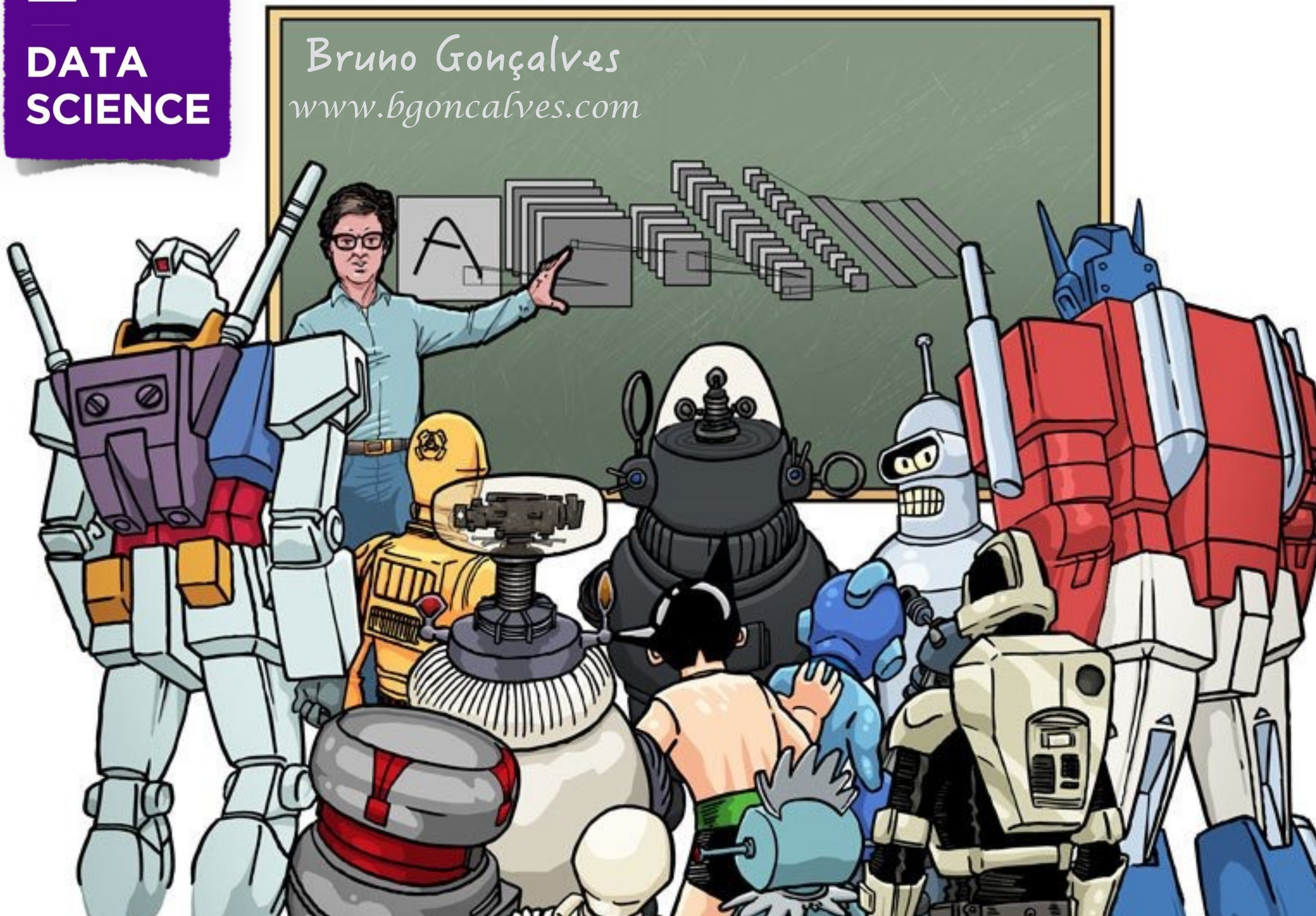
NYU

DATA SCIENCE

Machine(s) Learning and Data Science

Bruno Gonçalves
www.bgoncalves.com



A practical example - MNIST

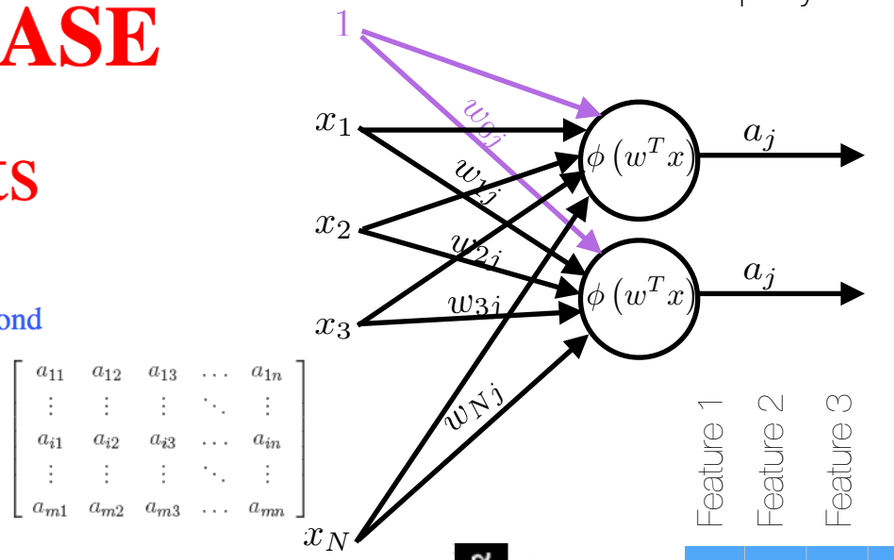
http://github.com/bmtgoncalves/Neural-Networks

http://yann.lecun.com/exdb/mnist/

THE MNIST DATABASE

of handwritten digits

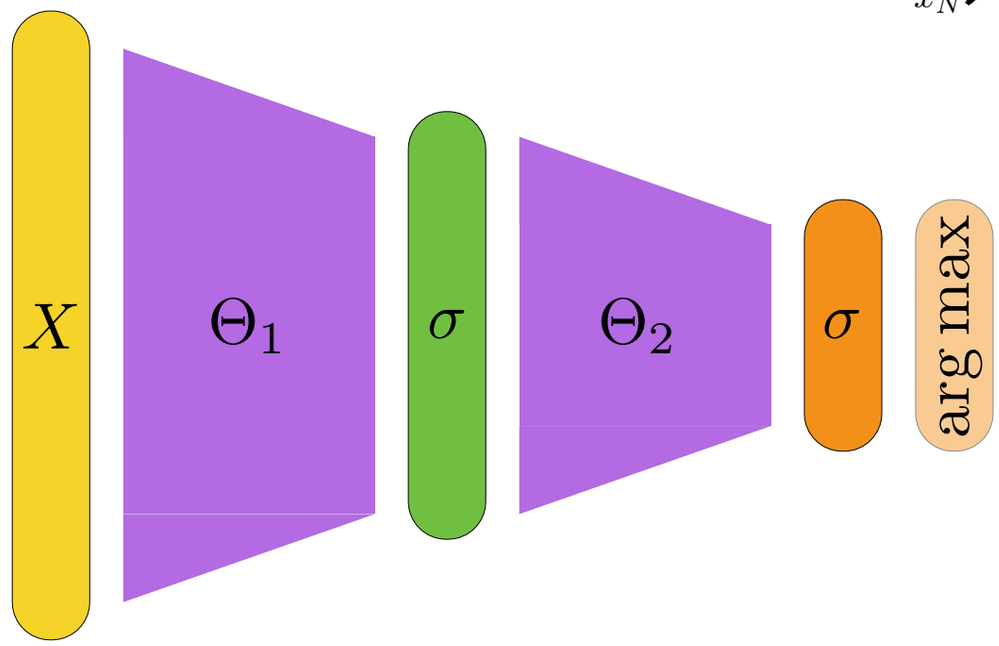
Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

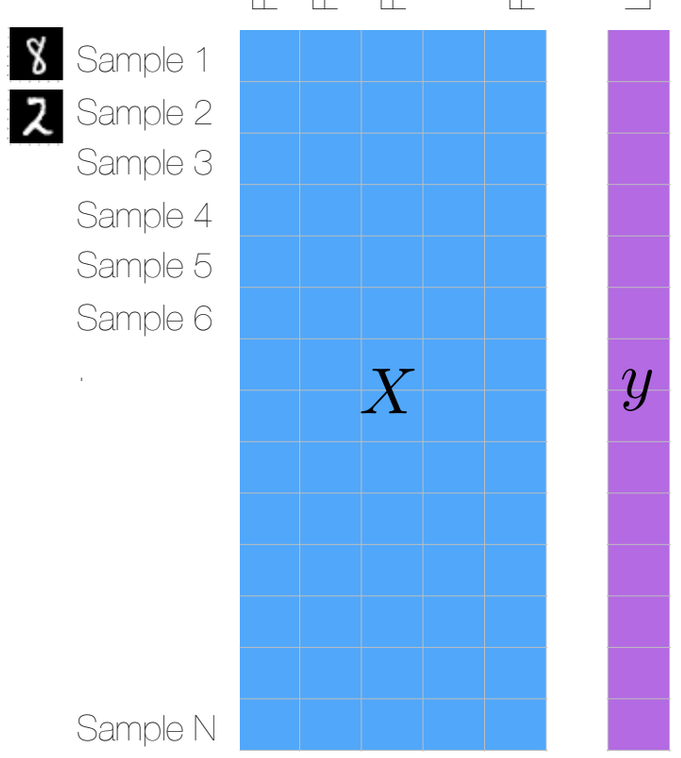


$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{in} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

3 layers:

