Machine Learning Methods for Text Classification

Bonan Min

bonanmin@gmail.com

Some slides are based on class materials from Thien Huu Nguyen, Dan Jurafsky, James H. Martin, Dhruv Batra, Fei-Fei Li, Justin Johnson

Recap on Text Classification

An example: opinion mining / sentiment analysis

- Classify whether a document expresses a positive or negative opinion (or no opinion) about an object or topic

Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

Different Types of Text Classification

Topics (e.g., politics, sports, science): by far the most frequent case, its applications are ubiquitous

Sentiment (e.g., positive, negative, neutral): useful in market research, online reputation management, customer relationship management, social sciences, political sciences

Languages (i.e., language identification): useful in query processing with search engines

Authors (i.e., authorship attribution): useful in forensics and cybersecurity

Machine Learning (ML) for Text Classification

ML classifiers

- A generic (task-independent) learning algorithm to train a classifier from a set of labeled examples
- The classifier learns, from these labeled examples, the characteristics of a new text should have in order to be assign to some label

Advantages

- Annotating/locating training examples is cheaper than writing rules
- Easier updates to changing conditions (annotate more data with new labels for new domains)

(This lecture) Focus on Bag-of-Words models to introduce ML approaches

- Also introduce a minimum amount of shallow features

ML for Text Classification

A sequence of words $X = \{w_1, \cdots, w_n\}$

A label set y $\in Y = \{c_1, \ldots, c_k\}$

Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

Training examples are pairs of input text and the corresponding labels: $D = \{(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\}$

Disjoint datasets used:

- Training: estimate model parameters
- Development: choose the best hyperparameters; prevent overfitting in training
- Test dataset: report model performance

ML for Text Classification

From the training dataset $D = \{(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\}$, learn a model/classifier/function that can predict the label for a new input text # (the classification problem):

$$f: X \rightarrow y \in Y$$

In a probabilistic formulation, this is done by computing the probability distribution over the possible classes in *Y* given the input document *X*:

$$P(y|X)$$

Then the label for a new document *X* is

$$\hat{y} = argmax_y P(y|X)$$

ML for Text Classification

ML models covered in this lecture

- Naïve Bayes classifiers
- Multi-class logistic regression
- Multi-layer perceptron
- Convolutional Neural Networks

Naive Bayes Classification (Recap)

Identify most likely class

$$s = \underset{t \in \{pos, neg\}}{\operatorname{argmax}} P(t \mid W)$$

Use Bayes' rule

$$\begin{aligned}
\operatorname{argmax}_t P(t|W) &= \frac{\operatorname{argmax}_t P(W|t)P(t)}{P(W)} \\
&= \operatorname{argmax}_t P(W|t)P(t) \\
&= \operatorname{argmax}_t P(w_1,\ldots,w_n|t)P(t) \\
&= \operatorname{argmax}_t \prod_i P(w_i|t)P(t)
\end{aligned}$$

Based on the naïve assumption of independence of the word probabilities

Training (Recap)

Estimate probabilities from the training corpus (N documents) using *maximum likelihood estimators*

$$P(t) = \text{count (docs labeled t)} / N$$

$$P(w_i \mid t) = \frac{\text{count (docs labeled t containing } w_i \text{)}}{\text{count (docs labeled t)}}$$

Text Classification: Flavors (Recap)

Bernoulli model: use presence (/ absence) of a term in a document as feature

- *formulas on previous slide*

Multinomial model: based on frequency of terms in documents:

- $P(t) = \frac{\text{total length of docs labeled t}}{\text{total size of corpus}}$

- $P(w_i \mid t) = \frac{\text{count (instances of } w_i \text{ in docs labeled t)}}{\text{total length of docs labeled t}}$

Better performance on long documents

The Importance of Smoothing (Recap)

Suppose a glowing review SLP2 (with lots of positive words) includes one word, “mathematical”, previously seen only in negative reviews

$$P(\text{positive} \mid \text{SLP2}) = 0$$

because $P(\text{“mathematical”} \mid \text{positive}) = 0$

The maximum likelihood estimate is poor when there is very little data

We need to ‘smooth’ the probabilities to avoid this problem

Add-One (Laplace) Smoothing (Recap)

A simple remedy is to add 1 to each count

- for the conditional probabilities P(w | t): Add 1 to each c(w, t)
- Increase the denominator by number of unique words (|V|). That is, add |V| to c(t) to keep them as probabilities (sum up to 1)

$$\sum_{w \in V} p(w|t) = 1$$

Features for Text Classification

Recall that a document is represented as a sequence of words $X = \{w_1, \cdots, w_n\}$

More features can be included

- Words in the title
- Author, length, date of document
- Sender, recipient of email
- Noun phrases or n-grams
- Number of punctuation marks
- …

Choices of features depend on the task

- Sentiment: punctuation marks are useful e.g., *?, !, !!!*
- Topic: titles are more important than the body

Features for Text Classification

Feature representation: Given a document/text, we extract the features, then represent them in a vector

The dog chased the cat.

