Tuesday • September 28, 2020

Neural Networks and Language

Yale

LING 380/780

Neural Network Models of Linguistic Structure

Full Neural Network Algorithm

- **Input:**
  - Training set
  - Validation set
  - Test set
  - Neural network architecture $f(\cdot, \boldsymbol{\theta})$ (multi-layer perceptron)

Full Neural Network Algorithm

- **Hyperparameters:**
  - Batch size b
  - Learning rate η
  - Sizes of neural network layers
  - Max number of epochs

Full Neural Network Algorithm

- Choose a set of hyperparameter values to try.
- For each hyperparameter configuration:
  - Train a model using the training set and validation set.
  - Test the model using the validation set.
- Find the model with the best performance on the validation set.
- Test this model using the test set.

Training Subroutine

- Initialize $\boldsymbol{\theta}$ to a random value, where the number of parameters is based on the hyperparameter settings (neural network layer sizes).
- Partition the training data into mini-batches $\mathbb{B}_1, \mathbb{B}_2, \ldots, \mathbb{B}_k$ of size b.

Training Subroutine

- Repeat for the max number of epochs:
  - For each mini-batch $\mathbb{B}_i$:
    - Form the computation graph for $\mathcal{L}$.
    - Set all gradients of the computation graph to 0.
    - Use backpropagation to compute $\nabla_{\boldsymbol{\theta}}\mathcal{L}$.
    - Set $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}$.
- Set $\boldsymbol{\theta}$ to be the parameter values that resulted in the best validation performance.

Variations: Hyperparameter Tuning

- **Hyperparameter tuning** is the process of trying different combinations of hyperparameter values.
- **Grid search:** Choose some range of values for each hyperparameter, and try all combinations of hyperparameters.

Variations: Hyperparameter Tuning

- **Grid Search Example:**
  - Batch size = 16, 32, 64
  - Learning rate = 1.0, .1, .01
  - Layer size = 50, 100, 200
  - Train a network with all 27 combinations of hyperparameters and keep the model with the best performance.

Variations: Hyperparameter Tuning

- Occasionally other methods of hyperparameter tuning are used as well, mainly because grid search requires too many trials.
- **Random search:** Randomly choose combinations of hyperparameter values.
- **Bayesian tuning:** Use a statistical model to choose new hyperparameter values based on previous results.

Variations: SGD Algorithms

- **Learning rate adaptation:** Change the learning rate throughout training.
- **Annealing:** Start out with a high learning rate, and gradually decrease it as the model improves.
  - A high learning rate causes the algorithm to “search the parameter space” by trying out a wide range of parameters
  - A low learning rate causes the algorithm to hone in on a local/global minimum

"hone" means to gradually and precisely approach towards a specific target

Variations: SGD Algorithms

- SGD typically uses an annealing **schedule.**
- **Step-Based Decay:** Every k epochs, multiply η by a **decay factor** d between 0 and 1
- **Patience-Based Decay:** If the max validation performance was not achieved within the last p epochs, multiply η by d

Variations: SGD Algorithms

- **Momentum:** Allow $\Delta\boldsymbol{\theta}$ to depend on the $\Delta\boldsymbol{\theta}$ from the previous time step.
- $\Delta\boldsymbol{\theta} = -\eta\nabla_{\boldsymbol{\theta}}\mathcal{L} + \mu\Delta\boldsymbol{\theta}'$
  - $\Delta\boldsymbol{\theta}'$ is $\Delta\boldsymbol{\theta}$ from the previous time step
  - $\mu > 0$ is the momentum factor
- Helps speed up SGD during the "parameter search" phase

Variations: Advanced SGD Algorithms

- AdaGrad (Adaptive Gradient)
- RMSProp (Root Mean Squared Propagation)
- Adam (Adaptive Moment)
    - Adam is the most popular. We will exclusively use Adam and SGD.

Variations: Regularization

- **Regularization** is any technique designed to prevent overfitting.
- A model overfits if it achieves high training performance but poor testing performance.
  - The model "memorizes" but does not generalize.

Variations: Regularization

- **L^2 Regularization:** Tries to reduce the norm of $\boldsymbol{\theta}$ by adding a **penalty term** to the objective.

$$\mathcal{L} = \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{D}} \left(L(f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{y}) + \lambda ||\boldsymbol{\theta}||^2 \right)$$

Variations: Regularization

- **L^2 Regularization:** Tries to reduce the norm of $\boldsymbol{\theta}$ by adding a **penalty term** to the objective.

$$\mathcal{L} = \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathbb{D}} \left(L(f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{y}) + \lambda \boldsymbol{\theta}^\top \boldsymbol{\theta} \right)$$

- **Occam's Razor:** "simpler" models are better.
- Supposedly, L^2 Regularization makes models "simpler."

Variations: Regularization

- Dropout: During each SGD iteration, temporarily set a random sample of weights to 0.
- The dropout rate is the percentage of weights to set to 0.
- The full neural network approximates the average of models obtained by dropping out units.

