Neural Networks in Python (PA6 Lab)

CS114 Lab 13

Kenneth Lai

May 1, 2020

Python Deep Learning Libraries

- CNTK
- Keras
- MXNet
- PyTorch
- TensorFlow
- Theano
- etc.

Python Deep Learning Libraries

- Numpy

Python Deep Learning Libraries

- Numpy
  - It builds character!

Python Deep Learning Libraries

- Numpy
  - It builds character!
    - Seriously, everyone should code a neural network from scratch at least once...

PA6 Tasks

- Preprocessing

PA6 Tasks

- Preprocessing
- Initialization

PA6 Tasks

- Preprocessing
- Initialization
- Training

PA6 Tasks

- Preprocessing
- Initialization
- Training
  - Forward propagation
  - Calculate the cross-entropy loss
  - Backpropagation
  - Update the model weights

PA6 Tasks

- Preprocessing
- Initialization
- Training
  - Forward propagation
  - Calculate the cross-entropy loss
  - Backpropagation
  - Update the model weights
- Testing

PA6 Tasks

- Preprocessing
- Initialization
- Training
  - Forward propagation
  - Calculate the cross-entropy loss
  - Backpropagation
  - Update the model weights
- Testing
  - Forward propagation

Preprocessing

- Feature extraction

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks
    - Last word in the stack
    - First word in the buffer
    - Dependent of the second-to-last word in the stack if there is one
    - etc.

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks
    - Last word in the stack
    - First word in the buffer
    - Dependent of the second-to-last word in the stack if there is one
    - etc.
  - Implemented for you

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks
    - Last word in the stack
    - First word in the buffer
    - Dependent of the second-to-last word in the stack if there is one
    - etc.
  - Implemented for you
- Embedding matrix

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks
    - Last word in the stack
    - First word in the buffer
    - Dependent of the second-to-last word in the stack if there is one
    - etc.
  - Implemented for you
- Embedding matrix
- Your job: put them together in `embedding_lookup`

Preprocessing

- Feature extraction
  - A Fast and Accurate Dependency Parser using Neural Networks
    - Last word in the stack
    - First word in the buffer
    - Dependent of the second-to-last word in the stack if there is one
    - etc.
  - Implemented for you
- Embedding matrix
- Your job: put them together in `embedding_lookup`
  - $\mathbf{x} = [\mathbf{E}_{w_1}, ..., \mathbf{E}_{w_m}] \in \mathbb{R}^{dm}$

Initialization

- Weights for each layer according to a random uniform distribution

Initialization

- Weights for each layer according to a random uniform distribution
- Advanced: if you want, you can read about Xavier uniform initialization

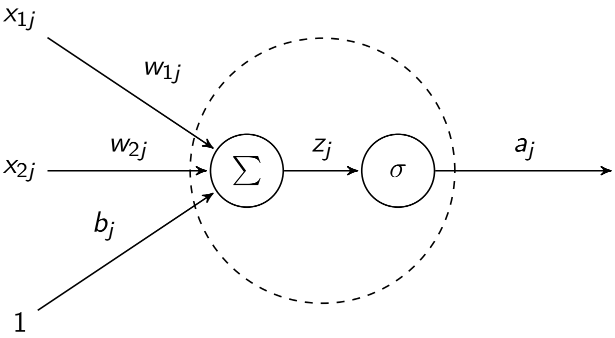
Initialization

- Weights for each layer according to a random uniform distribution
- Advanced: if you want, you can read about Xavier uniform initialization
  - Idea: the more inputs to a layer, the smaller the weights can be

Initialization

- Weights for each layer according to a random uniform distribution
- Advanced: if you want, you can read about Xavier uniform initialization
  - Idea: the more inputs to a layer, the smaller the weights can be
  - Recommended intervals very roughly correspond to this

Graphical Representation of a Neuron



Forward Propagation

- $\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{x}\mathbf{W_1} + \mathbf{b}_1)$
- $\mathbf{h}_2 = \mathrm{ReLU}(\mathbf{h_1}\mathbf{W_2} + \mathbf{b}_2)$
- $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{h_2}\mathbf{U} + \mathbf{b}_3)$

Forward Propagation

- $\mathbf{h}_1 = \text{ReLU}(\mathbf{x}\mathbf{W_1} + \mathbf{b}_1)$
- $\mathbf{h}_2 = \text{ReLU}(\mathbf{h_1}\mathbf{W_2} + \mathbf{b}_2)$
- $\hat{\mathbf{y}} = \text{softmax}(\mathbf{h_2}\mathbf{U} + \mathbf{b}_3)$
- Hint: note the order of multiplication!

Forward Propagation

- $\mathbf{h}_1 = \mathrm{ReLU}(\mathbf{x}\mathbf{W_1} + \mathbf{b}_1)$
- $\mathbf{h}_2 = \mathrm{ReLU}(\mathbf{h_1}\mathbf{W_2} + \mathbf{b}_2)$
- $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{h_2}\mathbf{U} + \mathbf{b}_3)$

- Hint: note the order of multiplication!
  - In the end, you don't need as many transposes