- 1 input layer
- 1 hidden layer
- 1 output layer

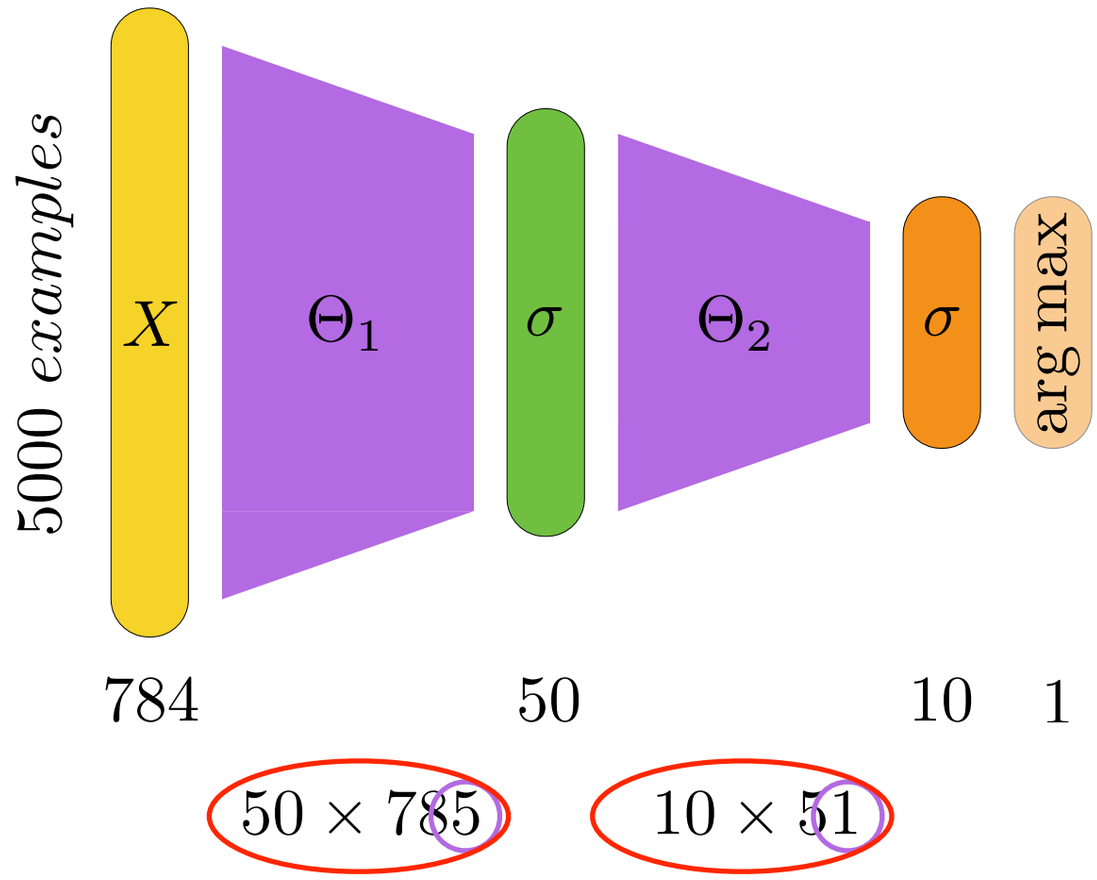




visualize_digits.py
convert_input.py

A practical example - MNIST

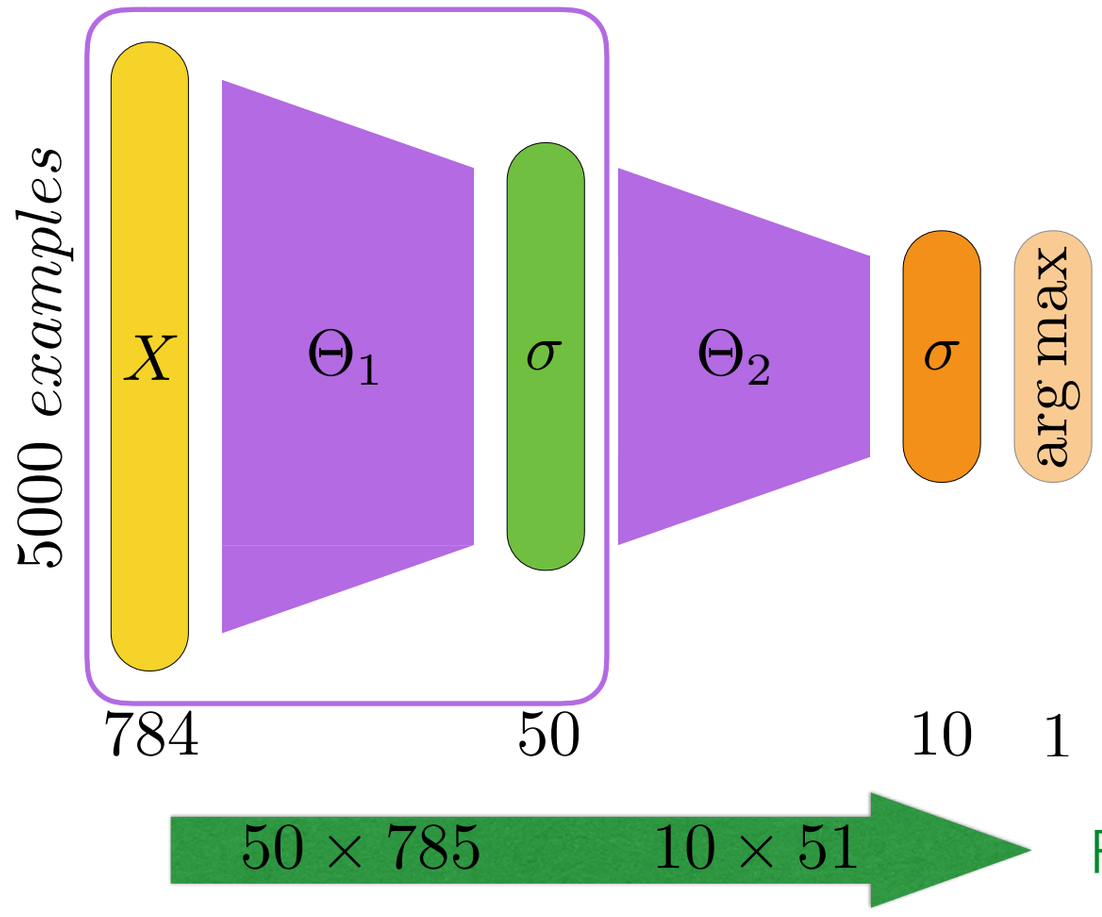
http://github.com/bmtgoncalves/Neural-Networks



nn.py
forward.py

A practical example - MNIST

http://github.com/bmtgoncalves/Neural-Networks



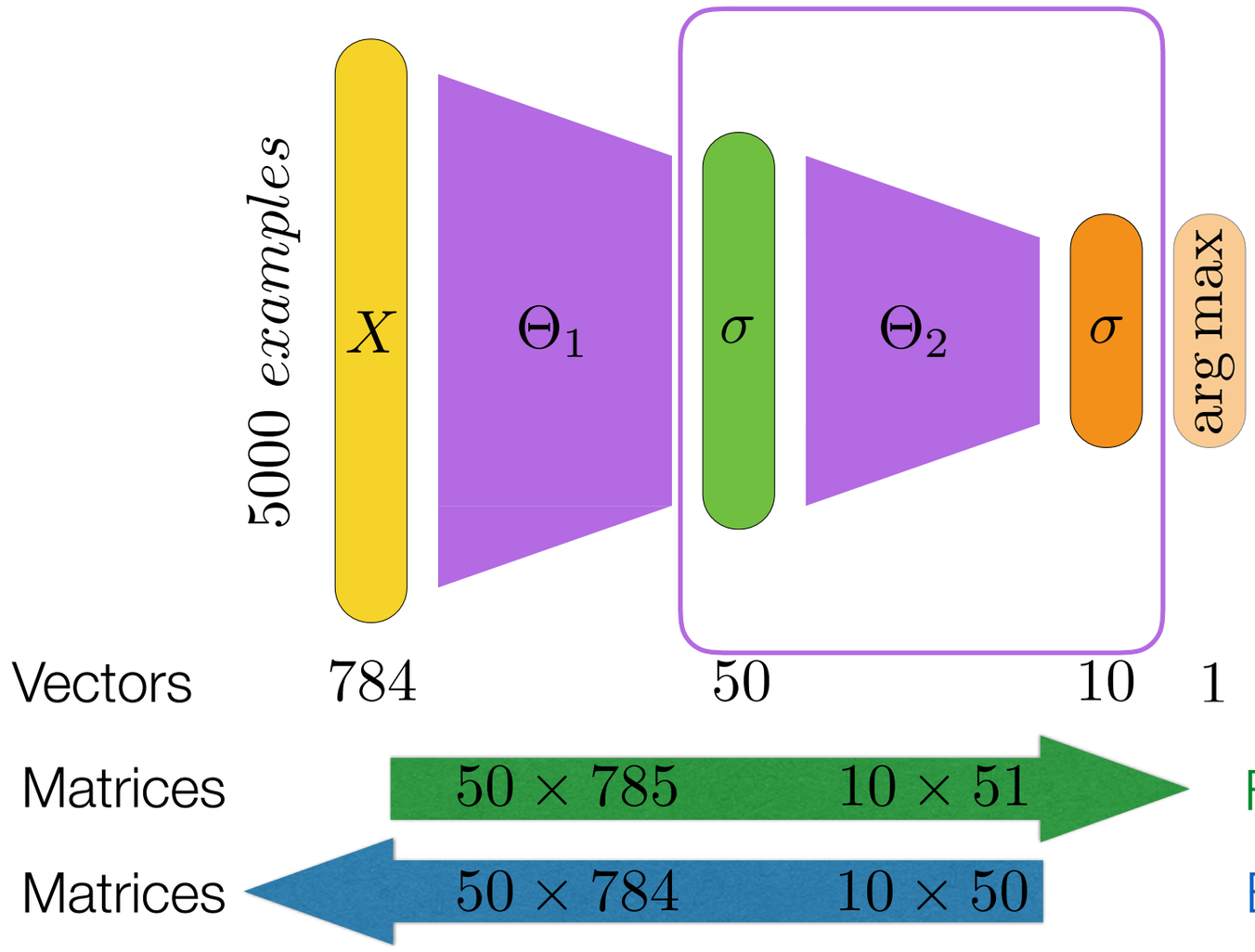

```python
def forward(Theta, X, active):
    N = X.shape[0]

    # Add the bias column
    X_ = np.concatenate((np.ones((N, 1)), X), 1)

    # Multiply by the weights
    z = np.dot(X_, Theta.T)

    # Apply the activation function
    a = active(z)

    return a
```

```python
def predict(Theta1, Theta2, X):
    h1 = forward(Theta1, X, sigmoid)
    h2 = forward(Theta2, h1, sigmoid)

    return np.argmax(h2, 1)
```

nn.py
forward.py

A practical example - MNIST

http://github.com/bmtgoncalves/Neural-Networks



nn.py
forward.py

Backprop

http://github.com/bmtgoncalves/Neural-Networks

```python
def backprop(Theta1, Theta2, X, y):
    N = X.shape[0]
    K = Theta2.shape[0]
    J = 0

    Delta2 = np.zeros(Theta2.shape)
    Delta1 = np.zeros(Theta1.shape)

    for i in range(N): # Forward propagation, saving intermediate results
        a1 = np.concatenate(([1], X[i]))  # Input layer
        z2 = np.dot(Theta1, a1)
        a2 = np.concatenate(([1], sigmoid(z2)))  # Hidden Layer
        z3 = np.dot(Theta2, a2)
        a3 = sigmoid(z3)  # Output layer
        y0 = one_hot(K, y[i])

        # Cross entropy
        J -= np.dot(y0.T, np.log(a3))+np.dot((1-y0).T, np.log(1-a3))

        # Calculate the weight deltas
        delta_3 = a3-y0
        delta_2 = np.dot(Theta2.T, delta_3)[1:]*sigmoidGradient(z2)
        Delta2 += np.outer(delta_3, a2)
        Delta1 += np.outer(delta_2, a1)

J /= N

Theta1_grad = Delta1/N
Theta2_grad = Delta2/N

return [J, Theta1_grad, Theta2_grad]
```

