CS 4248
Natural Language Processing

Professor NG Hwee Tou

Department of Computer Science
School of Computing
National University of Singapore
nght@comp.nus.edu.sg

Acknowledgment

- Materials from:
  - Neural Network Methods for Natural Language Processing, Yoav Goldberg, Synthesis Lectures on Human Language Technologies, March 2017.
  - Abbreviated as NNM4NLP
- Credit: Some figures in my slides extracted from NNM4NLP
- NNM4NLP Chapter 1, 2

Natural Language Processing

- Using supervised machine learning algorithms to infer usage patterns and regularities from a set of pre-annotated input and output pairs
- Example: spelling error correction, text classification, part-of-speech tagging

Neural Networks and Deep Learning

- Learning of parameterized differentiable mathematical functions
- Deep learning: Many layers of these differentiable functions are chained together
- Learning representations to appropriately represent the input data

Deep Learning in NLP

- Embedding layer: Mapping of discrete symbols to continuous vectors in a low dimensional space
- Distance between words = distance between vectors
- Representation of words as vectors is learned by the neural network during training
- Addresses discreteness and data sparsity

Neural Networks

- Simplify feature engineering
- Model designer specifies a small set of core, basic, “natural” features
- NN combines them into more meaningful higher-level features, or representations

Success of Neural Networks

- Non-linearity and use of pre-trained word embeddings often lead to better classification accuracy of neural networks

Linear Models

$$\boldsymbol{y} = f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}$$

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}} \quad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}} \quad \boldsymbol{b} \in \mathbb{R}^{d_{out}}$$

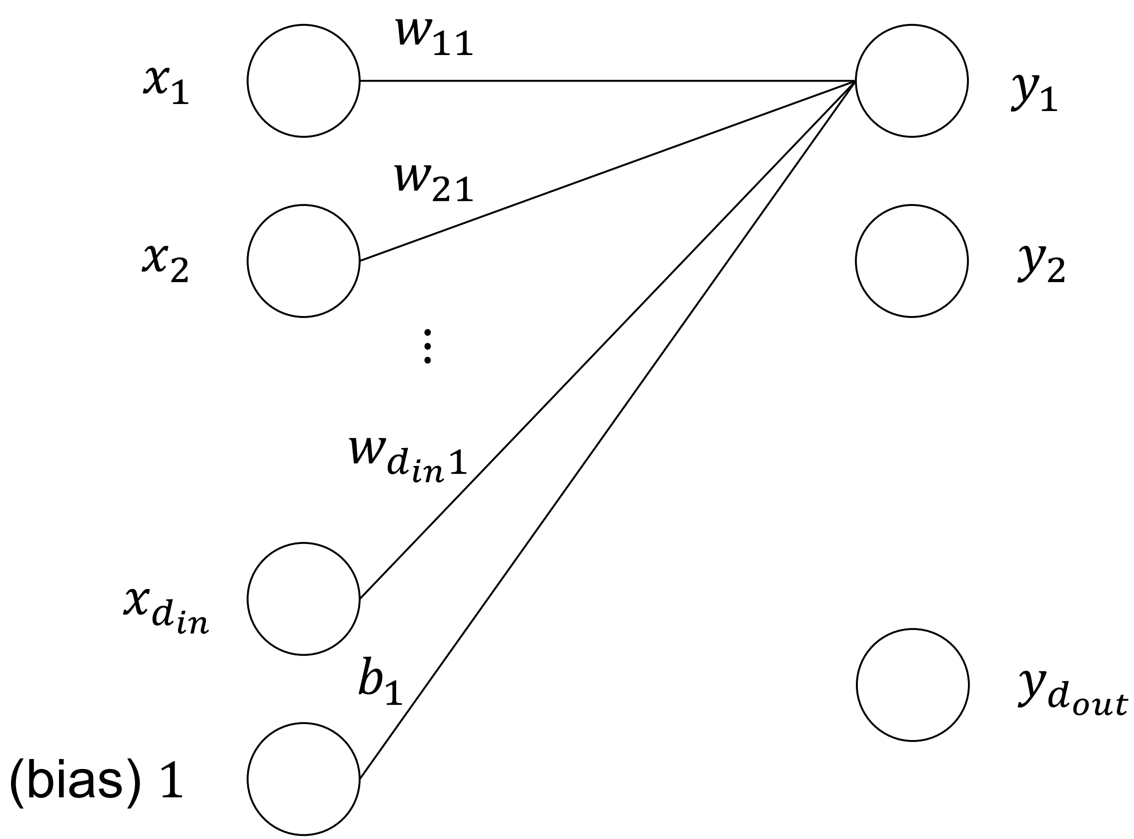
$\boldsymbol{x}$: input (row vector), $\boldsymbol{W}, \boldsymbol{b}$: parameters

