Week 3:
Recurrent Neural Networks

Tin D. Vo
@chata.ai

Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- Language Models
- Assignment

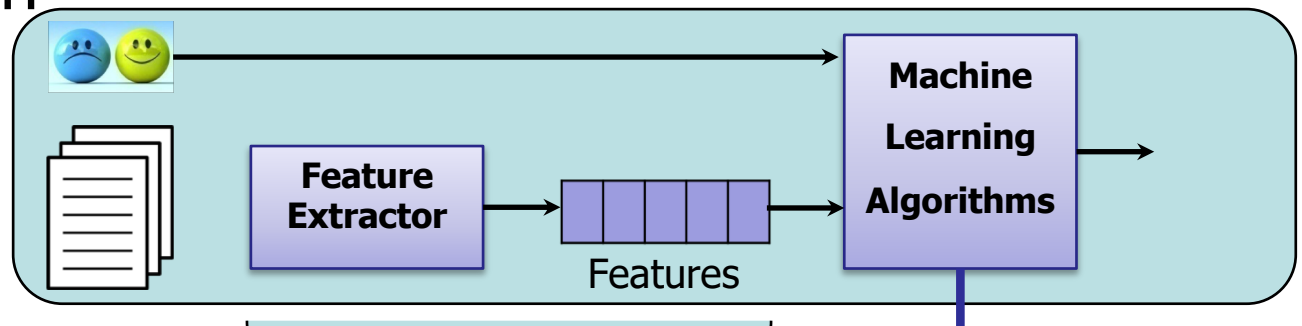
Outline

- **Introduction**
- Deep Learning Libraries
- Recurrent Neural Networks
- Language Models
- Assignment

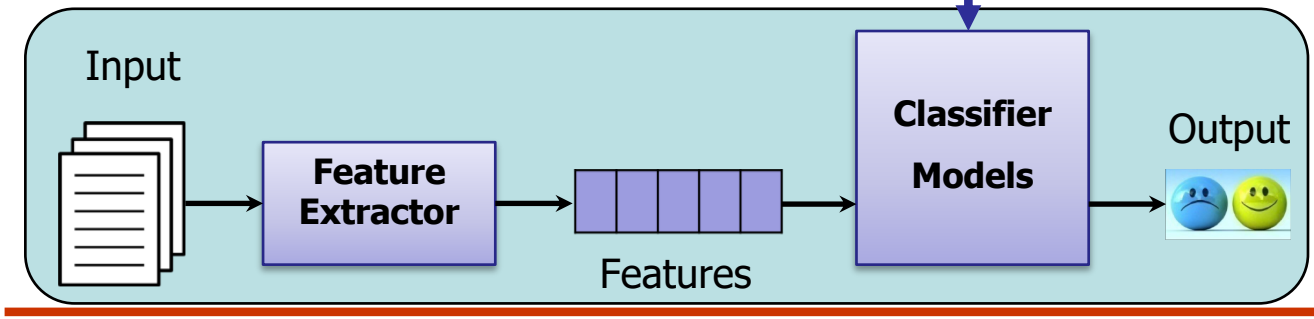
Statistical Models

General model:

- Train



- Predict

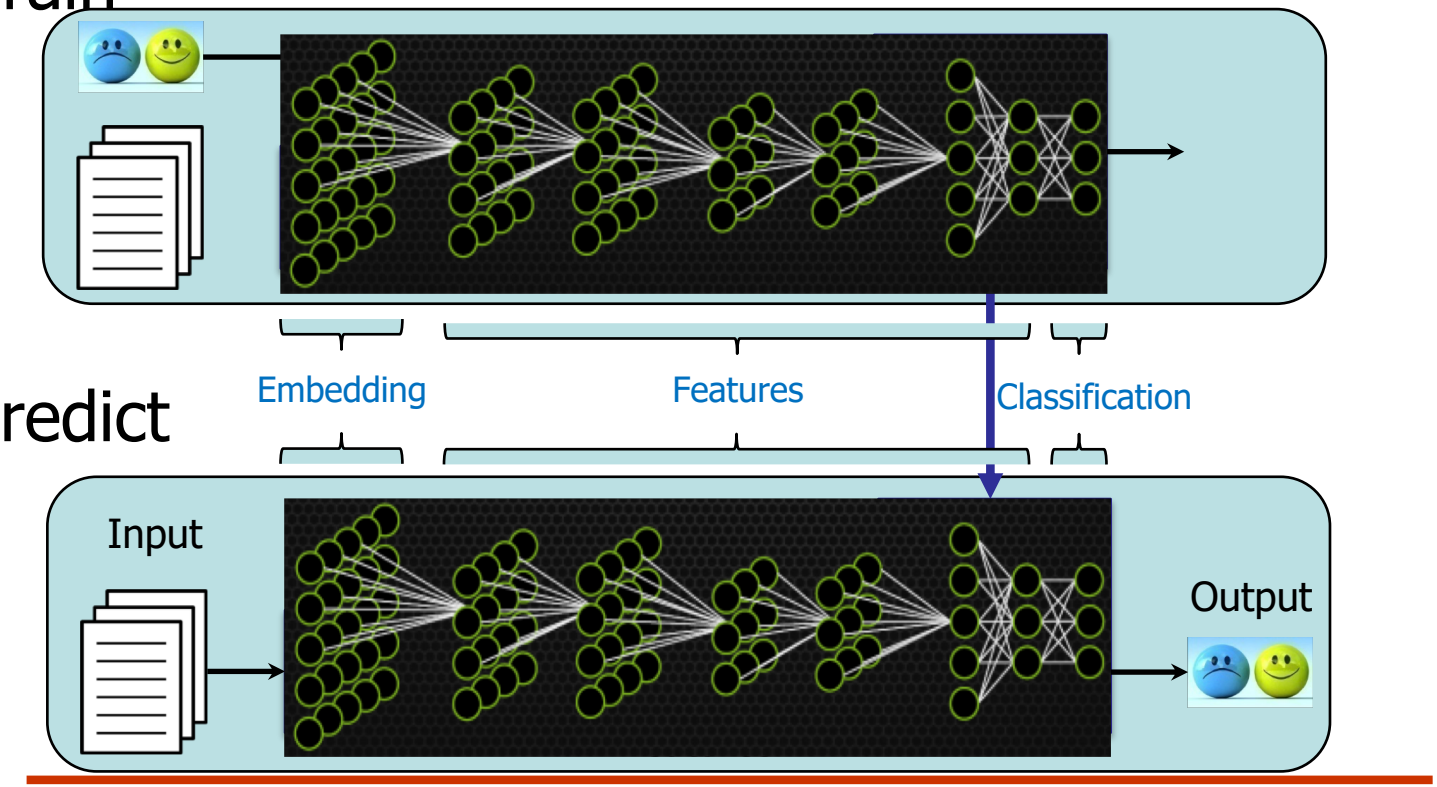


List of features:

