CS 533: Natural Language Processing

# Optimization, Introduction to Deep Learning

Karl Stratos

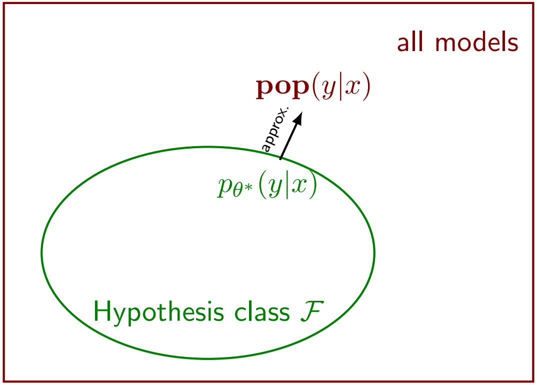
Rutgers University

## Review: Topic Classification with Linear Classifiers

- Classify a document $x \in \mathcal{X}$ to a topic $y \in \mathcal{Y}$
- Many annotated datasets, many related problems like sentiment analysis
- Text preprocessing, tokenization, vocabulary, Zipf's Law, bag-of-words (BOW) document representation $x \in \{0, 1\}^V$
- Linear classifier $\theta = (W = [w_1 \ldots w_L], b)$ defines the score of a document-label pair as $\textbf{score}_\theta(x, y) = w_y^\top x + b_y$
- Estimation
  - Softmax: $p_\theta(y|x) \propto \exp(\textbf{score}_\theta(x, y))$
  - Training Objective: Minimize cross entropy $H(\mathbf{pop}(y|x), p_\theta(y|x))$

## Review: Cross-Entropy Loss

$$\theta^* = \arg\min_{\theta \in \Theta} \mathop{\mathbf{E}}_{(x,y) \sim \mathbf{pop}} [-\log p_\theta(y|x)]$$



## Review: Empirical Cross-Entropy Loss

Don't know $\mathbf{pop}$, but can sample $\underbrace{(x_1, y_1) \ldots (x_N, y_N) \overset{\text{iid}}{\sim} \mathbf{pop}}_{\text{(training data)}}$

$$\widehat{J}_N(\theta) := -\frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i | x_i)$$

Important questions

- When is $\widehat{J}_N$ minimized for finite $N$?
- Are there multiple local minima (i.e., is $\widehat{J}_N$ nonconvex)?
- Can we optimize $\widehat{J}_N$ efficiently even if $N$ is really large?

## Minimizing Empirical Cross-Entropy Loss

$\widehat{J}_N(\theta) := -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|x_i) \geq 0$ is minimized to $0$ by $\theta^*$ that assigns $p_{\theta^*}(y_i|x_i) = 1$ for all $i = 1 \ldots N$.

## Minimizing Empirical Cross-Entropy Loss

$\widehat{J}_N(\theta) := -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|x_i) \geq 0$ is minimized to $0$ by $\theta^*$ that assigns $p_{\theta^*}(y_i|x_i) = 1$ for all $i = 1 \ldots N$.

- The model has memorized training data. Possible if expressive enough (in functional form & number of parameters)
- Is that a good thing?

## Minimizing Empirical Cross-Entropy Loss

$\widehat{J}_N(\theta) := -\frac{1}{N}\sum_{i=1}^{N} \log p_\theta(y_i|x_i) \geq 0$ is minimized to $0$ by $\theta^*$ that assigns $p_{\theta^*}(y_i|x_i) = 1$ for all $i = 1 \ldots N$.

- The model has memorized training data. Possible if expressive enough (in functional form & number of parameters)
- Is that a good thing?

Example: $N = 2$

$$\begin{aligned}
\mathcal{V} &= \{1:\texttt{stock}, 2:\texttt{game}, 3:\texttt{monday}, 4:\texttt{friday}\} \\
\mathcal{L} &= \{\texttt{business}, \texttt{sports}\} \\
x_1 &= (1,0,1,0),\ y_1 = \texttt{business} \\
x_2 &= (0,1,0,1),\ y_2 = \texttt{sports}
\end{aligned}$$

Find a linear classifier with zero training loss.

