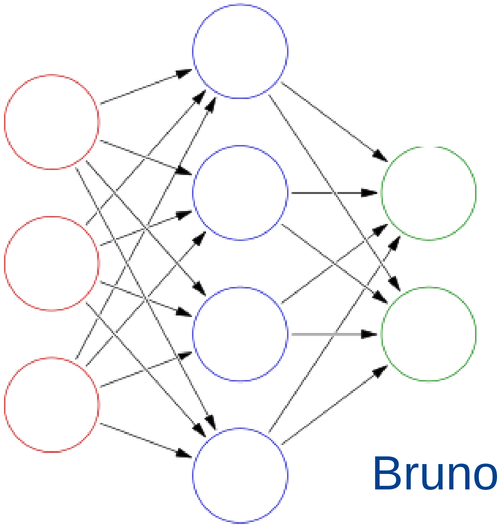
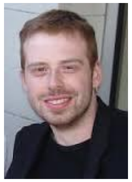
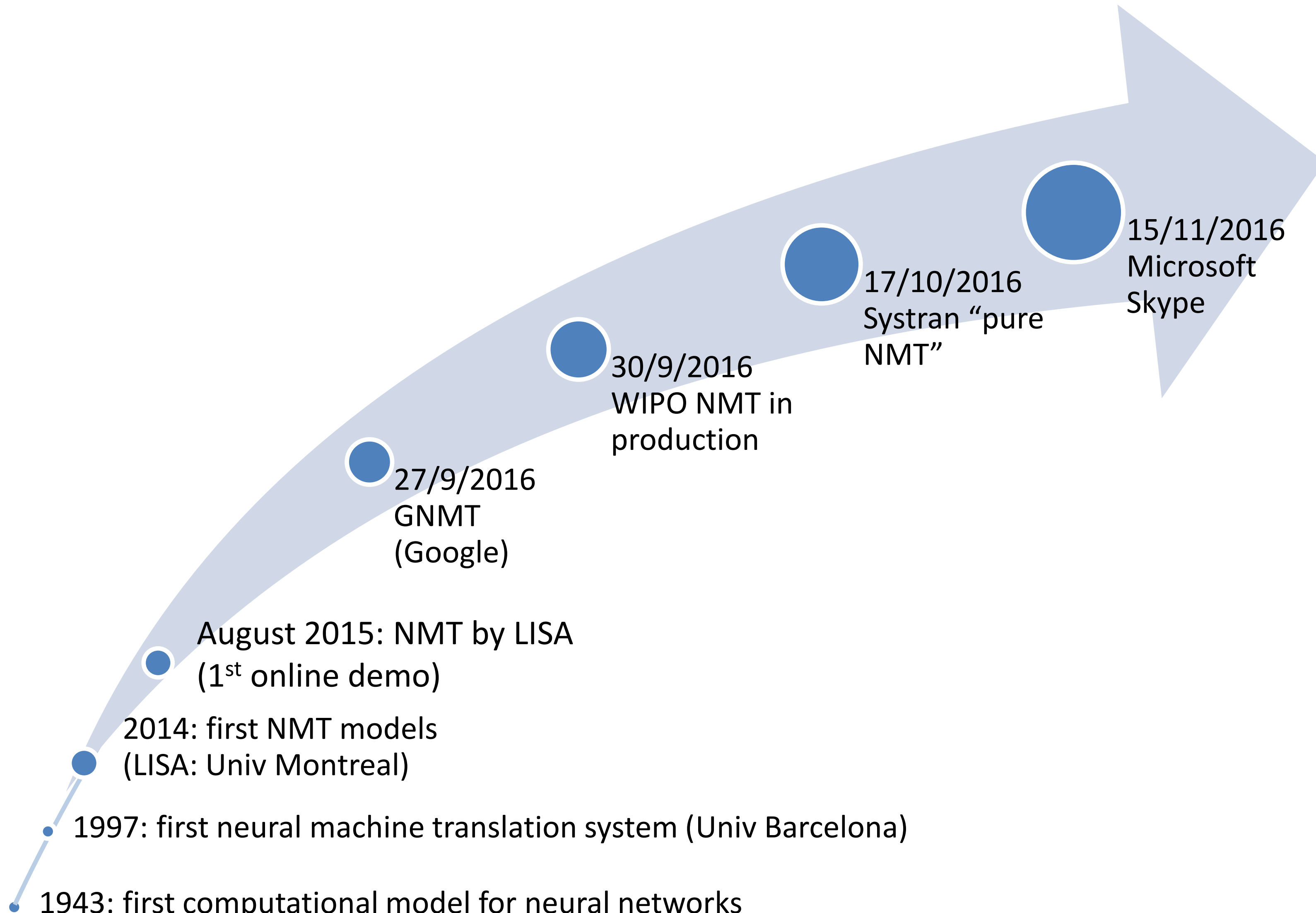
Lecture 7: Neural Machine Translation



Bruno Pouliquen
+ Marcin Junczys-Dowmunt



(quick) history of NMT



New trend in R&D in MT

Look at the Machine Translation Marathon organized this year

http://ufal.mff.cuni.cz/mtm16/programme.html

MT Marathon 2016 Programme

Going Neural this year!

Programme

We are still processing the videorecordings of the lectures and keynote talks. Once finished, they will be linked from this page.

Monday	**September 12**
9.00-10.30	**Lecture:** MT Evaluation **and** Significance Testing
S9	Lucia Specia (USFD), Yvette Graham (DCU)
11.00-12.00	**Keynote:** Neural Networks in MT: Past, Present and Future (historical video)
S9	Holger Schwenk (Facebook AI Research)
12.00-12.30	**Project Proposals**
S9	see the live list of proposed projects
	and Boaster slides (username and password: mtm)
14.00-15.30	**Lecture: Introduction to Neural Networks:** Linear Regression, Logistic Regression (slides source)
S9	Marcin Junczys-Dowmunt (AMU)
16.00-17.00	**Exercise:** Introduction to Theano (slides source)
SU2	Marcin Junczys-Dowmunt (AMU)

Tuesday	**September 13**
9.00-10.30	**Lecture:** Training Neural Networks, Backpropagation (slides source)
S9	Marcin Junczys-Dowmunt (AMU)
11.00-12.00	**Keynote:** Directed MT Research for Commercial Settings
S9	Adrià de Gispert (SDL Research and Cambridge University)
14.00-15.30	**Lab: Theano on CPUs** (continuation from Monday)
SU2	Marcin Junczys-Dowmunt (AMU)
16.00-17.00	**Lab:** Amazon EC2, SGE and GPU Warm-up; **Wiki:** EC2, ÚFAL cluster
SU2	Tomáš Musil (CUNI)

Tuesday	**September 13**
9.00-10.30	**Lecture:** Training Neural Networks, Backpropagation (slides source)
S9	Marcin Junczys-Dowmunt (AMU)
11.00-12.00	**Keynote:** Directed MT Research for Commercial Settings
S9	Adrià de Gispert (SDL Research and Cambridge University)
14.00-15.30	**Lab: Theano on CPUs** (continuation from Monday)
SU2	Marcin Junczys-Dowmunt (AMU)
16.00-17.00	**Lab:** Amazon EC2, SGE and GPU Warm-up; **Wiki:** EC2, ÚFAL cluster
SU2	Tomáš Musil (CUNI)

Wednesday	**September 14**
9.00-10.00	**Lecture:** N-Gram Language Modelling, including Feed-Forward NNs
S9	Kenneth Heafield (UEDIN)
10.00-10.30	**Lecture:** Word Embeddings, Introduction to Recurrent NNs
S9	David Vilar Torres (Nuance)
11.00-12.00	**Lecture:** Advanced Recurrent NNs (Backpropagation in Time, LSTM, ...)
S9	David Vilar Torres (Nuance)
14.00-15.30	**Lab:** Character-Level LMs in Practice (supplementary files)
SU2	David Vilar Torres (Nuance)
16.00-17.00	**Project mid-week reports**
S9	please commit to the SVN and browse here (username and password: mtm)
18.00-00.00	**Social event**
	Letná beer garden

Thursday	**September 15**
9.00-10.30	**Lecture:** Neural Machine Translation
S9	Rico Sennrich (UEDIN)
11.00-12.00	**Keynote:** Future Directions in Neural Machine Translation (motivating video)
S9	Orhan Firat (Middle East Technical University), Kyunghyun Cho (New York University)
12.00-12.20	**Boaster Session for Posters**
S9	see Poster Boasters (username and password: mtm)
14.00-15.30	**Lab:** Nematus (handout source for copy-paste)
SU2	Rico Sennrich (UEDIN)

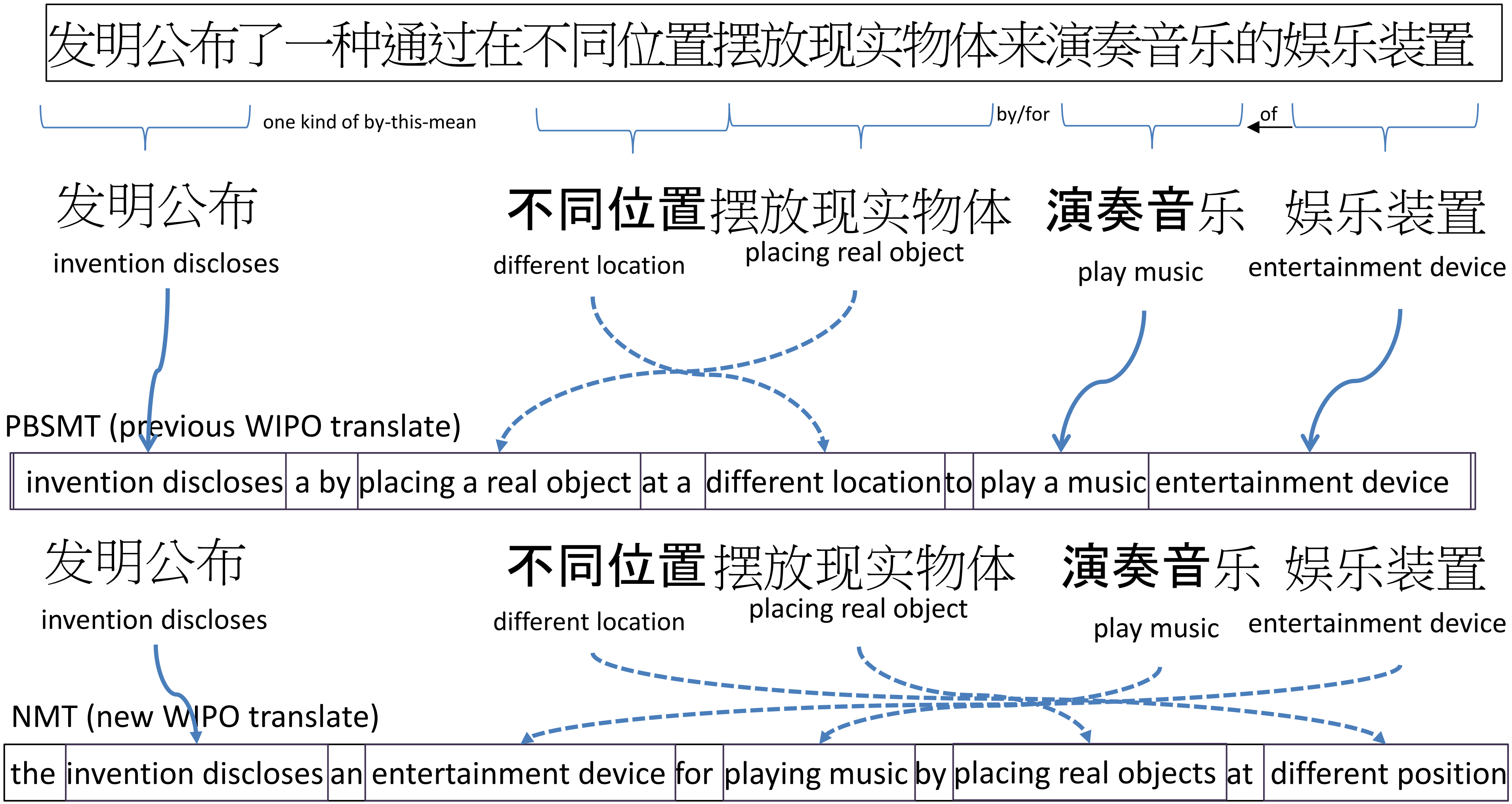
What is common/different? (between PBSMT and NMT)

- Common input/output

  Another machine learning method (bitexts as input / translation model as output)

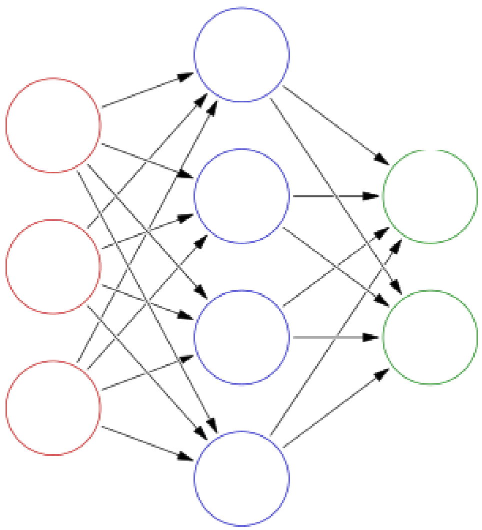
- Differences
  - Heavy maths for NMT, needs huge processing units (GPUs)
  - NMT models can “record” long distance dependencies (PBSMT cannot)
  - NMT cannot have huge dictionary (cannot handle OOVs)
  - PBMST models are huge, NMT models are “very” small
  - PBSMT models can be improved with huge language models, NMT cannot (at least not easily)

Why is NMT different?
(Phrase-based vs Neural-net)



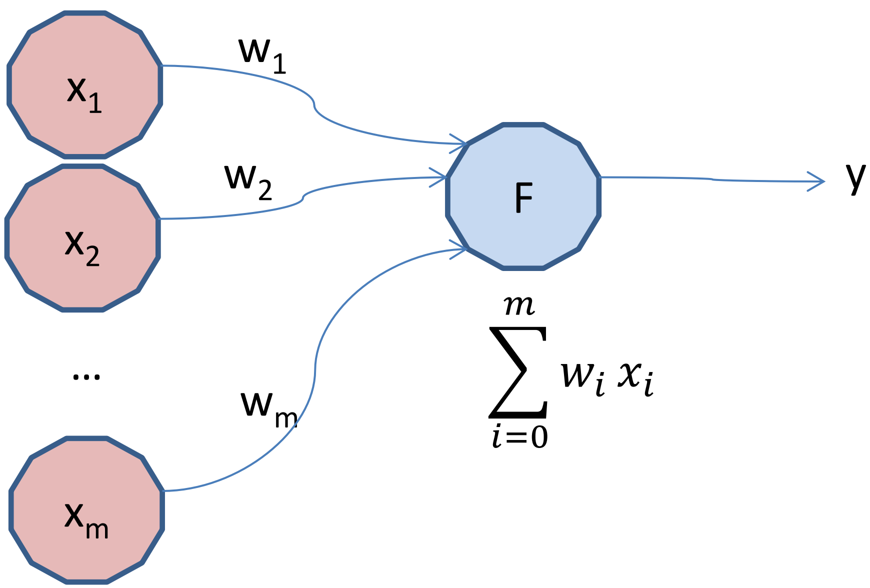
How does NMT work?