- One-hot vector
- N-grams
- Brown Clustering
- Lexicons
- Patterns
- POS
- …

Neural Network Models

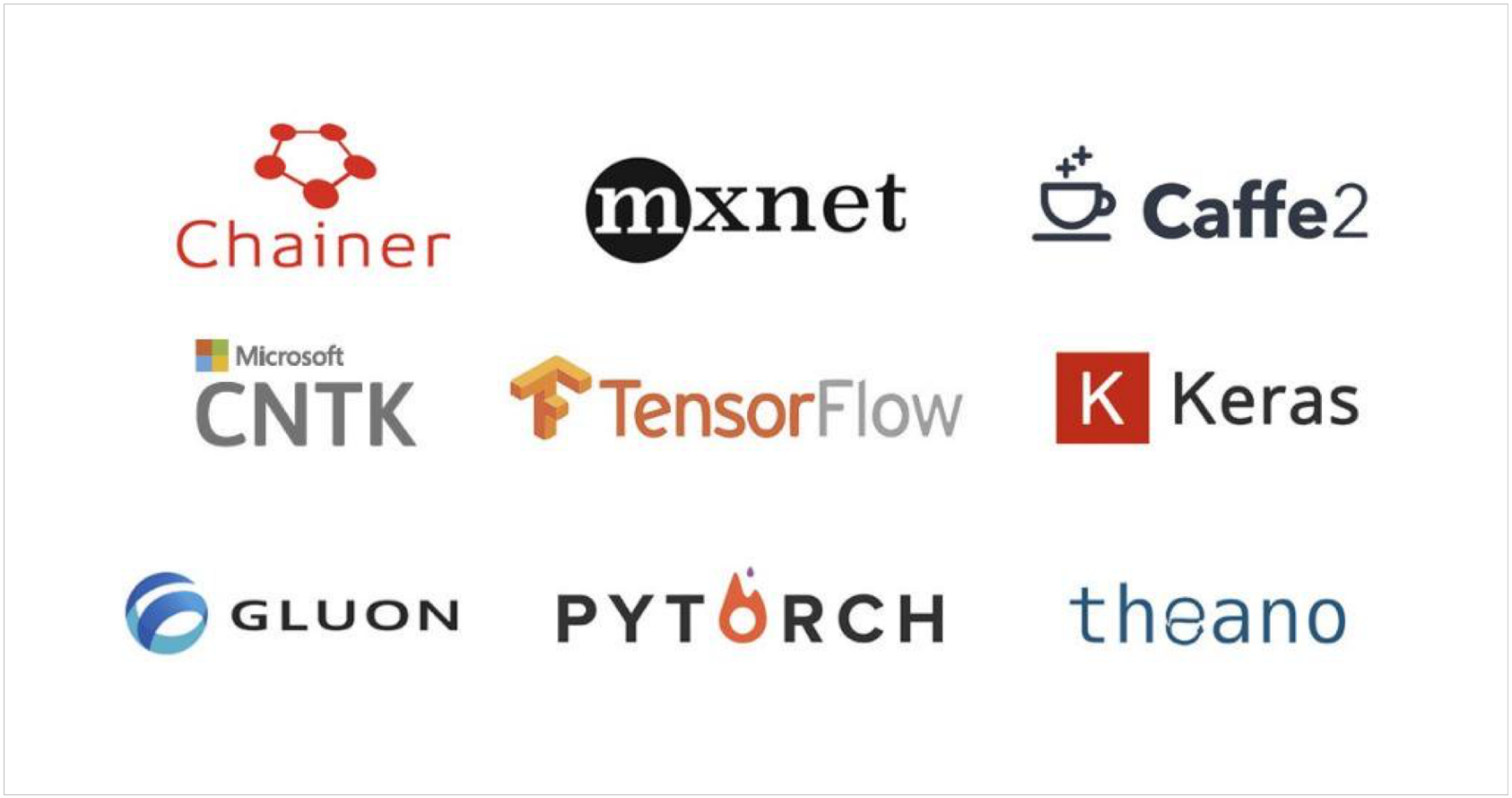
- Neural Network: a sub-area of machine learning
  - Train
  - Predict



Outline

- Introduction
- **Deep Learning Libraries**
- Recurrent Neural Networks
- Language Models
- Assignment

Deep Learning Libraries



Source: https://twitter.com/awscloud/status/987470361892720640

Tensorflow vs Pytorch

Tensorflow

- Google (11/2015)
- Static graph:
  - Operations (node)
  - Tensors (edge)

⇒ Communication with outer world via tf.Session

- Difficult to debugging
- Visualization: Tensorboard
- Deployment: TensorFlow Serving
- Manual designed parallelism

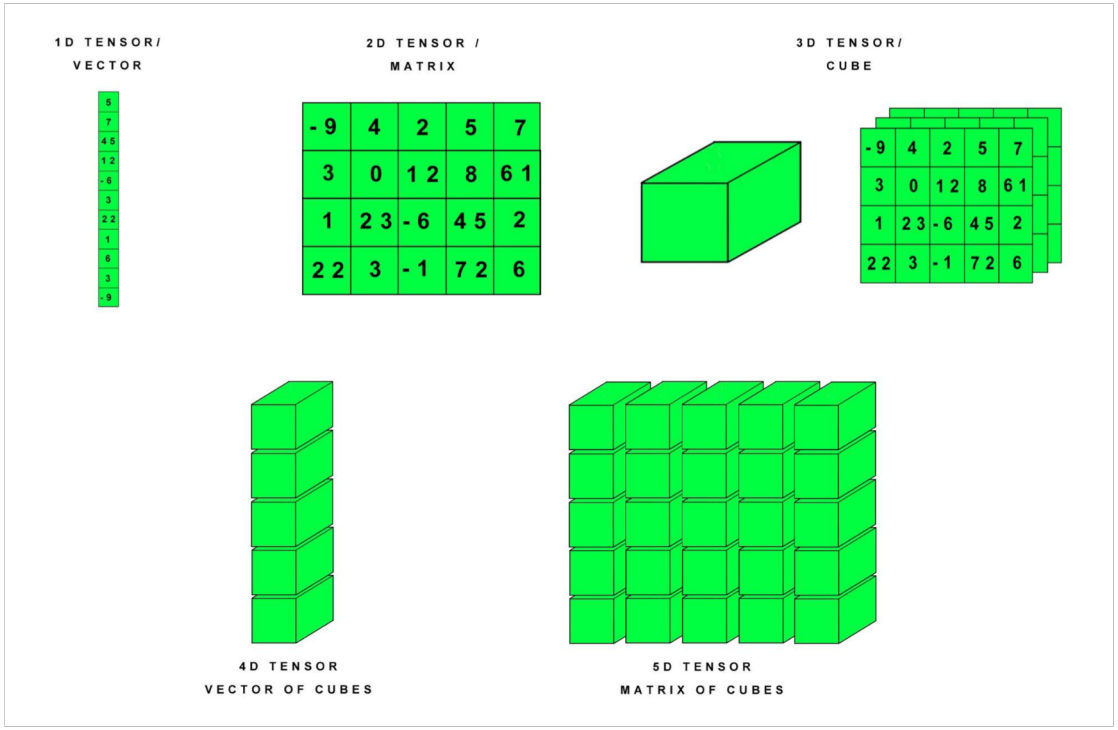
Pytorch

- Facebook (01/2017)
- Dynamic graph:
  - Operations (node)
  - Tensors (edge)

⇒ Does not have the concept of a session

- Easy to debugging
- Visualization: visom
- Deployment: Flask
- Declarative data parallelism

Basic Operators



https://www.datacamp.com/community/tutorials/investigating-tensors-pytorch

```
torch.rand([2, 3])
```

```
tensor([[0.0112, 0.1804, 0.1718],
        [0.2779, 0.4685, 0.4964]])
```

```
torch.zeros([4,5])
```

```
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]])
```

```
torch.randint(9,[2,5])
```

```
tensor([[5., 6., 7., 5., 4.],
        [8., 7., 2., 0., 6.]])
```

```
x = torch.tensor([[3.]*4]*2)
print(x)
```

```
tensor([[3., 3., 3., 3.],
        [3., 3., 3., 3.]])
```

```
y = torch.ones([2,4])
x - y
```

```
tensor([[2., 2., 2., 2.],
        [2., 2., 2., 2.]])
```

```
x + y
```

```
tensor([[4., 4., 4., 4.],
        [4., 4., 4., 4.]])
```