## Optimal Parameters

$W^* = [w^*_{\texttt{business}}, w^*_{\texttt{sports}}]$

$$\begin{aligned}
w^*_{\texttt{business}} &= (0, 0, 999, 0) \\
w^*_{\texttt{sports}} &= (0, 0, 0, 999) \\
\textbf{score}_{W^*}(x_1, \texttt{business}) &= 999 \\
\textbf{score}_{W^*}(x_1, \texttt{sports}) &= 0 \\
\textbf{score}_{W^*}(x_2, \texttt{business}) &= 0 \\
\textbf{score}_{W^*}(x_2, \texttt{sports}) &= 999 \\
p_{W^*}(\texttt{business}|x_1) &= \frac{\exp(999)}{\exp(999) + 1} \approx 1 \\
p_{W^*}(\texttt{sports}|x_1) &= \frac{1}{\exp(999) + 1} \approx 0 \\
p_{W^*}(\texttt{business}|x_2) &= \frac{1}{1 + \exp(999)} \approx 0 \\
p_{W^*}(\texttt{sports}|x_2) &= \frac{\exp(999)}{1 + \exp(999)} \approx 1
\end{aligned}$$

## Generalization Issues

Training examples:

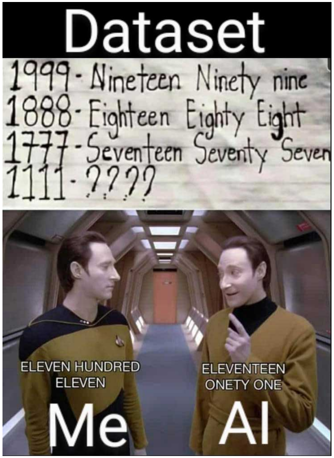
- ("stock monday", $\texttt{business}$)
- ("game tuesday", $\texttt{sports}$)

Test example: ("game monday", $\texttt{sports}$)

$$\begin{aligned}
\mathcal{V} &= \{1 : \texttt{stock}, 2 : \texttt{game}, 3 : \texttt{monday}, 4 : \texttt{friday}\} \\
x &= (0, 1, 1, 0) \\
w^*_{\texttt{business}} &= (0, 0, 999, 0) \\
w^*_{\texttt{sports}} &= (0, 0, 0, 999) \\
p_{W^*}(\texttt{business}|x) &= \frac{\exp(999)}{\exp(999) + 1} \approx 1 \\
p_{W^*}(\texttt{sports}|x) &= \frac{1}{\exp(999) + 1} \approx 0
\end{aligned}$$

## Overfitting

Model succeeds in fitting (finite) training data by exploiting spurious input-label correlations that do not generalize.


Dataset
1999 - Nineteen Ninety nine
1888 - Eighteen Eighty Eight
1777 - Seventeen Seventy Seven
1111 - ????
ELEVEN HUNDRED ELEVEN
ELEVENTEEN ONETY ONE
Me
AI

## Train-Validation-Test Split of Data

Always prepare a 3-way split of a labeled dataset

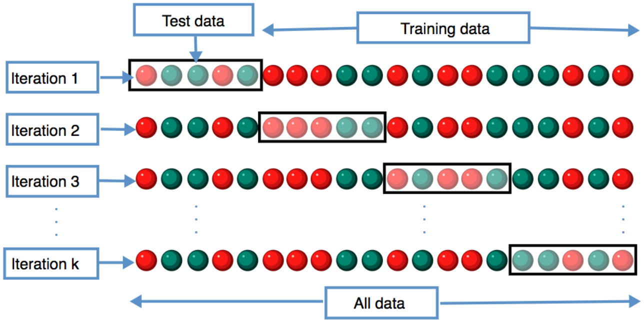
- **Training set**: Use this portion for training a supervised model. Majority of data ($> 90\%$)
- **Validation (or develepment/held-out) set**: Use this portion to check generalization performance of a trained model. About the same size as the test set
- **Test set**: After all experimentations and tuning, apply the trained model to this portion once and report final performance.