$$[y_1 \quad y_2 \quad \ldots \quad y_{d_{out}}]$$

$$= [x_1 \quad x_2 \quad \ldots \quad x_{d_{in}} \quad 1] \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d_{out}} \\ w_{21} & w_{22} & \cdots & w_{2d_{out}} \\ \cdots & \cdots & \cdots & \cdots \\ w_{d_{in}1} & w_{d_{in}2} & \cdots & w_{d_{in}d_{out}} \\ b_1 & b_2 & \cdots & b_{d_{out}} \end{bmatrix}$$

Linear Models

$$x_1 \cdot w_{11} + x_2 \cdot w_{21} + \cdots + x_{d_{in}} \cdot w_{d_{in}1} + 1 \cdot b_1 = y_1$$



Binary Classification

$$f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}$$

$$d_{out} = 1$$

$$sign(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

$$\hat{y} = sign\big(f(\boldsymbol{x})\big) = sign(\boldsymbol{x} \cdot \boldsymbol{W} + b)$$

Feature Extraction

- Maps a real-world object (document, sentence, word, etc.) to a vector of measurable quantities
- Informative features
- Design of the feature function (feature engineering)

NLP Task: Language Identification

Input: A document D

Output: The language in which the document is written (English or German)

$\boldsymbol{x}$: vector of normalized counts of character bigrams

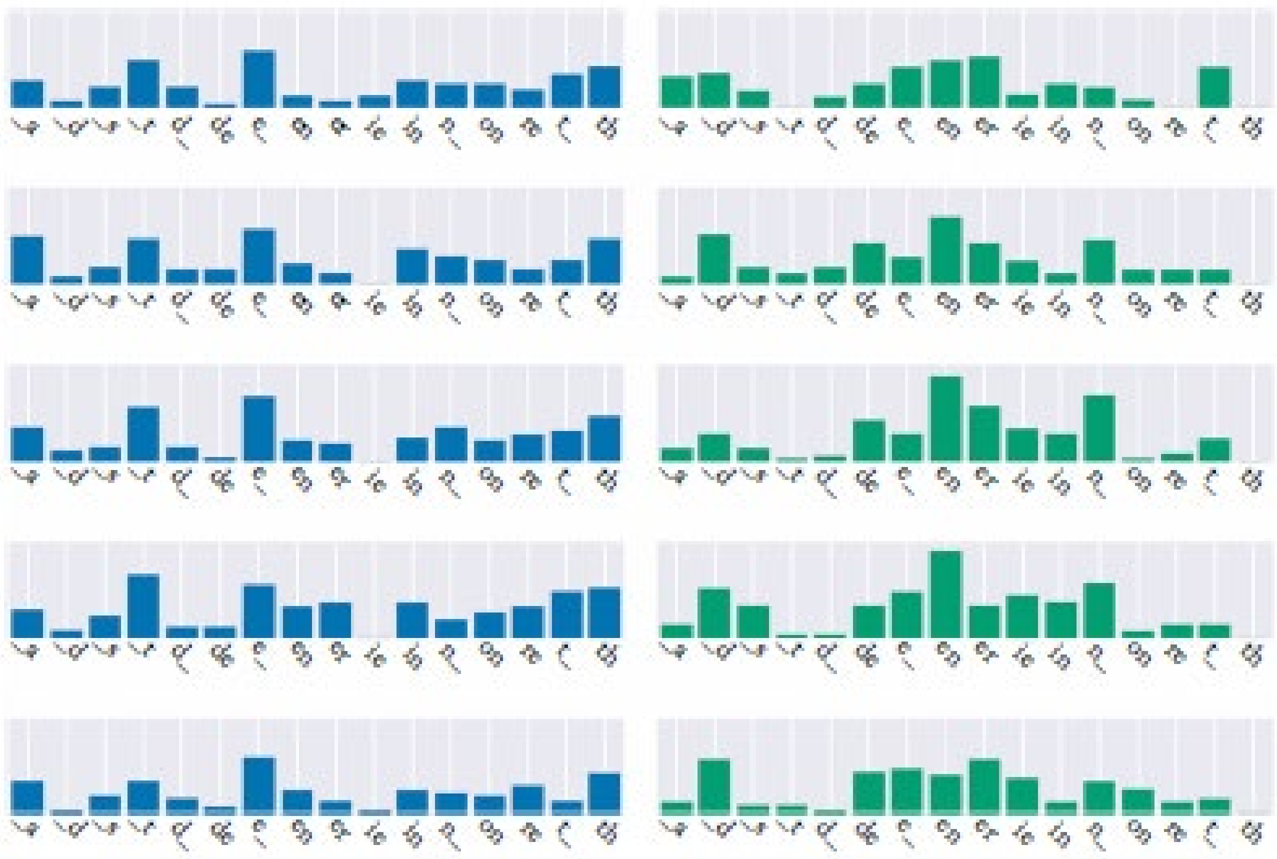
28 characters (a, b, …, z, _, other-char)