Autograd

```
x = torch.ones(2, 2, requires_grad=True)
print(x)
```

```
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)
```

```
def forward(x):
    y = x + 2
    z = y * y * 3
    print("z = ", z)
    return z.mean()
```

```
out = forward(x)
print("out = ", out)
```

```
z =  tensor([[27., 27.],
        [27., 27.]], grad_fn=<MulBackward>)
out =  tensor(27., grad_fn=<MeanBackward1>)
```

```
out.backward()
print(x.grad)
```

```
tensor([[4.5000, 4.5000],
        [4.5000, 4.5000]])
```

$$o = \frac{1}{4}\sum_i z_i; \quad z_i = 3(x_i + 2)^2$$

$$\Rightarrow o|_{x_i=1} = 27$$

$$\frac{\partial o}{\partial x_i} = \frac{\partial o}{\partial z_i}\frac{\partial z_i}{\partial x_i}$$

$$\frac{\partial o}{\partial x_i} = (\frac{1}{4} \times 1)\,[3 \times 2 \times (x_i + 2)]$$

$$\frac{\partial o}{\partial x_i} = (\frac{3}{2}(x_i + 2)$$

$$\Rightarrow \frac{\partial o}{\partial x_i}|_{x_i=1} = 4.5$$

NN Model

1. Data Processing
2. Design a neural network architecture
3. Determine an objective function
4. Select an optimization algorithm

Generate random data and parameters

```
x = torch.randn(batch_size, n_in)
y = torch.tensor([[1.0], [0.0], [0.0], [1.0], [1.0],
                  [1.0], [0.0], [0.0], [1.0], [1.0]])
```

```
n_in, n_h, n_out, batch_size = 10, 5, 1, 10
```

Construct a model

```
model = nn.Sequential(nn.Linear(n_in, n_h),
                      nn.ReLU(),
                      nn.Linear(n_h, n_out),
                      nn.Sigmoid())
criterion = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

NN Model

Train the model

```
for epoch in range(10):
    # Forward Propagation
    y_pred = model(x)
    # Compute and print loss
    loss = criterion(y_pred, y)
    print('epoch: ', epoch,' loss: ', loss.item())
    # Zero the gradients
    optimizer.zero_grad()

