Machine Learning for Language Modelling

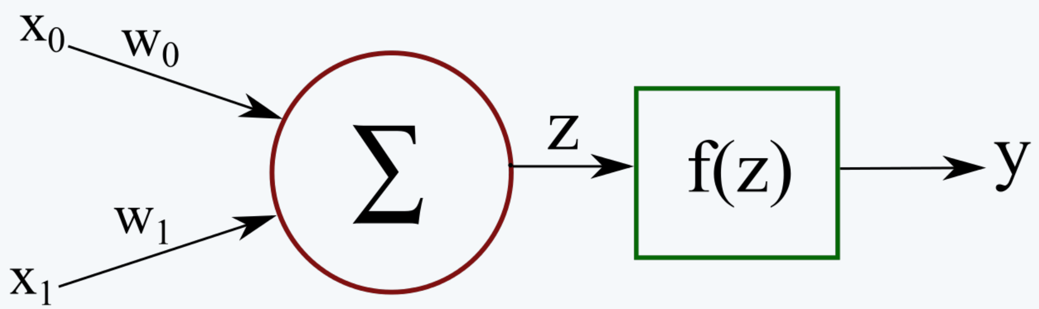
Part 4: Neural network LM optimisation

Marek Rei

UNIVERSITY OF TARTU 1632

UNIVERSITY OF CAMBRIDGE

Recap

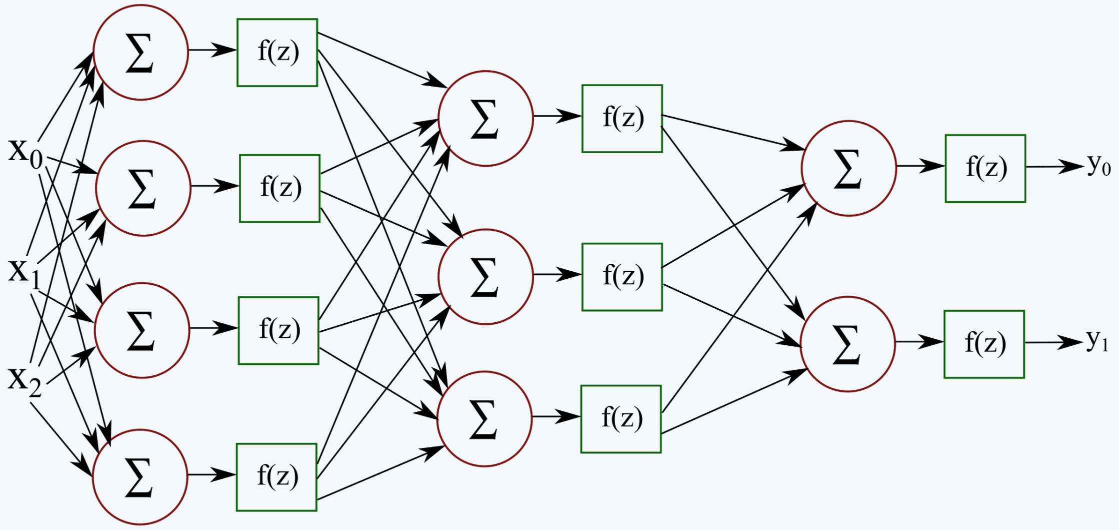


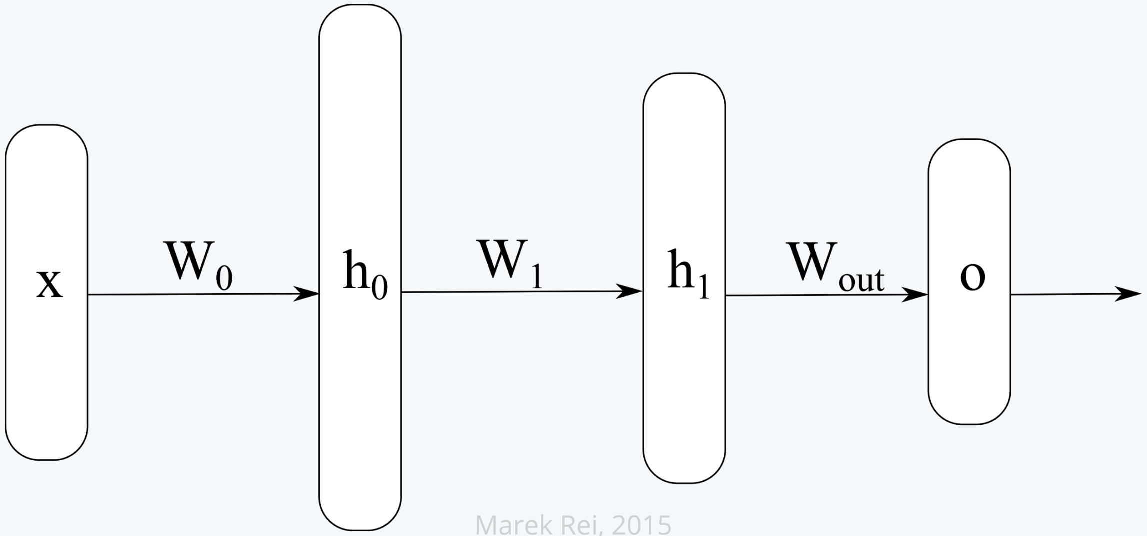
$$x = \begin{vmatrix} x_0 \\ x_1 \end{vmatrix} \qquad W = \begin{vmatrix} w_0 & w_1 \end{vmatrix}$$