- Basic knowledge about artificial neural networks
- Perceptron
- Feed forward Neural Network
- Recurrent Neural Network
- NMT



Simplest NN: Perceptron

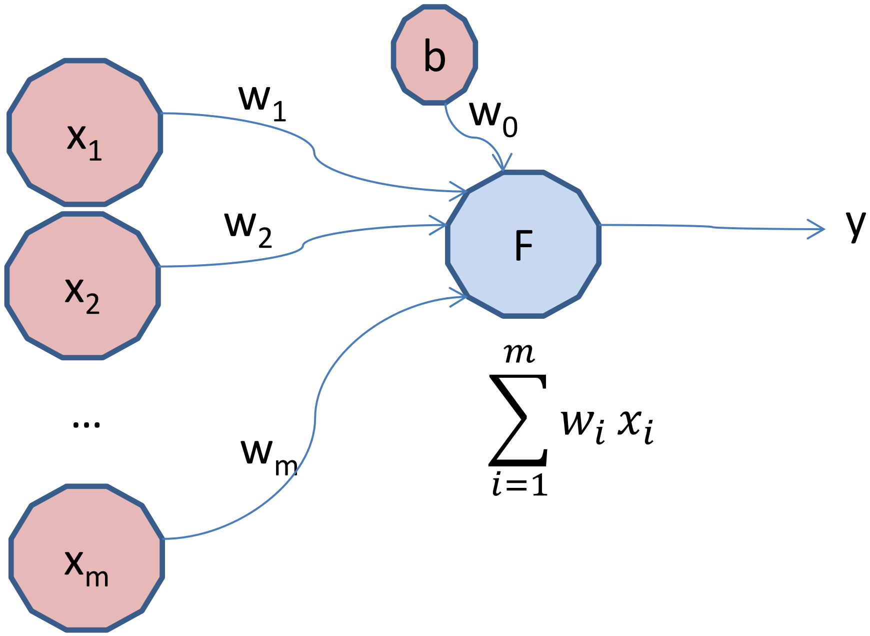
- 1957, Franck Rosenblatt.
- A linear classifier, able to output positive/negative value



$$f(x) = \begin{cases} 1, & if\ f > 0 \\ 0, & otherwise \end{cases}$$

Simplest NN: Perceptron with bias

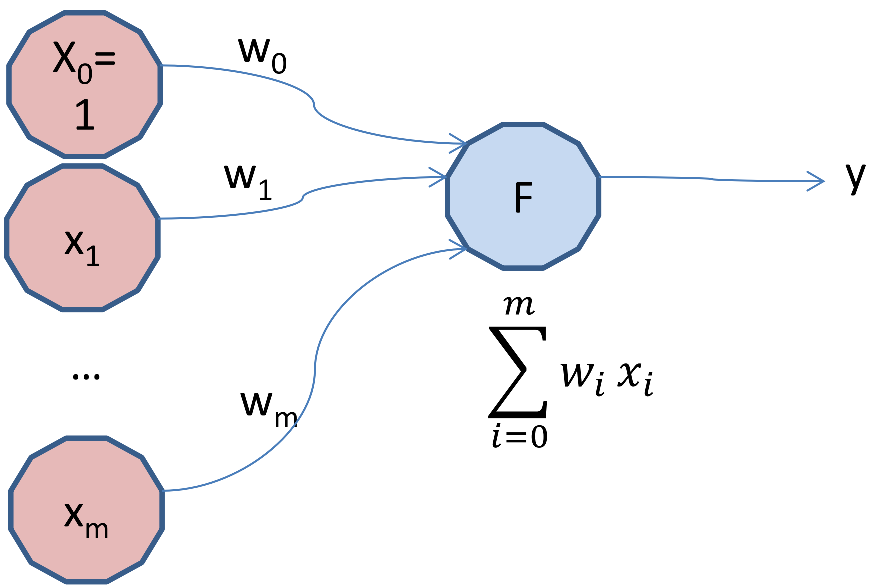
- Classifier output



$$f(x) = \begin{cases} 1, & if\ f > \mathrm{b} \\ 0, & otherwise \end{cases}$$

Simplest NN: Perceptron with bias

- Bias usually stored in X0



$$f(x) = \begin{cases} 1, & if\ f > 0 \\ 0, & otherwise \end{cases}$$

How a perceptron learns?

- Initialize randomly the weights
- Feed example(s), get the output value(s)

  $\sum_{i=0}^{m} w_i \, x_i$
- Compare with “expected” value(s)
  - “loss function”
  - e.g. $\mathrm{e} = \frac{1}{2}(predicted - expected)^{\,2}$
- Change the weights accordingly
  - Using gradient
    - Derivative of the “loss” function
    - Here: $\Delta = predicted - expected$
  - Each weight changes according to the gradient

    Should be: w_i=w_i- Δ. x_i

    but that would be too fast: hence a “learning factor” α (usually a “small” value)

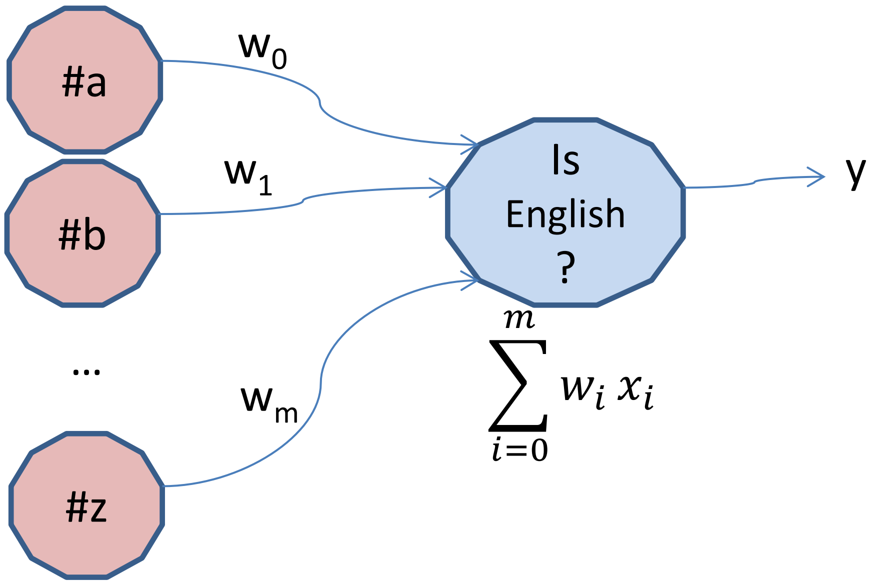
    w_i=w_i- α. Δ. x_i

Example in NLP

- Naïve language guesser
  - Take as input all characters in a text
  - Classify the text as being English or in another language (Dutch/French/Italian/Spanish…)
  - Training set: English sentences / other language sentences
  - Classifier: input a given text, output positive value if the text is English / negative otherwise

Language guesser

- For each sentence, input the number of time a character appears



$$\sum_{i=0}^{m} w_i x_i$$

$$f(x) = \begin{cases} 1, & if\ f > 0 \\ 0, & otherwise \end{cases}$$

Real example

- To be experienced during the lab:

  Run a minimum python script (40 lines) to classify on English/Spanish sentences (taken from patent domain)

  With 100 sample long sentences (half English, half Spanish) test the classifier on 10 sentences

Naïve language guesser code

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]      # the sentences read from the test file
training = []  # A matrix containing for each line the count of each char
category = []  # A vector: for each line its category 1(English)/0(other)
fileName='test.txt' # test file containing sentences (English begins with '+')
with open(fileName, 'r') as f:
   for line in f:
      lines.append(line)  # save the sentence
      training.append([line.lower().count(ch)
                   for ch in ascii_lowercase]) # add the counts
      category.append("+" in line)  # Adds the category
weights = np.random.rand(26) # initialise the weight randomly (26 x 0-1)
ntraining = np.array(training) # Create a "numpy" matrix
ncategory = np.array(catego
for example in range(100):
   input = ntraining[example
   goal_prediction = ncatego
   prediction = max(min(inpu
   error = 0.5*(goal_predicti
   delta = prediction - goal_p
   weights = weights - (alpha

   print("Ex: %d Error= %f Go
      % (example, error, goa

n = len(ntraining) - 1
for test in range(n, n - 10, -1
   input = ntraining[test]  # I
   goal_prediction = ncatego
   prediction = input.dot(we

   if (goal_prediction != pred
      print ("Test:" + str(test)
```


```
from string import ascii_lowercase

alpha = 0.01   # Learning rate
lines=[]      # the sentences read from the test file
training = []  # A matrix containing for each line the count of each char
category = []  # A vector: for each line its category 1(English)/0(other)
fileName='test.txt' # test file containing sentences (English begins with '+')

with open(fileName, 'r') as f:
   for line in f:
      lines.append(line)  # save the sentence
      training.append([line.lower().count(ch)
                   for ch in ascii_lowercase]) # add the counts
      category.append("+" in line)  # Adds the category
```

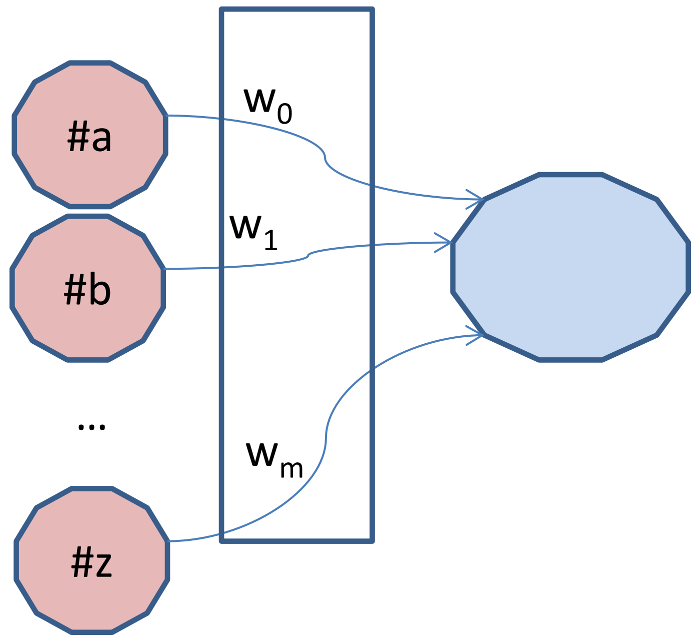
Naïve language guesser code

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]      # the sentences read from the test file
training = []  # A matrix containing for each line the count of each char
category = []  # A vector: for each line its category 1(English)/0(other)
fileName='test.txt' # test file containing sentences (English begins with '+')
with open(fileName, 'r') as f:
   for line in f:
      lines.append(line)  # save the sen
      training.append([line.lower().cou
                  for ch in ascii_lowercase
      category.append("+" in line)  # Adds the category
weights = np.random.rand(26) # initialise the weight randomly (26 x 0-1)
ntraining = np.array(training) # Create a "numpy" matrix
ncategory = np.array(category) # Create a "numpy" category vector
for example in range(100):     # for each of the 1st 140 lines (training data)
   input = ntraining[example] # Input vector: the counts for this line
   goal_prediction = ncategory[example]  # We should target this category
   prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
   error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
   delta = prediction - goal_prediction # gradient of the error
   weights = weights - (alpha * (input * delta)) # adjust the weights

   print("Ex: %d Error= %f Goal: %d Prediction: %d "
      % (example, error, goal_prediction, prediction))

n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
   input = ntraining[test]  # Input vector: the counts for this line
   goal_prediction = ncategory[test]
   prediction = input.dot(weights) > 0.5 # the computed category

   if (goal_prediction != prediction): # Here we detected the wrong category
      print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```


weights = np.random.rand(26) # initialise the weight randomly (26 x 0-1)

ntraining = np.array(training) # Create a "numpy" matrix

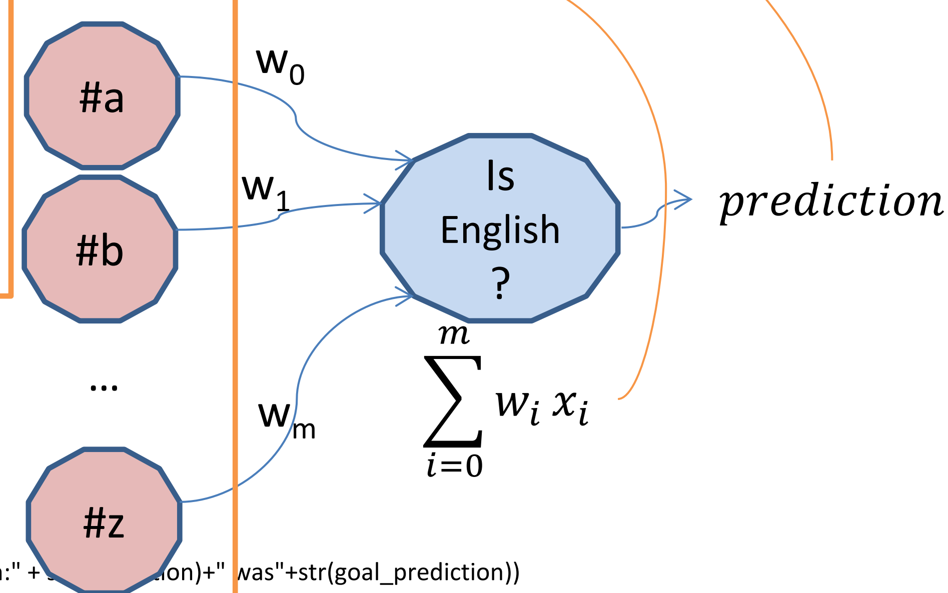
ncategory = np.array(category) # Create a "numpy" category vector



Naïve language guesser code

```
for example in range(100):     # for each of the 1st 100 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))
```

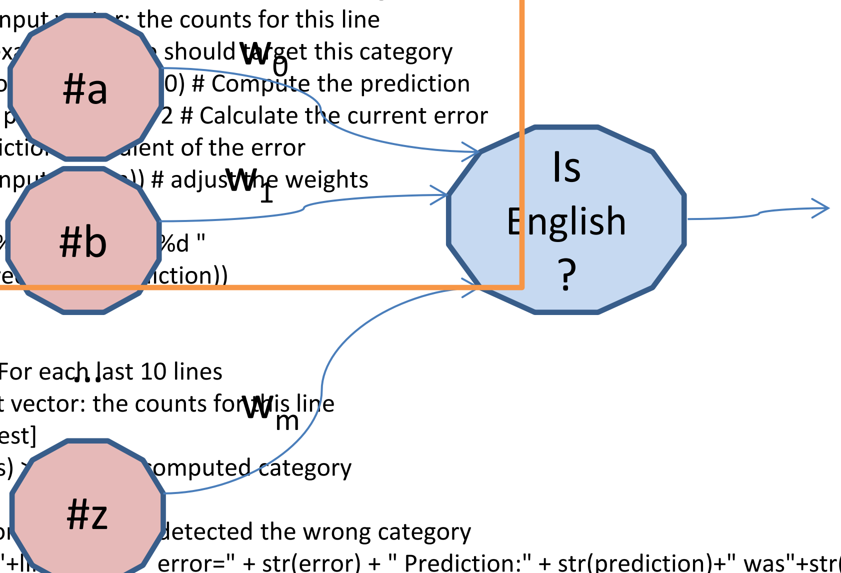
```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]      # the sentences read from
training = []  # A matrix containing fo
category = []  # A vector: for each line
fileName='test.txt' # test file contain
with open(fileName, 'r') as f:
    for line in f:
        lines.append(line)  # save the se
        training.append([line.lower().co
                for ch in ascii_lowercas
        category.append("+" in line)  # A
weights = np.random.rand(26) # initi
ntraining = np.array(training) # Creat
ncategory = np.array(category) # Cre
for example in range(100):     # for each of the 1st 140 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))

n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
    input = ntraining[test]  # Input vector: the counts for this line
    goal_prediction = ncategory[test]
    prediction = input.dot(weights) > 0.5 # the computed category

    if (goal_prediction != prediction): # Here we detected the wrong category
        print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + s    ion)+" was"+str(goal_prediction))
```



Naïve language guesser code

```
for example in range(100):     # for each of the 1st 100 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))
```

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]       # the sentences read from
training = []  # A matrix containing fo
category = []  # A vector: for each line
fileName='test.txt' # test file containi
with open(fileName, 'r') as f:
    for line in f:
        lines.append(line)  # save the se
        training.append([line.lower().co
                for ch in ascii_lowercas
        category.append("+" in line)  # A
weights = np.random.rand(26) # initi
ntraining = np.array(training) # Creat
ncategory = np.array(category) # Cre
for example in range(100):     # for each of the 1st 140 lines (training data)
    input = ntraining[example] # Input    r: the counts for this line
    goal_prediction = ncategory[ex        should target this category
    prediction = max(min(input.do         0) # Compute the prediction
    error = 0.5*(goal_prediction - p      2 # Calculate the current error
    delta = prediction - goal_predictio   ent of the error
    weights = weights - (alpha * (inpu    )) # adjust the weights

    print("Ex: %d Error= %f Goal: %       %d "
        % (example, error, goal  pre      iction))

n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
    input = ntraining[test]  # Input vector: the counts for this line
    goal_prediction = ncategory[test]
    prediction = input.dot(weights)       omputed category

    if (goal_prediction != predictio      detected the wrong category
        print ("Test:" + str(test) + " "+li    error=" + str(error) + " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```



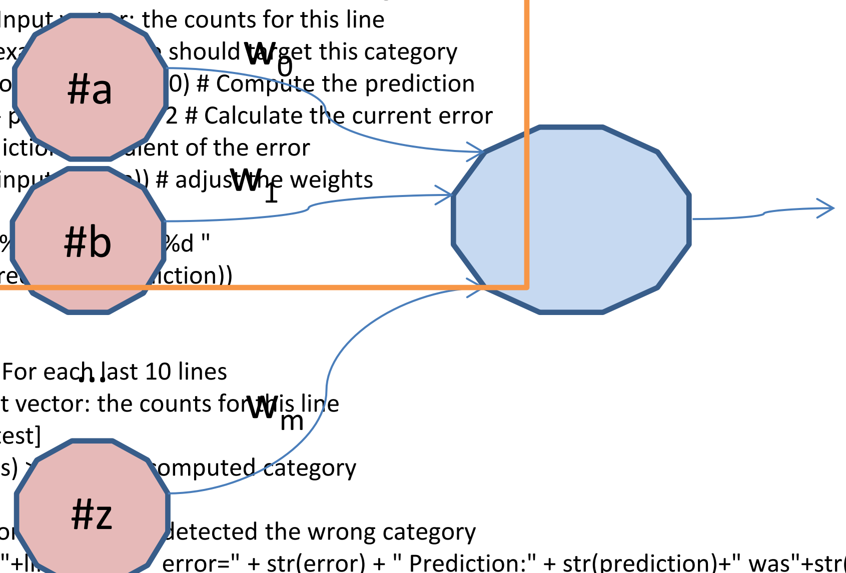
$prediction$

$$e = \frac{1}{2}(prediction - goal)^2$$

Naïve language guesser code