nn.py

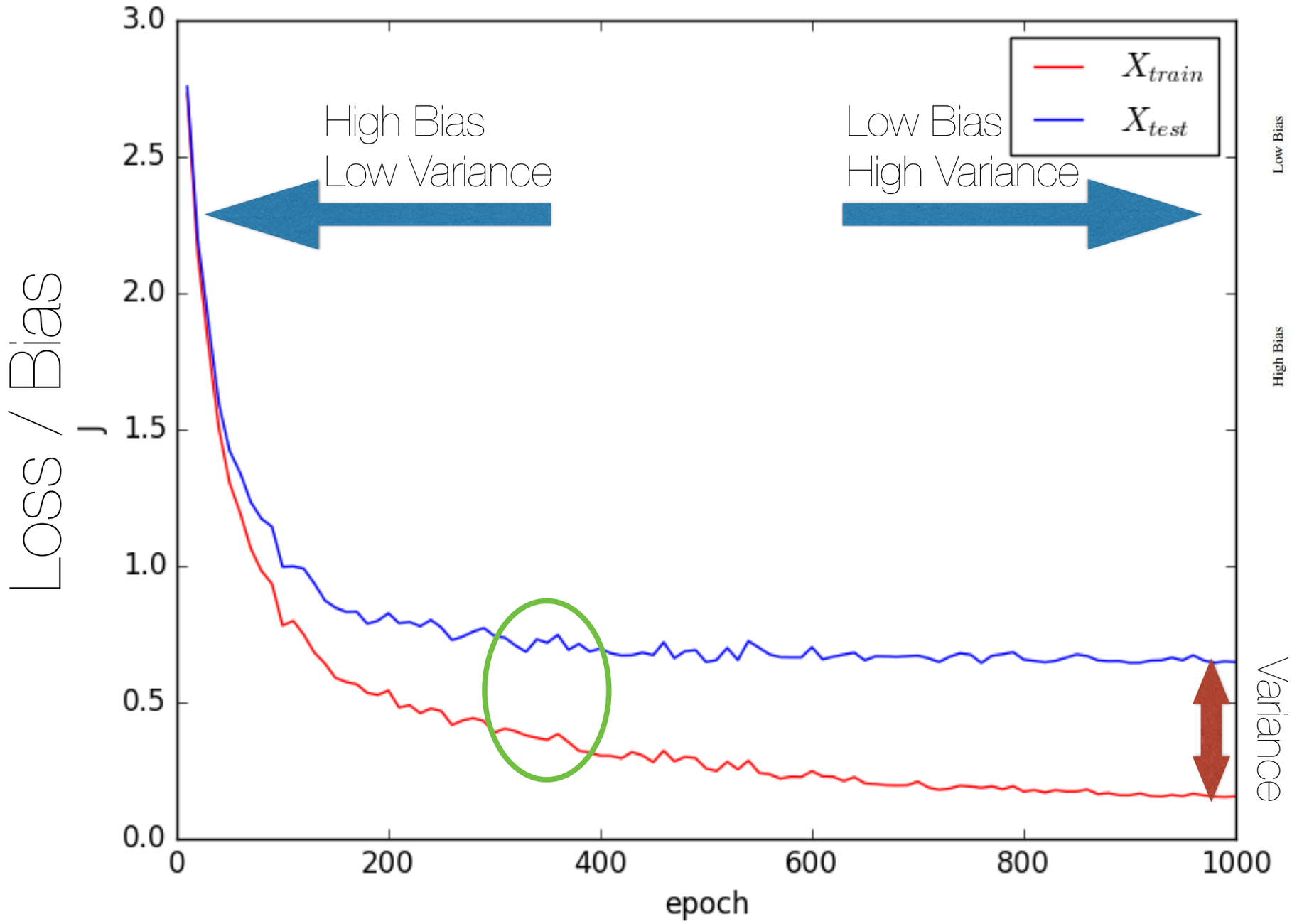
Training

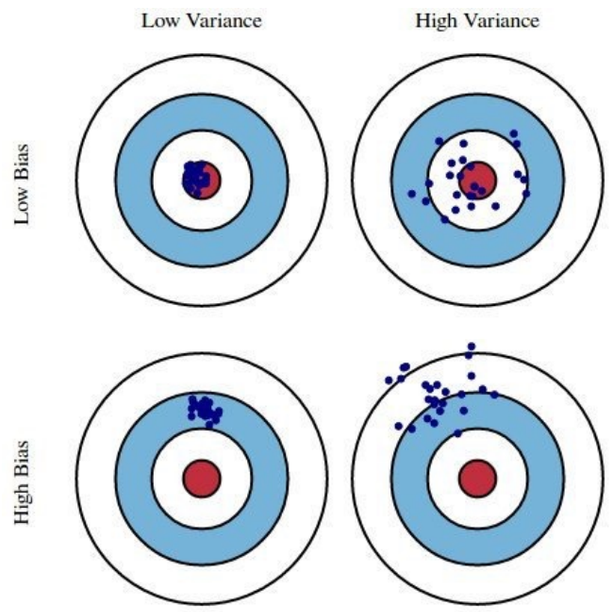
http://github.com/bmtgoncalves/Neural-Networks

```
Theta1 = init_weights(input_layer_size, hidden_layer_size)
Theta2 = init_weights(hidden_layer_size, num_labels)

iter = 0
tol = 1e-3
J_old = 1/tol
diff = 1

while diff > tol:
    J_train, Theta1_grad, Theta2_grad = backprop(Theta1, Theta2, X_train, y_train)

    diff = abs(J_old-J_train)
    J_old = J_train

    Theta1 -= .5*Theta1_grad
    Theta2 -= .5*Theta2_grad
```


nn.py

Bias-Variance Tradeoff

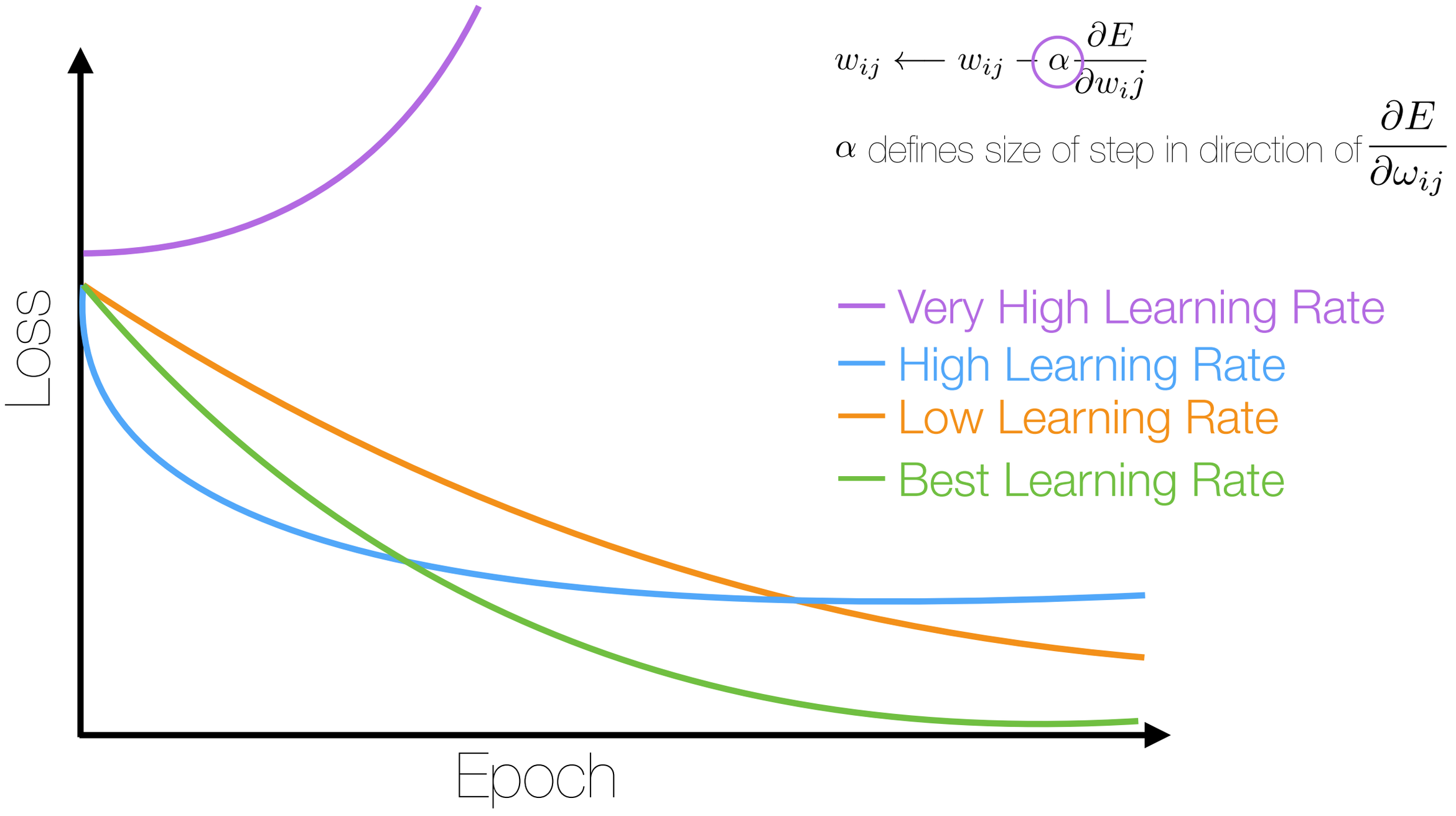




Learning Rate

$$w_{ij} \longleftarrow w_{ij} - \alpha \frac{\partial E}{\partial w_i j}$$

α defines size of step in direction of $\frac{\partial E}{\partial \omega_{ij}}$



Tips

- online learning - update weights after each case
  - might be useful to update model as new data is obtained
  - subject to fluctuations

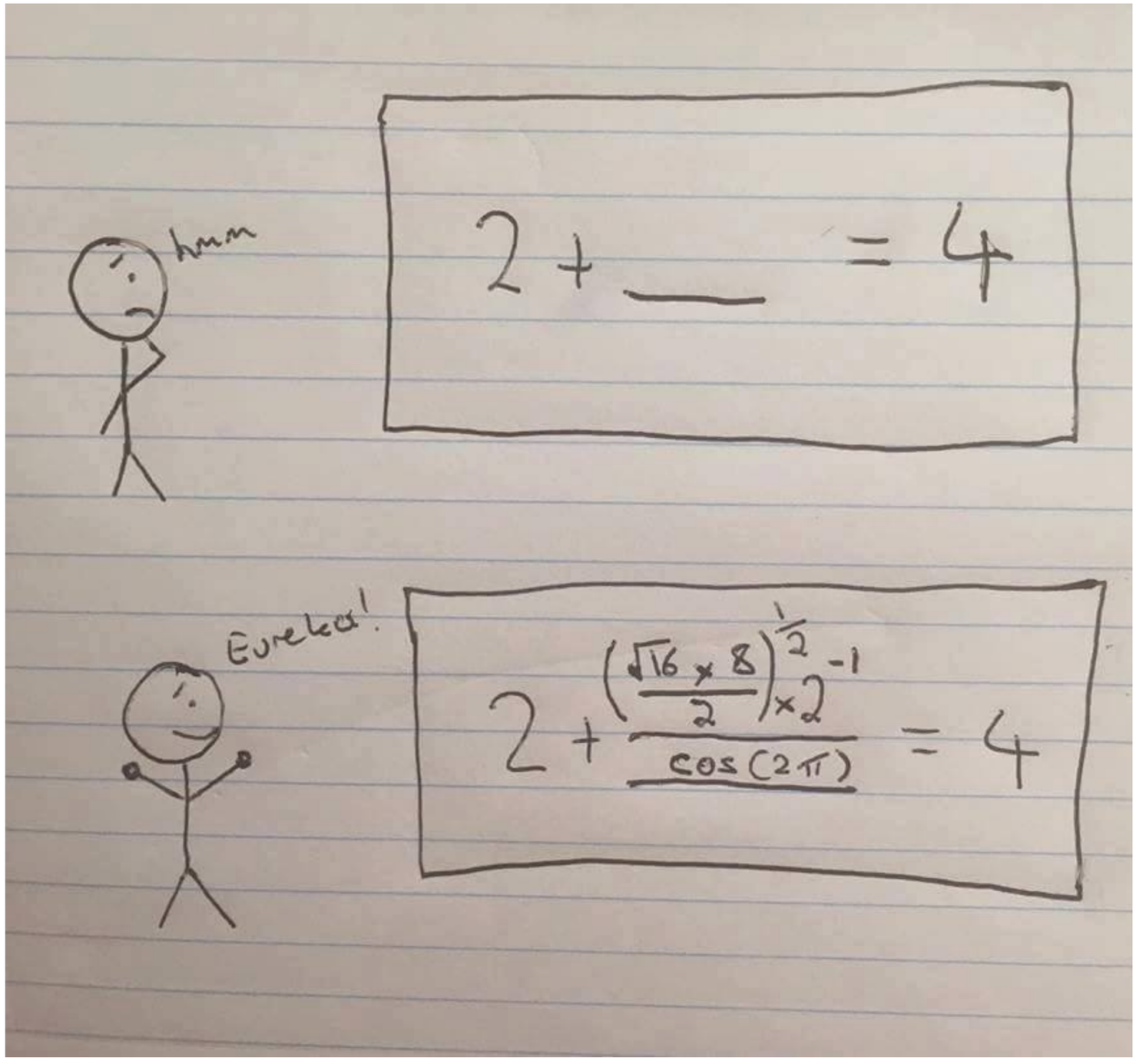
- mini-batch - update weights after a "small" number of cases
  - batches should be balanced
  - if dataset is redundant, the gradient estimated using only a fraction of the data is a good approximation to the full gradient.

- momentum - let gradient change the velocity of weight change instead of the value directly

- rmsprop - divide learning rate for each weight by a running average of "recent" gradients

- learning rate - vary over the course of the training procedure and use different learning rates for each weight

Interpretability



Teaching machines to read!

- Computers are really good at crunching numbers but not so much when it comes to words.

- Perhaps can we represent words numerically?

a	1
about	2
above	3
after	4
again	5
against	6
all	7
am	8
an	9
and	10
any	11
are	12
aren't	13
as	14
…	…

Teaching machines to read!

- Computers are really good at crunching numbers but not so much when it comes to words.