$$z = W \cdot x \qquad f(z) = \frac{1}{1 + e^{-z}}$$

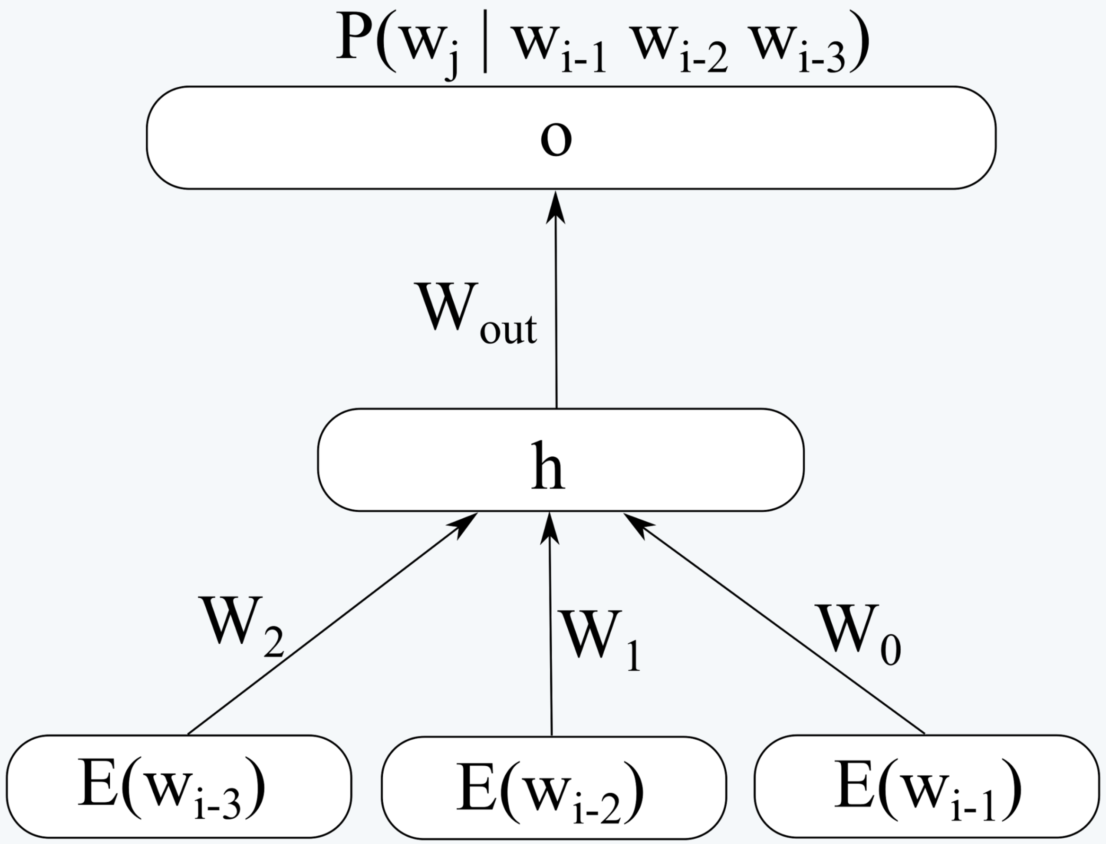
$$y = f(z) = f(W \cdot x)$$

Recap





Recap



$$\begin{aligned} z = & W_2 \cdot E(w_{i-3}) \\ & + W_1 \cdot E(w_{i-2}) \\ & + W_0 \cdot E(w_{i-1}) \end{aligned}$$

$$h = f(z)$$

$$s = W_{out} \cdot h$$

$$o = softmax(s)$$

Neural network training

How do we find values for the weight matrices?

Gradient descent training

Start with random weight values

For each training example:

1. Calculate the network prediction
2. Measure the error between prediction and the correct label
3. Update all the parameters, so that they would make a smaller error on this training example

Loss function

Loss function is used to calculate the error between the predicted value and the correct value.

For example: mean squared error

$$L = \sum_{k} \frac{1}{2}(y_k - \hat{y_k})^2$$

y	-	predicted value
$\hat{y}$	-	correct value (gold standard)
k	-	training examples

Loss function

$$L = \sum_k \frac{1}{2}(y_k - \hat{y_k})^2$$

We want to update our model parameters, so that the model has a smaller loss L

For each parameter w, we can calculate the derivative of L with respect to w:

$$\frac{\partial L}{\partial w}$$

Derivatives

Partial derivative - the derivative with respect to one parameter, while keeping the others constant.