```
for example in range(100):     # for each of the 1st 100 lines (training data)
   input = ntraining[example] # Input vector: the counts for this line
   goal_prediction = ncategory[example]  # We should target this category
   prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
   error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
   delta = prediction - goal_prediction # gradient of the error
   weights = weights - (alpha * (input * delta)) # adjust the weights

   print("Ex: %d Error= %f Goal: %d Prediction: %d "
      % (example, error, goal_prediction, prediction))
```

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]      # the sentences read from
training = []  # A matrix containing for
category = []  # A vector: for each line
fileName='test.txt' # test file containi
with open(fileName, 'r') as f:
   for line in f:
      lines.append(line)  # save the se
      training.append([line.lower().cou
                  for ch in ascii_lowercas
      category.append("+" in line)  # A
weights = np.random.rand(26) # initi
ntraining = np.array(training) # Creat
ncategory = np.array(category) # Cre
for example in range(100):     # for each of the 1st 140 lines (training data)
   input = ntraining[example] # Input v  r: the counts for this line
   goal_prediction = ncategory[ex        should target this category
   prediction = max(min(input.do        0) # Compute the prediction
   error = 0.5*(goal_prediction - p       2 # Calculate the current error
   delta = prediction - goal_predictio    ient of the error
   weights = weights - (alpha * (inpu     )) # adjust the weights

   print("Ex: %d Error= %f Goal: %        %d "
      % (example, error, goal  pre        iction))

n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
   input = ntraining[test]  # Input vector: the counts for this line
   goal_prediction = ncategory[test]
   prediction = input.dot(weights) >     omputed category

   if (goal_prediction != predictio       detected the wrong category
      print ("Test:" + str(test) + " "+li       error=" + str(error) + " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

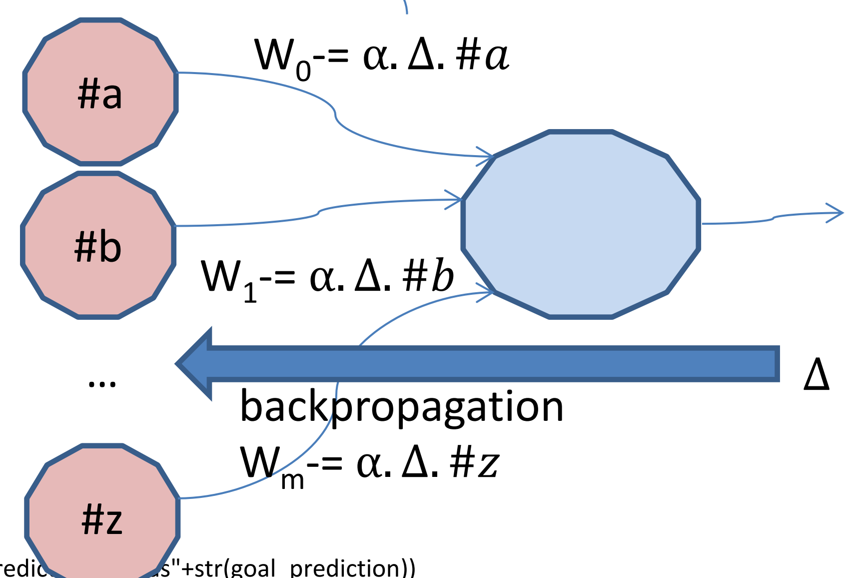


$$\Delta = \left(\frac{1}{2}(prediction - goal)^2\right)'$$

Naïve language guesser code

```
for example in range(100):     # for each of the 1st 100 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))
```

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]       # the sentences read from
training = []  # A matrix containing fo
category = []  # A vector: for each line
fileName='test.txt' # test file containi
with open(fileName, 'r') as f:
    for line in f:
        lines.append(line)  # save the se
        training.append([line.lower().cou
                for ch in ascii_lowercas
        category.append("+" in line)  # A
weights = np.random.rand(26) # initi
ntraining = np.array(training) # Creat
ncategory = np.array(category) # Cre
for example in range(100):     # for each of the 1st 140 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))

n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
    input = ntraining[test]  # Input vector: the counts for this line
    goal_prediction = ncategory[test]
    prediction = input.dot(weights) > 0.5 # the computed category

    if (goal_prediction != prediction): # Here we detected the wrong category
        print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + str(predic
s"+str(goal_prediction))
```

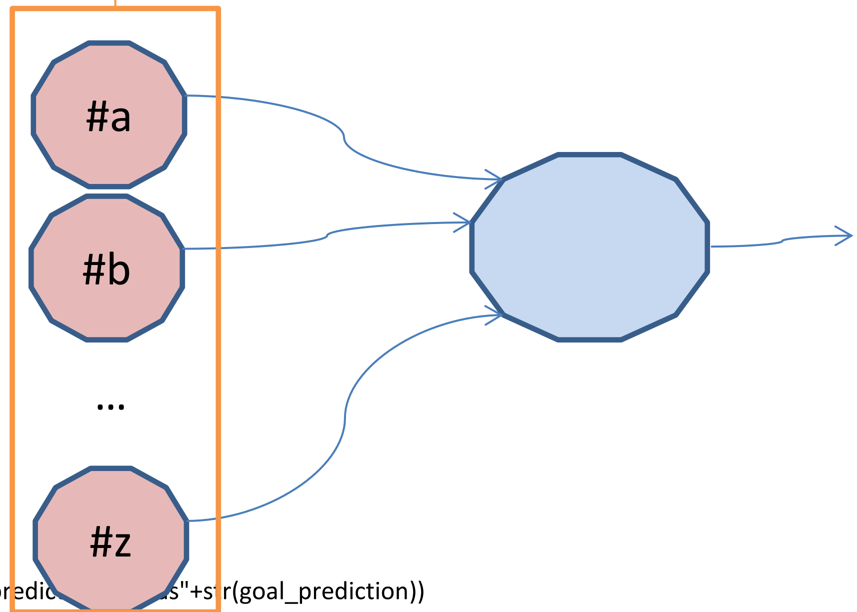


Naïve language guesser code

```
n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
   input = ntraining[test]  # Input vector: the counts for this line
   goal_prediction = ncategory[test]
   prediction = input.dot(weights) > 0.5 # the computed category

   if (goal_prediction != prediction): # Here we detected the wrong category
      print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) +
             " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

```
•   from string import ascii_lowercase
•   alpha = 0.01   # Learning rate
•   lines=[]      # the sentences read from
•   training = []  # A matrix containing for
•   category = []  # A vector: for each line
•   fileName='test.txt' # test file containi
•   with open(fileName, 'r') as f:
•      for line in f:
•         lines.append(line)  # save the ser
•         training.append([line.lower().cou
•                  for ch in ascii_lowercas
•         category.append("+" in line)  # A
•   weights = np.random.rand(26) # initi
•   ntraining = np.array(training) # Creat
•   ncategory = np.array(category) # Cre
•   for example in range(100):     # for each of the 1st 140 lines (training data)
•      input = ntraining[example] # Input vector: the counts for this line
•      goal_prediction = ncategory[example]  # We should target this category
•      prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
•      error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
•      delta = prediction - goal_prediction # gradient of the error
•      weights = weights - (alpha * (input * delta)) # adjust the weights
•
•      print("Ex: %d Error= %f Goal: %d Prediction: %d "
•         % (example, error, goal_prediction, prediction))
```


```
•   n = len(ntraining) - 1
•   for test in range(n, n - 10, -1): # For each last 10 lines
•      input = ntraining[test]  # Input vector: the counts for this line
•      goal_prediction = ncategory[test]
•      prediction = input.dot(weights) > 0.5 # the computed category
•
•      if (goal_prediction != prediction): # Here we detected the wrong category
•         print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + str(predic    s"+str(goal_prediction))
```

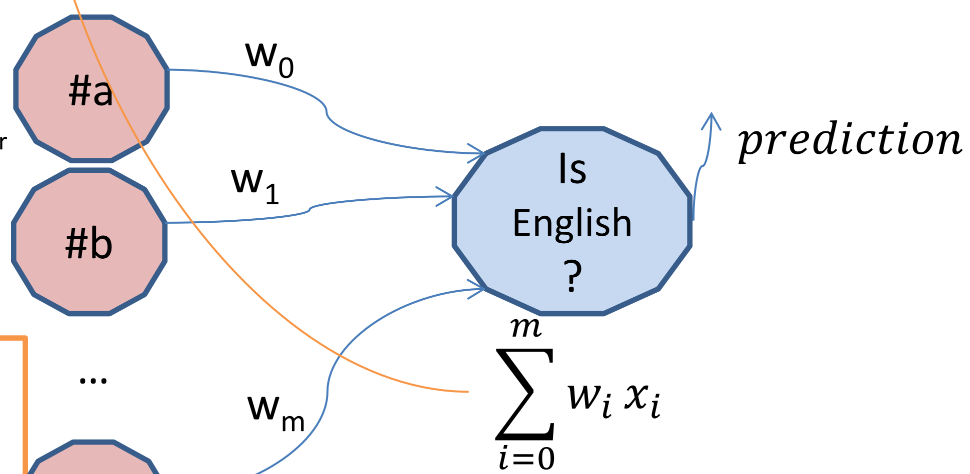



Naïve language guesser code

```
n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
    input = ntraining[test]  # Input vector: the counts for this line
    goal_prediction = ncategory[test]
    prediction = input.dot(weights) > 0.5 # the computed category

    if (goal_prediction != prediction): # Here we detected the wrong category
        print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) +
               " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

```
from string import ascii_lowercase
alpha = 0.01   # Learning rate
lines=[]       # the sentences read from
training = []  # A matrix containing fo
category = []  # A vector: for each line
fileName='test.txt' # test file containi
with open(fileName, 'r') as f:
    for line in f:
        lines.append(line)  # save the se
        training.append([line.lower().co
                for ch in ascii_lowercas
        category.append("+" in line)  # A
weights = np.random.rand(26) # initi
ntraining = np.array(training) # Creat
ncategory = np.array(category) # Cre
for example in range(100):     # for each of the 1st 140 lines (training data)
    input = ntraining[example] # Input vector: the counts for this line
    goal_prediction = ncategory[example]  # We should target this category
    prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
    error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
    delta = prediction - goal_prediction # gradient of the error
    weights = weights - (alpha * (input * delta)) # adjust the weights

    print("Ex: %d Error= %f Goal: %d Prediction: %d "
        % (example, error, goal_prediction, prediction))
```


```
n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
    input = ntraining[test]  # Input vector: the counts for this line
    goal_prediction = ncategory[test]
    prediction = input.dot(weights) > 0.5 # the computed category

    if (goal_prediction != prediction): # Here we detected the wrong category
        print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

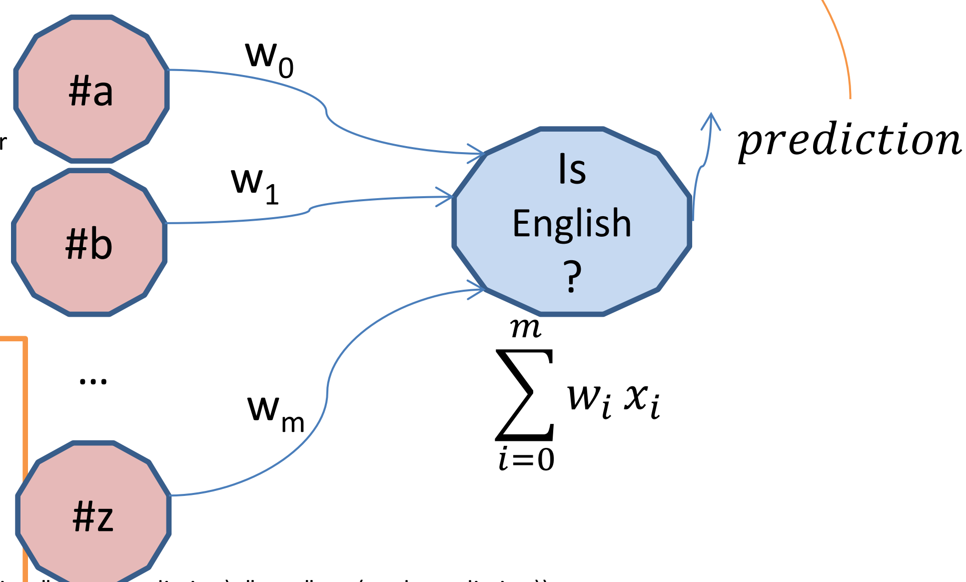



Naïve language guesser code

```
n = len(ntraining) - 1
for test in range(n, n - 10, -1): # For each last 10 lines
   input = ntraining[test]  # Input vector: the counts for this line
   goal_prediction = ncategory[test]
   prediction = input.dot(weights) > 0.5 # the computed category

   if (goal_prediction != prediction): # Here we detected the wrong category
      print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) +
                 " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

```
•   from string import ascii_lowercase
•   alpha = 0.01   # Learning rate
•   lines=[]       # the sentences read from
•   training = []  # A matrix containing fo
•   category = []  # A vector: for each line
•   fileName='test.txt' # test file containi
•   with open(fileName, 'r') as f:
•      for line in f:
•         lines.append(line)  # save the se
•         training.append([line.lower().cou
•                 for ch in ascii_lowercas
•         category.append("+" in line)  # A
•   weights = np.random.rand(26) # initi
•   ntraining = np.array(training) # Creat
•   ncategory = np.array(category) # Cre
•   for example in range(100):     # for each of the 1st 140 lines (training data)
•      input = ntraining[example] # Input vector: the counts for this line
•      goal_prediction = ncategory[example]  # We should target this category
•      prediction = max(min(input.dot(weights), 1), 0) # Compute the prediction
•      error = 0.5*(goal_prediction - prediction) ** 2 # Calculate the current error
•      delta = prediction - goal_prediction # gradient of the error
•      weights = weights - (alpha * (input * delta)) # adjust the weights
•
•      print("Ex: %d Error= %f Goal: %d Prediction: %d "
•          % (example, error, goal_prediction, prediction))
```

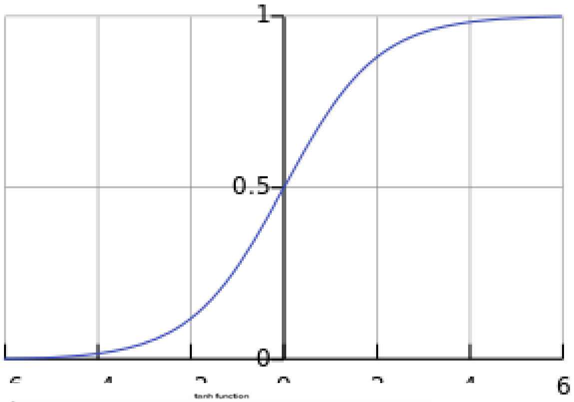

```
•   n = len(ntraining) - 1
•   for test in range(n, n - 10, -1): # For each last 10 lines
•      input = ntraining[test]  # Input vector: the counts for this line
•      goal_prediction = ncategory[test]
•      prediction = input.dot(weights) > 0.5 # the computed category
•
•      if (goal_prediction != prediction): # Here we detected the wrong category
•         print ("Test:" + str(test) + " "+lines[test]+" error=" + str(error) + " Prediction:" + str(prediction)+" was"+str(goal_prediction))
```

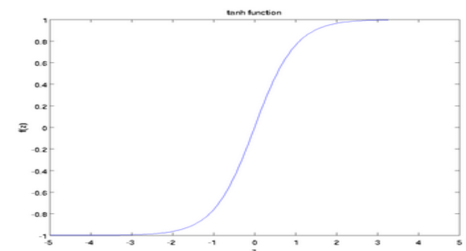
Perceptron: limits

- Perceptron can only classify linearly separable patterns
- Why? Their activation function is linear (0 / 1)
- Improvement: add a non-linear activation function
  - Sigmoid
  - tanh