- Perhaps can we represent words numerically?

$$v_{after} = (0, 0, 0, 1, 0, 0, \cdots)^T$$
$$v_{above} = (0, 0, 1, 0, 0, 0, \cdots)^T$$

One-hot encoding

- Can we do it in a way that preserves semantic information?

"You shall know a word by the company it keeps"
(J. R. Firth)

- Words that have similar meanings are used in similar contexts and the context in which a word is used helps us understand it's meaning.

The red house is beautiful.
The blue house is old.
The red car is beautiful.
The blue car is old.

Teaching machines to read!

"You shall know a word by the company it keeps"
(J. R. Firth)

➡Words with similar meanings should have similar representations.

➡From a word we can get some idea about the context where it might appear

___ ___ house __ _____.
___ ___ car __ _____.

$$\max p\left(C|w\right)$$

➡And from the context we have some idea about possible words

The red _____ is beautiful.
The blue _____ is old.

$$\max p\left(w|C\right)$$

word2vec

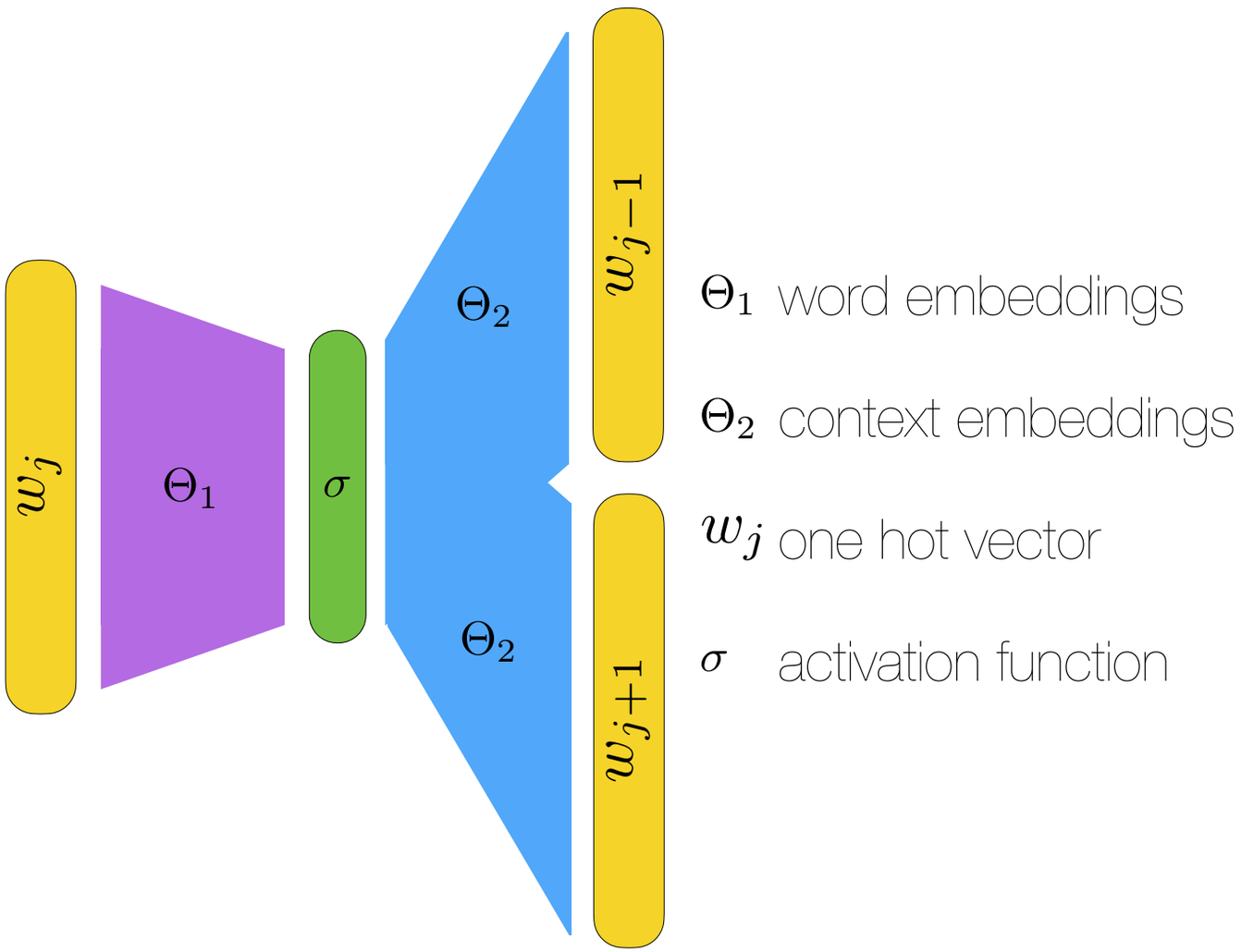
Mikolov 2013

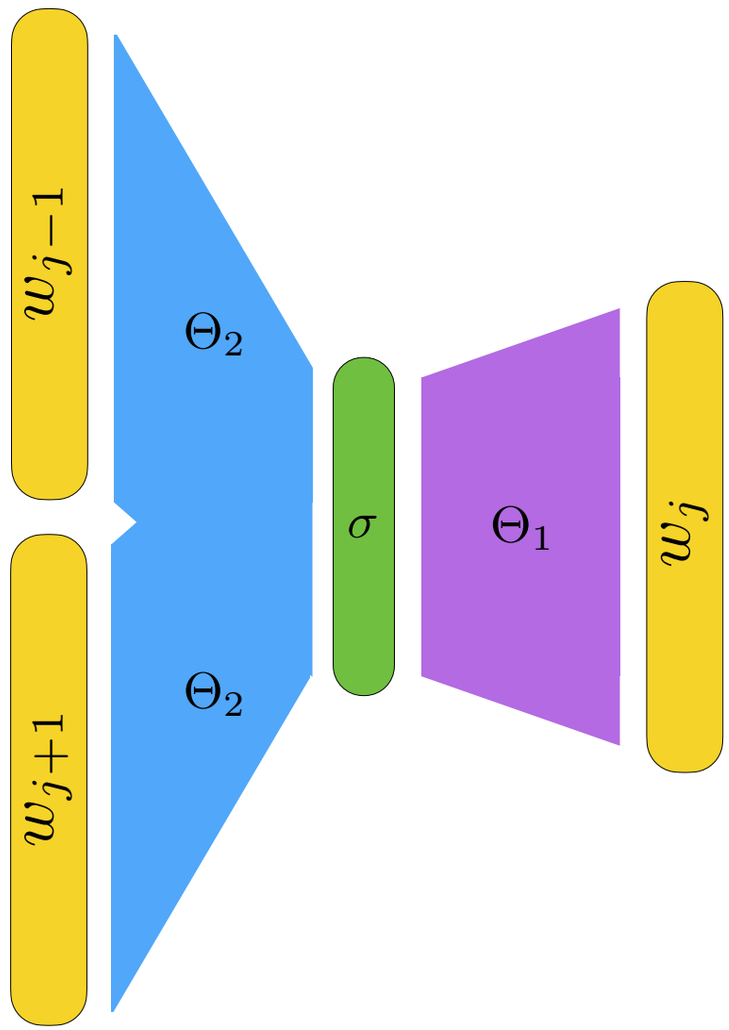
Skipgram

$$\max p\left(C|w\right)$$

Continuous Bag of Words

$$\max p\left(w|C\right)$$





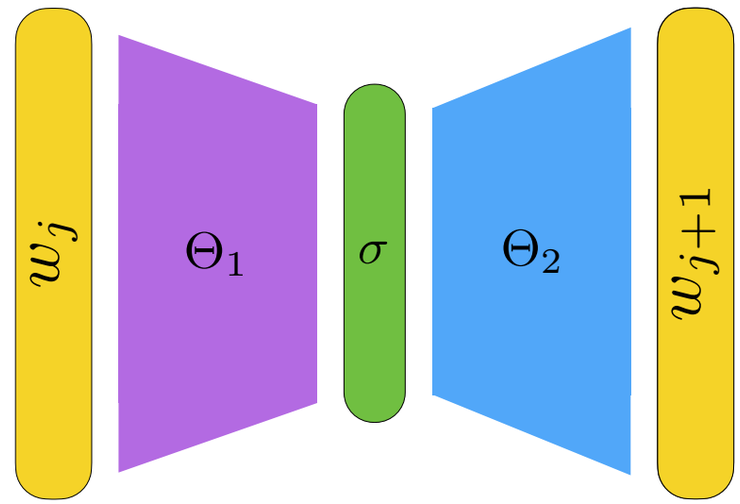
Skipgram

- Let us take a better look at a simplified case with a single context word

- Words are one-hot encoded vectors $w_j = (0, 0, 1, 0, 0, 0, \cdots)^T$ of length V

- Θ_1 is an $(M \times V)$ matrix so that when we take the product:

$$\Theta_1 \cdot w_j$$



Skipgram

- Let us take a better look at a simplified case with a single context word

- Words are one-hot encoded vectors $w_j = (0, 0, 1, 0, 0, 0, \cdots)^T$ of length V

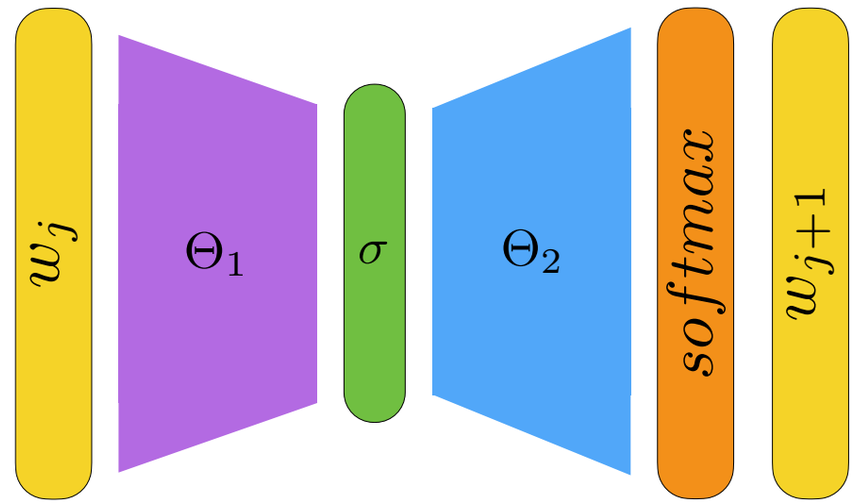
- Θ_1 is an $(M \times V)$ matrix so that when we take the product:

$$\Theta_1 \cdot w_j$$

- We are effectively selecting the j'th column of Θ_1:

$$v_j = \Theta_1 \cdot w_j$$

- The linear activation function simply passes this value along which is then multiplied by Θ_2, a $(V \times M)$ matrix.