$\frac{\partial L}{\partial w}$ - How much does L change, when we slightly increase w

$\frac{\partial L}{\partial w} > 0$ - If we increase w, L will also increase

$\frac{\partial L}{\partial w} < 0$ - If we increase w, L will decrease

Derivatives

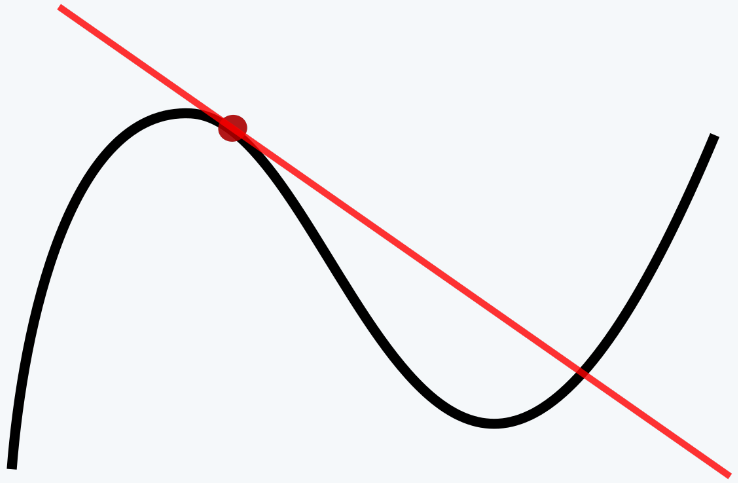
$$\frac{d}{dx}x^r = r \cdot x^{r-1}$$

$$\frac{d}{dx}e^x = e^x$$

$$\frac{d}{dx}log_e(x) = \frac{d}{dx}ln(x) = \frac{1}{x}$$

If $f(x) = h(g(x))$, then

$$f'(x) = h'(g(x)) \cdot g'(x)$$



$$(\frac{f}{g})' = \frac{f' \cdot g - f \cdot g'}{g^2}$$

$$(f \cdot g)' = f' \cdot g + f \cdot g'$$

Gradient descent

We always want to move any parameter w towards a smaller loss L

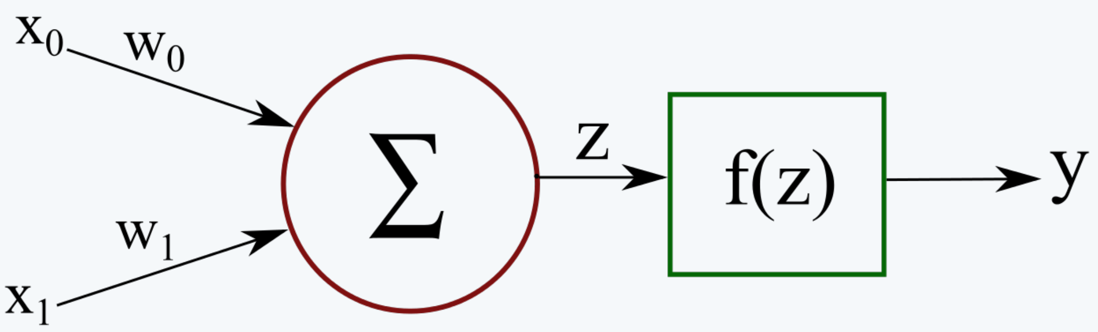
The parameters are updated following the rule:

$$w^{(t)} = w^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial w}$$

α - the learning rate, it controls how big is the update step that we take

$w^{(t)}$ - parameter w at time step t

Gradient descent



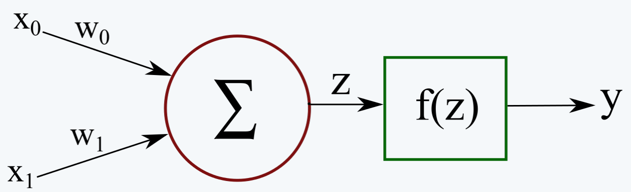
Let's calculate derivative of L with respect to w_0

$$\frac{\partial L}{\partial w_0} = ?$$

Can use the chain rule:

$$\frac{\partial L}{\partial w_0} = \boxed{\frac{\partial L}{\partial y}} \cdot \boxed{\frac{\partial y}{\partial z}} \cdot \boxed{\frac{\partial z}{\partial w_0}}$$

Gradient descent



$$L = \sum_k \frac{1}{2}(y_k - \hat{y_k})^2$$

$$\boxed{\frac{\partial L}{\partial y_k} = y_k - \hat{y_k}}$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$$

$$y = \sigma(z)$$

$$\boxed{\frac{\partial y}{\partial z} = y(1-y)}$$

$$z = w_0 \cdot x_0 + w_1 \cdot x_1$$

$$\boxed{\frac{\partial z}{\partial w_0} = x_0}$$

Gradient descent

We have calculated the partial derivative

$$\frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_0} = (y - \hat{y}) \cdot y \cdot (1 - y) \cdot x_0$$

Now we can update the weight parameter

$$w^{(t)} = w^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial w}$$

We do this for every parameter in the neural net, although this process can be simplified

Gradient

Gradient - a vector containing partial derivatives for all the model parameters.