Feature engineering

Features	the	dog	chased	mouse	cat	length	appCap	authorIsTom	authorisThien
Values	2	1	1	0	1	6	0	0	1

Normalization is helpful

- E.g., subtracting the mean from an individual raw score and then dividing the difference by the standard deviation

Can involve conjunction features, e.g., n-grams, combination of a feature and a label, to emphasize the co-occurrence of the features (thus highly interdependent)

Feature engineering usually requires linguistic intuition and analysis

Features for Text Classification

Naïve Bayes can be extended to include these features

However, the more features we include, the more likely they have dependencies with each other (violating the independency assumption of Naïve Bayes)

- We need methods that can handle the inter-dependency between features, thus allowing us to introduce as many features as we see fit to reflect our intuition about the problem

Maximum Entropy

Also known as **Multi-Class Logistic Regression*** or **MaxEnt**; it is a discriminative classification model:

$$\hat{y} = argmax_y \mathrm{P(y|x)}$$

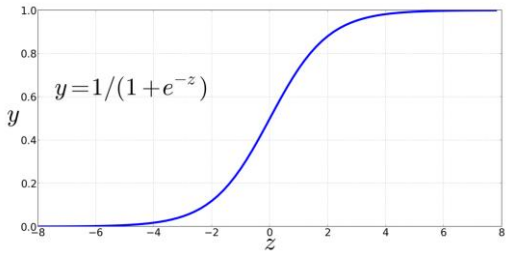
Logistic regression (binary):

- Pass z through the sigmoid/logistic function $\sigma(z)$
- Parameters : weight vector $\boldsymbol{w} \in \boldsymbol{R}^n$ and the bias vector $\boldsymbol{b} \in \boldsymbol{R}$

$$P(y = 1|x) = \sigma(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}) = \frac{1}{1 + \mathrm{e}^{-(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b})}}$$

Decision boundary is 0.5:

$$\hat{y} = \begin{cases} 1, \text{if } P(y = 1|x) > 0.5 \\ 0, \qquad \text{otherwise} \end{cases}$$



* On the equivalence of MaxEnt and multi-class logistic regression: http://www.win-vector.com/dfiles/LogisticRegressionMaxEnt.pdf

Maximum Entropy

Parameters : weight vector $\boldsymbol{w} \in \boldsymbol{R}^{n \times c}$ and the bias vector $\boldsymbol{b} \in \boldsymbol{R}^{c}$

- $\boldsymbol{w_{ij}}$ corresponds to the weight for *i*-th feature with regard to the j-th label

The likelihood vector for the types of *Y* is computed as

$$A = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b} = [a_1, a_2, \dots, a_c]$$

The likelihood vector is then normalized using the softmax function (a generalization of the sigmoid) to obtain a probability distribution over the classes:

$$softmax(A) = [\frac{e^{a_1}}{e^z}, \frac{e^{a_2}}{e^z}, \dots, \frac{e^{a_c}}{e^z}]$$

$$Z = \sum_{i=1}^{c} e^{a_i}$$

Loss Function: Cross-Entropy

Loss function: expresses how close the classifier output $\hat{y}$ is to the correct output y for a x

- Minimize the loss function = maximize the fit

Cross entropy loss

- Prefers the correct class labels of the training examples to be more likely
- This is called conditional maximum likelihood estimation: choose the parameters *w,b* that maximize the log probability of the true *y* labels in the training data given the observations *x*.
- The resulting loss function is the negative log likelihood loss, generally called the cross-entropy loss

Assuming *C* classes, the cross-entropy loss is the following:

$$L(\theta = \boldsymbol{w}, \boldsymbol{b}) = -\sum_{i=1}^{c} 1\{y = i\} \log P(y = i|x) = -\sum_{i=1}^{c} 1\{y = i\} \log \sum_{i=1}^{c} \frac{e^{a_i}}{e^{z}}$$