- Each element k of the output layer its then given by:

$$u_k^T \cdot v_j$$

- We convert these values to a normalized probability distribution by using the softmax

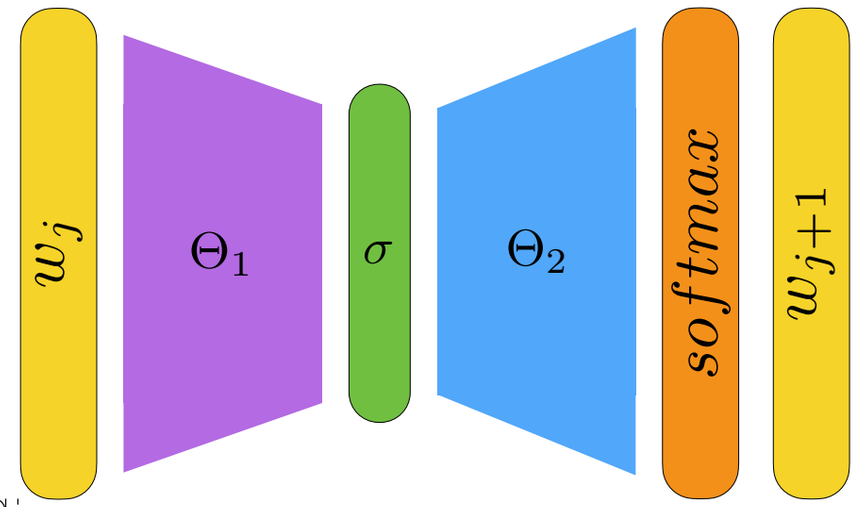
Softmax

- A standard way of converting a set of number to a normalized probability distribution:

$$softmax\left(x\right) = \frac{\exp\left(x_j\right)}{\sum_l \exp\left(x_l\right)}$$

- With this final ingredient we obtain:

$$p\left(w_k|w_j\right) \equiv softmax\left(u_k^T \cdot v_j\right) = \frac{\exp\left(u_k^T \cdot v_j\right)}{\sum_l \exp\left(u_l^T \cdot v_j\right)}$$



- Our goal is then to learn:

$$\Theta_1 \qquad \Theta_2$$

- so that we can predict what the next word is likely to be using:

$$p\left(w_{j+1}|w_j\right)$$

- But how can we quantify how far we are from the correct answer? Our error measure shouldn't be just binary (right or wrong)…

Cross-Entropy

- First we have to recall that what we are, in effect, comparing two probability distributions:

$$p\left(w_k|w_j\right)$$

- and the one-hot encoding of the context:

$$w_{j+1} = (0, 0, 0, 1, 0, 0, \cdots)^T$$

- The Cross Entropy measures the distance, in number of bits, between two probability distributions p and q:

$$H\left(p, q\right) = -\sum_k p_k \log q_k$$

- In our case, this becomes:

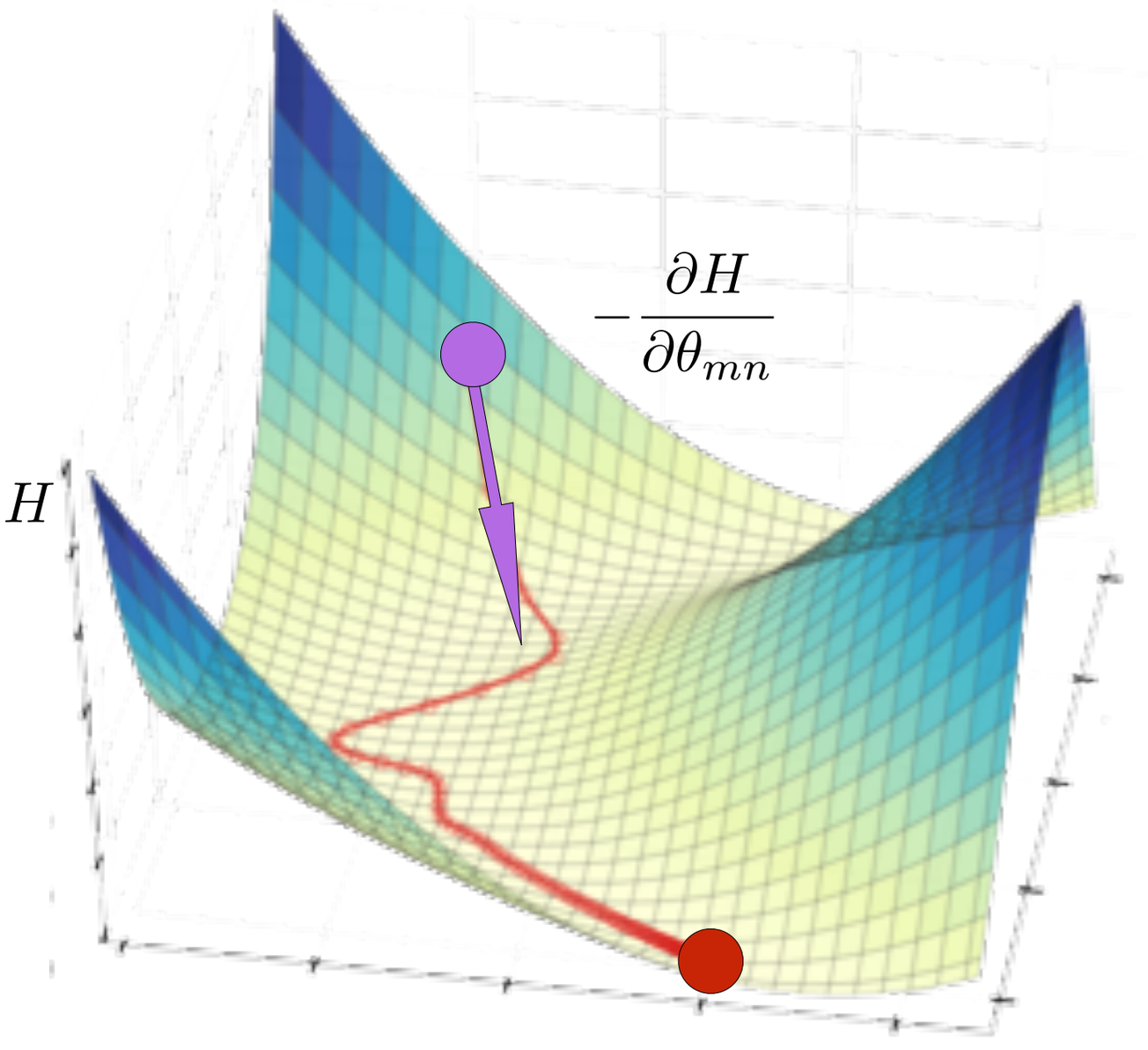
$$H\left[w_{j+1}, p\left(w_k|w_j\right)\right] = -\sum_k w_{j+1}^k \log p\left(w_k|w_j\right)$$

- So it's clear that the only non zero term is the one that corresponds to the "hot" element of w_{j+1}

$$H = -\log p\left(w_{j+1}|w_j\right)$$

- This is our Error function. But how can we use this to update the values of Θ_1 and Θ_2?

Gradient Descent



- Find the gradient for each training batch
- Take a step **downhill** along the direction of the gradient

$$\theta_{mn} \leftarrow \theta_{mn} - \alpha \frac{\partial H}{\partial \theta_{mn}}$$

- where α is the step size.
- Repeat until "convergence".

Chain-rule

- How can we calculate

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \log p\left(w_{j+1}|w_j\right) \qquad \theta_{mn} = \left\{\theta_{mn}^{(1)}, \theta_{mn}^{(2)}\right\}$$

- we rewrite:

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \log \frac{\exp\left(u_k^T \cdot v_j\right)}{\sum_l \exp\left(u_l^T \cdot v_j\right)}$$

- and expand:

$$\frac{\partial H}{\partial \theta_{mn}} = \frac{\partial}{\partial \theta_{mn}} \left[u_k^T \cdot v_j - \log \sum_l \exp\left(u_l^T \cdot v_j\right)\right]$$

- Then we can rewrite:

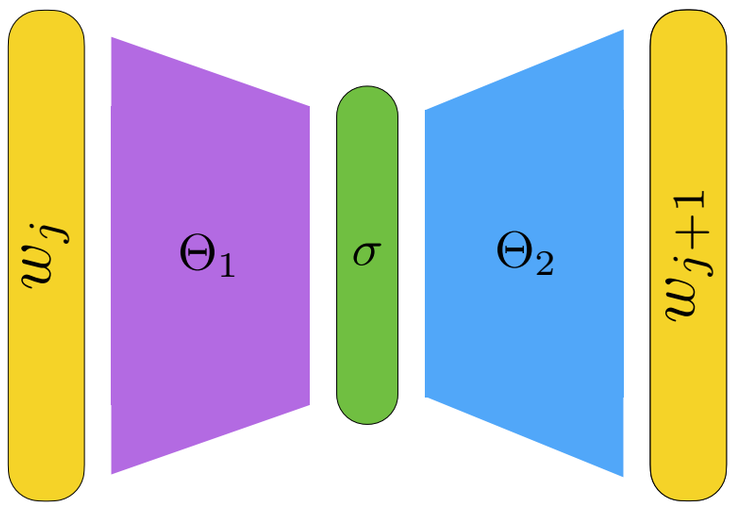
$$u_k^T \cdot v_j = \sum_q \theta_{kq}^{(2)} \theta_{qj}^{(1)}$$

- and apply the chain rule:

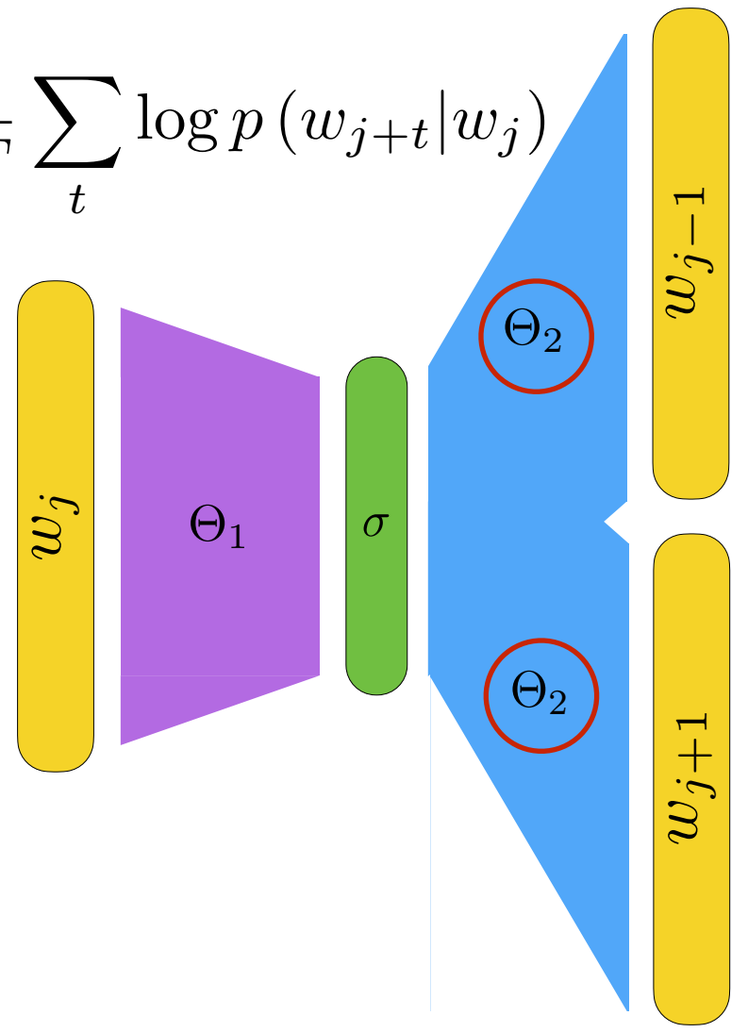
$$\frac{\partial f\left(g\left(x\right)\right)}{\partial x} = \frac{\partial f\left(g\left(x\right)\right)}{\partial g\left(x\right)} \frac{\partial g\left(x\right)}{\partial x}$$

SkipGram with Larger Contexts

$$H = -\log p\left(w_{j+1}|w_j\right)$$



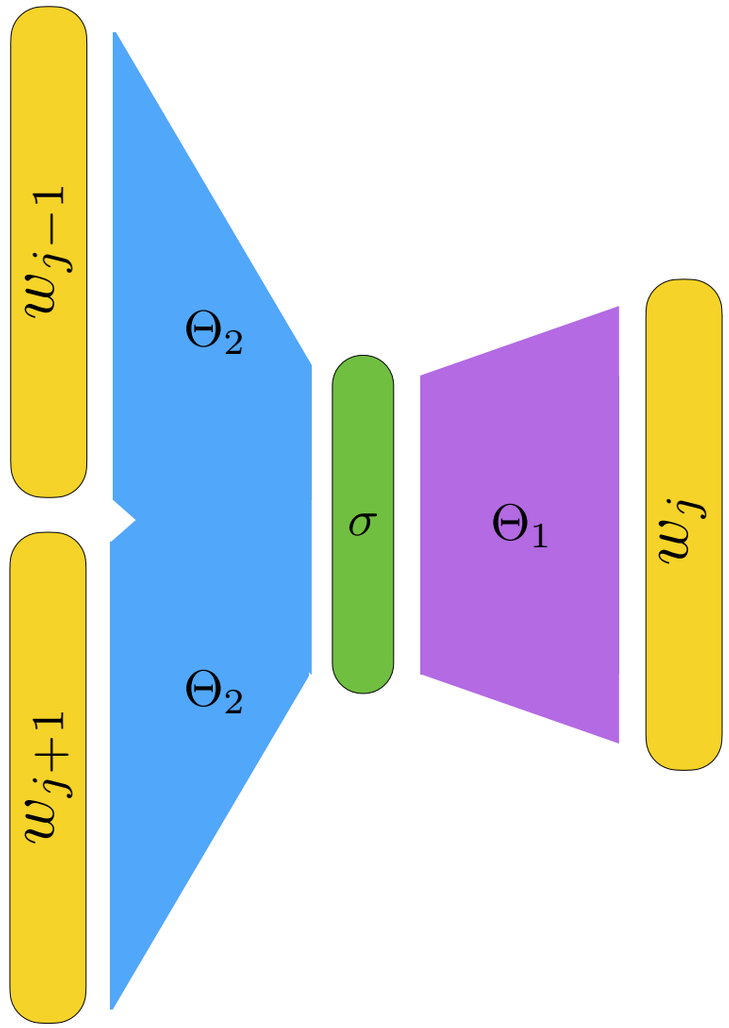
$$H = -\frac{1}{T}\sum_t \log p\left(w_{j+t}|w_j\right)$$



- Use the same Θ_2 for all context words.
- Use the average of cross entropy.
- word order is not important (the average does not change).