If dataset is missing a validation set, make one from random samples in the training set.

- If you don't have enough labeled data, consider $k$-fold **cross-validation**: a method for "reusing" held-out data

## $k$-fold cross-validation

- Partition training data into $k$ roughly equal parts
- Train on all but $i$-th part, test on $i$-th part



Use the average performance to assess a training configuration

- For final evaluation, train on all parts

## Regularization Methods

Ways to prevent overfitting.

1. Get more labeled data: best regularization method if possible!

2. **Early stopping**: stop training when validation performance fails to improve for a certain number of times ("patience")

3. Explicit regularization term: for some $\lambda > 0$ (to be tuned on dev set)

$$\min_{\theta \in \mathbb{R}^d} \widehat{J}_N(\theta) + \lambda \underbrace{\sum_{i=1}^{d} \theta_i^2}_{||\theta||_2^2} \qquad \text{or} \qquad \min_{\theta \in \mathbb{R}^d} \widehat{J}_N(\theta) + \lambda \underbrace{\sum_{i=1}^{d} |\theta_i|}_{||\theta||_1}$$

4. Other techniques (e.g., dropout, label smoothing)

## Empirical Cross-Entropy Loss for Linear Classifiers

Training data: $N$ iid samples of document-label pairs $(x_i, y_i) \in \mathbb{R}^V \times \{1 \ldots L\}$ from human annotators

$$\widehat{J}_N(W, b) = \frac{1}{N} \sum_{i=1}^{N} \log \left( \sum_{y=1}^{L} \exp(w_y^\top x_i + b_y) \right) - w_{y_i}^\top x_i - b_{y_i}$$

Training: Unconstrained optimization problem

$$(W^*, b^*) = \operatorname*{arg\,min}_{W \in \mathbb{R}^{V \times L},\, b \in \mathbb{R}^L} \widehat{J}_N(W, b)$$

How can we optimize $\widehat{J}_N$ efficiently even if $N$ is really large?

## Gradient of a Function

- Let $f : \mathbb{R}^d \to \mathbb{R}$ be any differentiable function.
- The $i$-th **partial derivative** of $f$ is the derivative of $f$ when viewed as a function of the $i$-th variable only: $\frac{\partial f(\theta)}{\partial \theta_i} \in \mathbb{R}$.
- The **gradient** of $f$ at $\theta \in \mathbb{R}^d$ is the vector

$$\nabla f(\theta) = \left( \frac{\partial f(\theta)}{\partial \theta_1}, \ldots, \frac{\partial f(\theta)}{\partial \theta_d} \right) \in \mathbb{R}^d$$

- Points to the direction of increase of $f$ at location $\theta \in \mathbb{R}^d$
  - Magnitude: rate of change
  - $\nabla f(\theta) = 0_d$ if $\theta$ is a stationary point
- $d = 1$ example $f(\theta) = (\theta - 3)^2$, $f'(\theta) = 2\theta - 6$
  - At $\theta = 5$: Increasing to the right at a rate of $4$
  - Stationary point $\theta = 3$

## Minimizing a Local Approximation

Taylor's theorem: First-order approximation of $f : \mathbb{R}^d \to \mathbb{R}$ around $\theta_0 \in \mathbb{R}^d$

$$f(\theta) \approx \underbrace{f(\theta_0)}_{\text{Current value}} + \underbrace{\nabla f(\theta_0)^\top (\theta - \theta_0)}_{\text{Change in value as we move away}}$$

The approximation is only good for a local neighborhood. Add an $l_2$ distance penalty, for some $\eta > 0$:

$$f_{\theta_0,\eta}(\theta) := f(\theta_0) + \nabla f(\theta_0)^\top (\theta - \theta_0) + \frac{1}{2\eta} ||\theta - \theta_0||^2$$

(Also a second-order approximation with $\nabla^2 f(\theta) \approx (1/\eta) I_{d \times d}$). The unique minimizer of $f_{\theta_0,\eta}$ is