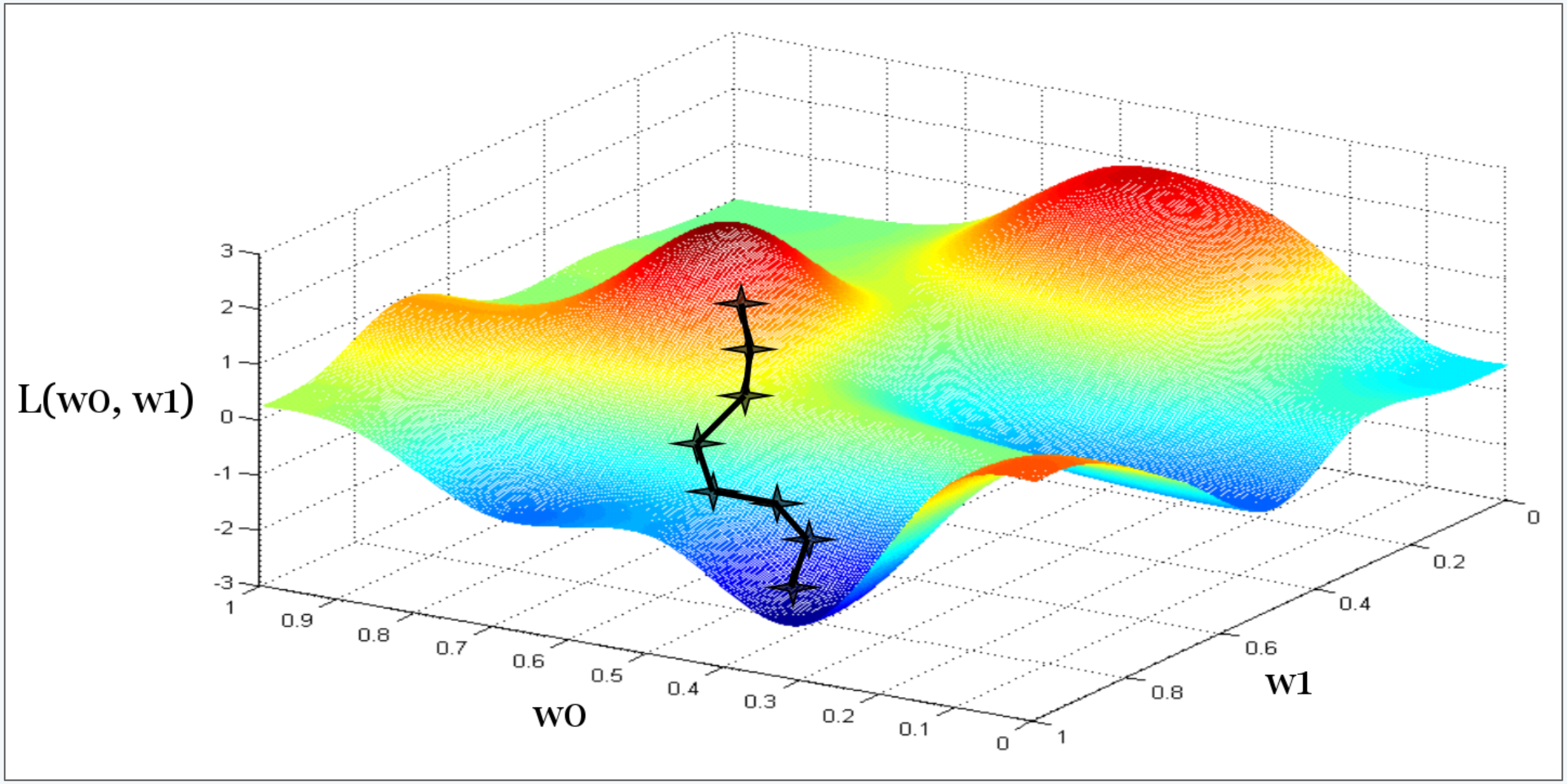
Can also call it the “error” vector

$$\nabla L = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \; \cdots \; , \frac{\partial L}{\partial w_n} \right)$$

In a high-dimensional parameter space, it shows us the way in which we want to move

