poorly.

Solution: Add a **regularization term** to the loss:

$$\mathcal{L}_{\text{reg}}(\theta) = \mathcal{L}(\theta) + \lambda R(\theta)$$

- **L2 regularization** (weight decay): $R(\theta) = \|\theta\|_2^2 = \sum_j \theta_j^2$
 - Encourages small weights; smooth decision boundary.
- **L1 regularization**: $R(\theta) = \|\theta\|_1 = \sum_j |\theta_j|$
 - Encourages **sparse** weights (feature selection).

$\lambda > 0$ controls the regularization strength. Tuned on a **validation set**.

Multi-Class Classification: Softmax

What if there are $C > 2$ classes?

Softmax regression (multinomial logistic regression):

- One weight vector per class: $\theta_1, \theta_2, \dots, \theta_C \in \mathbb{R}^d$.
- Equivalently, a weight matrix $\mathbf{W} \in \mathbb{R}^{C \times d}$, bias $\mathbf{b} \in \mathbb{R}^C$.

Compute scores (logits): $\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^C$.

Multi-Class Classification: Softmax

What if there are $C > 2$ classes?

Softmax regression (multinomial logistic regression):

- One weight vector per class: $\theta_1, \theta_2, \dots, \theta_C \in \mathbb{R}^d$.
- Equivalently, a weight matrix $\mathbf{W} \in \mathbb{R}^{C \times d}$, bias $\mathbf{b} \in \mathbb{R}^C$.

Compute scores (logits): $\mathbf{s} = \mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^C$.

Softmax function converts scores to probabilities:

$$P(y = c \mid \mathbf{x}) = \frac{e^{s_c}}{\sum_{c'=1}^C e^{s_{c'}}} \quad \text{for } c = 1, \dots, C$$

Properties: $\sum_{c=1}^C P(y = c \mid \mathbf{x}) = 1$ and $P(y = c \mid \mathbf{x}) > 0$ for all c .

Cross-Entropy Loss (Multi-Class)

Loss for a single example $(\mathbf{x}, y)$ where $y \in \{1, \dots, C\}$:

$$\ell(\mathbf{W}, \mathbf{b}) = -\log P(y \mid \mathbf{x}) = -\log \frac{e^{s_y}}{\sum_{c'=1}^C e^{s_{c'}}} = -s_y + \log \sum_{c'=1}^C e^{s_{c'}}$$

This is the **cross-entropy** $H(\mathbf{q}, \mathbf{p})$ between the one-hot gold-label distribution $\mathbf{q}$ and the predicted distribution $\mathbf{p}$.

Over the full dataset:

$$\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n} \sum_{i=1}^n \log P(y_i \mid \mathbf{x}_i)$$

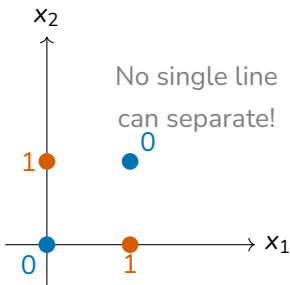
Connection to Lecture 1: This is exactly the cross-entropy $H(P_{\text{data}}, P_{\text{model}})$!
Minimizing cross-entropy $\Leftrightarrow$ maximizing likelihood.

Limitations of Linear Classifiers

Logistic regression learns a **linear** decision boundary.

Problem: Many NLP tasks require **nonlinear** decision boundaries.

Example: XOR problem

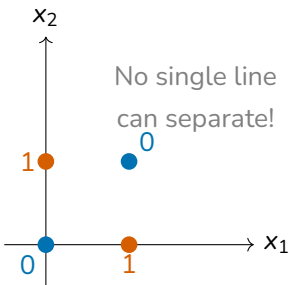


Limitations of Linear Classifiers

Logistic regression learns a **linear** decision boundary.

Problem: Many NLP tasks require **nonlinear** decision boundaries.

Example: XOR problem



Solution: Add nonlinearity → **neural networks**.

A Neural Layer: Generalized Perceptron

A neural layer = affine transformation + nonlinearity:

$$\text{neural_layer}(\mathbf{x}) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where g is an **activation function** applied element-wise.

Common activation functions:

Name	Formula	Range
Sigmoid	$\sigma(z) = \frac{1}{1 + e^{-z}}$	$(0, 1)$
Tanh	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$(-1, 1)$
ReLU	$\text{ReLU}(z) = \max\{0, z\}$	$[0, \infty)$

Why Nonlinearities?

Without activation functions, stacking layers is pointless:

$$\begin{aligned} \mathbf{z}^{(1)} &= \mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)} \\ \mathbf{z}^{(2)} &= \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)} \\ &= \mathbf{W}^{(1)}(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}) + \mathbf{b}^{(1)} \\ &= \underbrace{(\mathbf{W}^{(1)}\mathbf{W}^{(0)})}_{\mathbf{W}'}\mathbf{x} + \underbrace{(\mathbf{W}^{(1)}\mathbf{b}^{(0)} + \mathbf{b}^{(1)})}_{\mathbf{b}'} \end{aligned}$$

⇒ Just another linear transformation! No matter how many layers.

Why Nonlinearities?

Without activation functions, stacking layers is pointless:

$$\begin{aligned} \mathbf{z}^{(1)} &= \mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)} \\ \mathbf{z}^{(2)} &= \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)} \\ &= \mathbf{W}^{(1)}(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}) + \mathbf{b}^{(1)} \\ &= \underbrace{(\mathbf{W}^{(1)}\mathbf{W}^{(0)})}_{\mathbf{W}'}\mathbf{x} + \underbrace{(\mathbf{W}^{(1)}\mathbf{b}^{(0)} + \mathbf{b}^{(1)})}_{\mathbf{b}'} \end{aligned}$$

⇒ Just another linear transformation! No matter how many layers.