Continuous Bag of Words

- The process is essentially the same

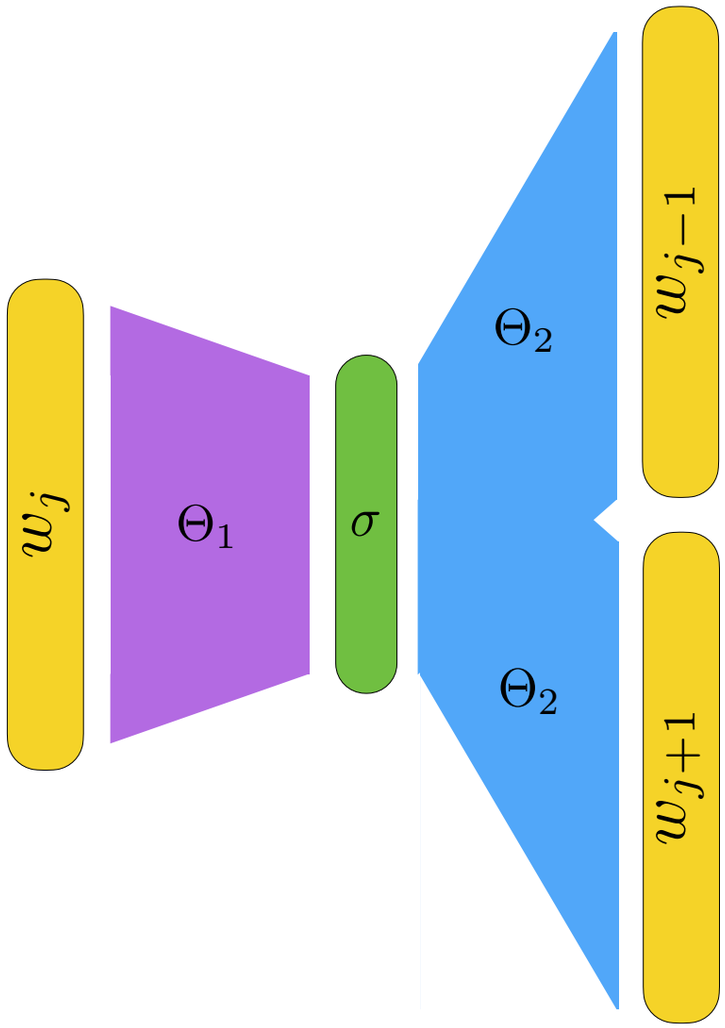


Variations

- Hierarchical Softmax:
  - Approximate the softmax using a binary tree
  - Reduce the number of calculations per training example from V to $\log_2 V$ and increase performance by orders of magnitude.
- Negative Sampling:
  - Under sample the most frequent words by removing them from the text before generating the contexts
  - Similar idea to removing stop-words — very frequent words are less informative.
  - Effectively makes the window larger, increasing the amount of information available for context

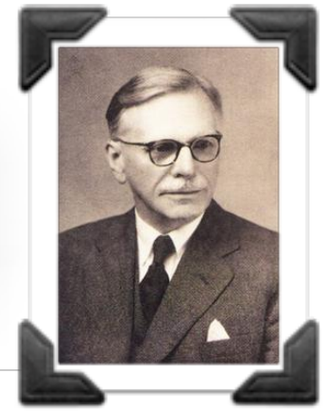
Comments

- word2vec, even in its original formulation is actually a family of algorithms using various combinations of:
  - Skip-gram, CBOW
  - Hierarchical Softmax, Negative Sampling
- The output of this neural network is deterministic:
  - If two words appear in the same context ("blue" vs "red", for e.g.), they will have similar internal representations in Θ_1 and Θ_2
  - Θ_1 and Θ_2 are vector embeddings of the input words and the context words respectively
- Words that are too rare are also removed.
- The original implementation had a dynamic window size:
  - for each word in the corpus a window size k' is sampled uniformly between 1 and k



Analogies

"You shall know a word by the company it keeps"
(J. R. Firth)



- The embedding of each word is a function of the context it appears in:

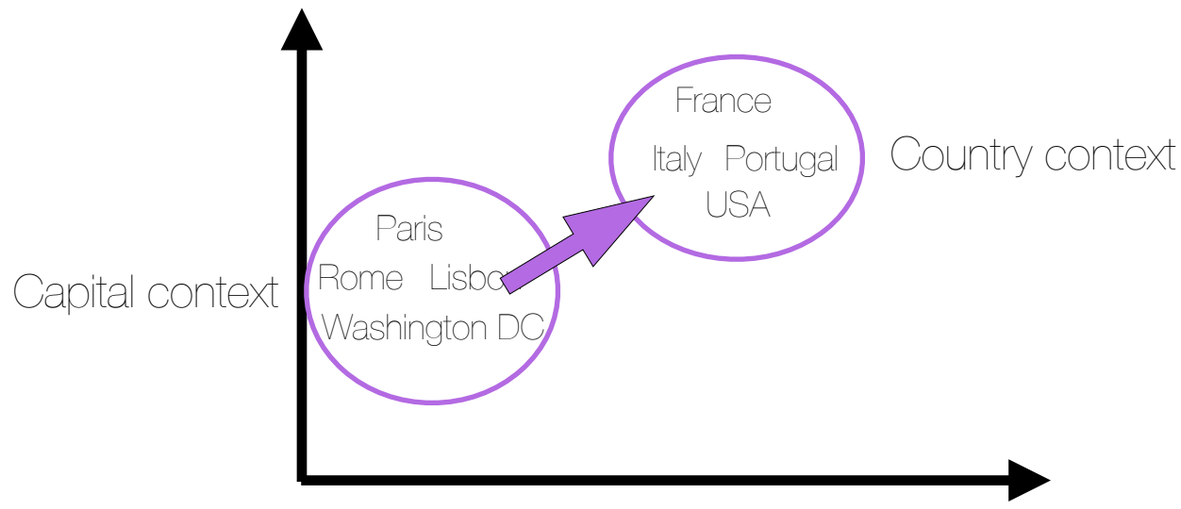
$$\sigma\left(red\right) = f\left(context\left(red\right)\right)$$

- words that appear in similar contexts will have similar embeddings:

$$context\left(red\right) \approx context\left(blue\right) \implies \sigma\left(red\right) \approx \sigma\left(blue\right)$$

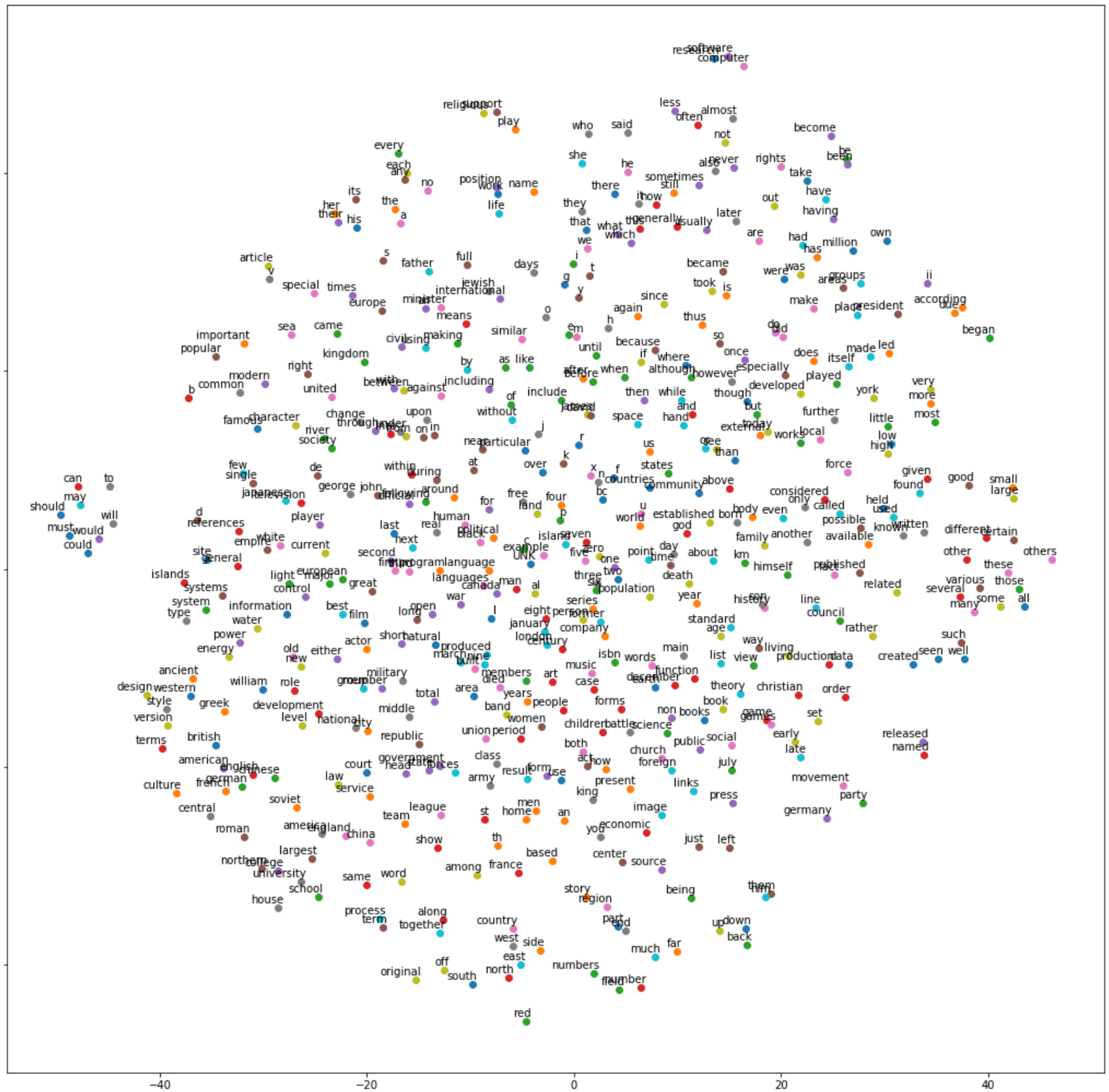
- "Distributional hypotesis" in linguistics

Geometrical relations between contexts imply semantic relations between words!