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



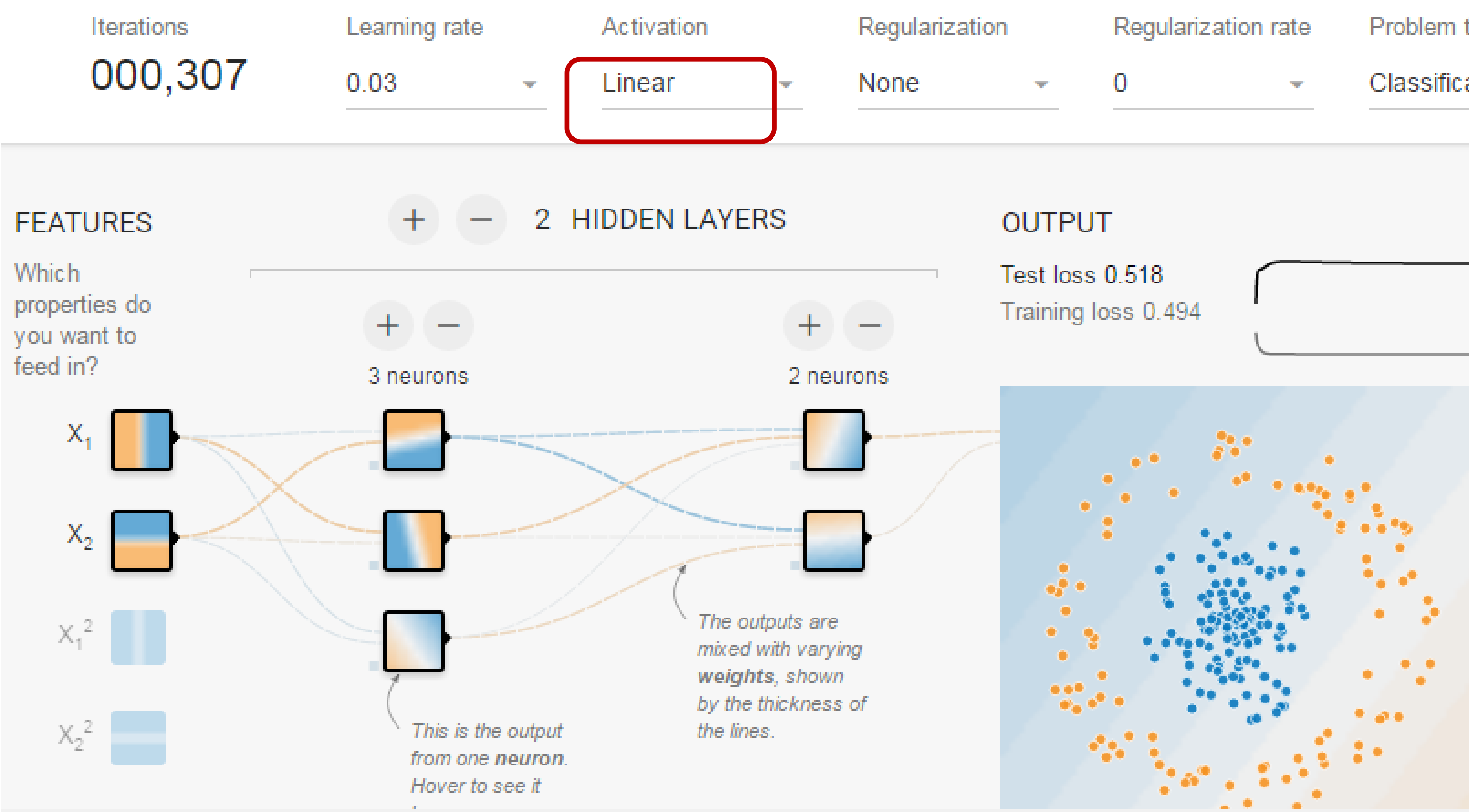
$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



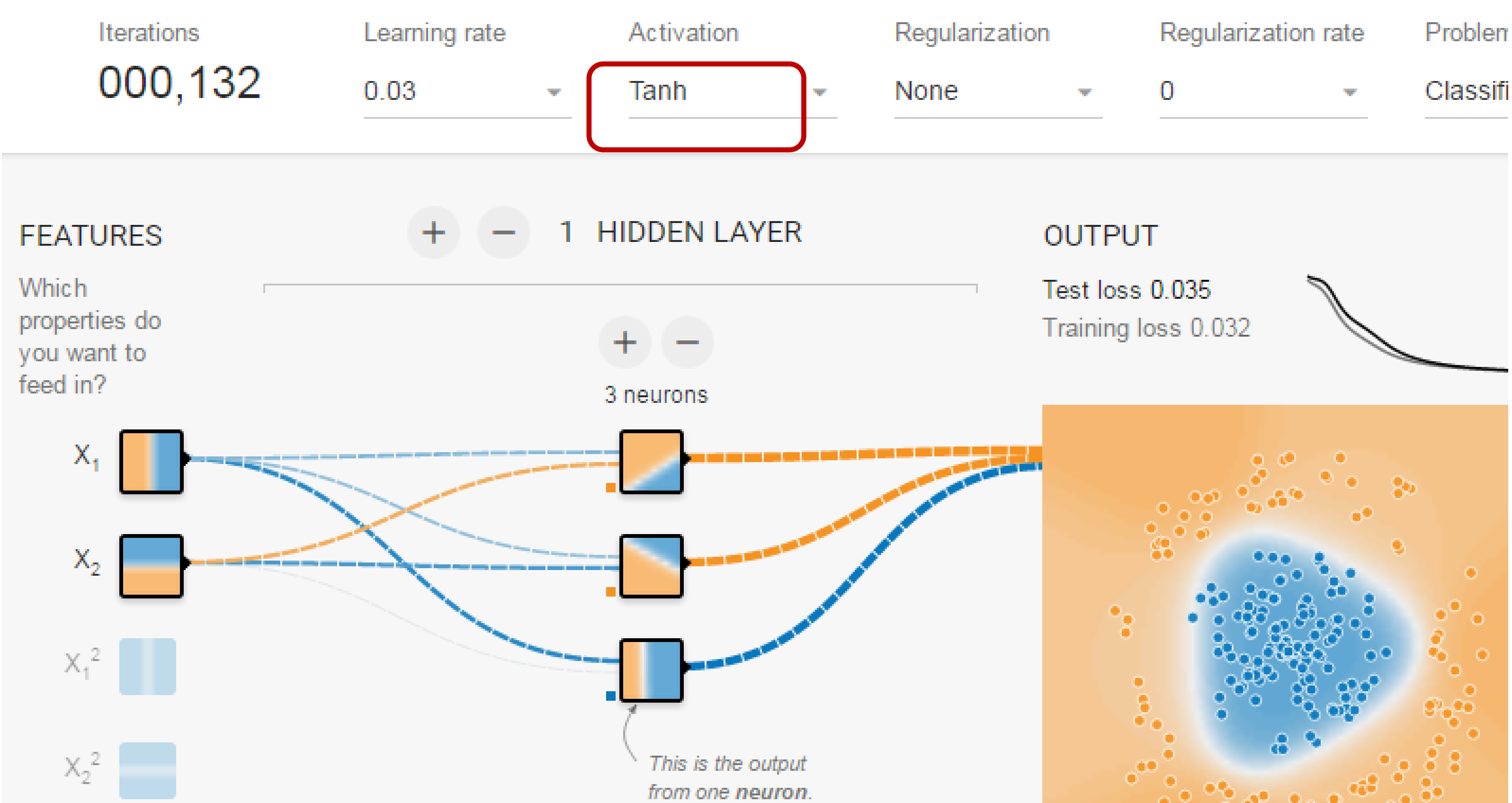
On “playground” linear perceptron

Try it yourself: http://playground.tensorflow.org/

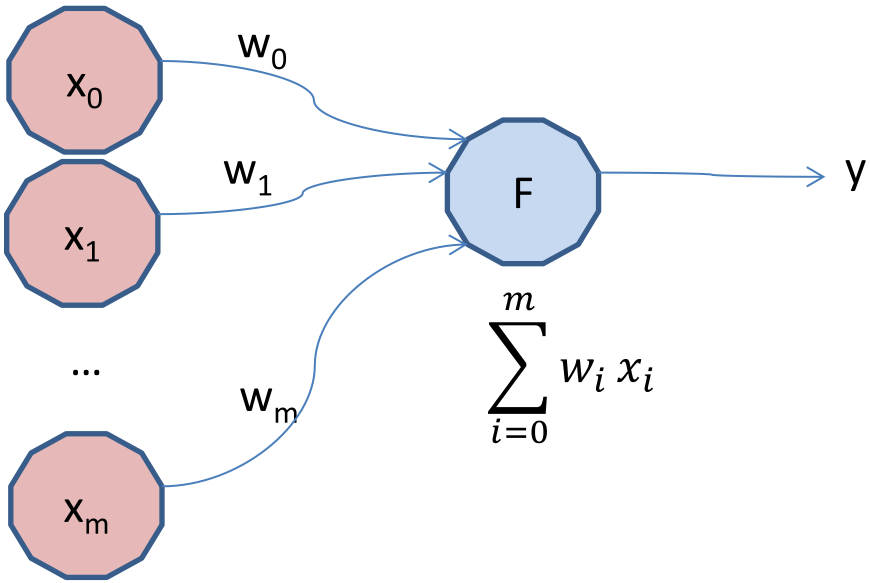
Even if we cascade linear output neurons…



- But it classify non-linear data, if the activation function is non-linear

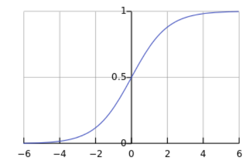


Feed forward NN



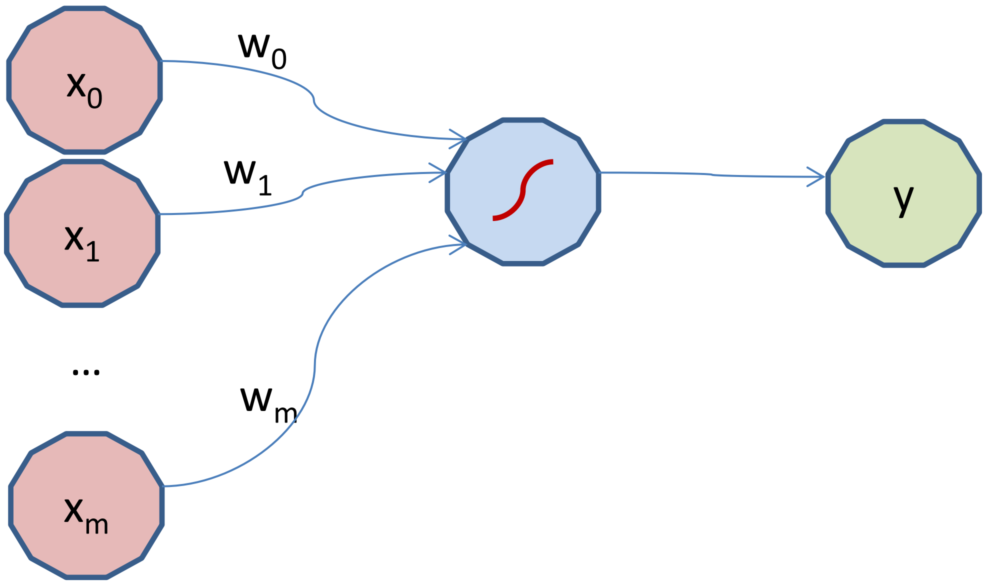
$$\sum_{i=0}^{m} w_i\, x_i$$

$$f(x) = \mathrm{sigmoid}(x)$$

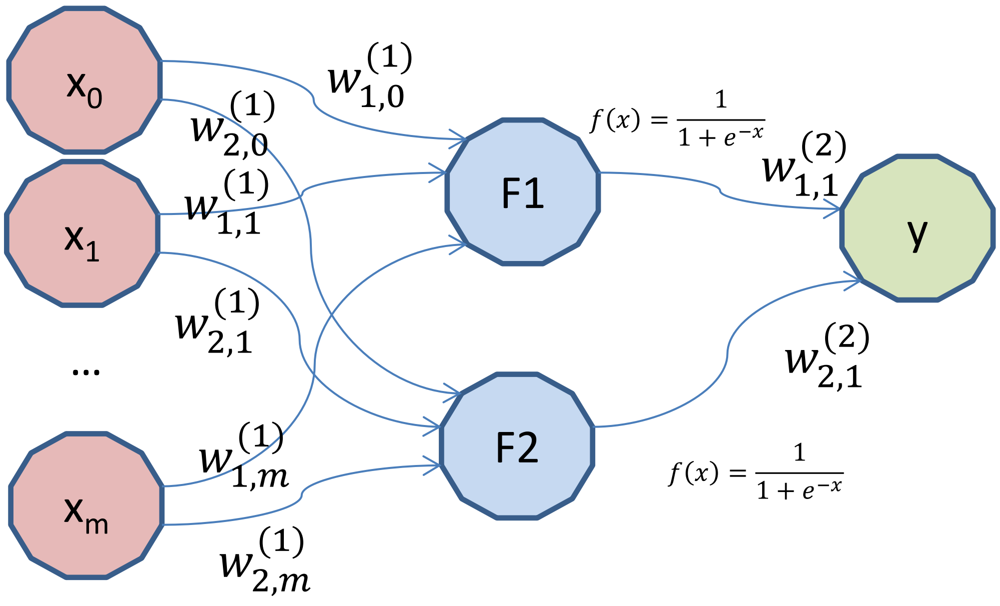


$$output = \frac{1}{1 + e^{-\sum_{i=0}^{m} w_i\, x_i}}$$

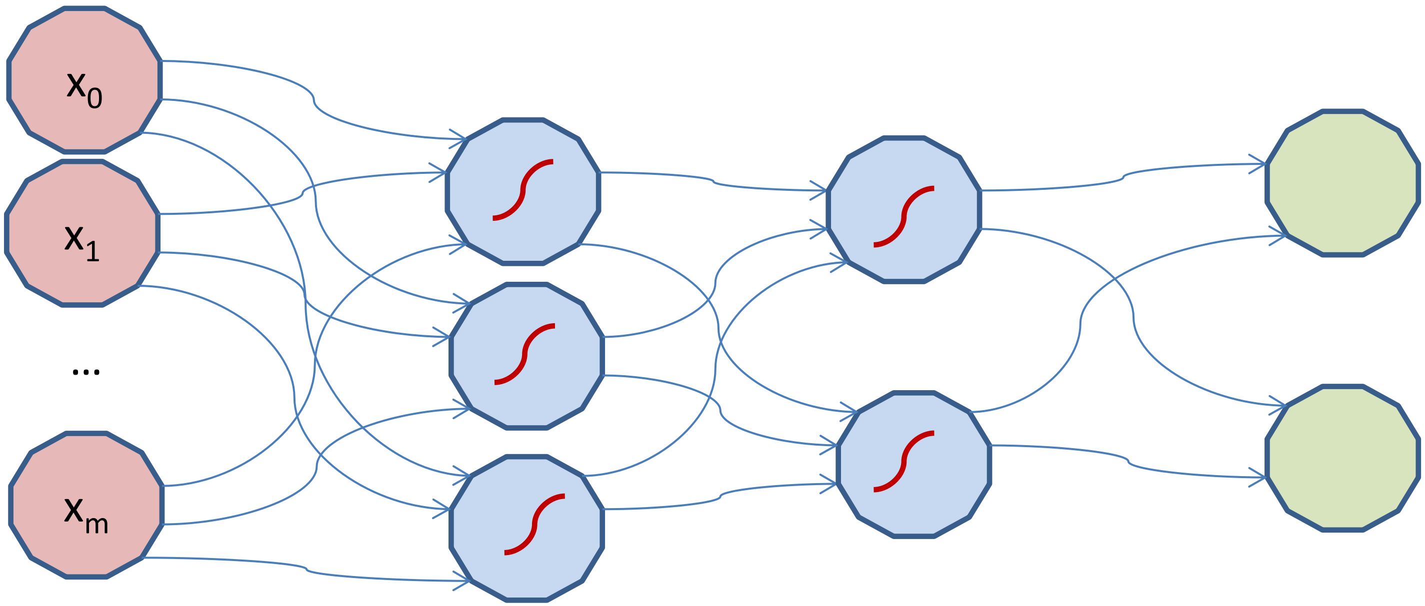
Feed-forward NN



Feed forward single hidden layer



Feed forward multiple hidden layer



Back Propagation on multiple layers

The gradient (derivative of the error) can also be back-propagated in a multi layer Neural Network…

.. But we would need a specific lecture about that!