$$x_{ab} = \frac{\#_{ab}}{|D|}$$

$\boldsymbol{x} \in \mathbb{R}^{784}$ (28 x 28 = 784)

$$\hat{y} = sign(x_{aa} \cdot w_{aa} + x_{ab} \cdot w_{ab} + \cdots + b)$$

Language Identification



Character bigrams
for English documents

Character bigrams
for German documents

Log-linear Binary Classification

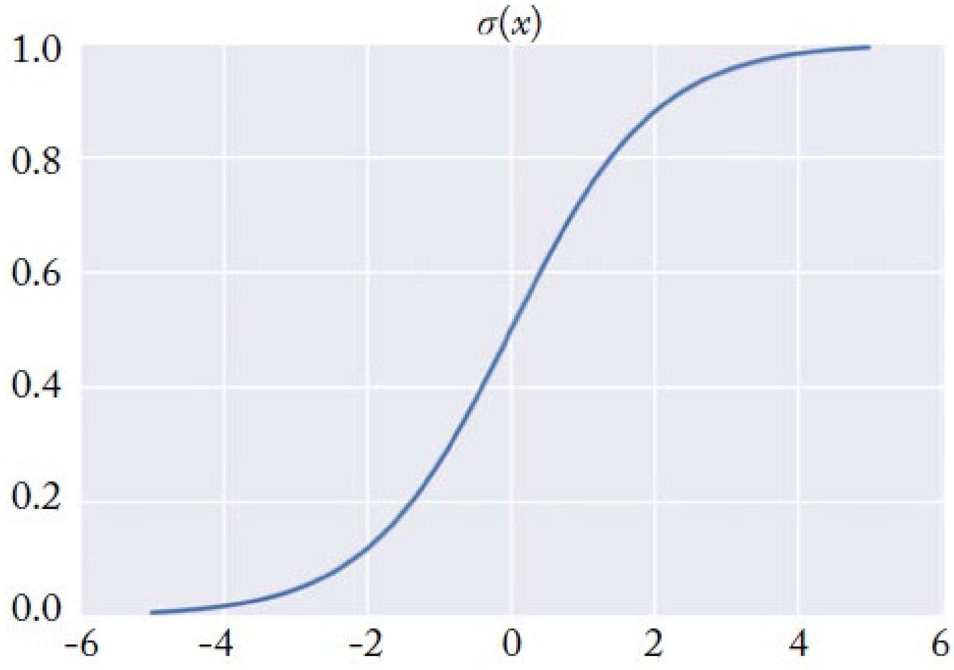
Probability that a classifier assigns to a class

Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

$$P(\hat{y} = 1|\boldsymbol{x}) = \sigma\big(f(\boldsymbol{x})\big) = \sigma(\boldsymbol{x} \cdot \boldsymbol{W} + b\ ) = \frac{1}{1+e^{-(\boldsymbol{x} \cdot \boldsymbol{W} + b)}}$$

$$P(\hat{y} = 0|\boldsymbol{x}) = 1 - P(\hat{y} = 1|\boldsymbol{x}) = 1 - \sigma\big(f(\boldsymbol{x})\big)$$

Sigmoid Function (Logistic Function)



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

Multi-class Classification

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}$$

$$\text{prediction} = \hat{y} = \underset{i}{\operatorname{argmax}} \, \hat{\boldsymbol{y}}_{[i]}$$

E.g., classify into 6 possible languages: English, French, German, Italian, Spanish, Other

$\boldsymbol{W} \in \mathbb{R}^{784 \times 6}$

Log-linear Multi-class Classification

$$\text{softmax}(\boldsymbol{x})_{[i]} = \frac{e^{\boldsymbol{x}_{[i]}}}{\sum_j e^{\boldsymbol{x}_{[j]}}}$$

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b})$$

$$\hat{\boldsymbol{y}}_{[\boldsymbol{i}]} = \frac{e^{(\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b})_{[i]}}}{\sum_j e^{(\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b})_{[j]}}}$$

softmax transforms the values in $\hat{\boldsymbol{y}}$ to be positive and sum to 1, making them interpretable as a probability distribution

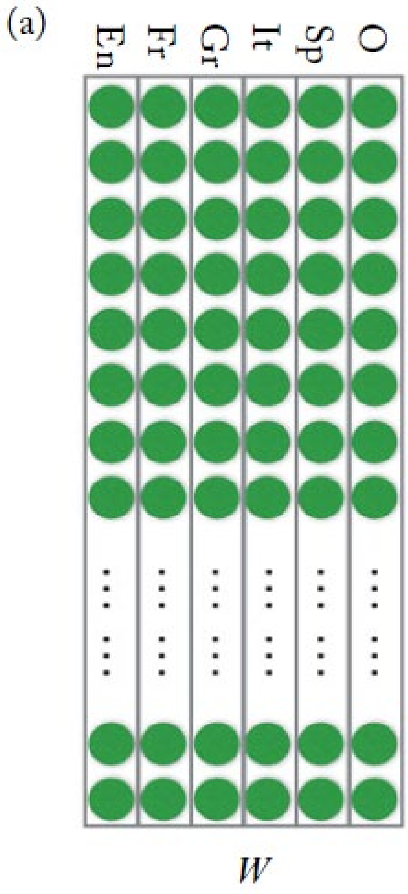
Linear Models

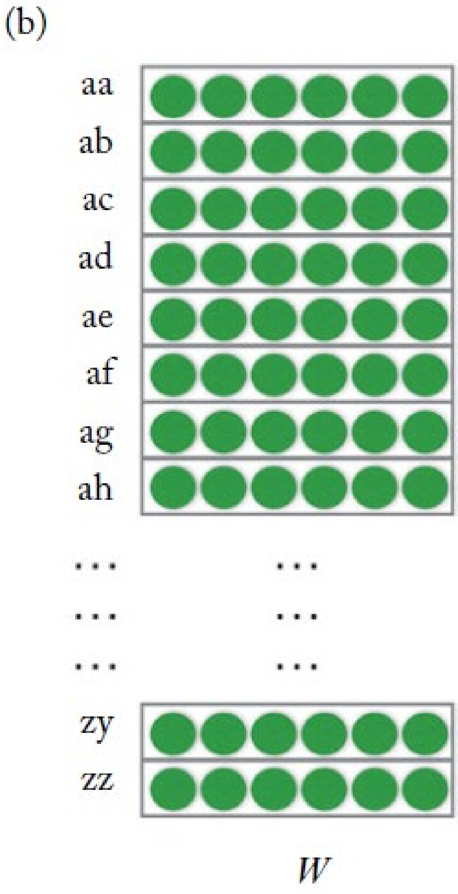
- Easy and efficient to train
- Trained models are interpretable
- Often effective in practice