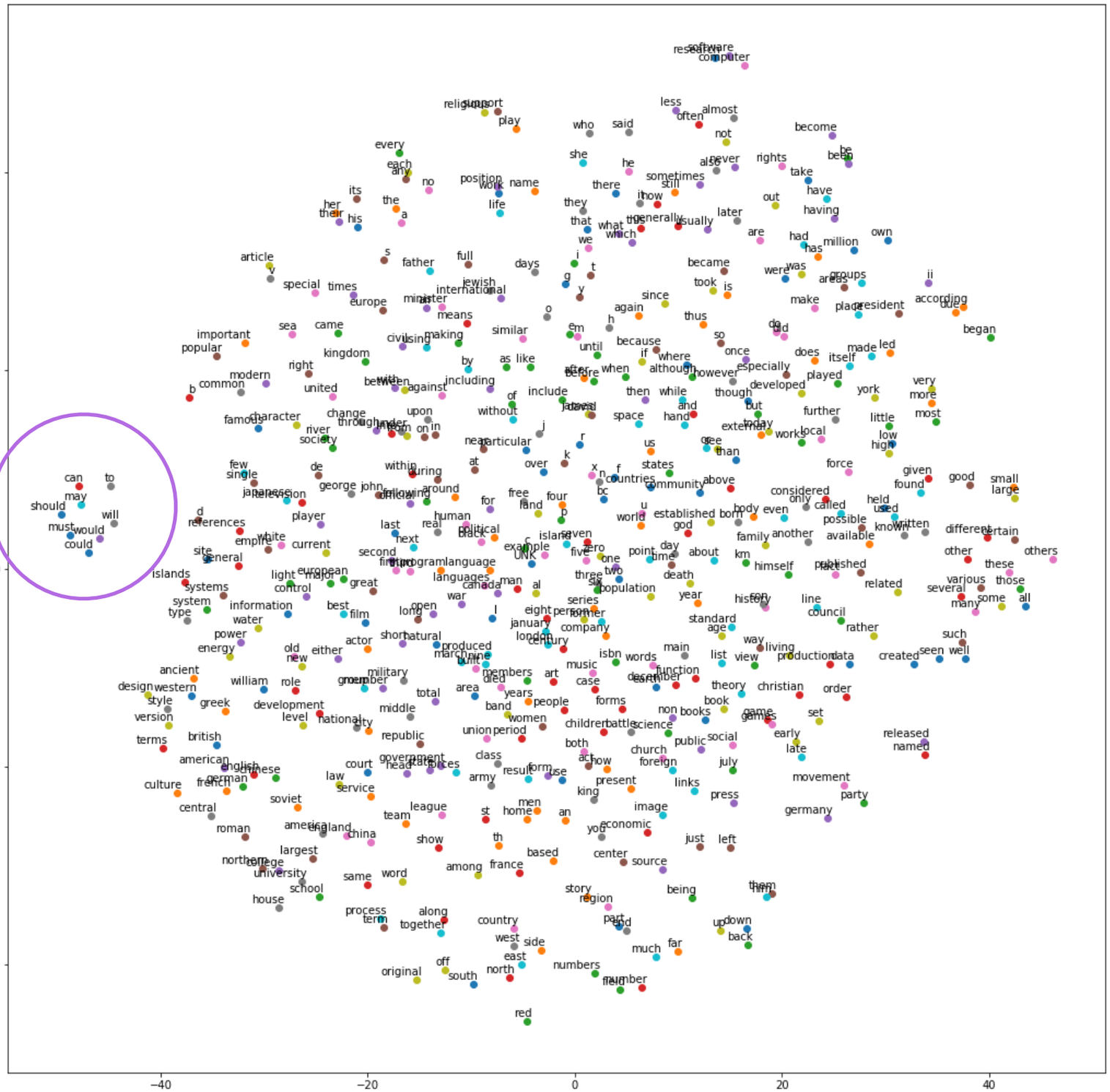


$$\sigma\left(France\right) - \sigma\left(Paris\right) + \sigma\left(Rome\right) = \sigma\left(Italy\right)$$