For a single label $y = i$, this is $L = -\log(\frac{e^{a_i}}{e^{z}})$

Training: Gradient Descent to Minimize the Loss

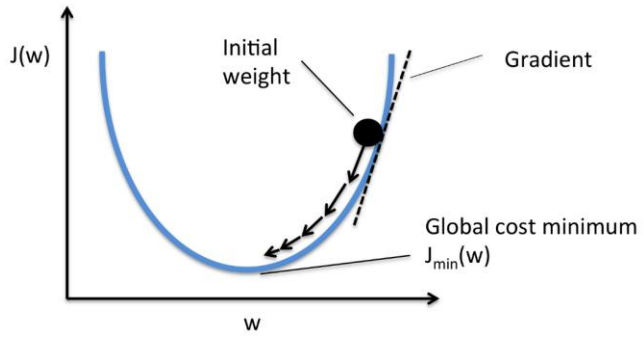
In 1-dimension, the derivative of a function

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

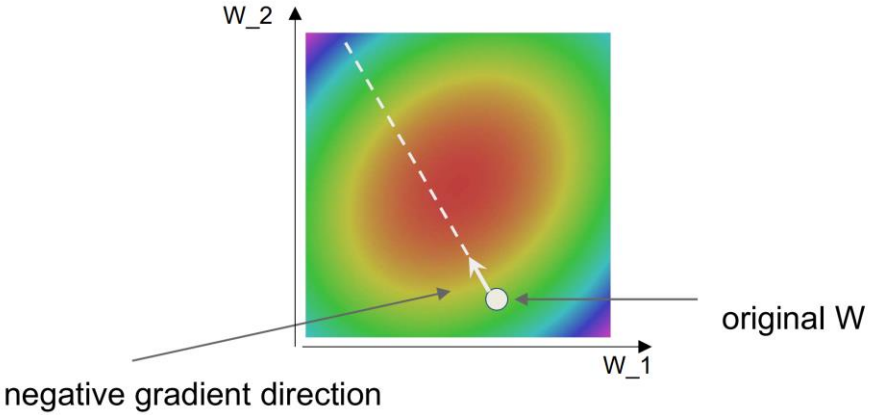
In multiple dimensions, the gradient is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient

The direction of **steepest descent** is the **negative gradient**



https://hackernoon.com/gradient-descent-aynk-7cbe95a778da



Gradient Descent

For a loss function $\mathrm{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, \theta)$

At step *t+1*, Model parameters can be updated with the following rule:

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta L(\theta_t)$$

In which

θ_t : model parameters at time t (θ_0 is the initial parameter)

γ : learning rate

Both θ_0 and γ are critical for convergence

Stochastic Gradient Descent (SGD)

Problem: Full sum is expensive when N is large; also the complexity grows linearly with N (size of the dataset)

$$\nabla_\theta L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta L_i(x_i, y_i; \theta)$$

Approximate sum using a minibatch of examples

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta L\left(x_i^t, y_i^t; \theta_t\right)$$

- Per-iteration complexity is independent of N
- The stochastic process $\{\theta_t | t = 1, \dots, \}$ depends on the examples (x_i^t, y_i^t) picked randomly at each iteration

Advanced Optimizers

Momentum

Adagrad

Adadelta

Adam

RMSprop

…

Some are adaptive learning rate methods (e.g., by leveraging second-order derivatives)

Most are built-in in Deep Learning (DL) toolkits, e.g., TensorFlow, PyTorch

- Also, state-of-the-art DL toolkits supports Automatic differentiation

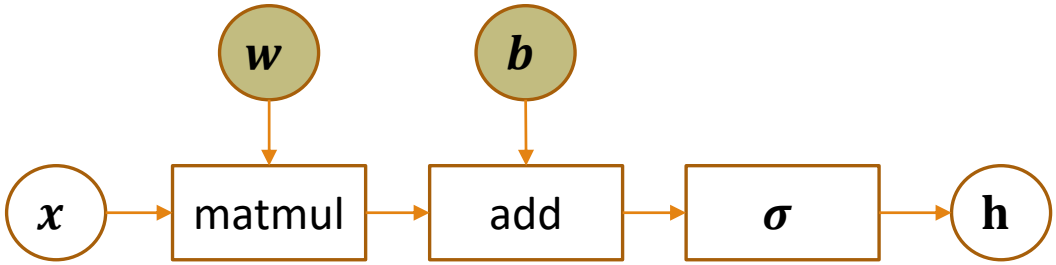
Layers in a Multi-Layer Perceptron (MLP) / Neural Networks