    # perform a backward pass (backpropagation)
    loss.backward()

    # Update the parameters
    optimizer.step()
```

```
epoch:  0  loss:  0.2677503526210785
epoch:  1  loss:  0.26763197779655457
epoch:  2  loss:  0.2675149142742157
epoch:  3  loss:  0.26739737391471863
epoch:  4  loss:  0.2672807574272156
epoch:  5  loss:  0.26716405153274536
epoch:  6  loss:  0.2670478820800781
epoch:  7  loss:  0.26693204045295715
epoch:  8  loss:  0.26681631803512573
epoch:  9  loss:  0.26670125126838684
```

1. Train the model
   a. Tune parameters
   b. Evaluate on validating data
   c. Save the best model
2. Inference
   a. Load the best model
   b. Predict unlabelled data

Assignment 1

Tfidf → fc nn → classification

embedding → fc nn → classification

Outline

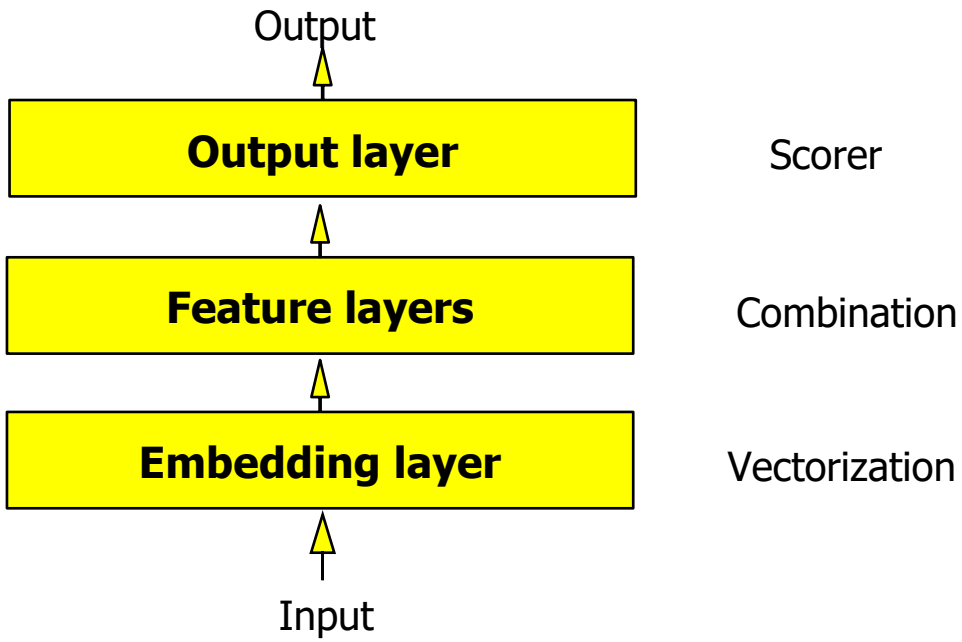
- Introduction
- Deep Learning Libraries
- **Recurrent Neural Networks**
  - Preview of Neural Networks
  - Feature Layers
  - Training Algorithms
- Language Models
- Assignment

Outline

- Introduction
- Deep Learning Libraries
- **Recurrent Neural Networks**
  - **Preview of Neural Networks**
  - Feature Layers
  - Training Algorithms
- Language Models
- Assignment

Overview

- General model:

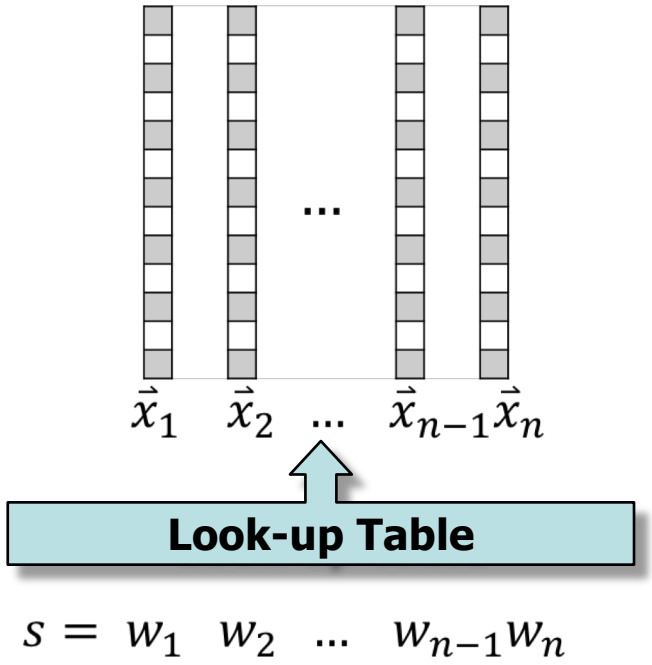


Overview

- Embedding Layer
  - Word to vector
  - Look up table

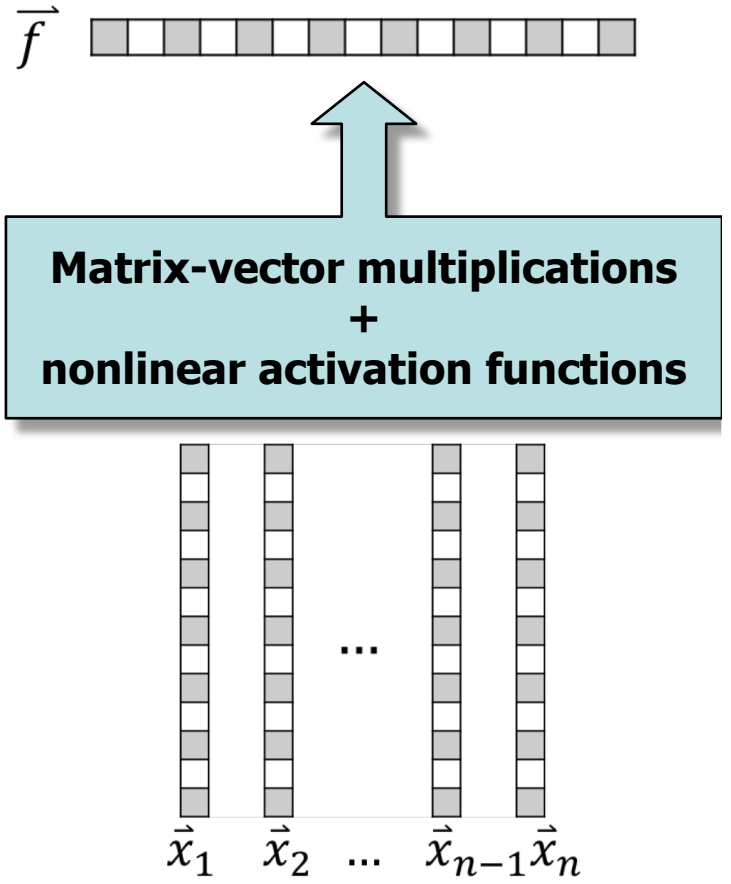
$$\vec{x}_i = \boldsymbol{W}_{|V|} \times \vec{I}_i$$

  - Where:
    - $\vec{x}_i \epsilon R^d$: word embedding
    - $\boldsymbol{W}_{|V|} \epsilon R^{d \times |V|}$: embedding matrix
    - $\vec{I}_i \epsilon R^{|V|}$: one-hot vector of word w_i
    - d: embedding dimension



Overview

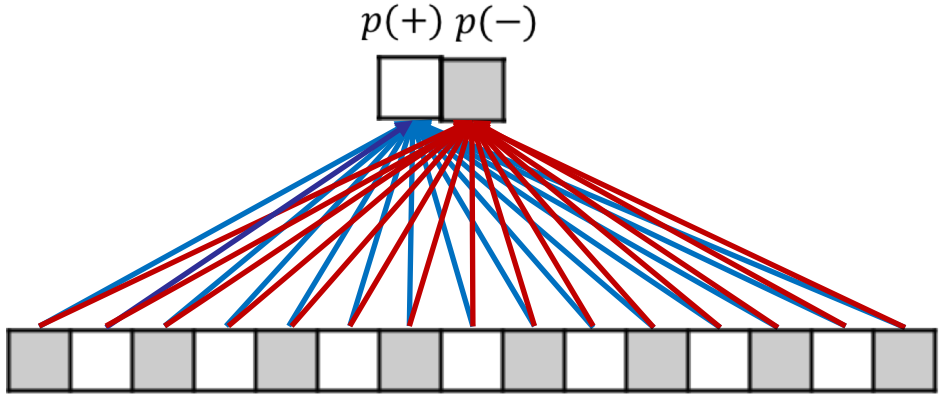
- Feature Layer
  - Automatically learn the representation of inputs
  - Matrix-vector multiplication
  - Element-wise composition
  - Non-linear transformation



Overview

- Output Layer
  - *Margin output:* $\vec{f}_{score} = \boldsymbol{W}_O \vec{f} + \vec{b}_O$
  - *Probability output*

$$O_c^{(i)} = P\left(Y = c \middle| x^{(i)}, \theta\right)$$
$$= softmax_c\left(\vec{f}_{score}\right)$$
$$= \frac{e^{\vec{w}_c \vec{f} + b_c}}{\sum_{c'} e^{\vec{w}_{c'} \vec{f} + b_{c'}}}$$



  - Predicted label: $\bar{y}^{(i)} = argmax(O^{(i)})$
  - *Where:*
    - θ*: set of parameters*
    - $\boldsymbol{W}_O$, $\vec{b}_O$*: weight and bias parameters of output layer*

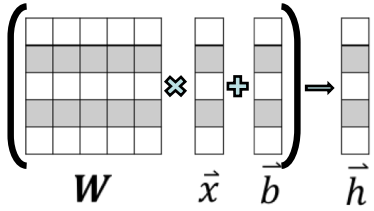
Outline

- Introduction
- Deep Learning Libraries
- **Recurrent Neural Networks**
  - Preview of Neural Networks
  - **Feature Layers**
  - Training Algorithms
- Language Models
- Assignment

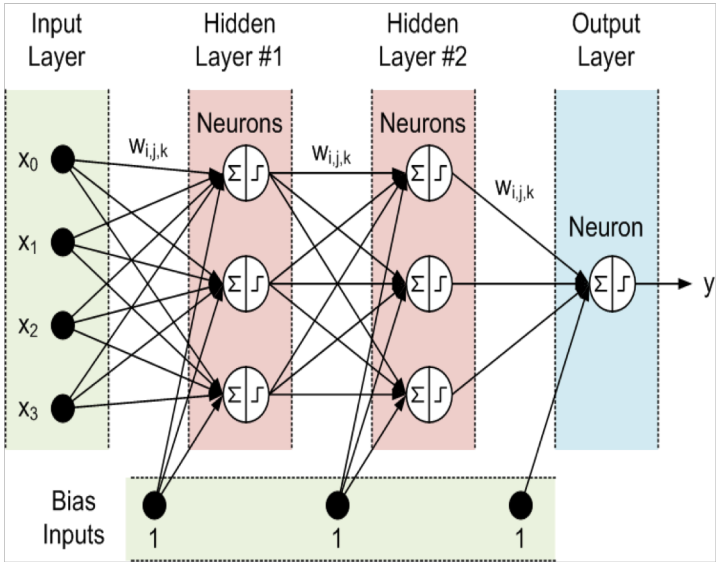
Feature Layers

- Feed Forward (MLP)

$$\vec{z} = \boldsymbol{W}\vec{x} + \vec{b}$$
$$\vec{h} = f(\vec{z})$$



  - Where:
    - $\vec{h}$: hidden features
    - $f(z)$: activation function
    - $\boldsymbol{W}, \vec{b}$: *weight and bias parameters of MLP*
    - $\vec{x}$: input vector



Source: https://www.mql5.com/pt/code/9002

Feature Layers

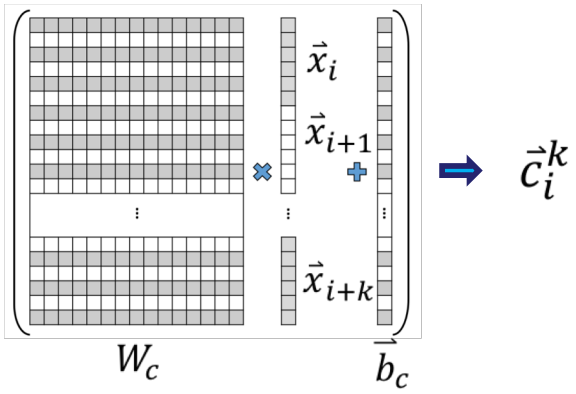
- *Acitivation functions* $f(z)$
  - $sigmoid(z) = \frac{1}{1+e^{-z}}$
  - $tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
  - $arctan(z) = tan^{-1}(z)$