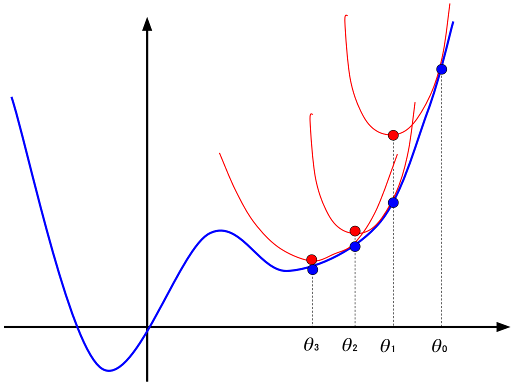
$$\theta = \theta_0 - \eta \nabla f(\theta_0)$$

## Gradient Descent

Idea: Start from some $\theta_0 \in \mathbb{R}^d$, repeatedly minimize local approx. $f_{\theta_t,\eta_t}(\theta)$ around $\theta_t$ by

$$\theta_{t+1} = \theta_t - \underbrace{\eta_t}_{\text{"step size" or "learning rate"}} \nabla f(\theta_t)$$

until $\nabla f(\theta_t) \approx 0_d$



# Properties of Gradient Descent

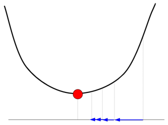
**Universal**: Can minimize *any* differentiable $f : \mathbb{R}^d \rightarrow \mathbb{R}$

- Only need the ability to calculate gradient $\nabla f : \mathbb{R}^d \rightarrow \mathbb{R}^d$

**Local search**: Only use *local* information around current location

- Initialization matters, small random values usually okay (e.g., $[\theta_0]_i \sim \mathrm{Unif}(-\alpha, \alpha)$ for $\alpha = 0.01$)
- Convergence at *some* stationary/critical point $\bar{\theta}$ (i.e., $\nabla f(\bar{\theta}) = 0_d$)
  1. Global minimum: $f(\bar{\theta}) = \min_{\theta \in \mathbb{R}^d} \; f(\theta)$
  2. Local minimum: $f(\bar{\theta} + u) \leq f(\bar{\theta})$ for all small nonzero $u \in \mathbb{R}^d$
  3. Saddle point: Not a local minimum

  (Global minimum is also local minimum.) Depends on initialization & function shape. If $f$ is *convex*, global convergence guaranteed for appropriate step sizes:



## Quick Gradient Estimation by Sampling

Often $f : \mathbb{R}^d \to \mathbb{R}$ averages "component" functions $f_i : \mathbb{R}^d \to \mathbb{R}$

$$f(\theta) = \frac{1}{N} \sum_{i=1}^{N} f_i(\theta) \tag{1}$$

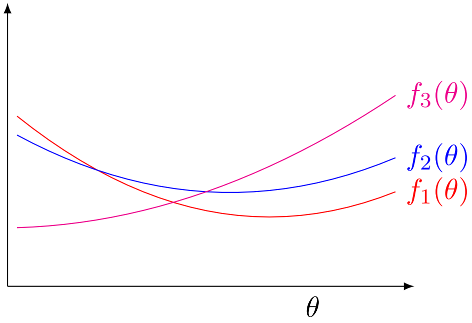
Given an equal-sized partition $\mathcal{I}_1 \ldots \mathcal{I}_K$ of $\{1 \ldots N\}$ (**mini-batches**) where $|\mathcal{I}_k| \ll N$ (**batch size**)

$$\nabla f(\theta) \approx \frac{1}{|\mathcal{I}_k|} \sum_{i \in \mathcal{I}_k} \nabla f_i(\theta) \qquad\qquad k \sim \mathrm{Unif}(\{1 \ldots K\})$$

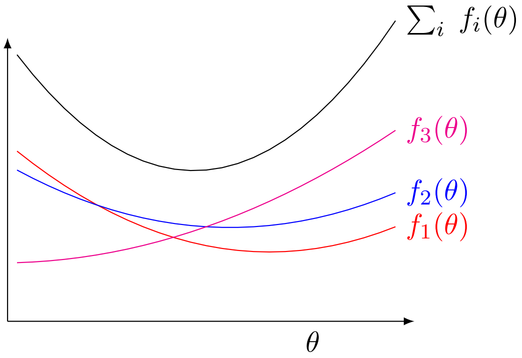
This allows us to estimate $\nabla f(\theta)$ quickly by averaging $\nabla f_i(\theta)$ in a single mini-batch, without considering all $N$ components.