So far we have only considered the logistic unit: $\mathrm{h} = \sigma\left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b\right)$ where $\mathrm{h} \in R, x \in R^p, w \in R^p$ and $b \in R$

Units can be composed **in parallel** to form a layer with *q* outputs:

$$\mathbf{h} = \boldsymbol{\sigma}\left(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + \boldsymbol{b}\right)$$

where $\mathrm{h} \in R^q, x \in R^p, w \in R^{p \times q}, b \in R^q$, and $\boldsymbol{\sigma}(\cdot)$ performs element-wise sigmoid function



Multi-Layer Perceptron (MLP) / Neural Nets

Similarly, layers can be composed **in series**, such that

$$\boldsymbol{h}_0 = \boldsymbol{x}$$

$$\boldsymbol{h}_1 = \sigma(\boldsymbol{W}_1^T \boldsymbol{h}_0 + \boldsymbol{b}_1)$$

…

$$\boldsymbol{h}_L = \sigma(\boldsymbol{W}_L^T \boldsymbol{h}_{L-1} + \boldsymbol{b}_L)$$

$f(x;\theta) = \hat{y} = h_L$, where θ denotes the model parameters $\{W_k, b_k, \dots | k = 1, \dots, L\}$

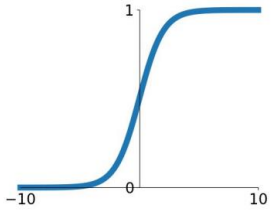
This model is the multi-layer perceptron, a.k.a., fully connected feedforward network

Homework: what if we don't use the non-linear functions?

Activation Functions

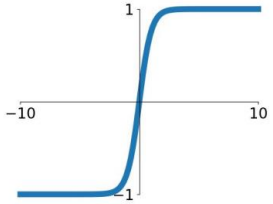
Sigmoid

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



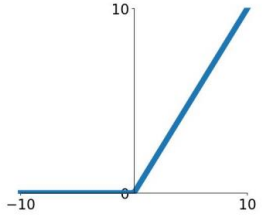
tanh

$$\tanh(x)$$



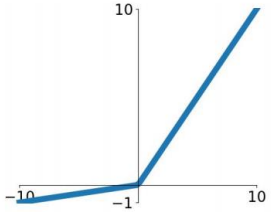
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

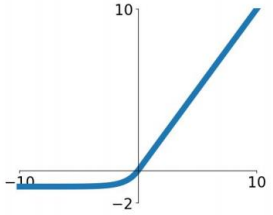


Maxout

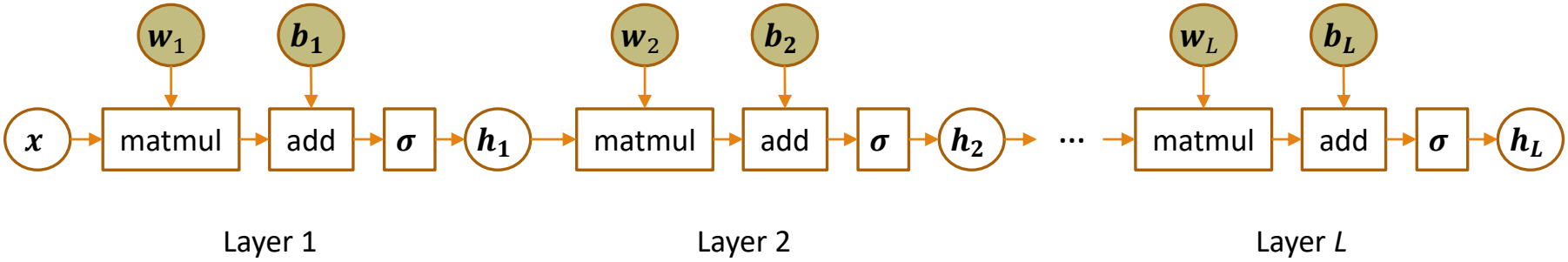
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Computational Graph



Final Layer: Classification

Binary classification:

- The width *q* of the last layer *L* is set to 1, which results in a single output $h_L \in [0,1]$ that models the probability $P(y = 1|\boldsymbol{x})$