  - $relu(z) = \max(0, z)$

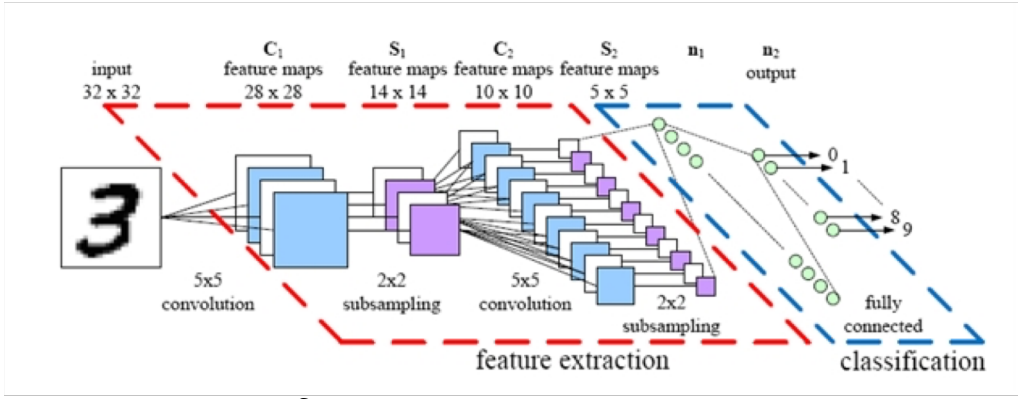
Feature Layers

- Convolutional neural network (CNN)

$$\vec{c}_i^k = f(\boldsymbol{W}_c(\vec{x}_i \oplus \vec{x}_{i+1} \oplus \ldots \oplus \vec{x}_{i+k-1}) + \vec{b}_c)$$

  - Where:
    - $\vec{c}_i^k$: convolutional features
    - $f(z)$: activation function
    - $\boldsymbol{W}_c$, $\vec{b}_c$: *weight and bias parameters of CNN*
    - $\vec{x}_i$: input vectors
    - k: window size (2,3 in common)
    - $\oplus$: concatenation





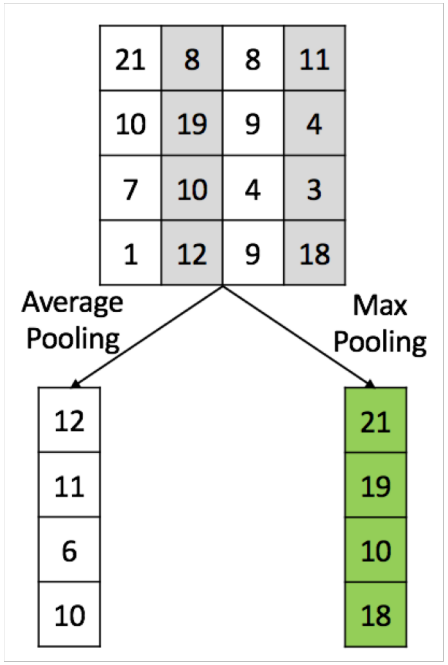
Source:
http://parse.ele.tue.nl/education/cluster2

Feature Layers

- Pooling

$$\vec{h}_i = pool(\boldsymbol{C}_i)$$

  - Where:
    - $\vec{h}_i$: hidden features
    - $pool$ is element-wise operations (max, average, min,...)
    - $\boldsymbol{C}_i$: input matrix

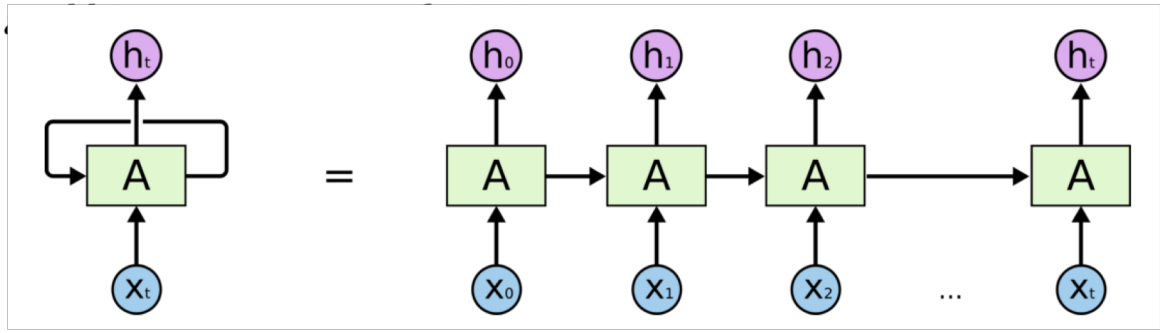


Feature Layers

- Recurrent Neural Network (RNN)

$$\vec{z}_i = \boldsymbol{W}_h \vec{h}_{i-1} + \boldsymbol{W}_x \vec{x}_i + \vec{b}$$
$$\vec{h}_i = f(\vec{z}_i)$$

  - Where:
    - $\vec{h}_i$: hidden features at time i
    - $f(z)$: activation function
    - $\boldsymbol{W}_h, \boldsymbol{W}_x, \vec{b}$: *weight*
    - $\vec{x}_i$: input vector



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Outline

- Introduction
- Deep Learning Libraries
- **Recurrent Neural Networks**
  - Preview of Neural Networks
  - Feature Layers
  - **Training Algorithms**
- Language Models
- Assignment

Training Algorithms

- Supervised Learning
- Randomly initialized model
- Compare model output with manual reference

Training Algorithms

- Loss functions
  - Cross Entropy Loss (Maximum Likelihood)

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i} p_i \log(q_i) = -\frac{1}{N}\sum_{i=1}^{N} l_{y^{(i)}} \log(O^{(i)})$$

    - Where:
      - θ: *set of parameters*
      - N: *number of samples*
      - $l_{y^{(i)}}$: one-hot vector corresponding to label $y^{(i)}$
      - $O^{(i)}$: *probability output of sample* $x^{(i)}$

Training Algorithms

- Loss functions
  - Hinge loss (maximum-margin)
    - Binary classification:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y^{(i)} {f_{score}}^{(i)})$$

    - Multiclass classification

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \max(0, 1 + \max_{c' \neq c} {f_{score}}_{c'} - {f_{score}}_{c})$$

    - Where:
      - θ: *set of parameters*
      - N: *number of samples*
      - $y^{(i)} \epsilon \{-1, 1\}$
      - f_{score}: margin output

Training Algorithms

- Loss functions
  - 0/1 Loss (large margin)

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} I_{y_i \neq \hat{y}_i}$$

    - Where:
      - θ: *set of parameters*
      - N: *number of samples*
      - I: indication function
      - y: ground-true labeled vector
      - $\hat{y}$: predicted vector

Training Algorithms

- Loss functions
  - MSE Loss (regression)

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

    - Where:
      - θ*: set of parameters*
      - N*: number of samples*
      - y is a ground-true labeled vector
      - $\hat{y}$ is a predicted vector

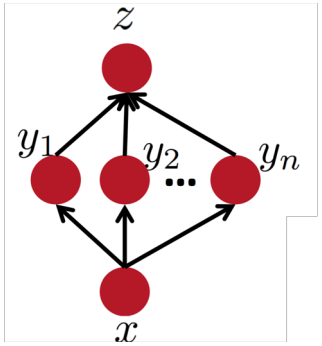
Training Algorithms

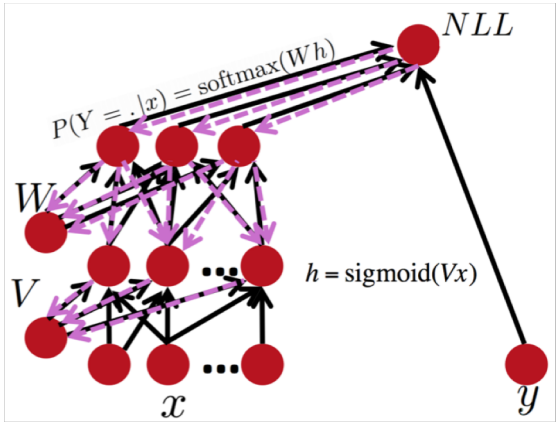