Please read: « Neural Networks - Backpropagation and beyond”, tutorial given by Marcin at MTM2016:

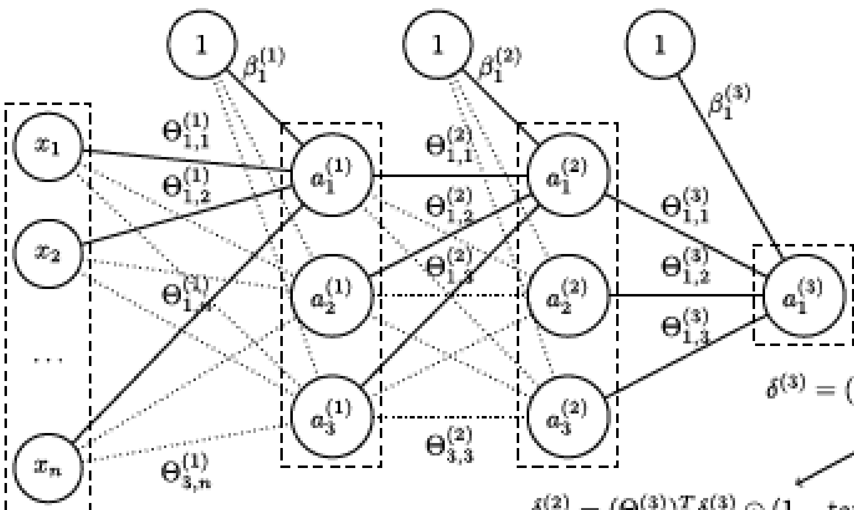
http://ufal.mff.cuni.cz/mtm16/files/06-nn-intro-backpropagation-marcin-junczys-dowmunt.pdf

Backpropagation through multiple layers

$$J(\Theta) = \frac{1}{2}(a^L - y)^2$$

$$\nabla_{a^L} J(\Theta) = a^L - y$$

$$\tanh'(x) = 1 - \tanh^2(x)$$



$$\delta^{(3)} = (a^{(3)} - y) \odot (1 - \tanh^2(z^{(3)}))$$

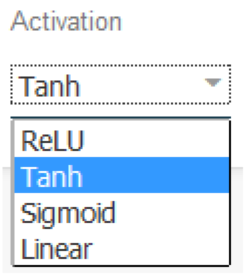
$$\delta^{(2)} = (\Theta^{(3)})^T \delta^{(3)} \odot (1 - \tanh^2(z^{(2)}))$$

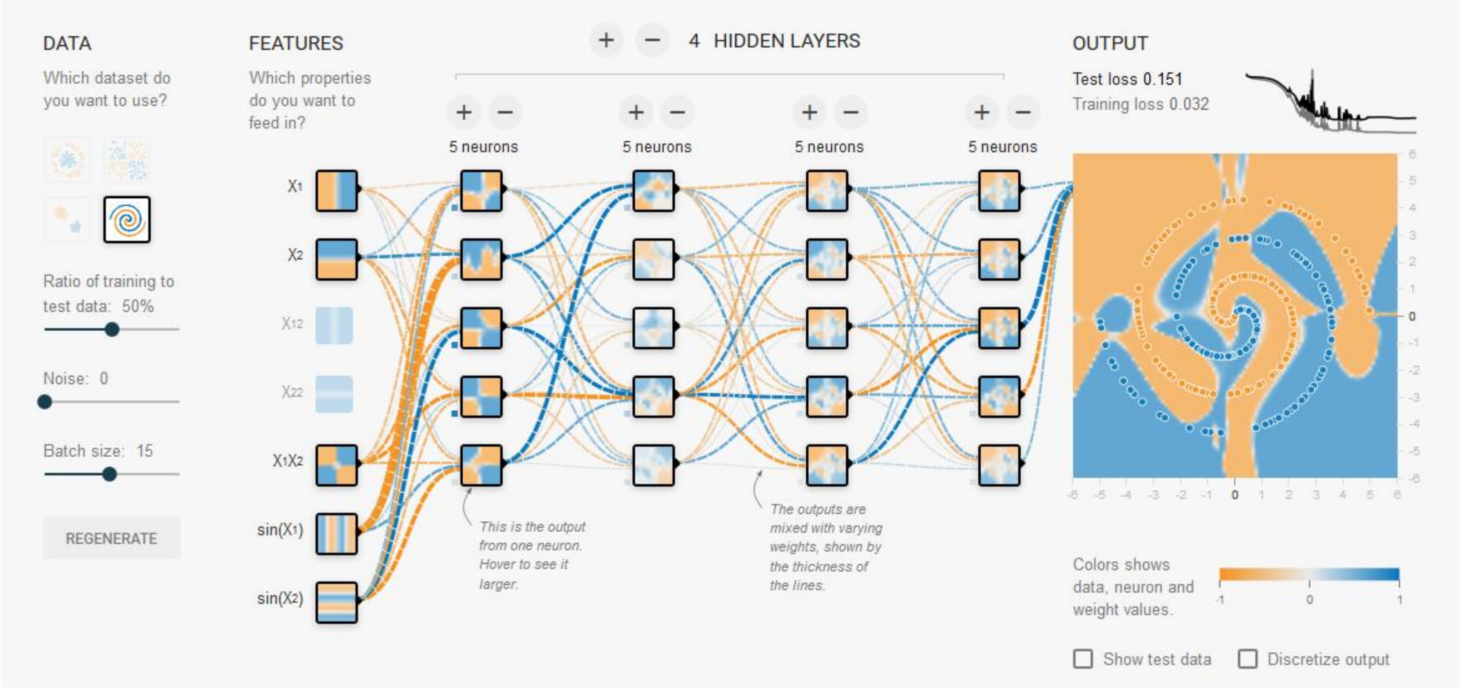
$$\delta^{(1)} = (\Theta^{(2)})^T \delta^{(2)} \odot (1 - \tanh^2(z^{(1)}))$$

$$a^{(0)} = x \longrightarrow \begin{array}{l} z^{(1)} = \Theta^{(1)} a^{(0)} + \beta^{(1)} \\ g^{(1)}(x) = \tanh(x) \\ a^{(1)} = g^{(1)}(z^{(1)}) \end{array} \longrightarrow \begin{array}{l} z^{(2)} = \Theta^{(2)} a^{(1)} + \beta^{(2)} \\ g^{(2)}(x) = \tanh(x) \\ a^{(2)} = g^{(2)}(z^{(2)}) \end{array} \longrightarrow \begin{array}{l} z^{(3)} = \Theta^{(3)} a^{(2)} + \beta^{(3)} \\ g^{(3)}(x) = \tanh(x) \\ a^{(3)} = g^{(3)}(z^{(3)}) \end{array}$$

http://ufal.mff.cuni.cz/mtm16/files/06-nn-intro-backpropagation-marcin-junczys-dowmunt.pdf

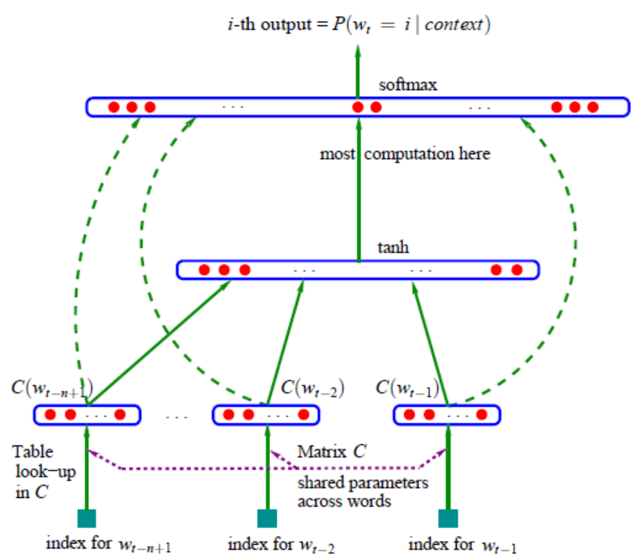
Tensorflow playground





In NLP: NN language model

- Predict next word, knowing previous *n* words
- Yoshua Bengio; Rejean Ducharme and Pascal Vincent , **A Neural Probabilistic Language Model, Bengio (2003)**



Feed forward NN for LM

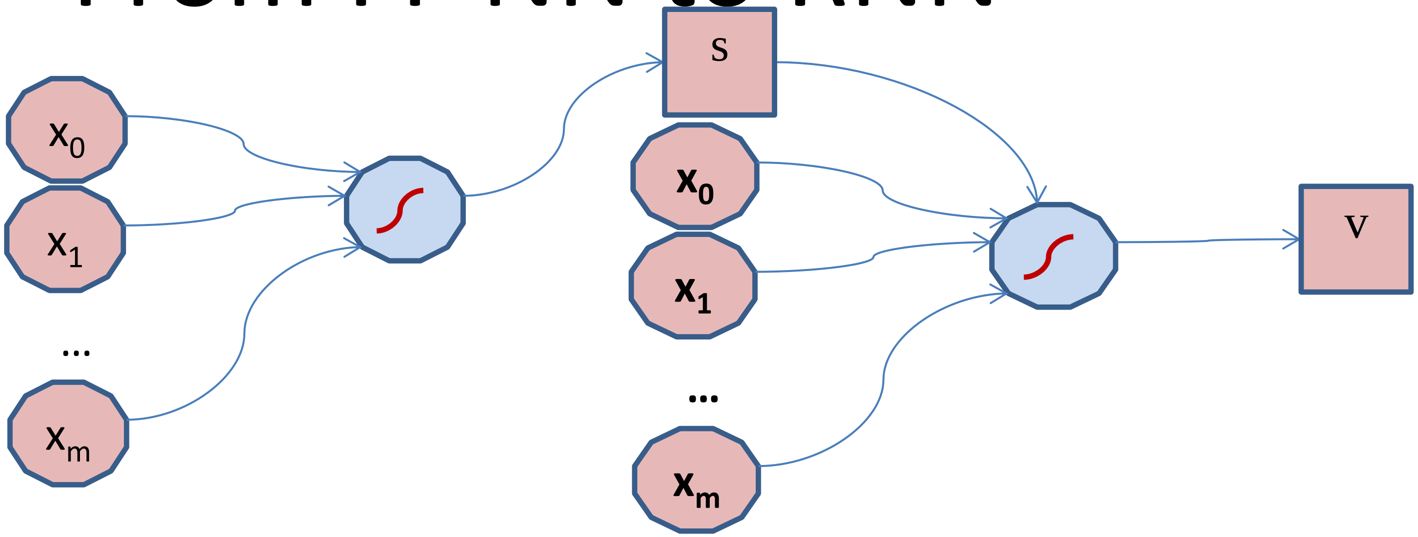
- But limited to a certain context
- New idea: use sequence to train a NN

Recurrent Neural Networks

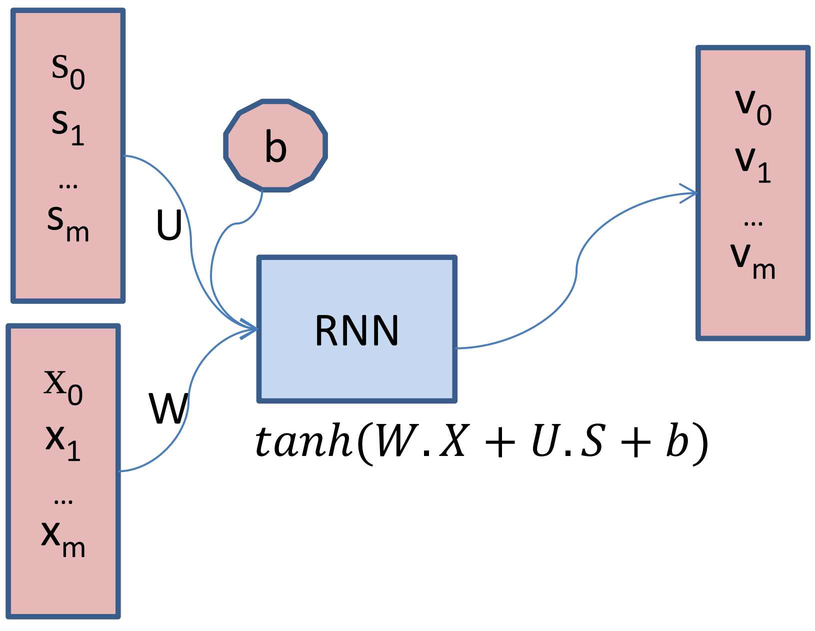
- Feedforward neural networks cannot handle "properly" sequences
- The idea is: let's produce an output and a "memory" (internal state) that can be fed to the next sequence

From FF NN to RNN

- FF NN:



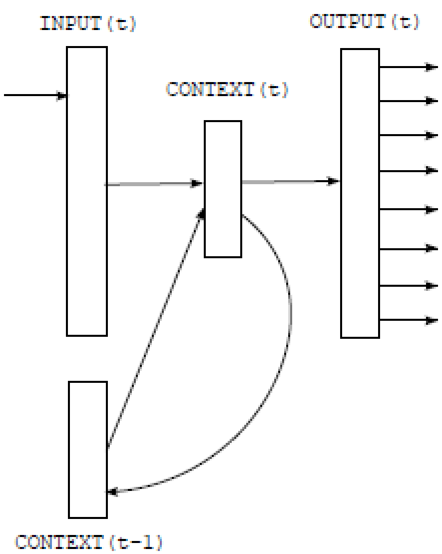
- RNN:



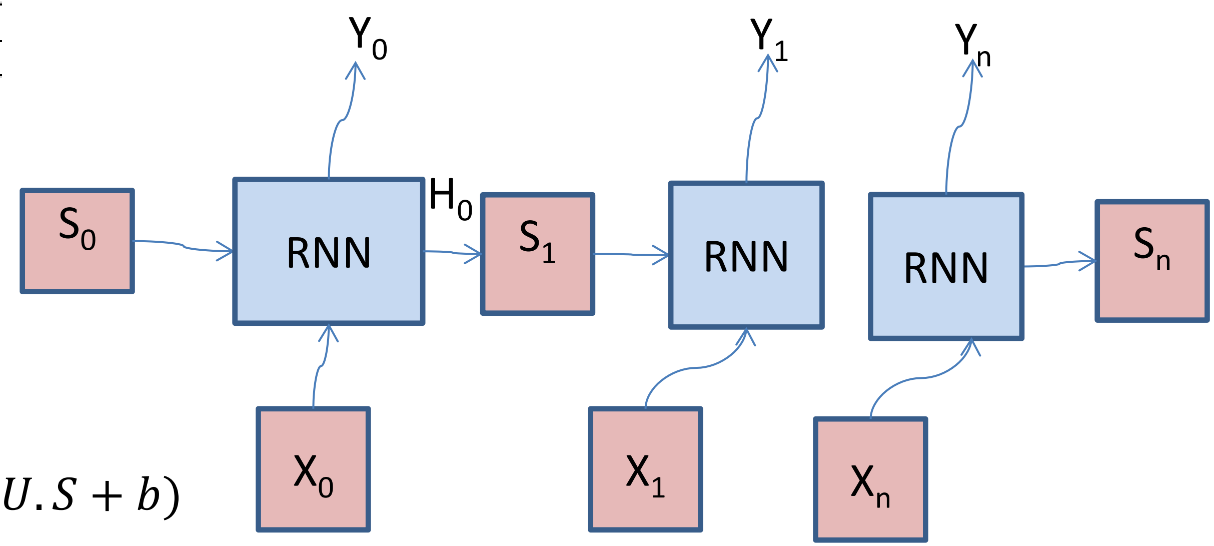
$tanh(W.X + U.S + b)$

RNN in NLP

- T Mikolov - 2010, **Recurrent Neural Network Based Language Model**



$$y_t = tanh(Wy.X + b)$$



$$h_t = tanh(W.X + U.S + b)$$

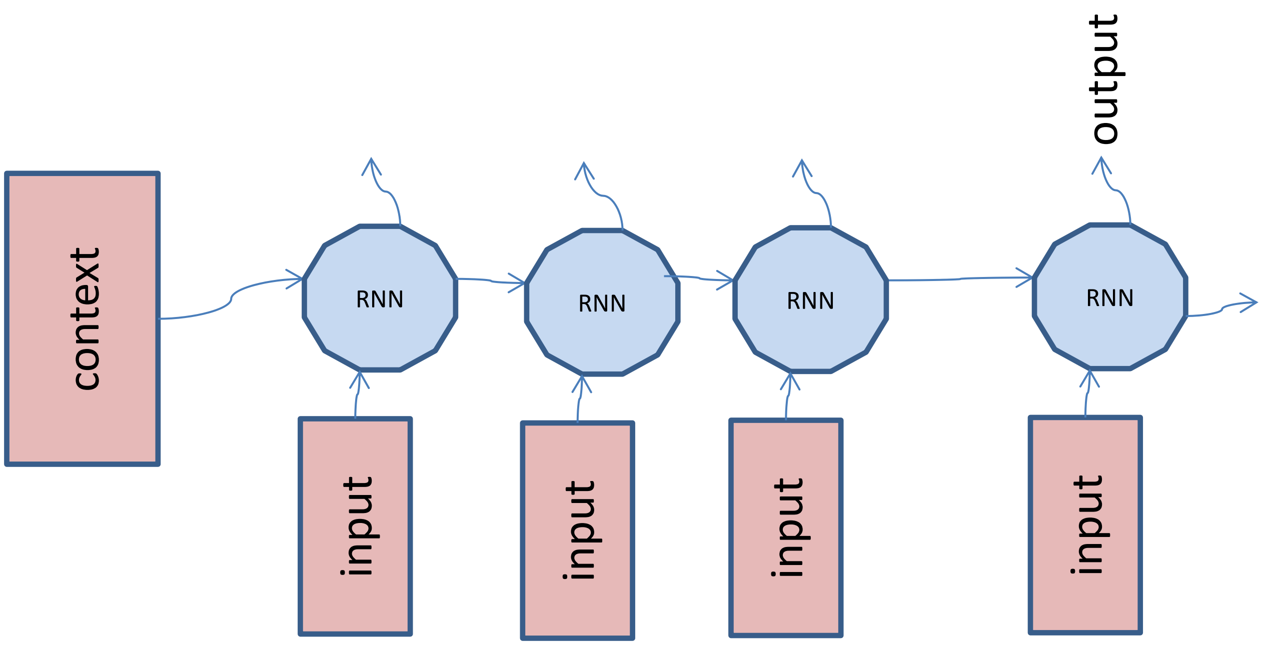
NLP example: character prediction

- How to train a sequence of characters to produce Shakespeare-like English?
- See the blog from Andrej Karpathy: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (together with a 100 line Python code to reproduce it).