- Consistent estimation (take expectation over $k$). This assumes the form (1). Not consistent for general $f$!

## Intuition

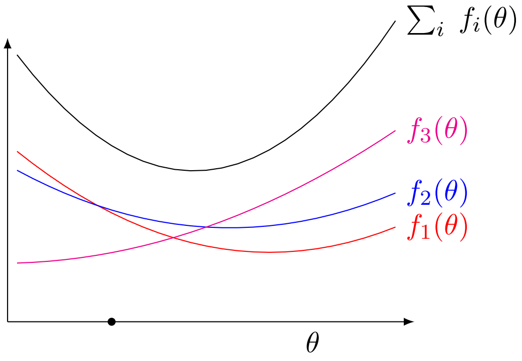


## Intuition



- Objective: $\min_\theta (1/N) \sum_{i=1}^{N} f_i(\theta)$

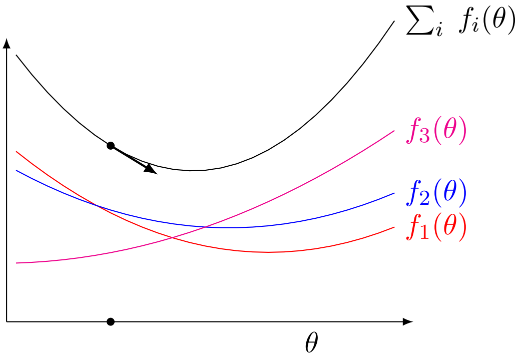
## Intuition



- Objective: $\min_\theta \, (1/N) \sum_{i=1}^{N} f_i(\theta)$
- Stochastic gradient approximation (batch size 1):

$$\frac{1}{N} \nabla f(\theta) \approx \nabla f_i(\theta)$$

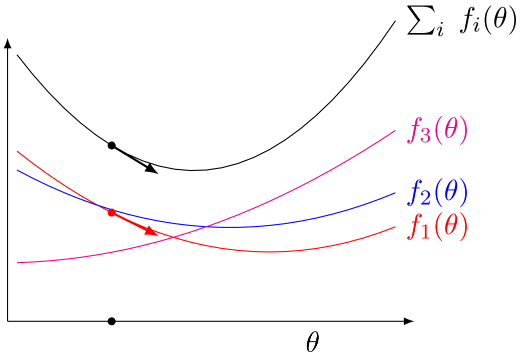
## Intuition



- Objective: $\min_\theta \, (1/N) \sum_{i=1}^N f_i(\theta)$
- Stochastic gradient approximation (batch size 1):

$$\frac{1}{N}\nabla f(\theta) \;\approx\; \nabla f_i(\theta)$$

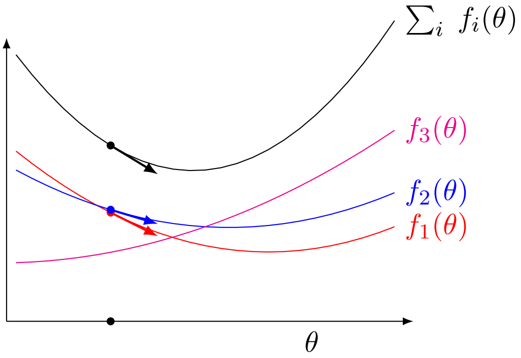
## Intuition



- Objective: $\min_\theta \; (1/N) \sum_{i=1}^{N} f_i(\theta)$
- Stochastic gradient approximation (batch size 1):