- Back Propagation
  - Goal
    - Find $\frac{\partial \mathcal{L}}{\partial \theta}$ for all parameters
    - Adjust parameters accordingly
  - Derivation
    - Chain Rule: if z= $f(y)$ and y= $g(x)$, then

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

    - Layer-wise calculation

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$





Training Algorithms

- Batch gradient descent is an algorithm in which we repeatedly make small steps downward on an error surface defined by a loss function of a set of parameters over the full training set (N samples)

$$\theta^{k+1} = \theta^{k} - \eta \frac{\partial \mathcal{L}(\theta)}{\partial \theta}$$

  - Where
    - θ *: set of parameters*
    - η: learning rate

➔ Problem: N is a very large number

Training Algorithms

- SGD: Stochastic gradient descent works according to the same principles as batch gradient descent, but proceeds more quickly by estimating the gradient from just one example at a time instead of the entire training set

$$\theta^{k+1} = \theta^k - \eta \frac{\partial \mathcal{L}(\theta, x^{(i)}, y^{(i)})}{\partial \theta}$$

- Mini-batch SGD (MSGD) works identically to SGD, except that we use more than one training example to make each estimate of the gradient

$$\theta^{k+1} = \theta^k - \eta \frac{\partial \mathcal{L}\left(\theta, x^{(i:i+n)}, y^{(i:i+n)}\right)}{\partial \theta}$$

- Problem: manually adjust learning rate

Training Algorithms

- Momentum: helps to accelerate SGD in the relevant direction by adding a fraction γ of the update vector of the past time step to the current update vector

$$v_k = \gamma v_{k-1} - \eta \frac{\partial \mathcal{L}(\theta, x^{(i)}, y^{(i)})}{\partial \theta}$$

$$\theta^{k+1} = \theta^k - v_k$$

Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. In Proceedings of Neural Networks, 145-151.

Training Algorithms

- AdaGrad: adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters

$$\theta^{k+1} = \theta^{k} - \eta^{k} g^{k}$$

  - Where:
    - g^k: the gradient of $\mathcal{L}$ w.r.t θ at k
    - $\eta^k = \frac{\eta}{\sqrt{\sum_{\tau=1}^{k} g_\tau^2 + \varepsilon}}$
    - ε: a smoothing term that avoids division by zero

➔ Problem: learning rate need to be initialized and gradually shrunk to an infinitesimally small number

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. In Proceeding of *The Journal of Machine Learning Research* 12, 2121-2159.

Training Algorithms

- RMSprop*: adjusts the Adagrad method in a very simple way in an attempt to reduce its aggressive, monotonically decreasing learning rate. In particular, it uses a moving average of squared gradients instead

$$\theta^{k+1} = \theta^k + \Delta\theta^k,$$
$$\Delta\theta^k = -\frac{\eta}{RMS[g]_k} g_k$$

  - Where:
    - RMS: root mean square
    - $RMS[g]_k = \sqrt{E[g^2]_k + \varepsilon}$, $E[g^2]_k = \rho E[g^2]_{k-1} + (1-\rho) g_k^2$

*currently unpublished adaptive learning rate method. However, it is usually to cite slide 29 of Lecture 6 of Geoff Hinton's Coursera class.

Training Algorithms

- AdaDelta: is an extension of Adagrad to handle the problem of continual decay of learning rates.

$$\theta^{k+1} = \theta^k + \Delta\theta^k,$$