With nonlinearities, each layer can learn a different nonlinear feature, and the network can approximate arbitrarily complex functions ([universal approximation theorem](#)).

Multi-Layer Perceptron (MLP)

Stack multiple neural layers to build a **feed-forward neural network**:

$$\mathbf{z}^{(1)} = g(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}) \quad \text{(hidden layer 1)}$$

$$\mathbf{z}^{(2)} = g(\mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}) \quad \text{(hidden layer 2)}$$

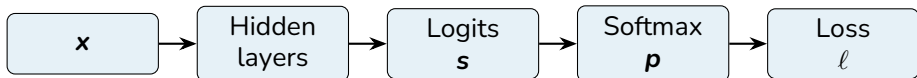
$$\vdots$$

$$\mathbf{s} = \mathbf{W}^{(L)}\mathbf{z}^{(L)} + \mathbf{b}^{(L)} \quad \text{(output logits)}$$

- $\mathbf{z}^{(\ell)}$: **hidden units** at layer ℓ (learned representations).
- The final layer produces C-dimensional logits $\mathbf{s}$ (no activation).
- Apply **softmax** to $\mathbf{s}$ to get class probabilities.
- Also called a *multi-layer perceptron* (MLP).

Neural Network Training: Softmax + Cross-Entropy

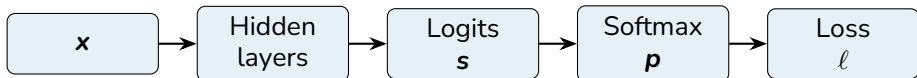
Forward pass:



$$\mathbf{p} = \text{softmax}(\mathbf{s}), \quad \ell = -\log p_y = -\log \frac{e^{s_y}}{\sum_{c'} e^{s_{c'}}}$$

Neural Network Training: Softmax + Cross-Entropy

Forward pass:



$$p = \text{softmax}(s), \quad \ell = -\log p_y = -\log \frac{e^{s_y}}{\sum_{c'} e^{s_{c'}}}$$

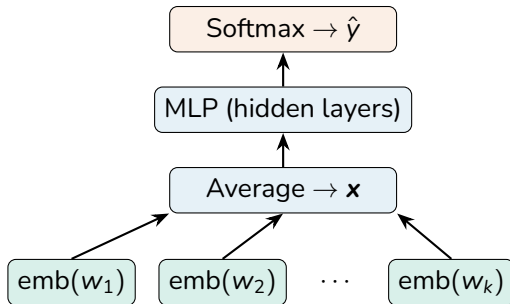
Backward pass (backpropagation):

- Propagate $\frac{\partial \ell}{\partial s}$ backward using the **chain rule**; each nonlinear layer contributes g' :

$$\frac{\partial \ell}{\partial \mathbf{W}^{(0)}} = \frac{\partial \ell}{\partial \mathbf{s}} \cdot \frac{\partial \mathbf{s}}{\partial \mathbf{z}^{(1)}} \cdot \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}}$$

- Update all parameters with SGD.

Putting It All Together: Neural Text Classifier



$$\mathbf{x} = \frac{1}{k} \sum_{i=1}^k \text{emb}(w_i), \quad \mathbf{z}^{(1)} = g(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}), \quad \mathbf{s} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

The embedding matrix can be **pretrained** (Word2Vec/GloVe) or **trained from scratch**.

Comparing the Approaches

	Rules	Naïve Bayes	Log. Reg.	Neural Net
Learning	none	counting	SGD	SGD + backprop
Decision boundary	manual	linear*	linear	nonlinear
Representation	manual	BoW	BoW / emb.	learned
Training data	none	small OK	moderate	large
Interpretable	high	high	moderate	low
Accuracy	varies	good	good	best

*Naïve Bayes is linear in log-space.

Trend in NLP: rules → statistical models → neural networks.

Most of the remaining lectures focus on neural approaches, especially

Transformers (Lecture 7).

Summary

- **Text classification** is the simplest supervised NLP task: map text to a label.
- Two representation strategies: **feature engineering** (BoW) vs. **learned representations** (embeddings).
- **Naïve Bayes** (generative): model $P(s \mid y)P(y)$, learn by counting, smooth to avoid zeros.
- **Logistic regression** (discriminative):
 - Binary: sigmoid + binary cross-entropy.
 - Multi-class: softmax + cross-entropy.
 - Learn by SGD; regularize to prevent overfitting.
- **Neural networks** add nonlinear hidden layers $\rightarrow$ more expressive classifiers.
- Train with **backpropagation** + SGD.
- Cross-entropy loss connects classification to the information-theoretic concepts from Lecture 1.

Readings

Recommended:

- Jurafsky & Martin, *SLP 3*, [Chapter 4: Logistic Regression and Text Classification](#).

Optional / further reading:

- Jurafsky & Martin, *SLP 3*, [Naïve Bayes Classification \(web appendix\)](#).
- Goldberg, Y. (2015 draft). *A Primer on Neural Network Models for Natural Language Processing*. u.cs.biu.ac.il/~yogo/nlp.pdf, Sections 1–5.
- Ng, A.Y. & Jordan, M.I. (2001). [On discriminative vs. generative classifiers: A comparison of logistic regression and naïve Bayes](#). *NeurIPS*.

Next Lecture

Statistical Language Models

- *N*-gram models: MLE, smoothing (Laplace, Kneser-Ney).
- Evaluation: perplexity, bits-per-character.
- Shannon's prediction experiments.