Rectified Linear Unit

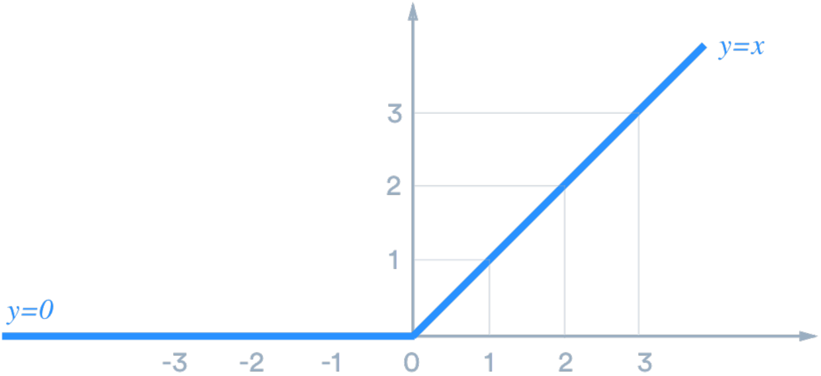
- Common neural network activation function

Rectified Linear Unit

- Common neural network activation function
- $\mathrm{ReLU}(z) = \max(z, 0)$

Rectified Linear Unit

- Common neural network activation function
- $\text{ReLU}(z) = \max(z, 0)$



Source

Minibatch Processing

- Much more efficient than parsing one sentence at a time

Minibatch Processing

- Much more efficient than parsing one sentence at a time
  - But only if your code is fully broadcasted

Minibatch Processing

- Much more efficient than parsing one sentence at a time
  - But only if your code is fully broadcasted
- All layer outputs should have shape `[batch_size, ...]`

Cross-entropy Loss

- $J(\theta) = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{3} y_i \log \hat{y}_i$

Cross-entropy Loss

- $J(\theta) = CE(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{i=1}^{3} y_i \log \hat{y}_i$
- To compute the loss for the training set, we average this $J(\theta)$ across all training examples

Backpropagation

- Recall the equations of backpropagation:

Backpropagation

- Recall the equations of backpropagation:
- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$

Backpropagation

- Recall the equations of backpropagation:
- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$

Backpropagation

- Recall the equations of backpropagation:
- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\delta_K \cdot \mathbf{W}_K^T) \odot \sigma'(\mathbf{z}_J)$

Backpropagation

- Recall the equations of backpropagation:
- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\delta_K \cdot \mathbf{W}_K^T) \odot \sigma'(\mathbf{z}_J)$
    - Remember to use the derivative of ReLU rather than sigmoid

Backpropagation

- Recall the equations of backpropagation:

- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\delta_K \cdot \mathbf{W}_K^T) \odot \sigma'(\mathbf{z}_J)$
    - Remember to use the derivative of ReLU rather than sigmoid

- Hint: note the order of multiplication (again)!

Backpropagation

- Recall the equations of backpropagation:

- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\delta_K \cdot \mathbf{W}_K^T) \odot \sigma'(\mathbf{z}_J)$
    - Remember to use the derivative of ReLU rather than sigmoid

- Hint: note the order of multiplication (again)!
- Also, note that the equations apply to one sentence at a time

Backpropagation

- Recall the equations of backpropagation:
- For all layers l:
  - $\nabla_l L = \mathbf{x}_l \odot \delta_l$
- For an output layer $\mathcal{L}$:
  - $\delta_{\mathcal{L}} = \hat{\mathbf{y}} - \mathbf{y}$
- For a non-output layer J (with next layer K):
  - $\delta_J = (\delta_K \cdot \mathbf{W}_K^T) \odot \sigma'(\mathbf{z}_J)$
    - Remember to use the derivative of ReLU rather than sigmoid
- Hint: note the order of multiplication (again)!
- Also, note that the equations apply to one sentence at a time
  - Broadcasting will turn one Hadamard product into a matrix (dot) product

Minibatch Gradient Descent

- HW1: minibatch gradient is average of individual gradients

Minibatch Gradient Descent

- HW1: minibatch gradient is average of individual gradients
- PA6: minibatch gradient is sum of individual gradients

Minibatch Gradient Descent

- HW1: minibatch gradient is average of individual gradients
- PA6: minibatch gradient is sum of individual gradients
  - Average will also work, but you'll need to adjust the default learning rate accordingly

Minibatch Gradient Descent

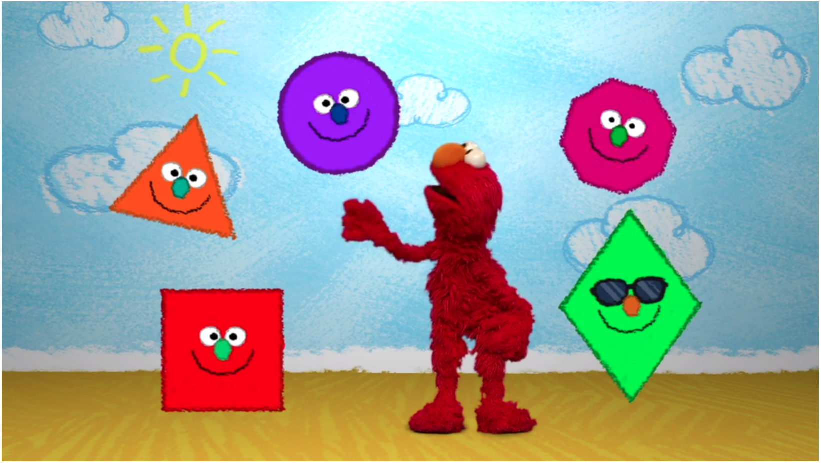
- HW1: minibatch gradient is average of individual gradients
- PA6: minibatch gradient is sum of individual gradients
  - Average will also work, but you'll need to adjust the default learning rate accordingly
- $\theta_{t+1} = \theta_t - \eta \nabla L$

General Advice

- Know your shapes!

General Advice

- Know your shapes!



Source