$$\Delta\theta^k = -\frac{RMS[\Delta\theta]_{k-1}}{RMS[g]_k} g_k$$

  - Where:
    - RMS: root mean square
    - $RMS[\Delta\theta]_{k-1} = \sqrt{E[\Delta\theta^2]_{k-1} + \varepsilon}$, $E[\Delta\theta^2]_{k-1} = \rho E[\Delta\theta^2]_{k-2} + (1-\rho)\Delta\theta_{k-1}^2$
    - $RMS[g]_k = \sqrt{E[g^2]_k + \varepsilon}$, $E[g^2]_k = \rho E[g^2]_{k-1} + (1-\rho)g_k^2$

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Training Algorithms

- Adaptive Moment Estimation (ADAM): is another method that computes adaptive learning rates for each parameter. It is similar to RMSProp with momentum. The simplified ADAM update looks as follows

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k$$
$$v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2$$
$$\theta^{k+1} = \theta^k - \eta \frac{m_k}{\sqrt{v_k} + \varepsilon}$$

Diederik Kingma, and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Training Algorithms

- Regularization
  - L2: $J(\theta) = \mathcal{L}(\theta) + \lambda \|\theta\|_2^2$
    - Where:
      - λ: regularisation weight
  - Dropout: $\vec{f}' = \vec{f} \circ r$
    - Where:
      - r: a masking vector of Bernoulli random variables with probability p of being 1
- Challenges:
  - Hyper-parameters
  - OOV, fine-tune embedding, etc.

Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- **Language Models**
  - Overview
  - Scoring LM
  - Probabilistic LM
- Assignment

Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- **Language Models**
  - **Overview**
  - Scoring LM
  - Probabilistic LM
- Assignment

Overview

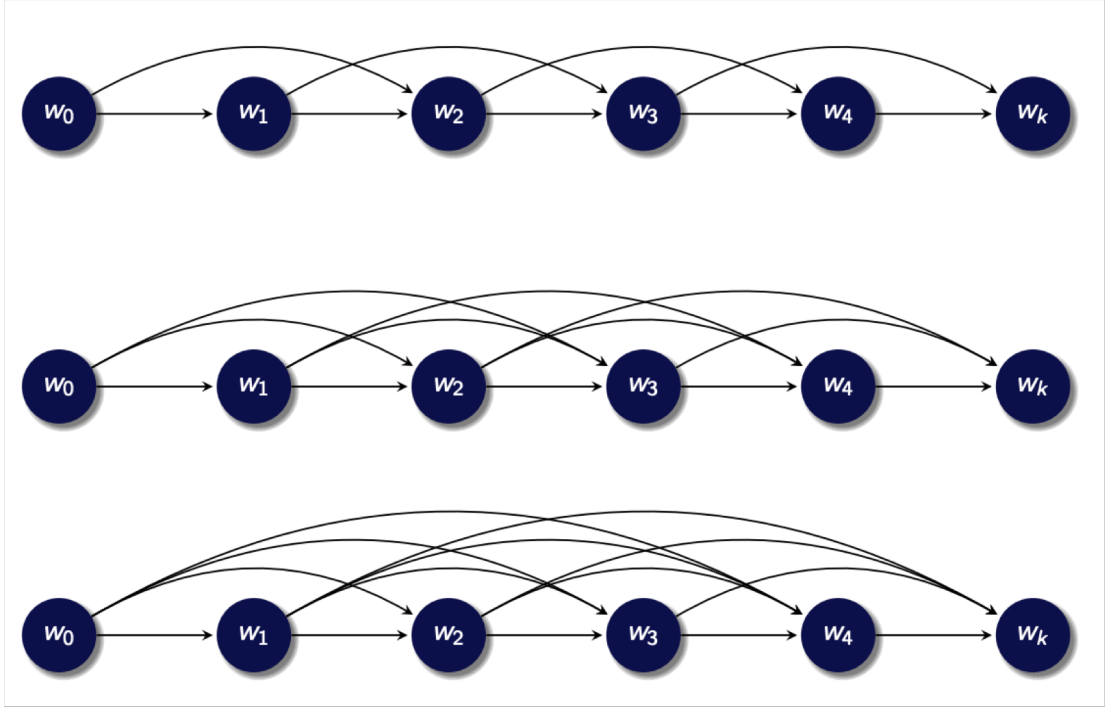
- Statistical Language Model

$$
\begin{aligned}
& P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) \\
= \; & P(X_1 = x_1) \times P(X_2 = x_2 | X_1 = x_1) \\
& \times \prod_{i=3}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \\
= \; & \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
\end{aligned}
$$

$$
P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})
$$

$$
= q(x_i | x_{i-2}, x_{i-1})
$$

$$
q(w_i \mid w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}
$$



Source: https://jon.dehdari.org/tutorials/lm_overview.pdf

Overview

- Evaluating metric:
  - Perplexity

- We have some test data, m sentences

$$s_1, s_2, s_3, \ldots, s_m$$

- We could look at the probability under our model $\prod_{i=1}^{m} p(s_i)$. Or more conveniently, the *log probability*

$$\log \prod_{i=1}^{m} p(s_i) = \sum_{i=1}^{m} \log p(s_i)$$

- In fact the usual evaluation measure is *perplexity*

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

and M is the total number of words in the test data.

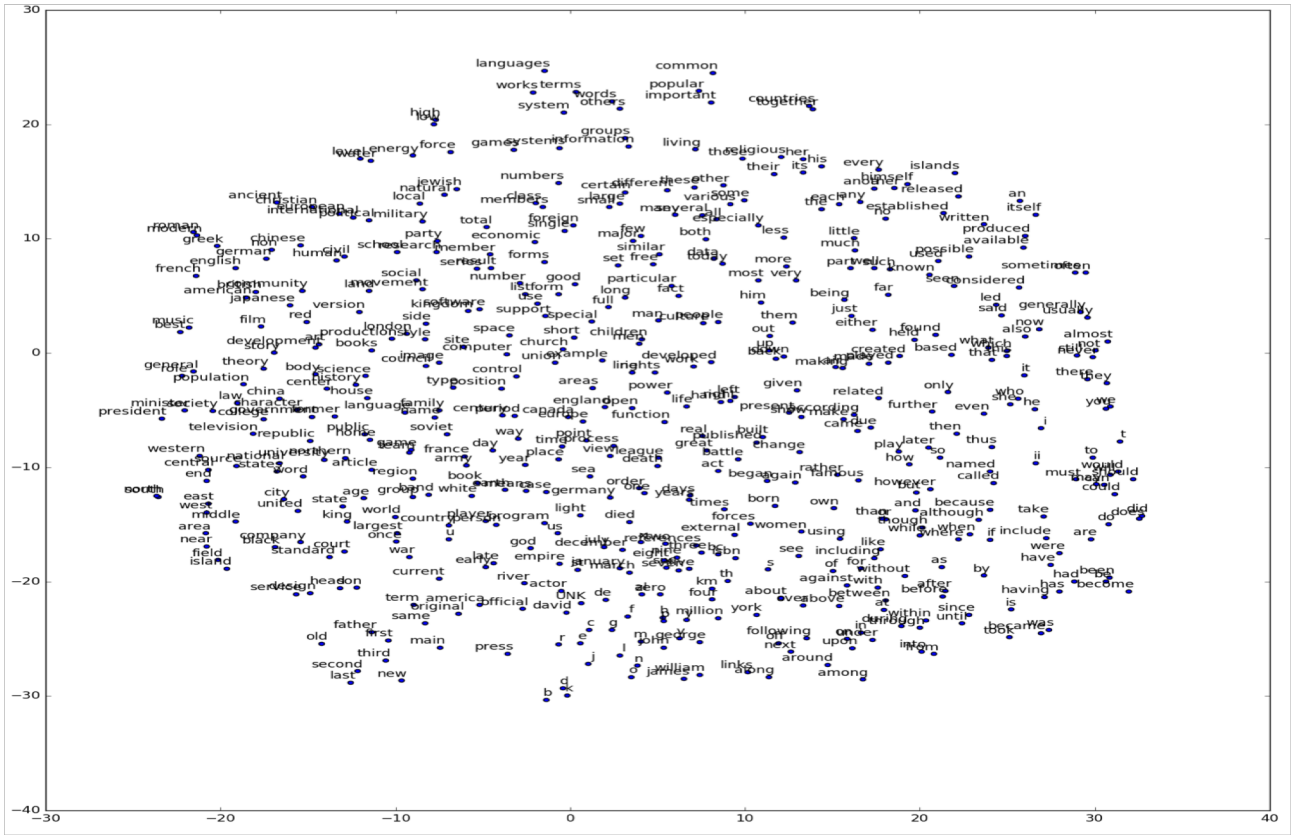
Source: http://www.cs.columbia.edu/~mcollins/cs4705-fall2018/slides/lmslides.pdf

Overview

- Applications:
  - Google suggest
  - Machine translation
  - Assisting people with motor disabilities. For example, Dasher
  - Speech Recognition (ASR)
  - Optical character recognition (OCR) and handwriting recognition
  - Information retrieval / search engines
  - Data compression
  - Language identification, as well as genre, dialect, and idiolect identification (authorship identification)
  - Software keyboards