Multi-class classification:

- The sigmoid activation σ in the last layer needs to be generalized to produce a (normalized) vector of probability estimates $P(y = i|\boldsymbol{x})$, this leads to the softmax function in which its i-th output is defined as

$$softmax(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{c} \exp(z_j)}$$

for $\mathrm{i} = 1, \ldots, c$

Automatic Differentiation

To minimize $L(\theta)$ with SGD, we need the gradient $\nabla_\theta L(\theta_t)$

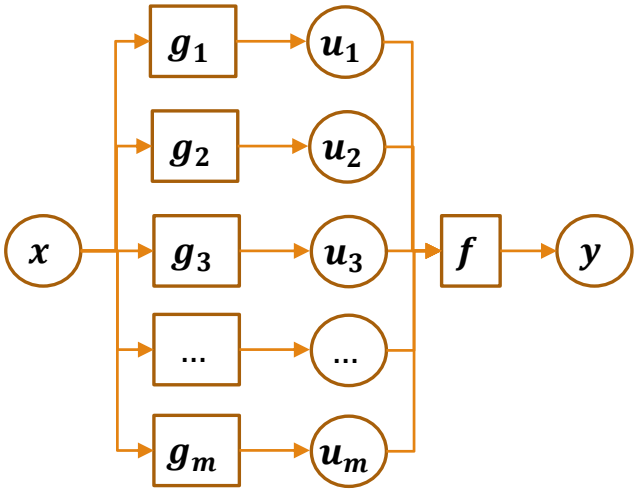
Therefore, we require the evaluation of the (total) derivatives

$$\frac{dL}{d\boldsymbol{W}_k}, \frac{dL}{d\boldsymbol{b}_k}$$

Of the loss $L$with respect to all model parameters $\boldsymbol{W}_k, \boldsymbol{b}_k$, for $k = 1, \dots, L$

These derivatives can be evaluated automatically from the **computational graphs** of L using **automatic differentiation**

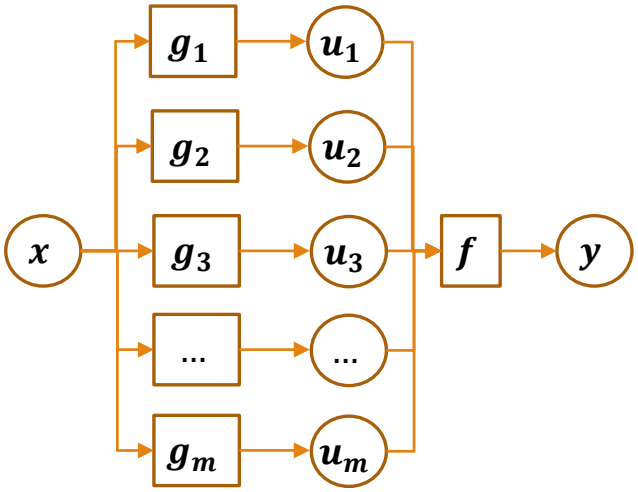
Chain Rule



Let us consider a 1-d output composition $f \circ g$, such that

$$y = f(u)$$
$$u = g(x) = (g_1(x), \dots, g_m(x))$$

Chain Rule



Let us consider a 1-d output composition $f \circ g$, such that

$$y = f(u)$$
$$u = g(x) = (g_1(x), \dots, g_m(x))$$

The chain rule states that $(f \circ g)' = (f' \circ g)g'$

For the total derivative, the chain rule generalizes to

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \sum_{k=1}^{m} \frac{\partial y}{\partial u_k} \frac{\mathrm{d}u_k}{\mathrm{d}x}$$

Reverse Automatic Differentiation

A Neural Network is a composition of differential functions, the total derivatives of the loss can be evaluated backward, by applying the chain rule recursively over its computational graph