Gradient descent

Move in the direction of the negative gradient

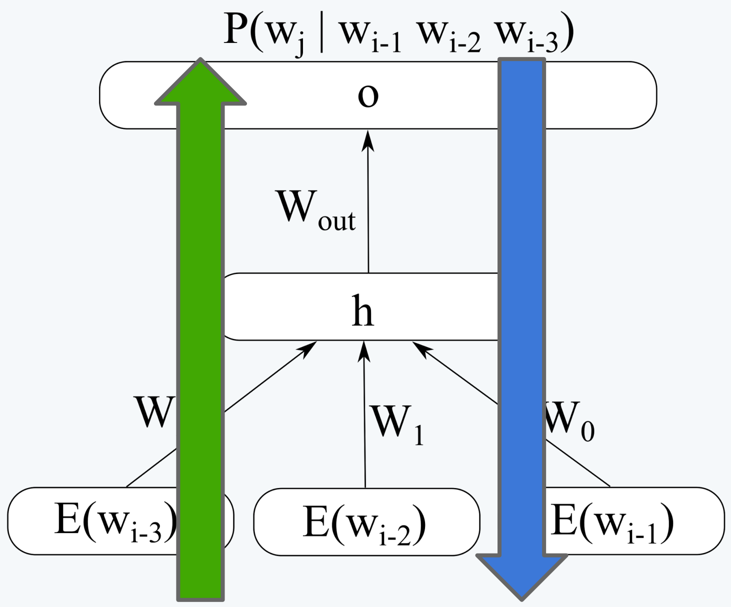


Language model loss function

$$L_k = -\sum_j \hat{y_j} \cdot log(p_j)$$

L_k	- loss for training example k
j	- possible words in the output layer
$\hat{y_j}$	- binary variable for the correct answer 1 if j is the correct word 0 otherwise
p_j	- out predicted probability for j

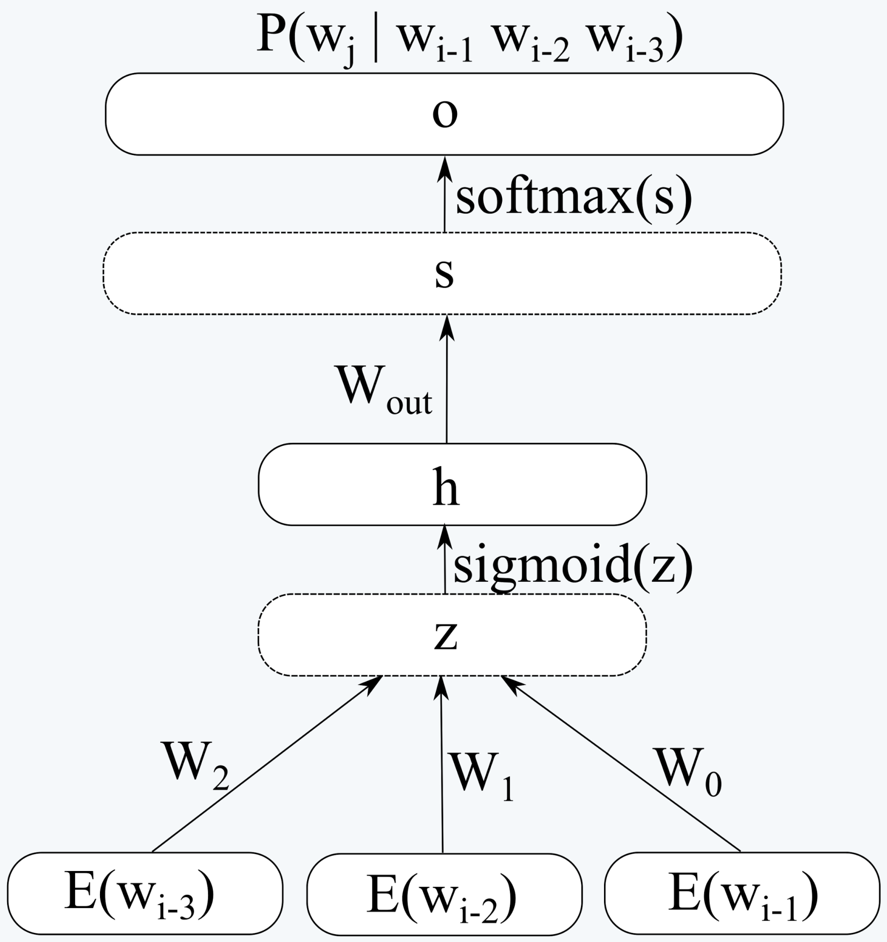
Backpropagation



Activation: we start with the input and move through the network to make a prediction

Backpropagation: we start from the output layer and move towards the input layer, calculating partial derivatives and weight updates