Representations

- $\boldsymbol{x}$: representation of a document, in terms of the normalized character bigram counts of the document
- $\widehat{\boldsymbol{y}}$: representation of a document, in terms of the scores of the different languages
- $\widehat{\boldsymbol{y}}$: more compact (6 entries instead of 784) and more specialized for the language prediction objective

Two Views of the W matrix





Representations

- $\boldsymbol{W} \in \mathbb{R}^{784 \times 6}$: learned representations
- Two views of $\boldsymbol{W}$:
  - Each column of $\boldsymbol{W}$: a 784-dimensional vector representation of the language in terms of its characteristic character bigram patterns
  - Each row of $\boldsymbol{W}$: a 6-dimensional vector representation of the character bigram in terms of the languages that it prompts

Representations

- Representations are central to deep learning
- The main power of deep learning is the ability to learn good representations
- But learned representations are often not interpretable in multi-layer NNs
- At the boundaries of the model (input and output), we get vector representations that correspond to the input and the output

Training as Optimization

A training set of n training examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
with corresponding labels $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$

Goal of learning: Find a function f
such that the predictions $\widehat{\boldsymbol{y}} = f(\boldsymbol{x})$
over the training set are accurate

Loss Functions

- A loss function $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ maps two vectors, predicted label $\hat{\boldsymbol{y}}$ and true label $\boldsymbol{y}$, to a scalar quantifying the loss suffered due to the inaccurate prediction

Training as Optimization

Training: Find parameters Θ that minimize the sum of the loss function and regularization term:

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\Big(\underbrace{\frac{1}{n}\sum_{i=1}^{n} L(f(\boldsymbol{x}_i;\Theta),\boldsymbol{y}_i)}_{\text{loss}} + \underbrace{\lambda R(\Theta)}_{\text{regularization}}\Big)$$

Regularization: control the complexity of the parameter values and avoid overfitting

λ: control the amount of regularization. A hyperparameter set manually based on development set

Loss Functions

- Binary cross-entropy loss (logistic loss)
  - Used in binary classification with conditional probability output
  - $y \in \{0,1\}$
  - $0 < \hat{y} < 1$ $(\hat{y} = \sigma(\tilde{y}) = P(y = 1|\boldsymbol{x}))$

$$L_{\text{logistic}}(\hat{y}, y) = -y\ \log_2 \hat{y} - (1 - y)\ \log_2(1 - \hat{y})$$

Maximize the log conditional probability $\log_2 P(y|\boldsymbol{x})$ for each training example $(\boldsymbol{x}, y)$

Loss Functions

- Categorical cross-entropy loss (negative log likelihood)
- $\boldsymbol{y} = \boldsymbol{y}_{[1]}, \dots, \boldsymbol{y}_{[n]}$: a vector representing the true multinomial distribution over the labels $1, \dots, n$
- $\widehat{\boldsymbol{y}} = \widehat{\boldsymbol{y}}_{[1]}, \dots, \widehat{\boldsymbol{y}}_{[n]}$: the classifier's output, transformed by softmax
- $\widehat{\boldsymbol{y}}_{[i]} = P(y = i|\boldsymbol{x})$
- $L_{\text{cross-entropy}}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_i \boldsymbol{y}_{[i]} \log_2(\widehat{\boldsymbol{y}}_{[i]})$
- For hard classification problem with a single correct class t: $L_{\text{cross-entropy}}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = -\log_2(\widehat{\boldsymbol{y}}_{[t]})$

Loss Functions

- Ranking loss (margin-based)
  - We have only positive training examples, and generate negative training examples by corrupting positive training examples
  - Given a pair of positive training example $\boldsymbol{x}$ and negative training example $\boldsymbol{x}'$
- Ranking loss (margin-based)
  - Aim: $f(\boldsymbol{x}) - f(\boldsymbol{x}') > 1$

$$L_{\text{ranking(margin)}}(\boldsymbol{x}, \boldsymbol{x}') = \max(0,\ 1 - (f(\boldsymbol{x}) - f(\boldsymbol{x}')))$$