RNN in practice

Sequence to output:

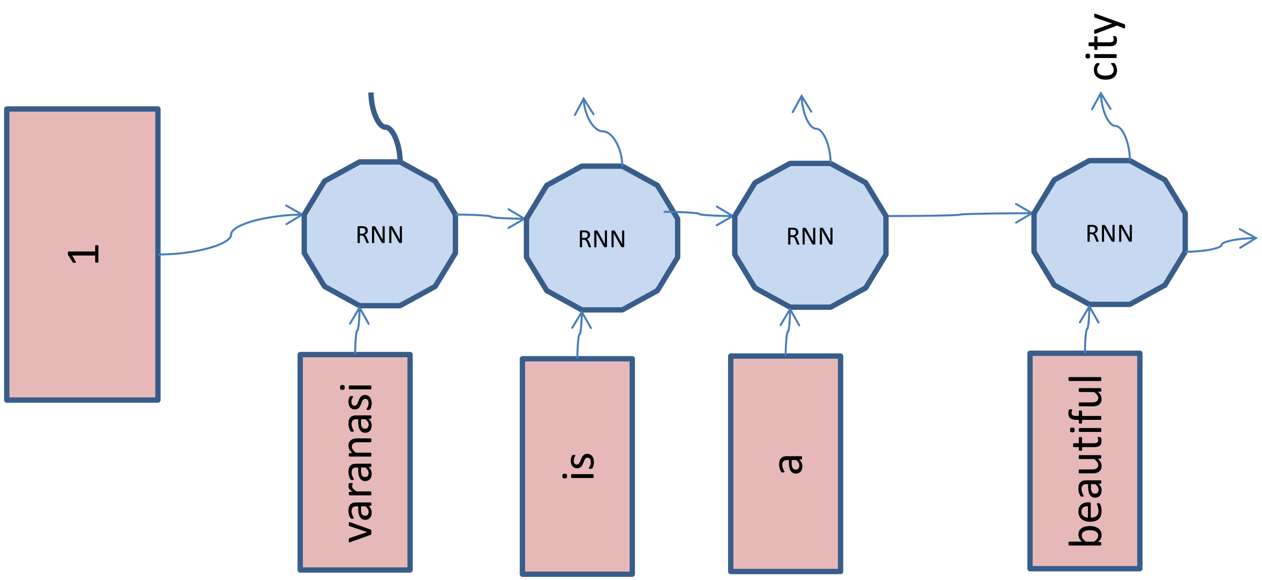
- LM: from a sequence of words/characters, predict the next one



RNN in practice

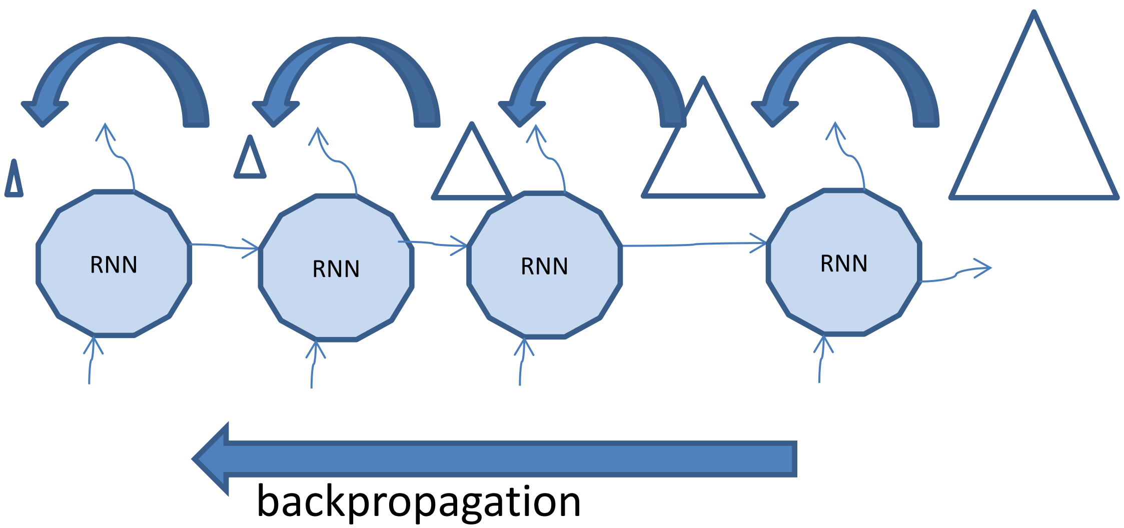
Sequence to output: NN language model

Predict the next word…



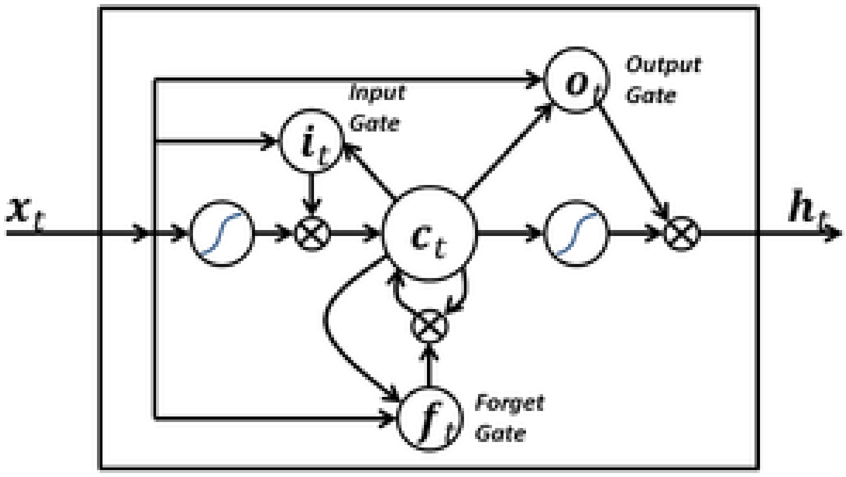
But…

- When the sequence is long, the RNN cannot really back-propagate the error correctly: gradient vanishing / gradient exploding



Solution

- Use specific RNN architecture that can keep "memory".
- LSTM (Long Short Term Memory) are able to « decide » to keep the memory or forget it.



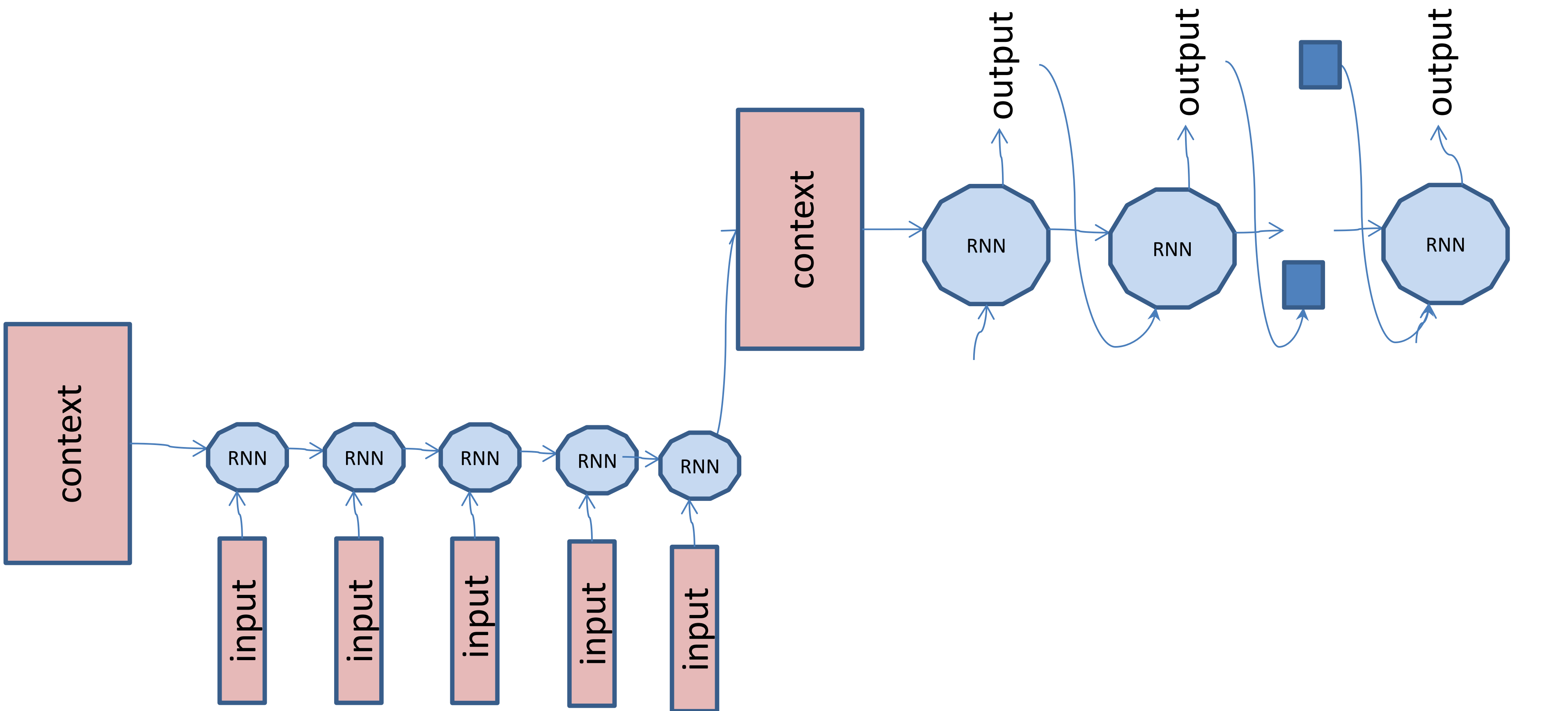
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Solution(2)

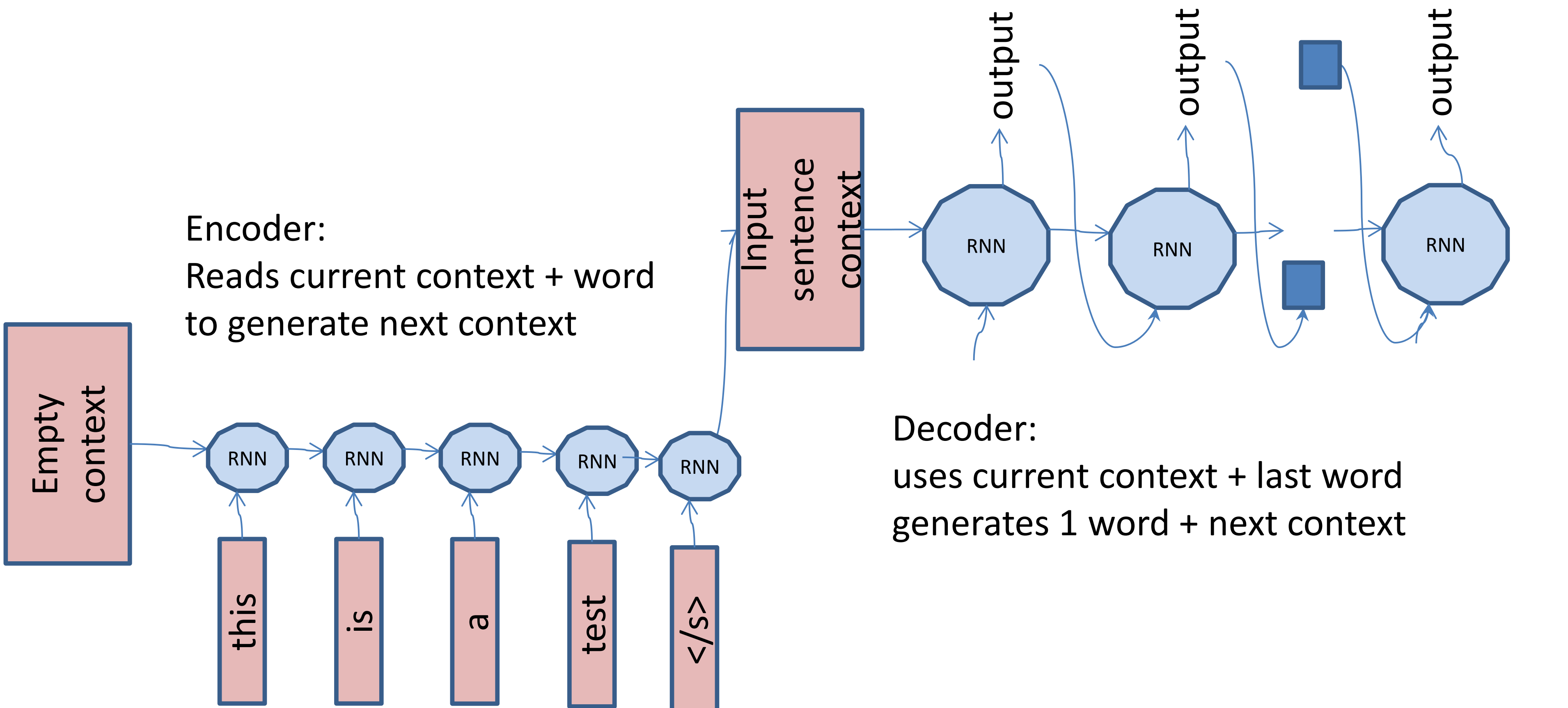
- GRU: Gated Recurrent Units, combine “input” and “forget” gates of LSTRMs by a single “update” gate
- Thanks to LSTMs / GRUs, RNNs can now have a long “memory”, therefore, in NLP, handle long sentences, even the last RNN, reading the last word “remembers” the first word of the sentence

RNN in practice

- Sequence to sequence: from a sequence of words, predict a sequence of translated words

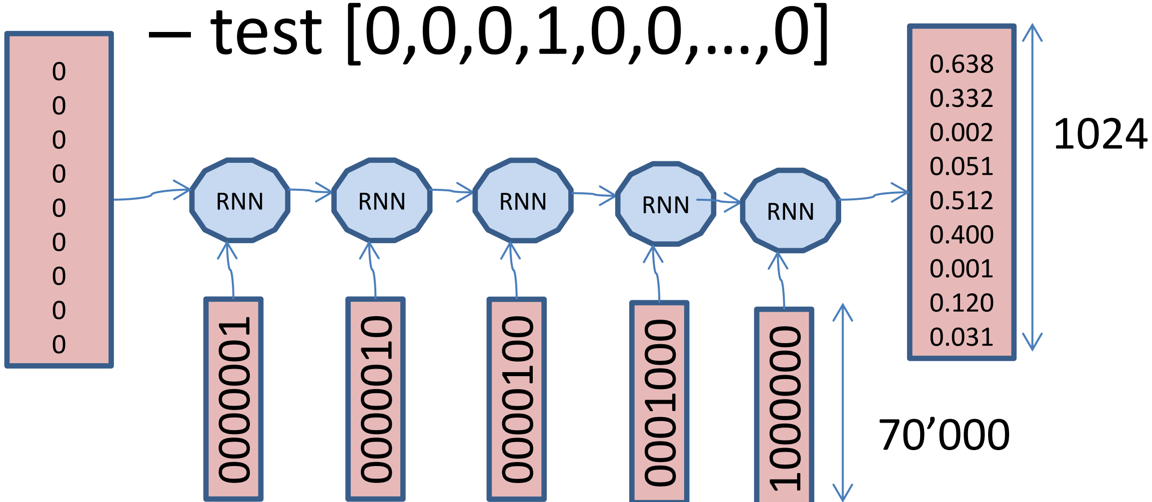


RNN in practice



Neural Networks need inputs as vectors

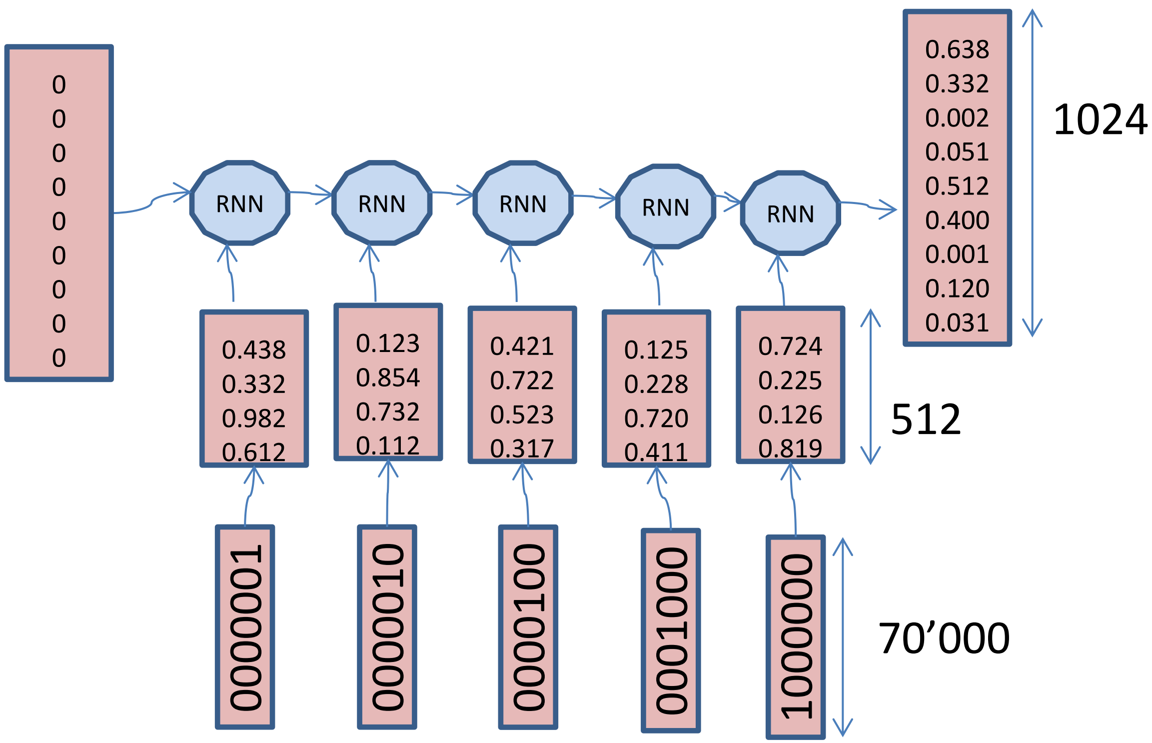
- In NLP, inputs are words (not vectors)
- Represent them as hot-vector
  - this [1,0,0,0,0,0,…,0]
  - is [0,1,0,0,0,0,…,0]
  - a [0,0,1,0,0,0,…,0]
  - test [0,0,0,1,0,0,…,0]