$$\frac{1}{N} \nabla f(\theta) \; \approx \; \nabla f_i(\theta)$$

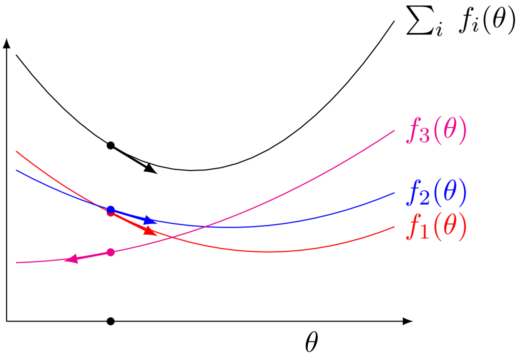
## Intuition



- Objective: $\min_\theta \, (1/N) \sum_{i=1}^{N} f_i(\theta)$
- Stochastic gradient approximation (batch size 1):

$$\frac{1}{N} \nabla f(\theta) \; \approx \; \nabla f_i(\theta)$$

## Intuition



- Objective: $\min_\theta \, (1/N) \sum_{i=1}^N f_i(\theta)$
- Stochastic gradient approximation (batch size 1):

$$\frac{1}{N} \nabla f(\theta) \; \approx \; \nabla f_i(\theta)$$

- Could be a noisy estimate

## Stochastic Gradient Descent (SGD)

- **Input**: $N$ examples $(x_i, y_i)$ defining per-example losses $\widehat{J}_i : \mathbb{R}^d \to \mathbb{R}$
- **Objective**: Average loss $\widehat{J}_N(\theta) = (1/N) \sum_{i=1}^N \widehat{J}_i(\theta)$
- **Hyperparameters**: Number of "epochs" $E$, batch size $B$, learning rate $\eta$
- **Algorithm**: Initialize $\theta \in \mathbb{R}^d$ and for $E$ epochs,
  1. Shuffle $(1 \ldots N)$ and partition into $B$-sized batches $\mathcal{I}_1 \ldots \mathcal{I}_K$ ($K \approx N/B$).
  2. For $k = 1 \ldots K$, take a gradient step

$$\theta \leftarrow \theta - \eta \left( \frac{1}{B} \sum_{i \in \mathcal{I}_k} \nabla \widehat{J}_i(\theta) \right)$$

- **Memory footprint**: $O(d)$ to store $\theta$ and gradients

## Gradient for Linear Classifiers

Recall: training data $(x_i, y_i) \in \mathbb{R}^V \times \{1 \ldots L\}$, loss

$$\widehat{J}_N(W, b) = \frac{1}{N} \sum_{i=1}^{N} \log \left( \sum_{y=1}^{L} \exp(w_y^\top x_i + b_y) \right) - w_{y_i}^\top x_i - b_{y_i}$$

Can show $\widehat{J}_N$ is convex, so SGD will converge to a global minimum. Exercise: derive the gradients

$$\nabla_{w_y} \widehat{J}_N(W, b) = \frac{1}{N} \sum_{i=1}^{N} \left( p_{W,b}(y|x_i) - \underbrace{[[y = y_i]]}_{\text{1 if true, 0 else}} \right) x_i$$

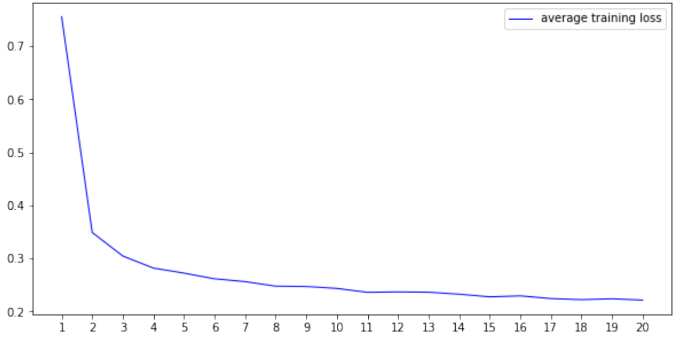
$$\nabla_{b_y} \widehat{J}_N(W, b) = \frac{1}{N} \sum_{i=1}^{N} p_{W,b}(y|x_i) - [[y = y_i]]$$

Intuitive: adjust the difference between model prediction and ground truth

## Rule 1: Always Monitor Training Loss

$\widehat{J}_N(\theta)$ should (almost certainly) strictly decrease each epoch: this is what we are optimizing!