Backpropagation

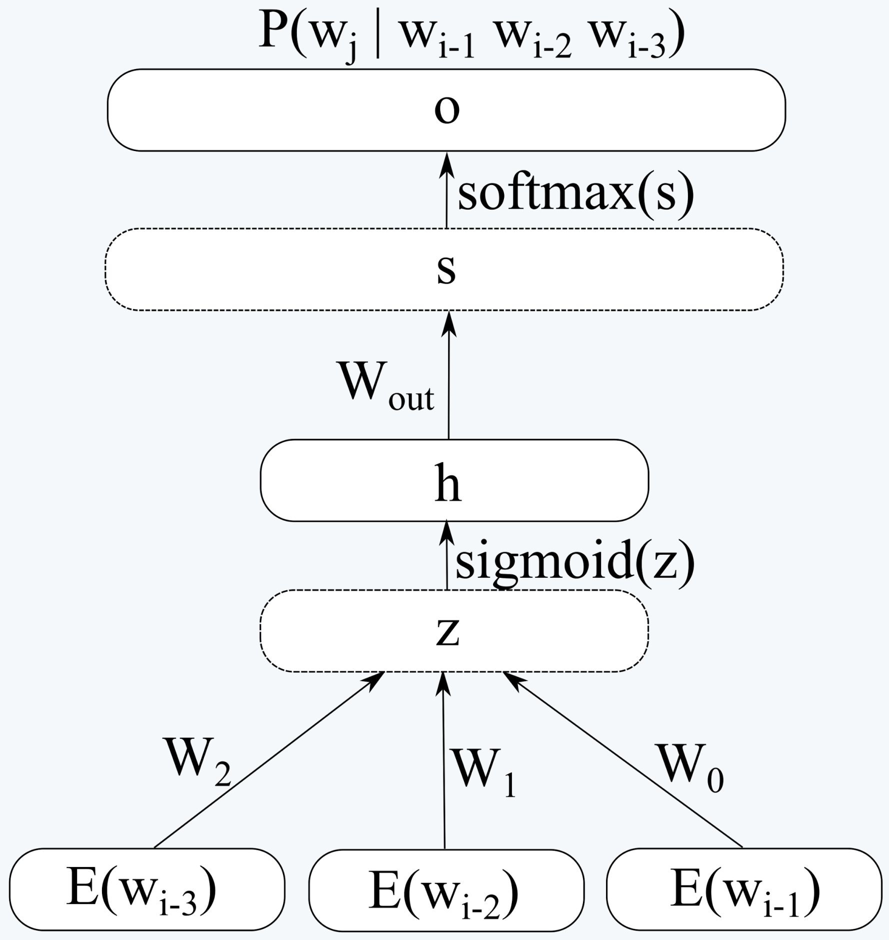


We'll go through backpropagation for a neural language model

The derivatives can be found the same way as we did for a simple neuron

The graph explicitly shows stages z and s for clarity

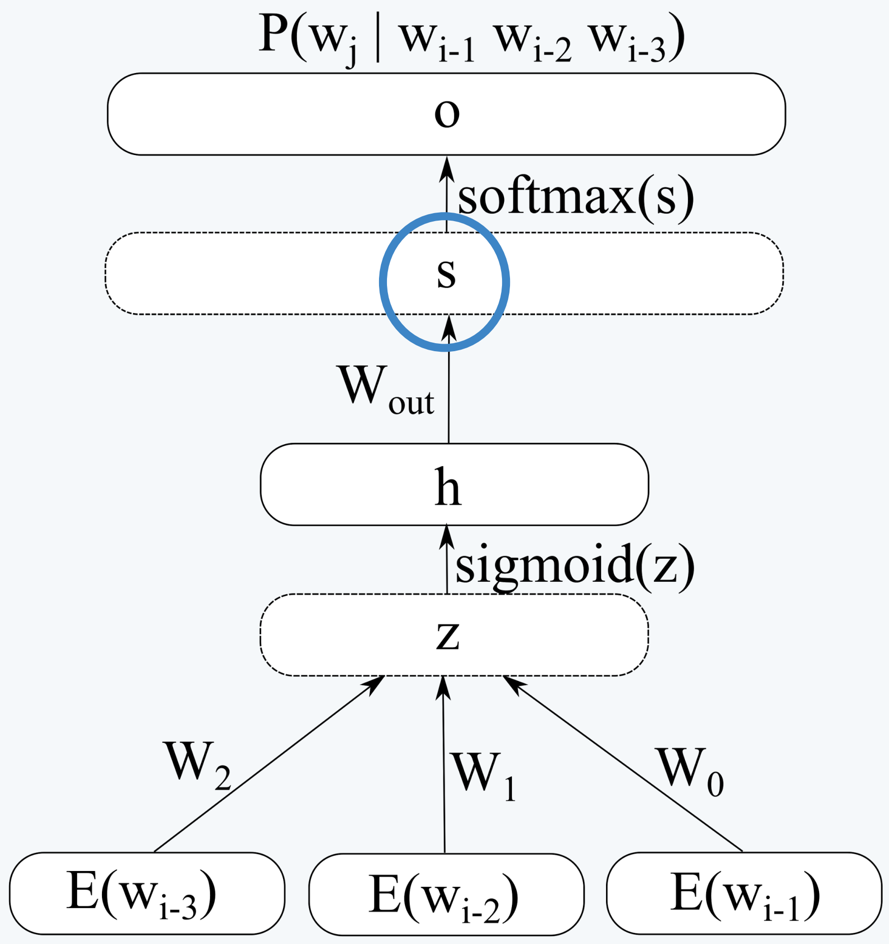
Backpropagation



1. Take a training example and perform a feedforward pass through the network.

Values in output vector o will be word probabilities.