Problem: the hot vectors are as big as your vocabulary (could be 70’000)

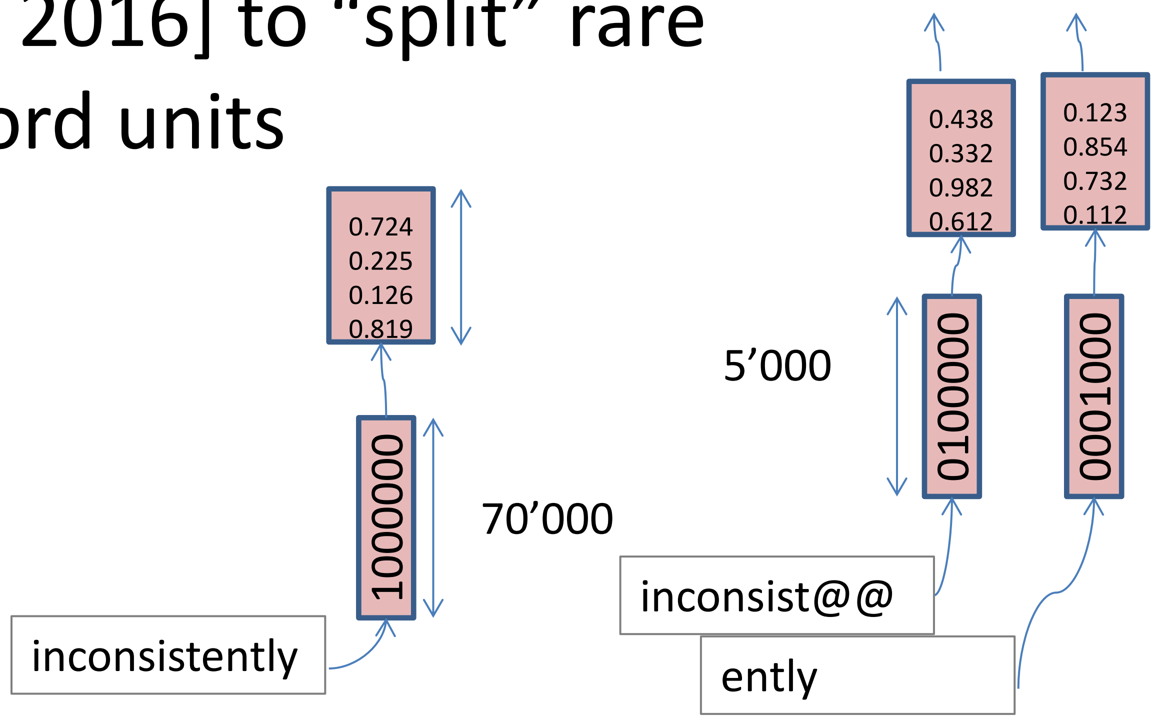
Word embeddings

- Add a hidden layer with less dimension
- RNN can now process less dimensions



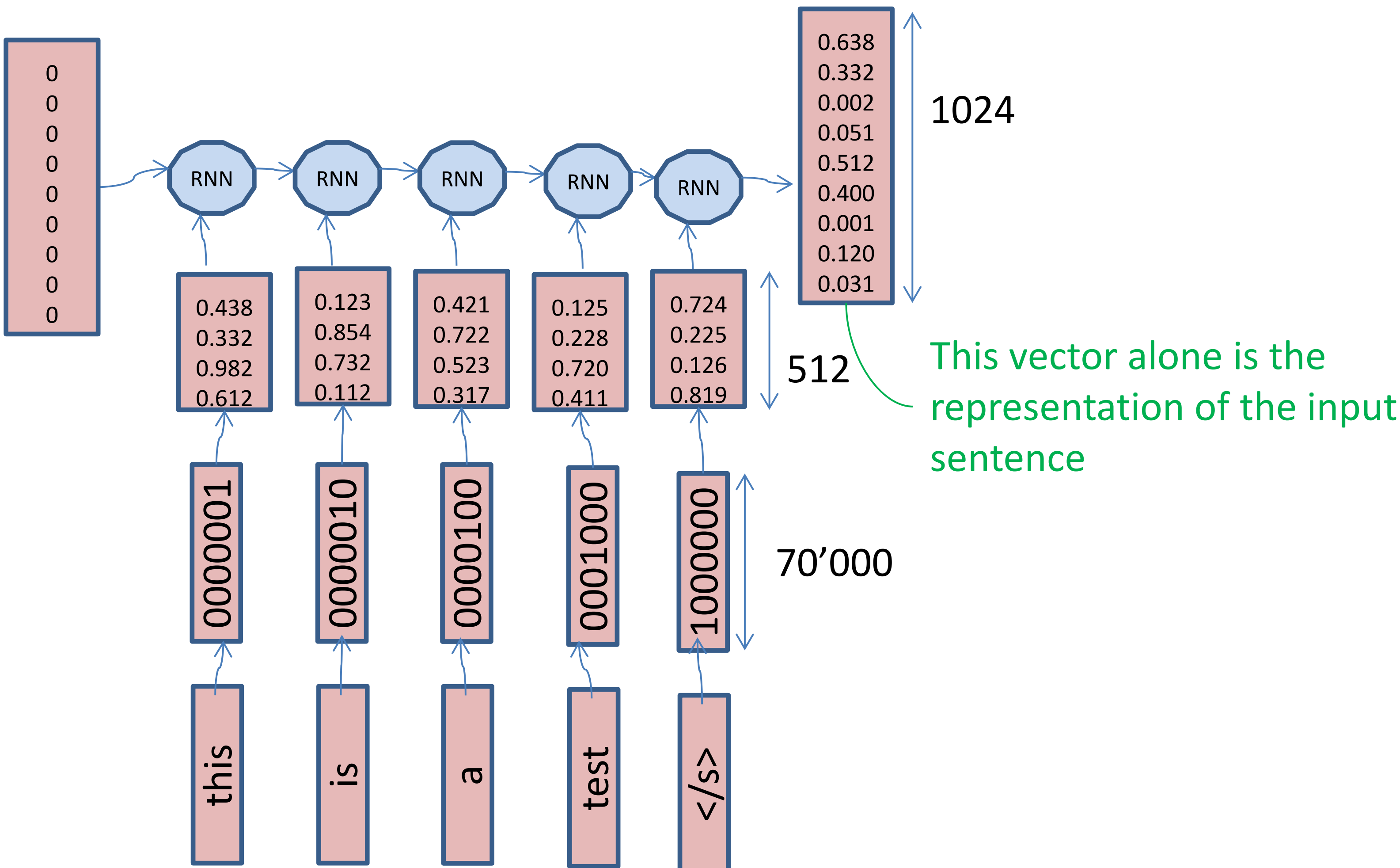
Reducing input vocabulary

- Even with embeddings, the input vocabulary size is an issue.
- We use Byte Pair Encoding algorithm [Sennrich et al. 2016] to “split” rare words in subword units

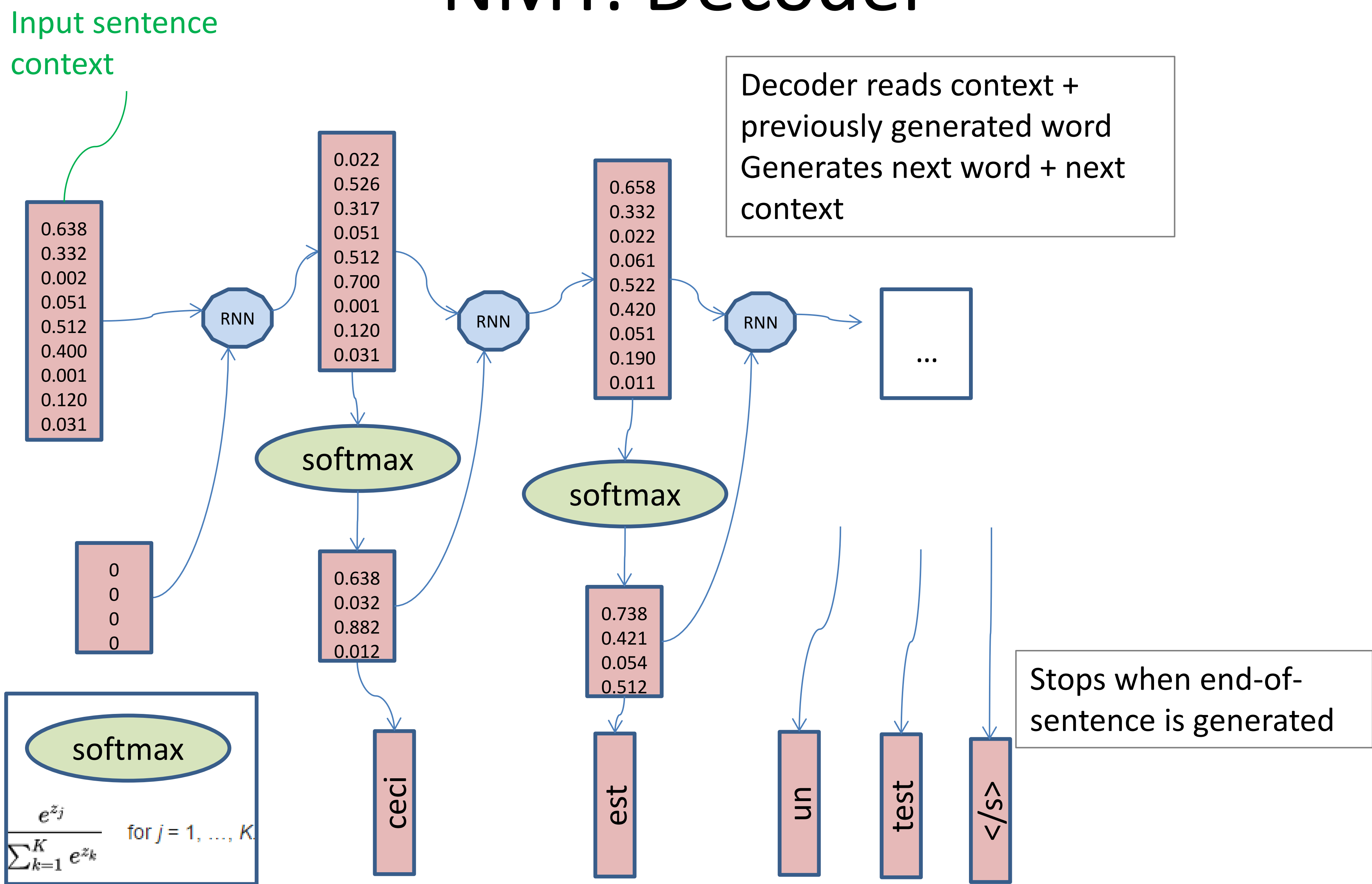


NMT: Encoder (forward only)

Encoder reads words, generate sentence context



NMT: Decoder



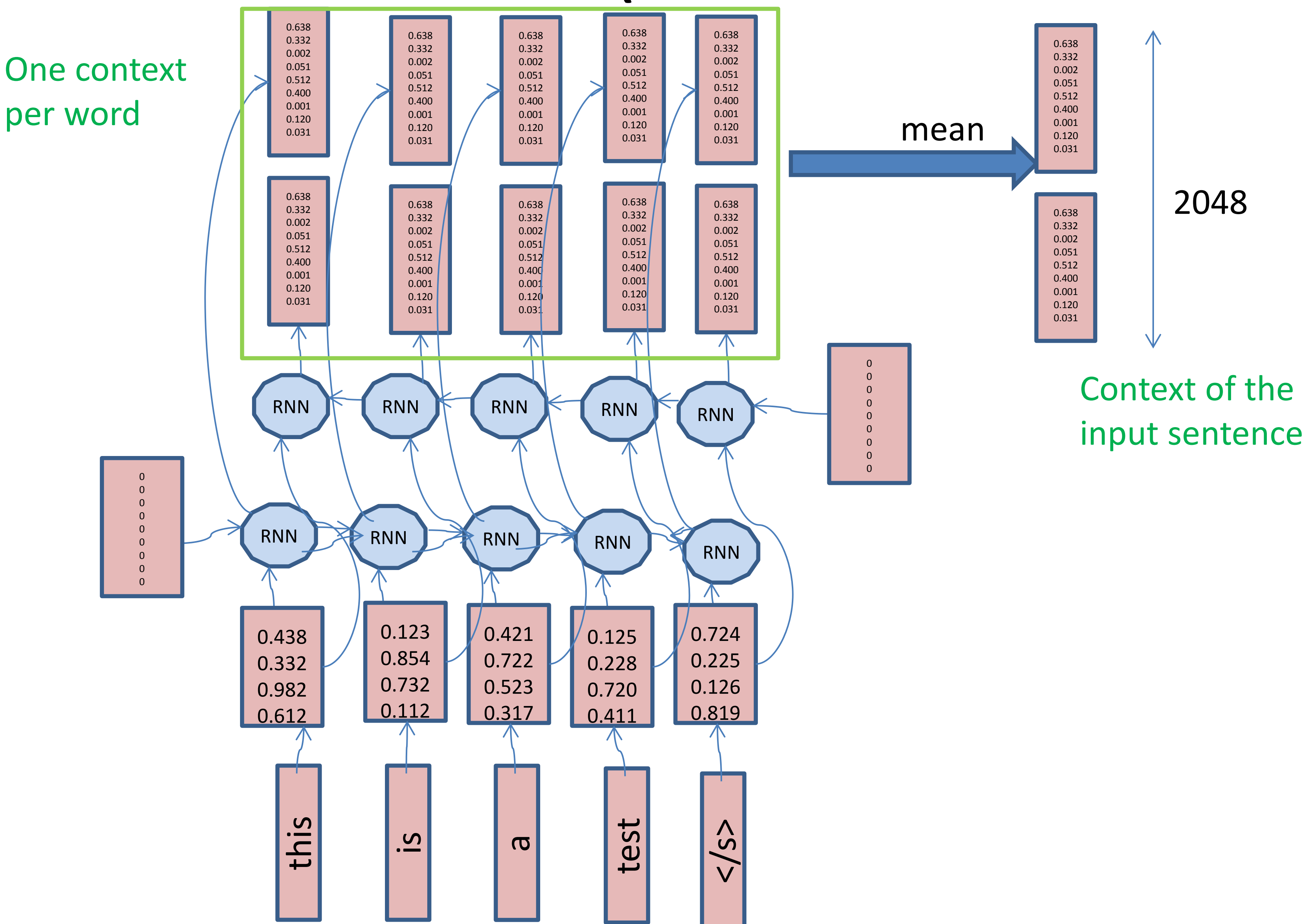
Input sentence context
Decoder reads context + previously generated word
Generates next word + next context
0.638
0.332
0.002
0.051
0.512
0.400
0.001
0.120
0.031
RNN
0.022
0.526
0.317
0.051
0.512
0.700
0.001
0.120
0.031
RNN
0.658
0.332
0.022
0.061
0.522
0.420
0.051
0.190
0.011
RNN
...
softmax
softmax
0
0
0
0
0.638
0.032
0.882
0.012
0.738
0.421
0.054
0.512
softmax
$\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$ for $j = 1, \ldots, K$.
ceci
est
un
test
</s>
Stops when end-of-sentence is generated

Decoder needs some information

- Apart from the source context vector, the previous model does not take into account specific source words
- ➔Creation of an attention mechanism