  - Surface realization in natural language generation
  - Password cracking
  - Cipher cracking

Overview

- Neural network LM



Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- **Language Models**
  - Overview
  - **Scoring LM**
  - Probabilistic LM
- Assignment

Scoring LM

- Unsupervised Learning
  - Pairwise-ranking neural network language models
    - Input: a pair of
      - n-grams:

$$t = w_{i-k} w_{i-k+1} \dots \boldsymbol{w_i} \dots w_{i+k-1} w_{i+k}$$

      - corrupted n-grams:

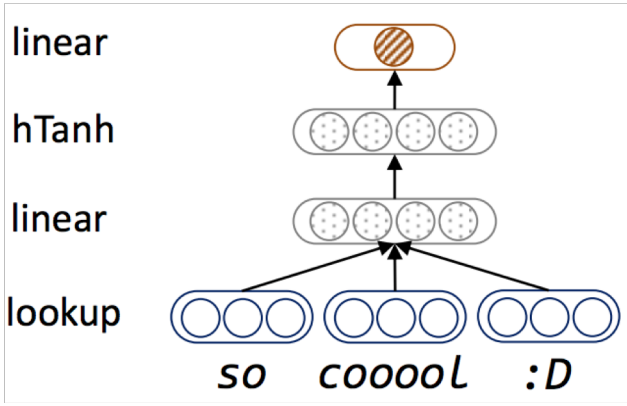
$$t^r = w_{i-k} w_{i-k+1} \dots \boldsymbol{w_i^r} \dots w_{i+k-1} w_{i+k}$$

    - Output:
      - margin scores $f(t), f(t^r)$
  - Objective function

$$\boldsymbol{loss_{cw}(t, t^r) = \max(0, 1 + f(t^r) - f(t))}$$

➔ Problem: Deep structure

➔ Still high computation



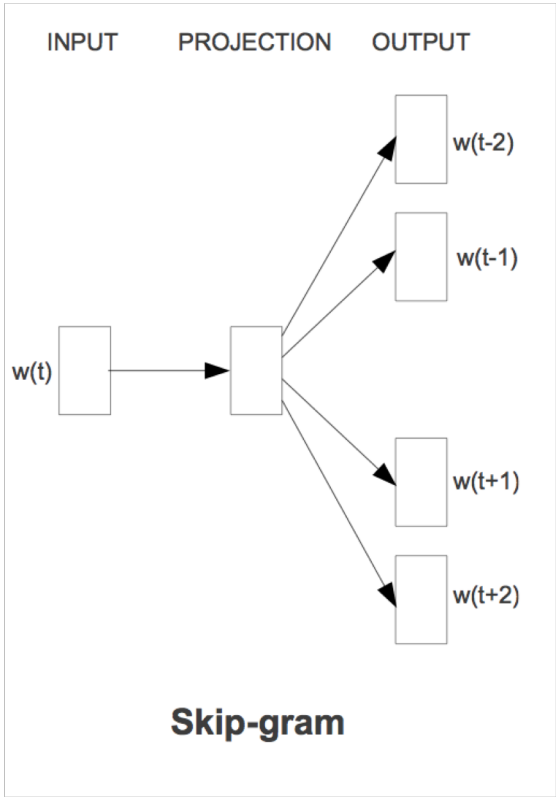
Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *J. Mach. Learn. Res.* 12, 2493-2537.

Scoring LM

- Unsupervised Learning
  - Simple neural network language models
    - Input:
      - target word /n-grams context
    - Output:
      - probability score of the context words given a word or vice versa
  - Objective function

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, c\neq 0} logP(w_{t+j}|w_t)$$

  - Optimization
    - Hierarchical softmax
    - Negative sampling



Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In Proceedings of *NIPS*, 3111-3119.

Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- **Language Models**
  - Overview
  - Scoring LM
  - **Probabilistic LM**
- Assignment

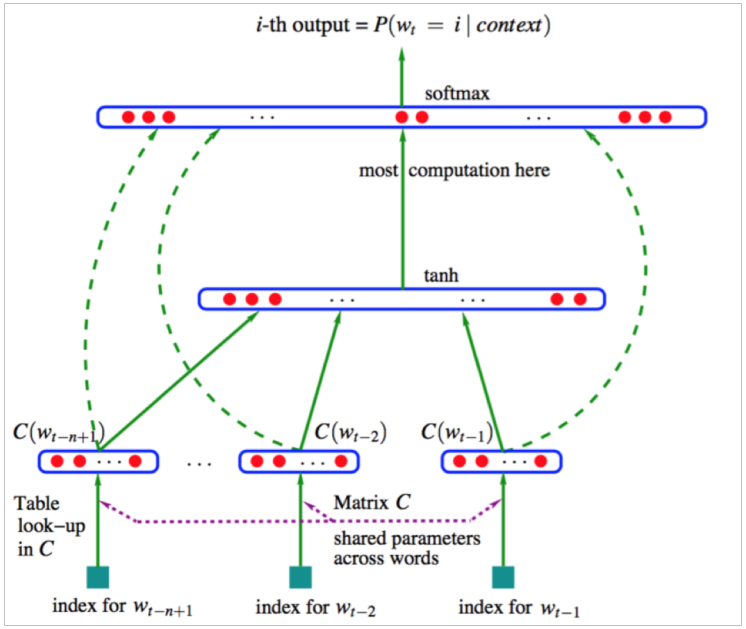
Probabilistic LM

- Unsupervised Learning
  - Basic neural network language models:
    - Input:
      - n-grams
    - Output:
      - probability score of the word given previous words
  - Objective function

$$P\left(w_t \middle| w_1^{t-1}\right) \approx P\left(w_t \middle| w_{t-n+1}^{t-1}\right)$$

➔ Problem: probability score

➔ High computation



$$x = (C(w_{t-1}), C(w_{t-2}), \cdots, C(w_{t-n+1}))$$

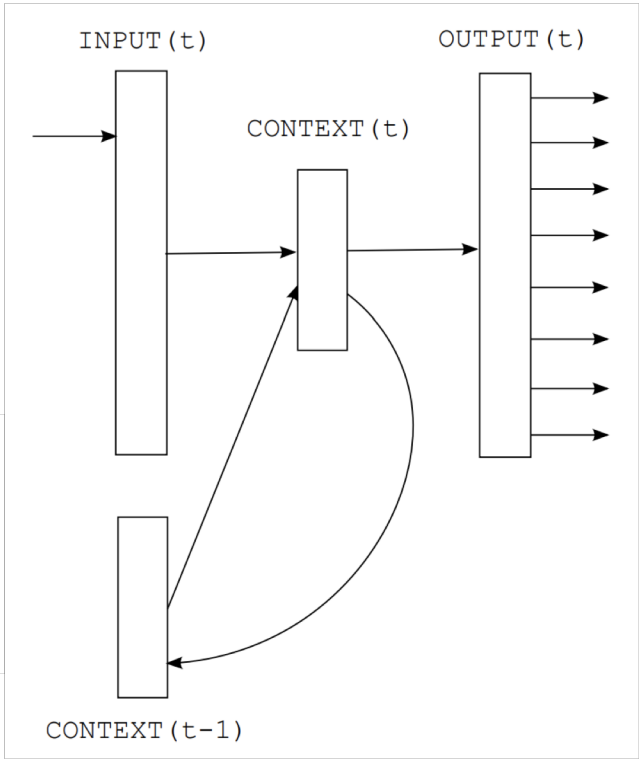
$$y = b + Wx + U\tanh(d + Hx)$$

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3, 1137-1155.

Probabilistic LM

- Similar to Bengio et al. (2003) on:
  - Input layer
  - Output layer
- Different at the feature layer
  - RNN

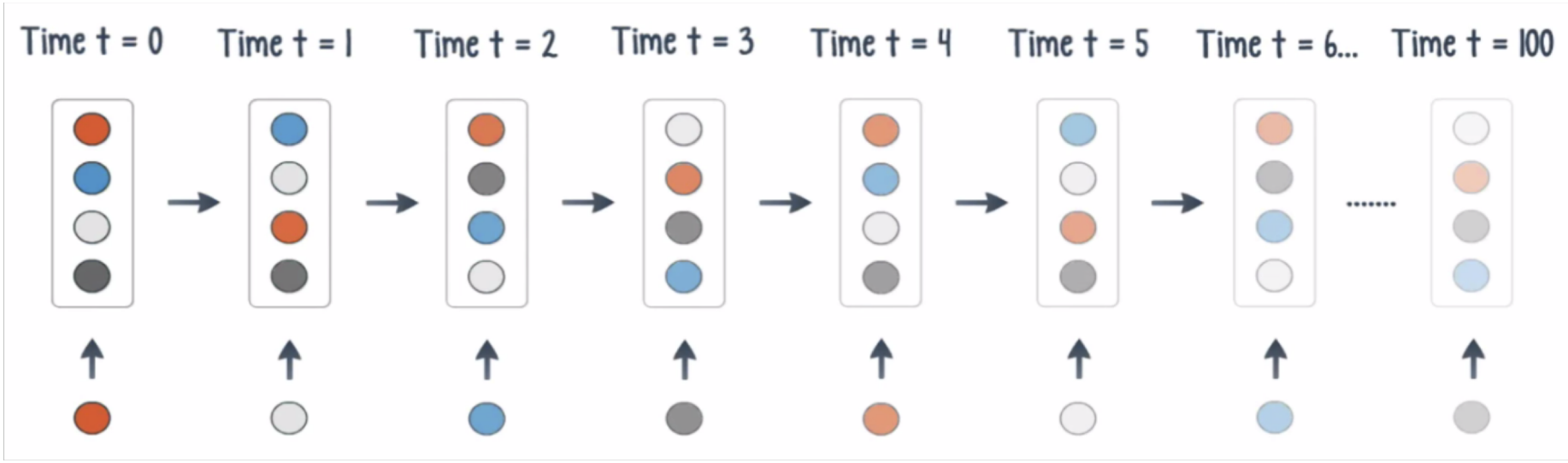
$$\vec{z}_i = \boldsymbol{W}_h \vec{h}_{i-1} + \boldsymbol{W}_x \vec{x}_i + \vec{b}$$
$$\vec{h}_i = f(\vec{z}_i)$$



Tomáš Mikolov, et al. "Recurrent neural network based language model." *Eleventh Annual Conference of the International Speech Communication Association*. 2010.

Probabilistic LM

- Problem: The Vanishing Gradient
- Solution: Backpropagation through time (BPTT)



Source: https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257

Outline

- Introduction
- Deep Learning Libraries
- Recurrent Neural Networks
- Language Models
- **Assignment**

Assignment

- Build RNN language model
  - Training phase:
    - Write logic code for RNN
    - Train the model using yelp reviews (data will be provided)
    - Measure the perplexity score
  - Inference phase:
    - Predict the next work given current context
    - Generate random reviews

Q&A