The implementation is called reverse-mode automatic differentiation

An Example

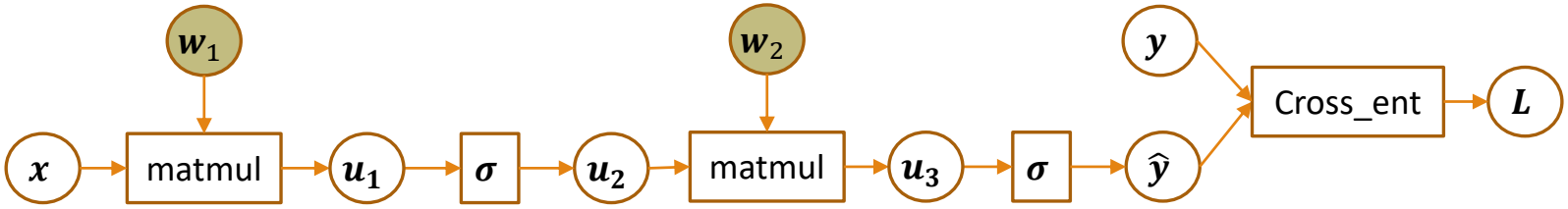
Let us consider a simplified 2-layer MLP with a cross-entropy loss:

$$f(\boldsymbol{x}; \boldsymbol{W}_1, \boldsymbol{W}_2) = \sigma\left(\boldsymbol{W}_2^T \sigma(\boldsymbol{W}_1^T \boldsymbol{x})\right)$$

$$L(y, \hat{y}; \boldsymbol{W}_1, \boldsymbol{W}_2) = cross\_ent(y, \hat{y})$$

for $\boldsymbol{x} \in \boldsymbol{R}^{\boldsymbol{p}}, y \in \boldsymbol{R}, \mathbf{W_1} \in \boldsymbol{R}^{\boldsymbol{p}\times\boldsymbol{q}}$ and $\boldsymbol{W_2} \in \boldsymbol{R}^{\boldsymbol{q}}$

In the feedforward pass, intermediate values are all computed from inputs to outputs, which results in the automatic computational graph below:

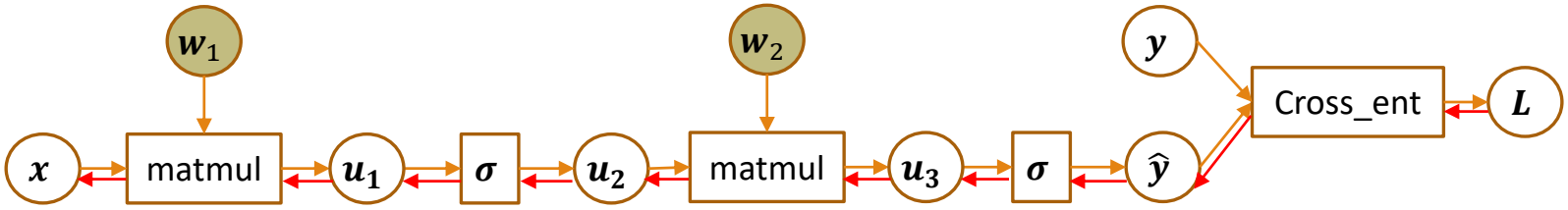


An Example

The total derivative can be computed through a backward pass, by walking through all paths from outputs to parameters in the computational graph and accumulating the terms.

For example, for $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{W}_1}$, we have

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{W}_1} = \frac{\partial L}{\partial \hat{y}} \frac{\mathrm{d}\hat{y}}{\mathrm{d}\boldsymbol{W}_1}, \qquad \frac{\mathrm{d}\hat{y}}{\mathrm{d}\boldsymbol{W}_1} = \cdots$$



An Example

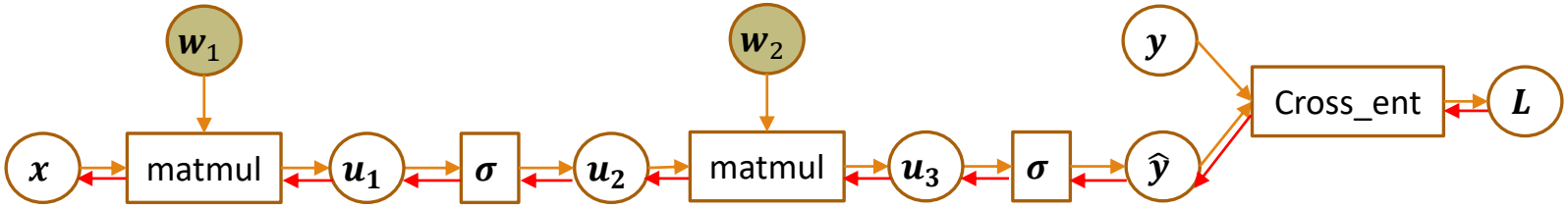
Let us zoom in on the computation of the output $\hat{y}$ and its derivatives with respect to $\boldsymbol{W}_1$:

- Forward pass: values u_1, u_2, u_3 and $\hat{y}$ are computed by traversing the graph from inputs to outputs given $\boldsymbol{x}$, $\mathbf{W}_1$, and $\mathbf{W}_2$
- Backward pass: by the chain rule, we have

$$\frac{\mathrm{d}\hat{y}}{\mathrm{d}\boldsymbol{W}_1} = \frac{\partial \hat{y}}{\partial u_3}\frac{\partial u_3}{\partial u_2}\frac{\partial u_2}{\partial u_1}\frac{\partial u_1}{\partial \boldsymbol{W}_1}$$

$$= \frac{\partial \sigma(u_3)}{\partial u_3}\frac{\partial \sigma(u_2)}{\partial u_2}\frac{\partial \sigma(u_1)}{\partial u_1}\frac{\partial \boldsymbol{W}_1^T u_1}{\partial \boldsymbol{W}_1}$$

Note that evaluating the partial derivatives requires the intermediate values computed forward



Back-propagation

This algorithm is also known as back-propagation

Since differentiation is a linear operator, automatic differentiation can be implemented efficiently in terms of tensor operations

Back-propagation and automatic differentiations are built-in in Deep Learning (DL) toolkits, e.g., TensorFlow, PyTorch

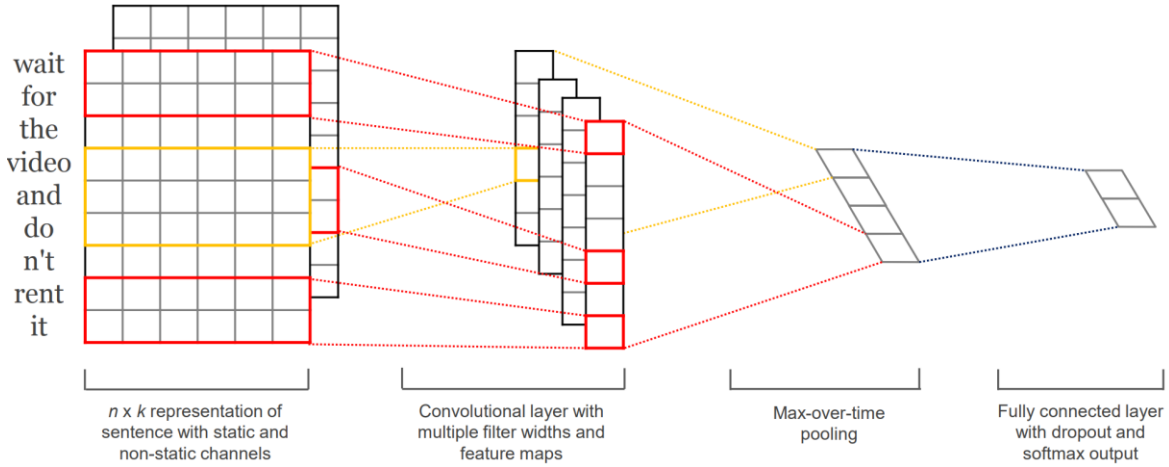
Convolutional Neural Networks

Instead of using fully-connected layers, it is more useful to have limited-window features