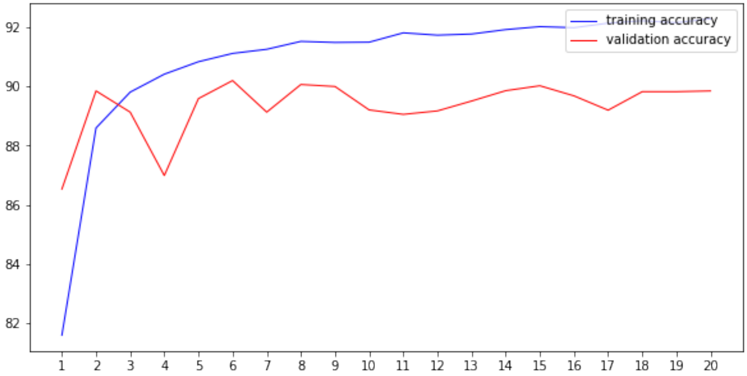
- Easy to track: accumulate losses over batches and divide by $N$



## Rule 2: Always Monitor Validation Performance

Check validation performance (at least) every epoch and do early stopping to prevent overfitting.

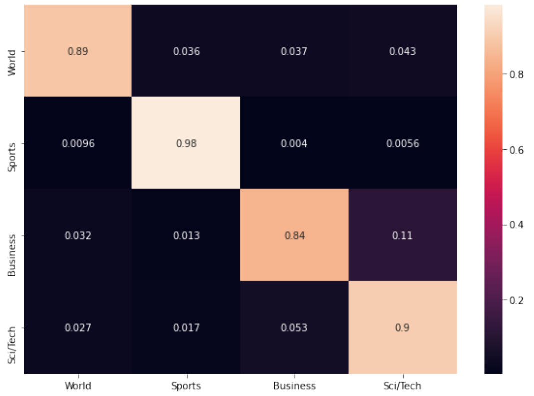
- Performance for topic classification is simple accuracy (# correct / # test examples), but it can be more complicated for other tasks (structured prediction?)



## Rule 3: Look at Errors

Once performance seems okay by quantitative metrics, do some *qualitative* analysis of errors to get an actual understanding of challenges in the task and how to improve

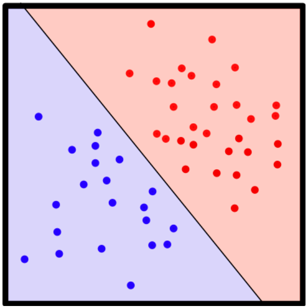
- Confusion matrix works for simple classification, but may need to be creative to analyze complex problems (translation?)



| | World | Sports | Business | Sci/Tech |
|---|---|---|---|---|
| World | 0.89 | 0.036 | 0.037 | 0.043 |
| Sports | 0.0096 | 0.98 | 0.004 | 0.0056 |
| Business | 0.032 | 0.013 | 0.84 | 0.11 |
| Sci/Tech | 0.027 | 0.017 | 0.053 | 0.9 |

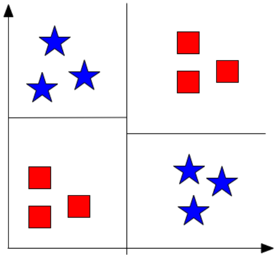
## Limitations of a Linear Classifier

Linearly separable



Accuracy $100\%$ ✓

Not linearly separable (e.g., XOR)



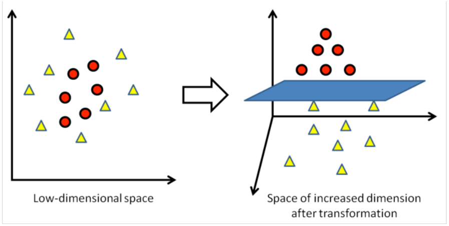
Accuracy $\leq 50\%$ ✗

Solutions

1. **Feature engineering:** Specify better input representation
2. **Feature learning:** Representation $=$ part of model to learn