Backpropagation



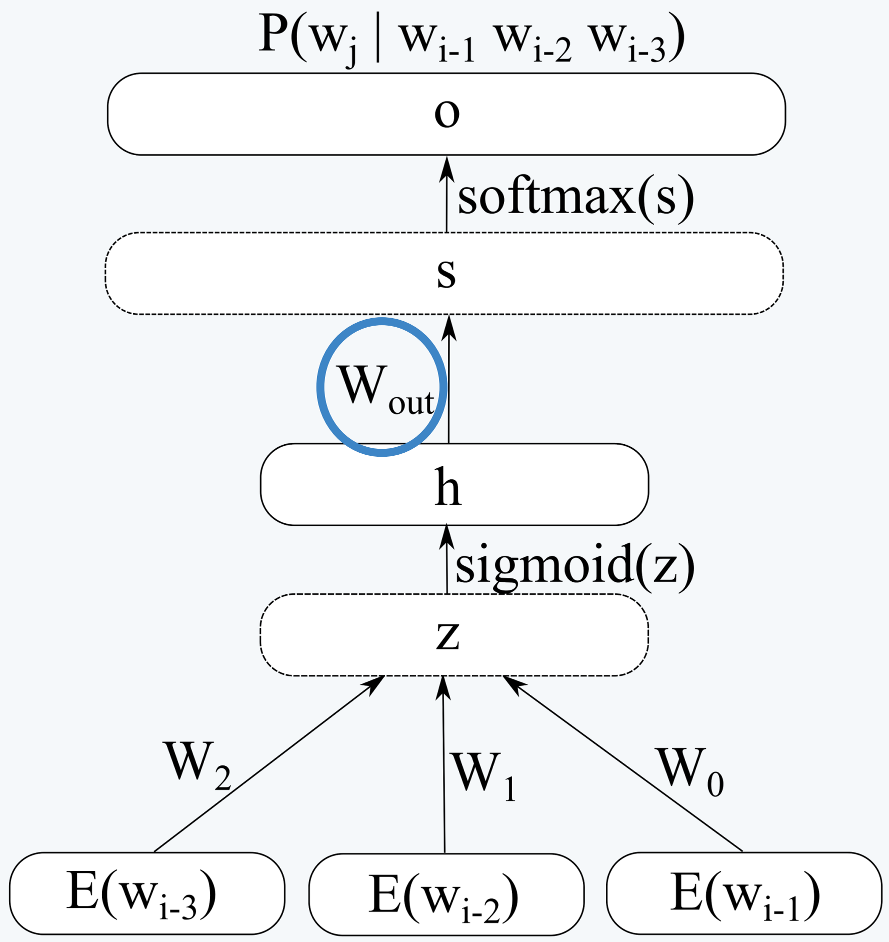
2. Calculate the error in the output layer

$$o = softmax(s)$$

$$\frac{\partial L}{\partial s} = o - \hat{y}$$

$\hat{y}$ - Vector of correct answers. Position j will be 1 if word j is the correct word. 0 otherwise.