- E.g., n-grams, skip-word n-grams, word pairs that are nearby

Convolutional Network Networks, in each each CNN layer consists of

- Convolution
- Max-pooling



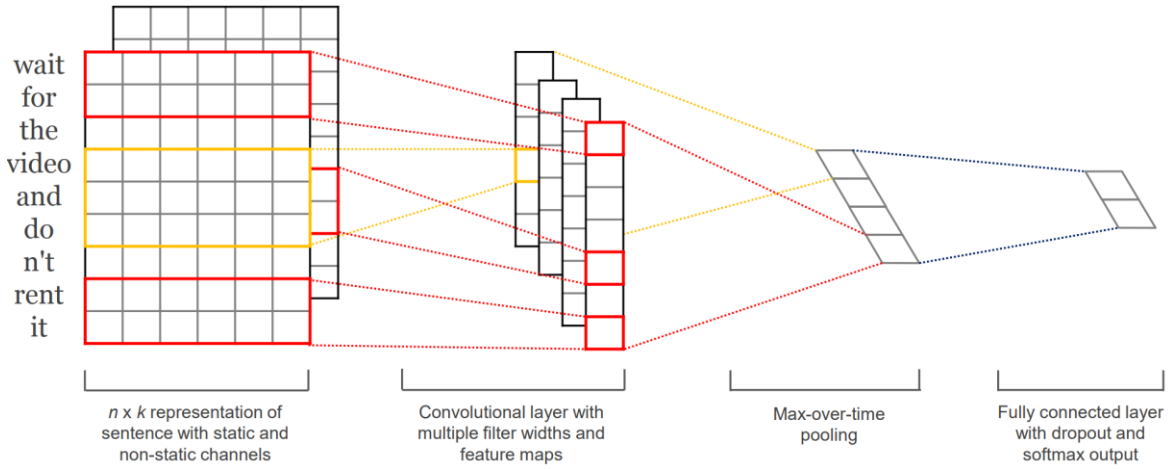
Convolutional Neural Networks

Convolution: for the i-th convolution filter of window size *k*, the convolution operation on an input sentence $\boldsymbol{x}$ produce

$$\boldsymbol{z} = (z_1, z_2, \ldots, z_{t-k+1})$$

$$z_j = g_1\left(\boldsymbol{w}_i \boldsymbol{x}_j + b_i\right)$$

- $\boldsymbol{w}_i \in \boldsymbol{R}^{\boldsymbol{kn_1}}$ are the linear transform parameters
- $\boldsymbol{x}_j$ denotes the j-th context window in $\boldsymbol{x}$
- g_1 is an activation function such as the rectified linear unit.



Convolutional Neural Networks

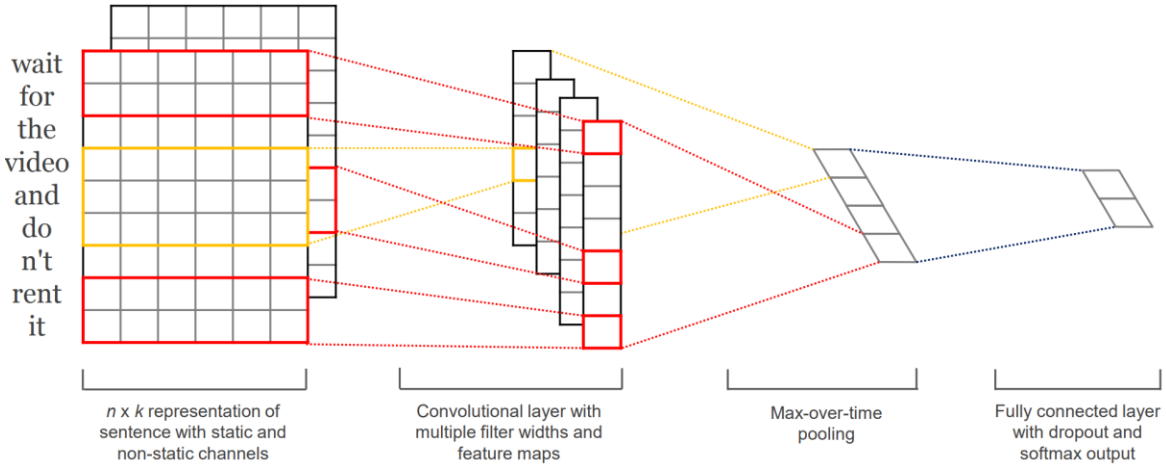
Max-pooling: $m_i = \max(\boldsymbol{z})$

Often replicated with

- **Multiple filters** to capture most important (weighted) n-gram (context windows of *n* words) features
- **Varying window sizes** to allow variable-length n-grams

May have multiple Conv+max-pooling layers

The rest are standard feedforward networks



Convolutional Neural Networks for Sentence Classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Convolutional Neural Networks for Sentence Classification, Kim et al., 2014 (EMNLP)

Experimentation

Experimentation cycle

- Learn parameters (e.g. model probabilities) on training set
- Tune hyperparameters on development/held-out set
- Compute accuracy on test set

Overfitting and generalization

- Want a classifier which does well on *test* data
- Overfitting: fitting the training data very well, but not generalizing well to unsee data (i.e., poor results on *test*)
- Common ways to improve generalization
  - Dropout
  - L1, L2 regularization
  - Cross-validation
  - Early stopping

Summarize

ML models for text classification

- Naïve Bayes classifiers
- Multi-class logistic regression
- Multi-layer perceptron
- Convolutional Neural Networks

Homework Assignment #1

Posted on Canvas

Due in 2 weeks