## Feature Engineering

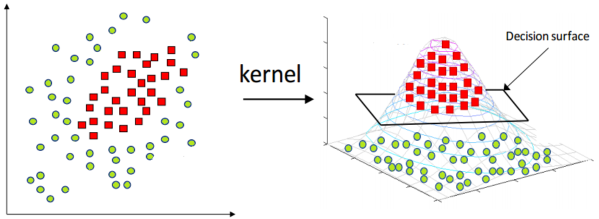
- You can always add new dimensions until data is separable



- Much of past NLP research spent in feature engineering. For instance, for topic classification consider
  - Higher-order features: bag-of-$n$-grams $x \in \{0,1\}^{V^n}$?
  - Side information (e.g., author identity? date? length?)
- Pros: Requires a deep understanding of the specific problem, interpretable weights, can work well even with small data by specifying right features
- Con: Have to do this for every new problem, no way to know how much engineering is enough

## Aside: Kernel Trick

Technique to *implicitly* enrich input representation, applicable whenever model/learning involves only dot product between inputs (e.g., SVMs)



No need to manually engineer good features, but has other cons (not easily scalable to large data, still have to choose the kernel)

- Largely out of the scope of this course
- Active research on connections between kernel machines and deep learning

## Feature Learning

- This is “deep learning”.
- Parameterizes an **encoder** $\textbf{enc}_\theta : \mathcal{X} \to \mathbb{R}^H$, which computes the input representation
- Only score changes: score of $(x, y) \in \mathcal{X} \times \{1 \dots L\}$ now

  $$\textbf{score}_\theta(x, y) := w_y^\top \textbf{enc}_\theta(x) + b_y$$

  $\theta$ now includes all parameters associated with the encoder, as well as the linear classifier parameters $(W = [w_1 \dots w_L], b)$
- Conditional label distribution defined the same way

  $$p_\theta(y|x) = \frac{\exp(\textbf{score}_\theta(x, y))}{\sum_{y'=1}^{L} \exp(\textbf{score}_\theta(x, y'))} \qquad \forall y = 1 \dots L$$

- Same training scheme: gradient descent on the cross-entropy loss

## Example: One Hidden Layer Feedforward Network

- Parameters: $\theta \in \mathbb{R}^{100V+20200+201L}$ referring to
  - **Embedding matrix**: $E = [e_1 \ldots e_V] \in \mathbb{R}^{100 \times V}$, $e_i \in \mathbb{R}^{100}$ is a dense, 100-dimensional vector representation of word $i \in \mathcal{V}$
  - **Hidden layer**: $U \in \mathbb{R}^{100 \times 200}$ and $a \in \mathbb{R}^{200}$
  - **Linear layer**: $W = [w_1 \ldots w_L] \in \mathbb{R}^{200 \times L}$ and $b \in \mathbb{R}^L$
- Encoder: given an initial BOW representation $x \in \{0,1\}^V$ of a document, let $\mathbf{avg}_E(x) := (1/|x|) \sum_{i:x_i=1} e_i$ and compute

$$\mathbf{enc}_\theta(x) = \max \left\{ 0,\ \underbrace{U^\top}_{200 \times 100} \underbrace{\mathbf{avg}_E(x)}_{100 \times 1} + \underbrace{a}_{200 \times 1} \right\} \in \mathbb{R}^{200}$$

  where $\max\{0, v\}$ is elementwise.
- $\mathbf{score}_\theta(x,y) = w_y^\top \mathbf{enc}_\theta(x) + b_y$
- $p_\theta(y|x) \propto \exp(\mathbf{score}_\theta(x,y))$

## Training

Use SGD to optimize

$$\min_{\theta \in \mathbb{R}^d} \; -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp(\textbf{score}_\theta(x_i, y_i))}{\sum_{y=1}^{L} \exp(\textbf{score}_\theta(x_i, y))}$$

Recall: $\theta$ denotes parameters of the encoder as well as $(W, b)$

Important questions

- How should we define the encoder? Does it matter?
- How can we calculate gradients?
- How can we make training efficient with so many parameters?