Variations: Regularization

- **Early Stopping:** If the max validation performance was not achieved within the last p epochs, stop training immediately.
- p is the **patience**.
- Overfitting often occurs when training has gone on for too long.

Applying Neural Networks to Language

Applying Neural Networks to Language

- We have now finished learning the basic technology behind neural networks.
- **This lecture (and the rest of the course):** Apply neural networks to language!

Word2Vec Revisited

Skip-Gram with Negative Sampling (SGNS):

$$y = \sigma(\langle c \rangle^\top [\![w]\!])$$

- $[\![w]\!]$ = the word embedding for $w \in \mathbb{V}$
- $\langle c \rangle$ = the context embedding for $c \in \mathbb{V}$
- y = the probability that w and c "occur together"

Word2Vec Revisited

Skip-Gram **without** Negative Sampling (SG):

$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{C}[\![w]\!])$$

- $[\![w]\!]$ = the **word embedding** for $w \in \mathbb{V}$
- $\boldsymbol{C}$ = the matrix of **all context embeddings** as row vectors
- $\boldsymbol{y}$ = the probability that w and c "occur together" **for all context words c**

The Softmax Function

Softmax turns vectors into probability vectors:

$$\operatorname{softmax}(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}}}{\mathbf{1}^\top e^{\boldsymbol{x}}}$$

- NumPy: `np.exp(x) / np.exp(x).sum()`
- SciPy: `scipy.special.softmax(x)`
- PyTorch: `F.softmax(x)`

Softmax vs. Sigmoid

- Sigmoid is used for binary classification.
  - $y = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$ is the probability of class 1
  - $1 - y$ is the probability of class 0
- Softmax is used for multinomial classification (more than 2 classes).
  - $\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ contains the probability of all classes

Softmax vs. Sigmoid

- Sigmoid is softmax when the confidence (logit) score of class 0 is always 0.

$$\operatorname{softmax}\left(\begin{bmatrix} x \\ 0 \end{bmatrix}\right)_0 = \frac{e^x}{e^x + e^0} = \frac{e^x}{e^x + 1} = \sigma(x)$$

Multinomial Classification

Binary Cross-Entropy Loss Function:

$$L_{CE}(\hat{y}, y) = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y})$$

where $0 < \hat{y} < 1$ and $y \in \{0, 1\}$

Multinomial Cross-Entropy Loss Function:

$$L_{CE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\mathbf{1}^\top(\boldsymbol{y} \odot \ln(\hat{\boldsymbol{y}}))$$

where $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ are both probability vectors

Multinomial Classification

Simplified Cross-Entropy Loss Function:

$$L_{CE}(\hat{\boldsymbol{y}}, y) = -\ln(\hat{y}_y)$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^n$ is a probability vector and $y \in \{1, 2, \ldots, n\}$ is a class label

Multinomial Classification

- **Skip-Gram with Negative Sampling:** Do words w and c occur together? (Yes or No)
- **Skip-Gram without Negative Sampling:** Which word is most likely to occur with w?
- SGNS *classifies*; SG *predicts*.

Dataset Preparation

Text: Brazil's health minister has tested positive for the coronavirus while in New York for the United Nations General Assembly, where President Jair Bolsonaro spoke on Tuesday.

- **Dataset Cleaning:** Remove metadata, formatting, non-linguistic characters, etc.
- **Optional:** Make everything lowercase, remove common “stopwords,” remove morphological affixes, etc.

Dataset Preparation

- **Tokenization:** Convert the text into a sequence of tokens from a fixed vocabulary.

```
['[BOS]', '[BOS]', 'Brazil', "'", 's', 'health',
'minister', 'has', 'tested', 'positive', 'for', 'the',
'co', '##rona', '##virus', 'while', 'in', 'New', 'York',
'for', 'the', 'United', 'Nations', 'General',
'Assembly', ',', 'where', 'President', 'Jai', '##r',
'Bo', '##lson', '##aro', 'spoke', 'on', 'Tuesday', '.',
'[EOS]', '[EOS]']
```

Dataset Preparation

- **Indexation:** Assign each vocabulary item an integer index.

```
[101, 101, 3524, 112, 188, 2332, 3907, 1144, 7289, 3112,
1111, 1103, 1884, 15789, 27608, 1229, 1107, 1203, 1365,
1111, 1103, 1244, 3854, 1615, 2970, 117, 1187, 1697,
19183, 1197, 9326, 15590, 14452, 2910, 1113, 9667, 119,
102, 102]
```

Embedding Layer

The **embedding layer** contains a lookup table that maps indices to their word embeddings.

- $\mathrm{forward}(\boldsymbol{x}) = ?$
  - $\boldsymbol{x}$: array of indices of shape (number of tokens,)
  - $\mathrm{forward}(\boldsymbol{x})$: an array of shape (number of tokens, embedding size) where each index has been replaced with its embedding
- $\mathrm{backward}(\boldsymbol{\delta}) =$ None, but the gradient of the word embedding is $\boldsymbol{\delta}$