Visualization

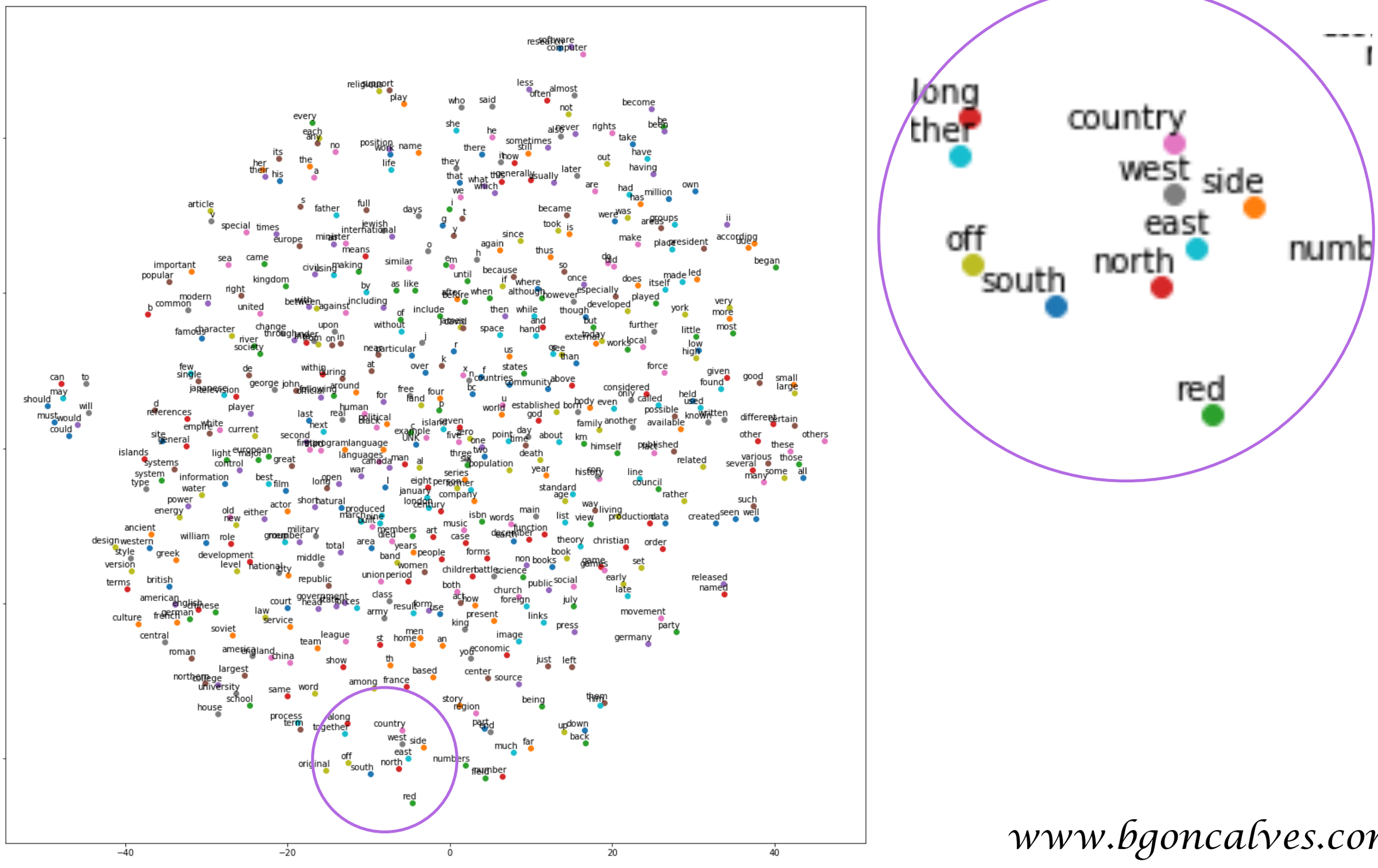


Visualization



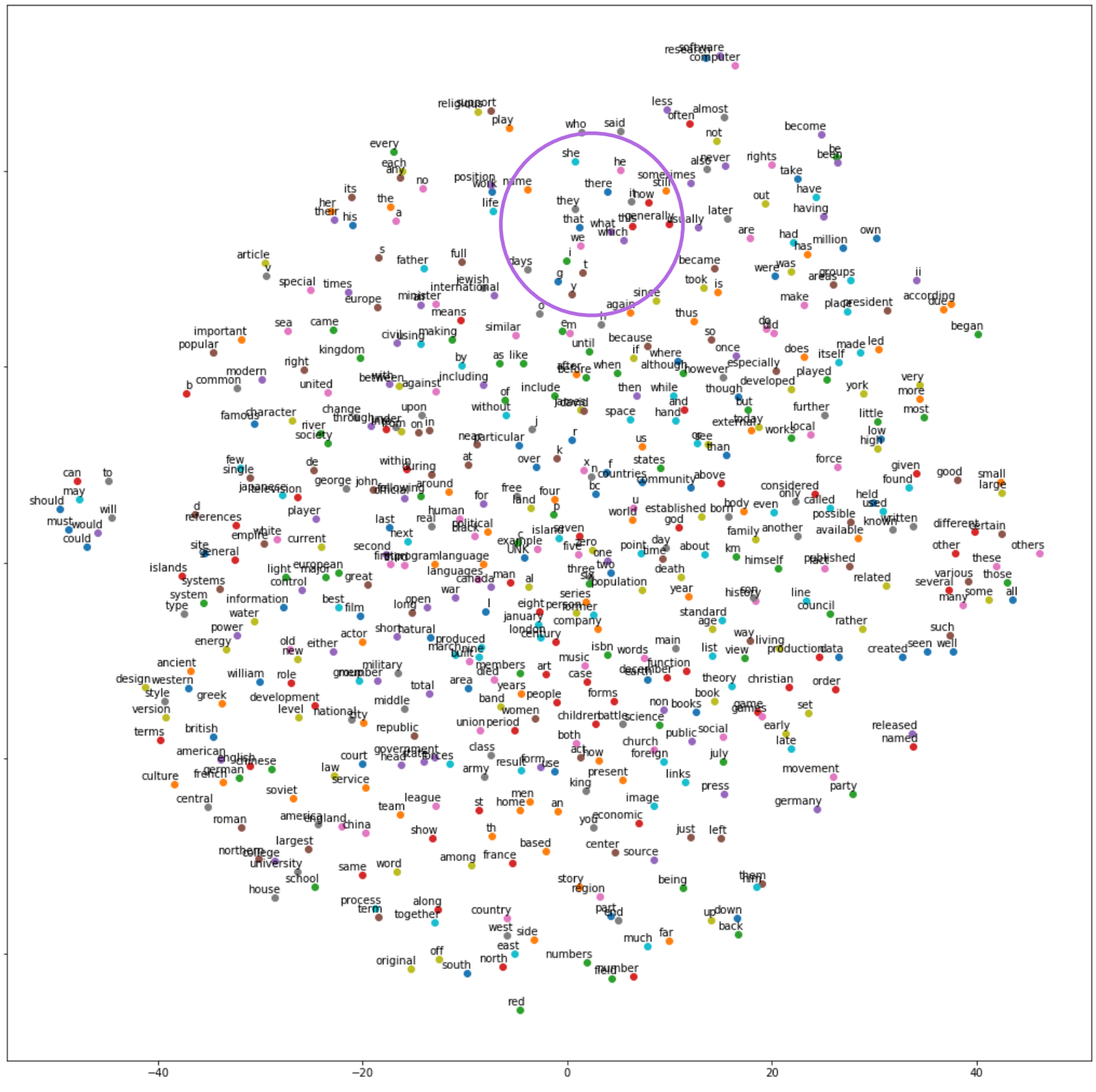
www.bgoncalves.com

Visualization



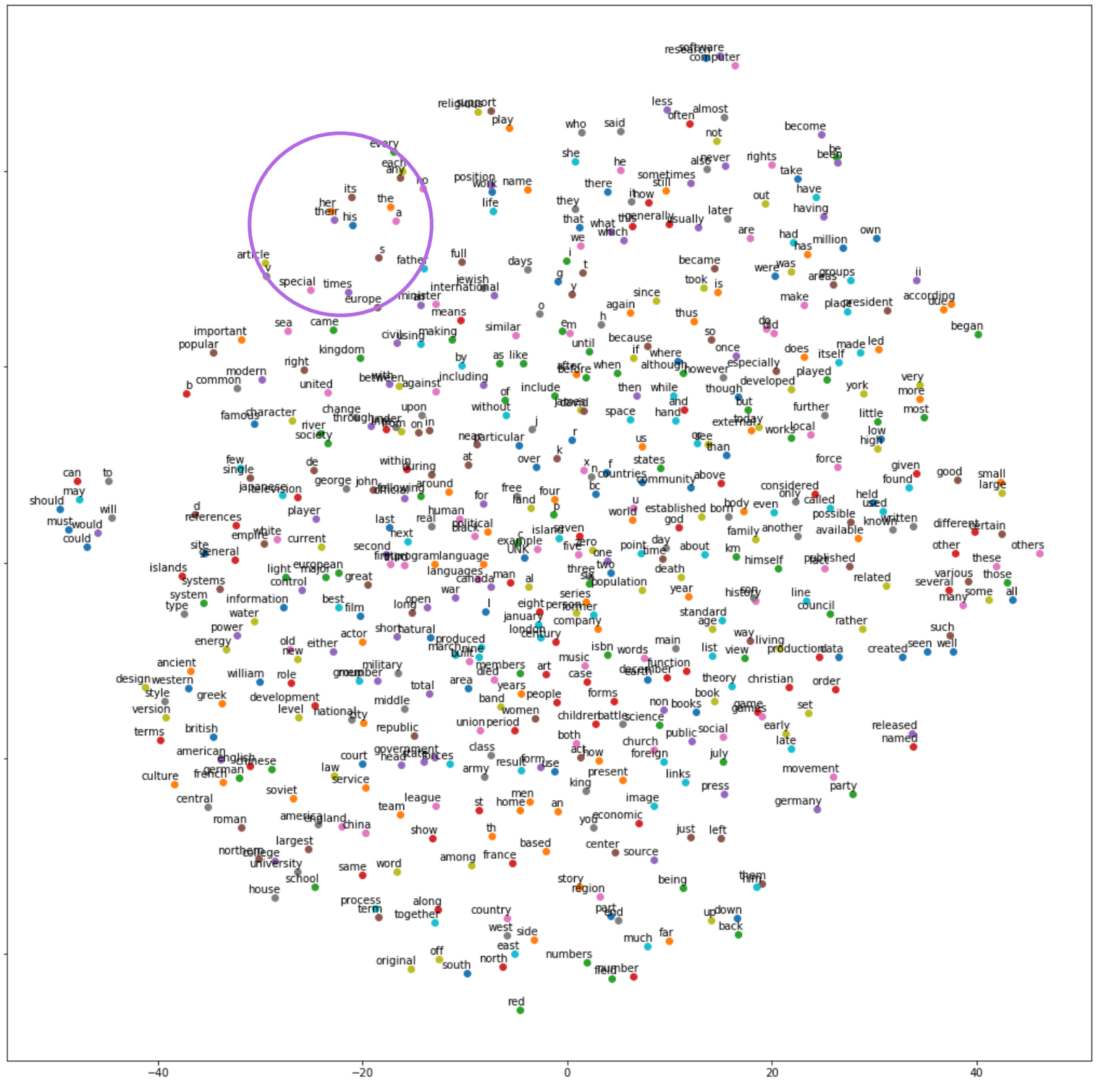
www.bgoncalves.com

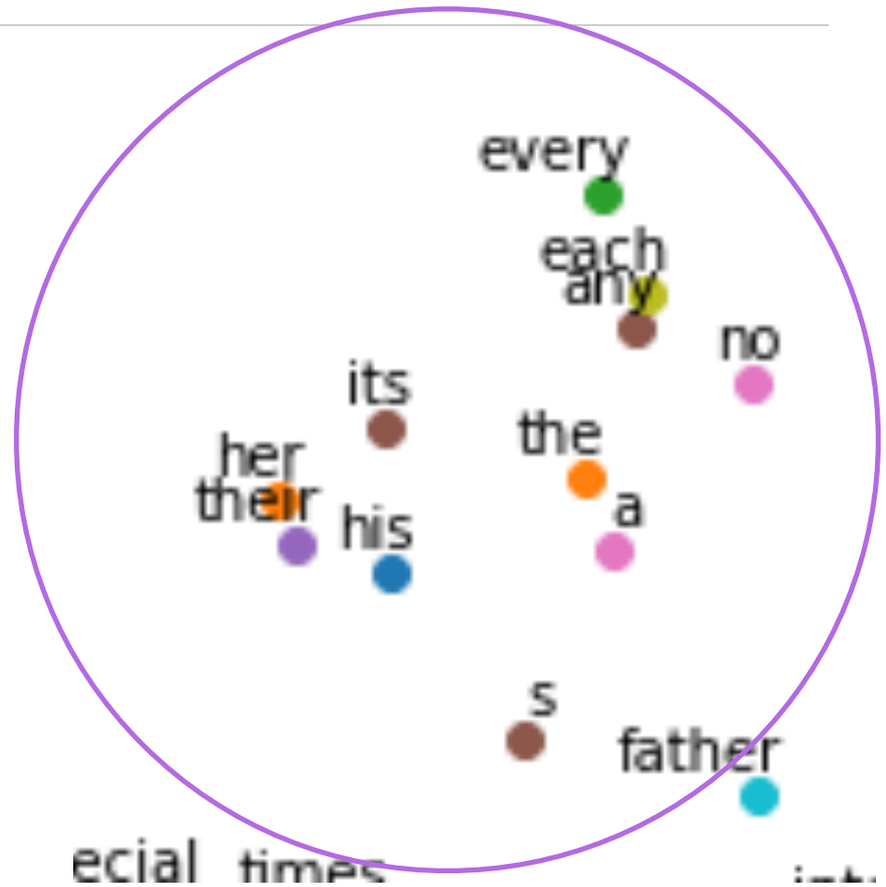
Visualization



www.bgoncalves.com

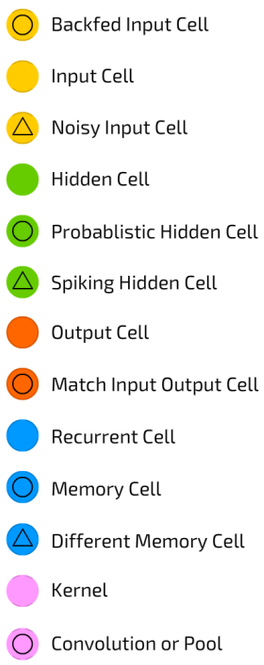
Visualization



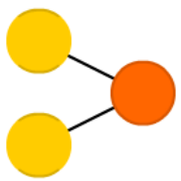


Deep Learning...

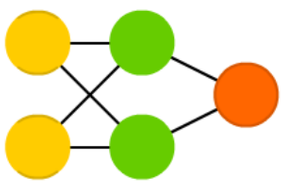
Neural Network Architectures



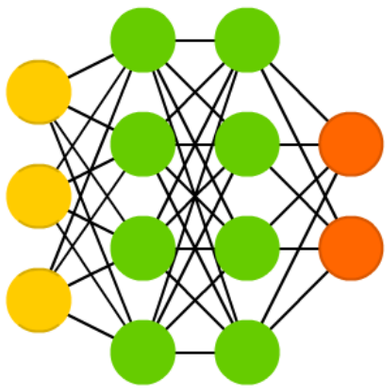
Perceptron (P)



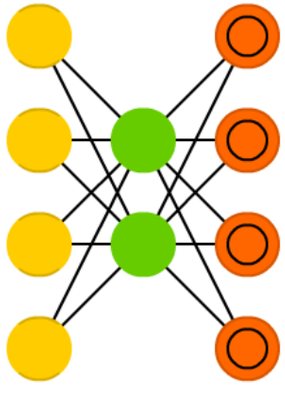
Feed Forward (FF)



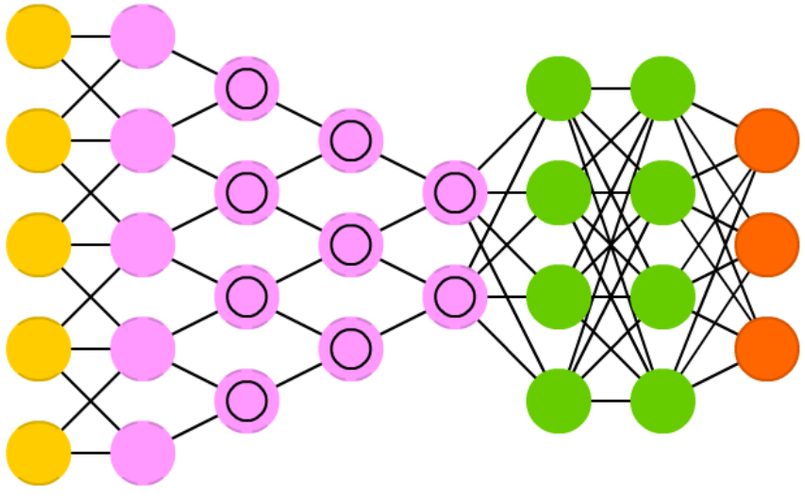
Deep Feed Forward (DFF)



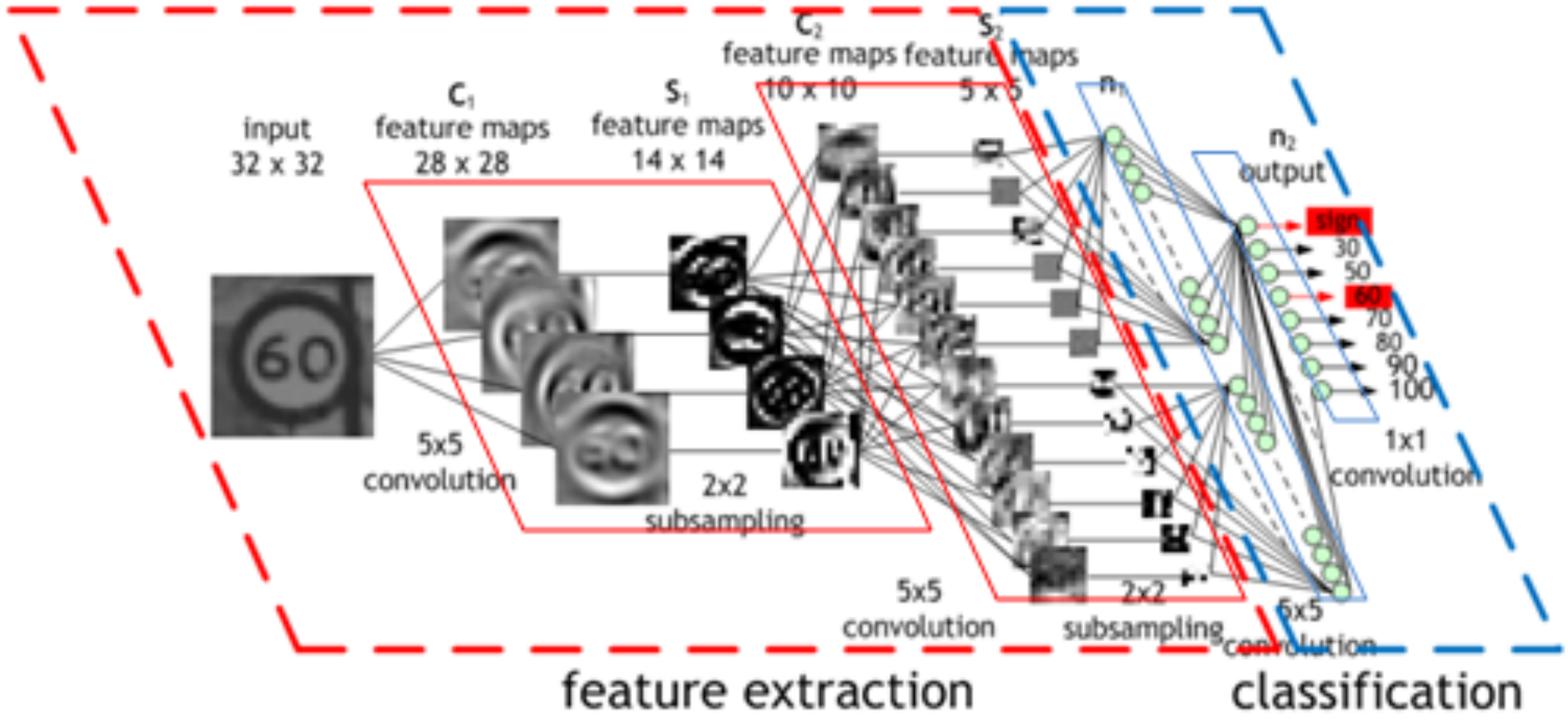
Auto Encoder (AE)



Deep Convolutional Network (DCN)



Convolutional Neural Networks



Convolutional Neural Networks