Loss Functions

- Squared (quadratic) loss

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^2$$

Regularization

- In practice, R equates complexity with large weights
- L_2 regularization
  - $R_{L_2}(\boldsymbol{W}) = ||\boldsymbol{W}||_2^2 = \sum_{i,j}(\boldsymbol{W}_{[i,j]})^2$
- L_1 regularization
  - $R_{L_1}(\boldsymbol{W}) = ||\boldsymbol{W}||_1 = \sum_{i,j} |\boldsymbol{W}_{[i,j]}|$
- Elastic-Net: combines both L_1 and L_2 regularization
  - $R_{\text{elastic}-\text{net}}(\boldsymbol{W}) = \lambda_1 R_{L_1}(\boldsymbol{W}) + \lambda_2 R_{L_2}(\boldsymbol{W})$

Gradient Descent

- A generic numerical optimization method
- Use a series of successive approximations

$$L(w_0, w_1, \dots, w_{m-1}) = L(\boldsymbol{w})$$

$$L(\boldsymbol{w_1}) > L(\boldsymbol{w_2}) > \cdots > \min$$

Given $\boldsymbol{w}$, find $\boldsymbol{v}$ such that $L(\boldsymbol{w}) > L(\boldsymbol{w} + \boldsymbol{v})$ and $||\boldsymbol{v}||$ is small

Gradient Descent

Since $||\boldsymbol{v}||$ is small, we can approximate $L(\boldsymbol{w}+\boldsymbol{v})$ using a Taylor series expansion:

$$L(\boldsymbol{w}+\boldsymbol{v}) \approx L(\boldsymbol{w}) + \boldsymbol{v} \cdot \nabla L(\boldsymbol{w})$$

$$\nabla L(w_0, w_1, \ldots, w_{m-1}) = (\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \cdots, \frac{\partial L}{\partial w_{m-1}})$$

is a direction vector corresponding to the steepest slope

Gradient Descent

We obtain the steepest descent when we choose $\boldsymbol{v}$ collinear to $\nabla L(\boldsymbol{w})$:

$$\boldsymbol{v} = -\alpha \nabla L(\boldsymbol{w}) \qquad \alpha > 0$$

$$L(\boldsymbol{w} - \alpha \nabla L(\boldsymbol{w})) \approx L(\boldsymbol{w}) - \alpha \|\nabla L(\boldsymbol{w})\|^2$$

Gradient Descent

Hence,

$L(\boldsymbol{w}) > L(\boldsymbol{w} - \alpha \nabla L(\boldsymbol{w}))$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha_t \nabla L(\boldsymbol{w}_t)$$

α_t is the learning rate or step size (a small positive number)

Iteration stops when $||\nabla L(\boldsymbol{w})||$ is less than a predefined threshold

Faster convergence if α_t decreases over the iterations

Gradient Descent

- A convex function (second derivative is always non-negative) has a single minimum point.
- If $L(\boldsymbol{w})$ is a convex function, then gradient descent will converge to the global minimum. Otherwise, gradient descent only finds a local minimum in general.

Gradient-Based Optimization

Algorithm 2.1 Online stochastic gradient descent training.

Input:
- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \dots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \dots, \boldsymbol{y_n}$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$
3: Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
4: $\hat{\boldsymbol{g}} \leftarrow$ gradients of $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: **return** Θ

Gradient-Based Optimization

Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:
- Function $f(x;\Theta)$ parameterized with parameters Θ.
- Training set of inputs $x_1, \ldots, x_n$ and desired outputs $y_1, \ldots, y_n$.
- Loss function L.

while stopping criteria not met **do**
    Sample a minibatch of m examples $\{(x_1, y_1), \ldots, (x_m, y_m)\}$
    $\hat{g} \leftarrow 0$
    for $i = 1$ to m **do**
        Compute the loss $L(f(x_i;\Theta), y_i)$
        $\hat{g} \leftarrow \hat{g} +$ gradients of $\frac{1}{m}L(f(x_i;\Theta), y_i)$ w.r.t Θ
    $\Theta \leftarrow \Theta - \eta_t \hat{g}$
return Θ

A higher value of minibatch size *m* provides better estimate of the corpus-wide gradient, while a smaller value allows more updates and faster convergence.