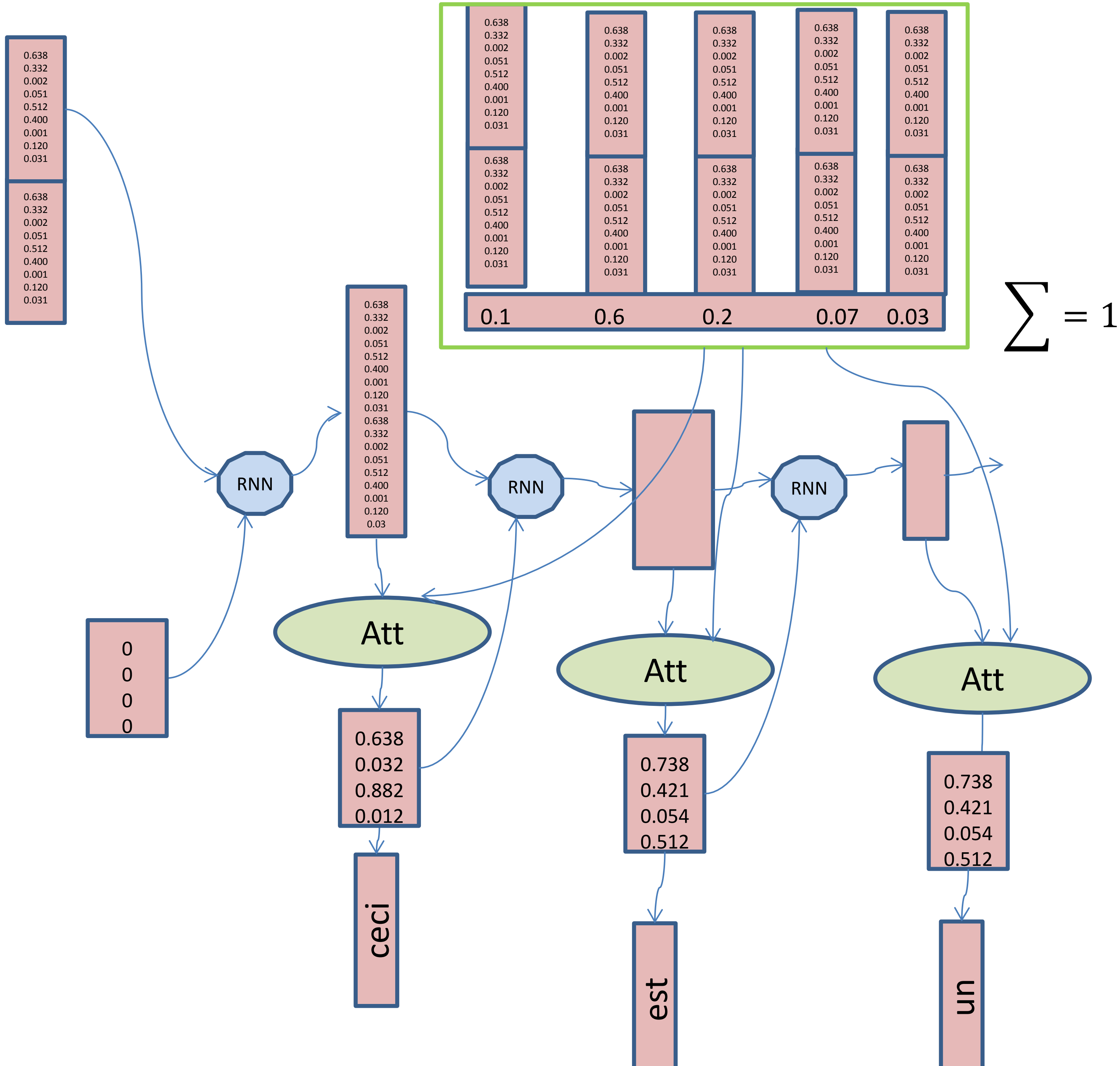
- … first try to create a context for each word
- to make it more efficient, parse the sentence for left to right … and from right to left

NMT: Encoder (forward-backward)



Decoder with “attention model”

- Now that we built a “context” for each word during encoding…
- We feed this context to the decoder
- Each RNN is augmented by an “attention mechanism” that reads the word context before producing the output
- As a “side effect” our NMt can now produce alignments



0.638 0.332 0.002 0.051 0.512 0.400 0.001 0.120 0.031
0.638 0.332 0.002 0.051 0.512 0.400 0.001 0.120 0.031
0.638 0.332 0.002 0.051 0.512 0.400 0.001 0.120 0.031 0.638 0.332 0.002 0.051 0.512 0.400 0.001 0.120 0.03
0.1 0.6 0.2 0.07 0.03
$\sum = 1$
RNN
RNN
RNN
0 0 0 0
Att
Att
Att
0.638 0.032 0.882 0.012
0.738 0.421 0.054 0.512
0.738 0.421 0.054 0.512
ceci
est
un

… and it works!

- NMT are now giving better results than SMT
- And are ready to be put in production

Machine translation

1. (CN106090640) 一种具有双重散热结构的LED照 灯具

National Biblio. Data | Description | Claims

(ZH) Led (light emitting diode) illuminating lamp with double radiating structures

(ZH) The invention discloses an LED (light emitting diode) illumination lamp with a double heat dissipation structure. The LED illumination lamp comprises a lens, a heat dissipation portion and a mounting portion, one end of the heat dissipation part is fixed together with the installation part. one end of the heat dissipation part is fixed together with the installation part, the other end of the LED integrated board is provided with an LED integrated board and is fixed together with the lens, a cooling fan and a driving power supply are arranged in the mounting part, and the driving power supply is respectively connected with the cooling fan and the LED integrated board, the heat dissipation portion is provided with a heat dissipation groove for reinforcing heat dissipation. The heat dissipation portion is provided with a heat dissipation groove used for reinforcing heat dissipation. The heat dissipation device and the heat dissipation fan are simultaneously provided with the heat dissipation portion and the heat dissipation fan., the heat dissipation portion dissipates heat into the air through contact

Permanent Link/ Bookmark:

Application Number: 102016000409796 **Appli**

Publication Number: 106090640 **Publication**

Publication Kind : A

IPC:

Applicants: 江门市新中光科技有限公司

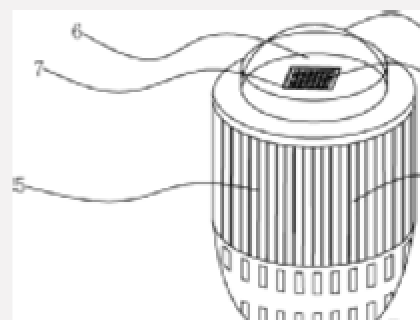
Inventors: 周昌平

Agents: 广州嘉权专利商标事务所有限公司 44205

Priority Data:

Title: (ZH) 一种具有双重散热结构的LED照明灯具

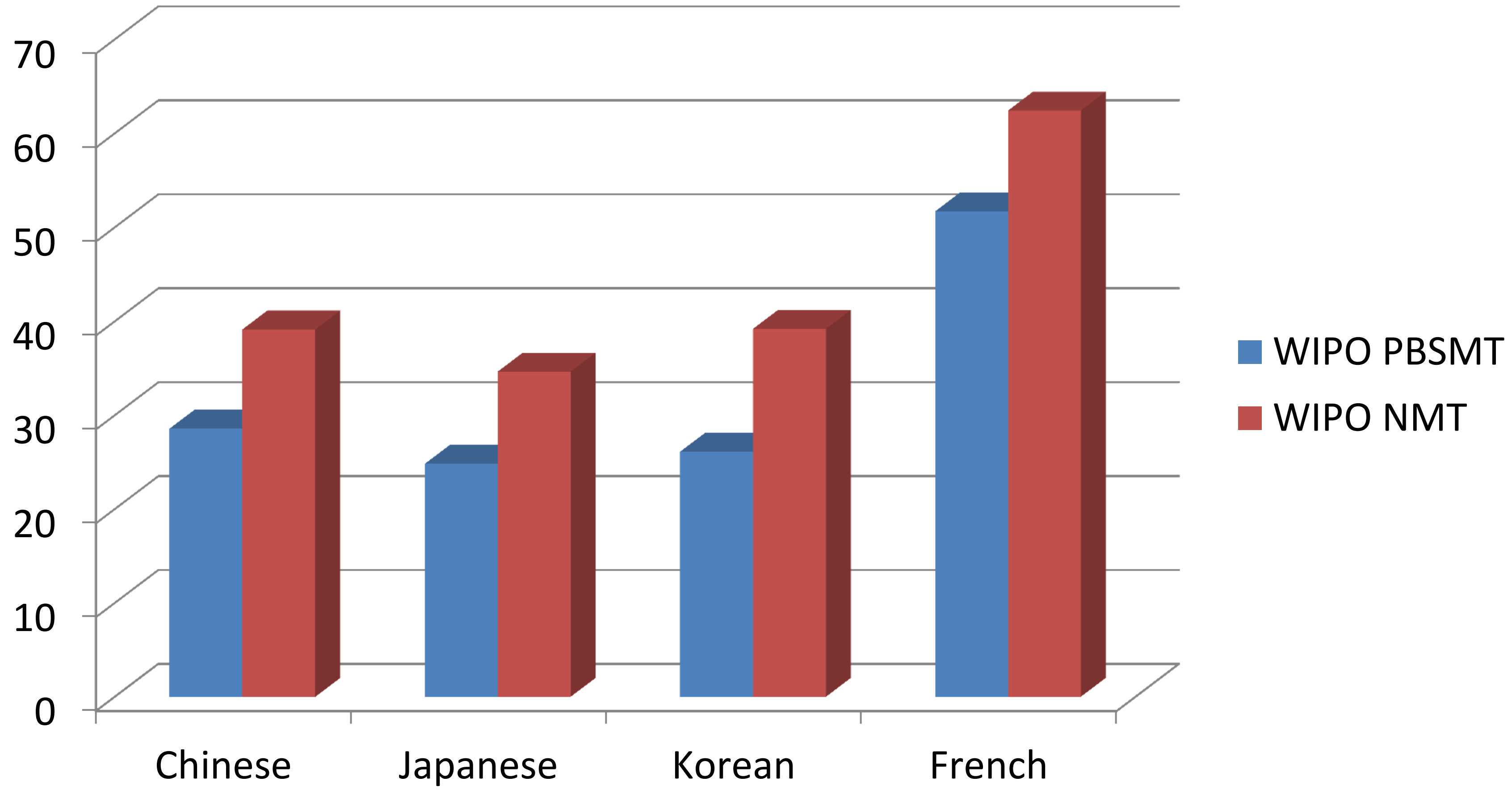
Abstract: (ZH) 本发明公开了一种具有双重散热结构的LED照明灯具，包括透镜、散热部以及安装部，所述散热部的一端与安装部固定在一起，另一端设置有LED集成板并与透镜固定在一起，所述安装部内设置有散热风扇和驱动电源，驱动电源分别与散热风扇和LED集成板连接，所述散热部设置有用于加强散热性的散热凹槽。本发明同时设置有散热部和散热风扇进行散热，散热部通过与空气的接触，将热量散发到空气中，而散热风扇可以加速空气的流动，增强散热效果，这两种相互配合，使得LED照明灯具的散热效果获得了大幅的提升。

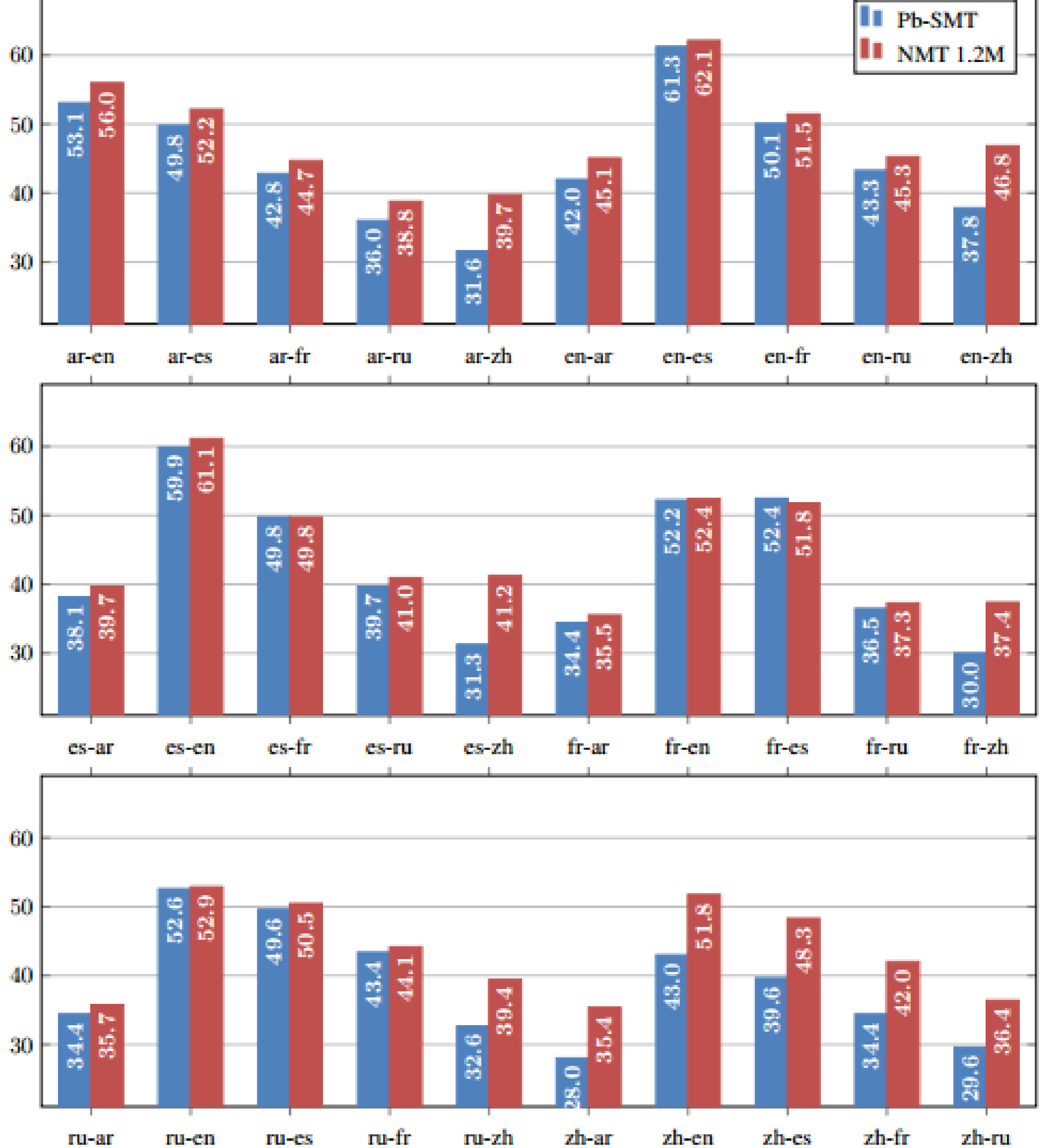


Using UN corpus

- Done on freely available UN corpus [Ziemski et al., 2016]

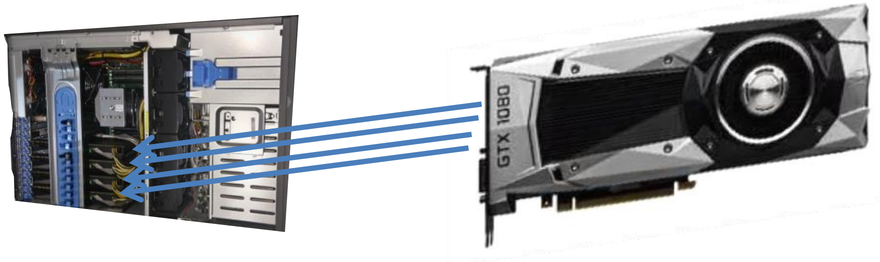
in WIPO: NMT compared to PBSMT




Pb-SMT
NMT 1.2M
60
50
40
30
53.1
56.0
49.8
52.2
42.8
44.7
36.0
38.8
31.6
39.7
42.0
45.1
61.3
62.1
50.1
51.5
43.3
45.3
37.8
46.8
ar-en
ar-es
ar-fr
ar-ru
ar-zh
en-ar
en-es
en-fr
en-ru
en-zh
38.1
39.7
59.9
61.1
49.8
49.8
39.7
41.0
31.3
41.2
34.4
35.5
52.2
52.4
52.4
51.8
36.5
37.3
30.0
37.4
es-ar
es-en
es-fr
es-ru
es-zh
fr-ar
fr-en
fr-es
fr-ru
fr-zh
34.4
35.7
52.6
52.9
49.6
50.5
43.4
44.1
32.6
39.4
28.0
35.4
43.0
51.8
39.6
48.3
34.4
42.0
29.6
36.4
ru-ar
ru-en
ru-es
ru-fr
ru-zh
zh-ar
zh-en
zh-es
zh-fr
zh-ru

Why is NMT different in practice?

- Trained on GPUs



- Model size smaller
- Could be faster (on GPUs). Both for training and decoding
- Early studies tend to prove that post-edition of NMT is easier (Bentivogli et al. 2016[1])

system	BLEU	HTER	mTER
PBSY	25.3	28.0	21.8
HPB	24.6	29.9	23.4
SPB	25.8	29.0	22.7
NMT	31.1*	21.1*	16.2*

Post-editors need less work to "correct" NMT output

[1] https://www.arxiv.org/abs/1608.04631

What are the existing tools?

- A lot of open-sources are now available
  - Librairies:
    - Theano
    - Tensorflow
    - Torch
    - Caffe,
    - DL4J…
  - Tools for training/decoding:
    - Nematus
    - NeuralMonkey
    - etc.
  - Tool for decoding only: AmunMT

e.g. Toy Hindi-English example

- Attentional encoder-decoder [Bahdanau et al., 2014], implemented in Nematus [Sennrich et al., 2016];
- Mini-batches of size 80, maximum sentence length of 80 words, word embeddings of size 256, and hidden layers of size 250.
- On a Nvidia GTX 1080;
- After 5 hours, reached a Moses PBSMT similar BLEU score (still low: 7)
- Model size: 25Mb
- Can be run (even on CPU, thanks to AmunMT)

Challenges with training NMT

- Setting the right parameters
  - If BPE, what is the size of the vocabulary
  - Embedding vector size
  - Context vector size
  - Learning factor
  - Activation function
  - Mini batch size
  - Number of epochs
  - …

Future of NMT?

- A lot has still to be explored
  - Include context information in input
    - Sentence level context
    - Document level context
  - Include more monolingual data to the training [Sennrich et al., 2016b]
  - Multilingual system:
    - Multi-source NMT
    - Multi-target NMT
    - Many-to-many NMT