Backpropagation

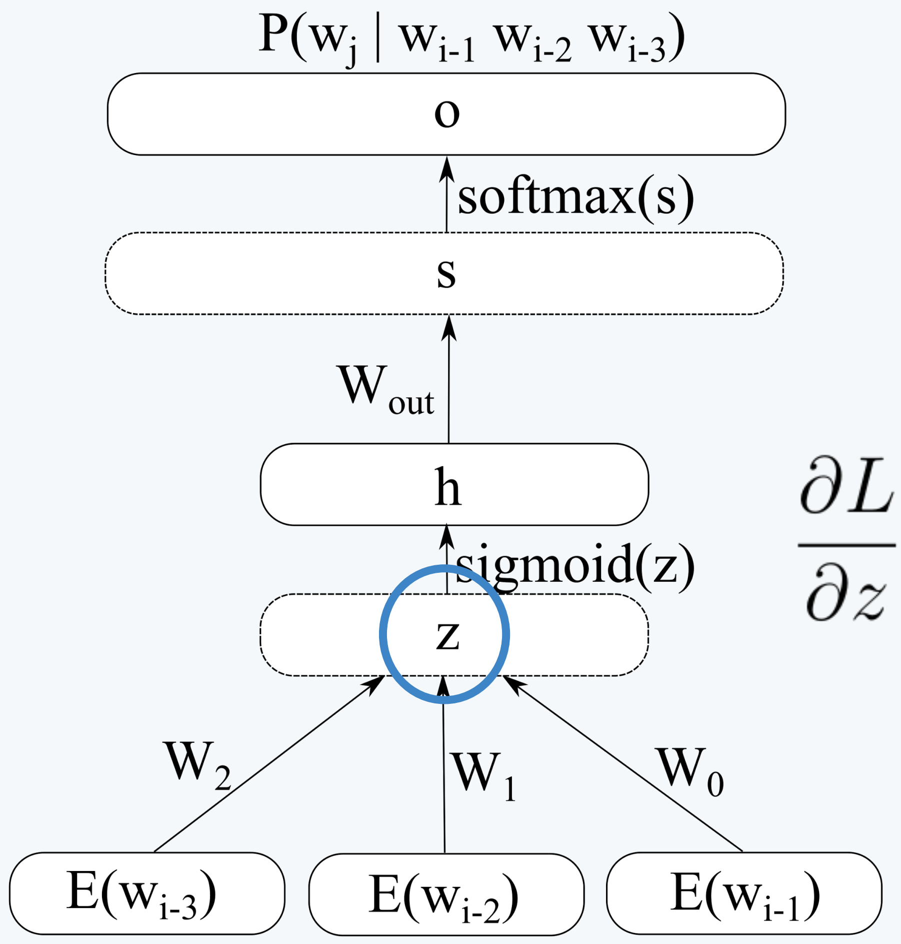


3. Calculate the error for the output weights

$$s = W_{out} \cdot h$$

$$\frac{\partial L}{\partial W_{out}} = \frac{\partial L}{\partial s} \cdot h^T$$

Backpropagation



4. Calculate error at hidden layer z

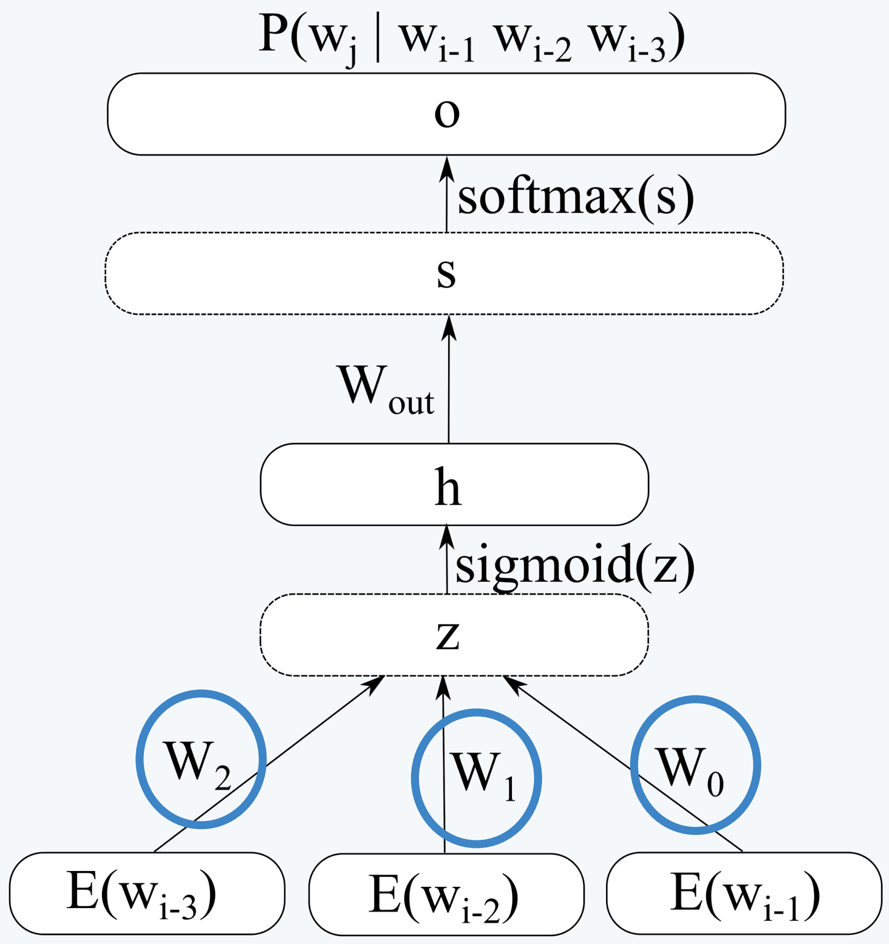
$$s = W_{out} \cdot h$$

$$h = sigmoid(z) = \sigma(z)$$

$$\frac{\partial L}{\partial z} = \underbrace{(W_{out}^T \cdot \frac{\partial L}{\partial s})}_{\text{partial derivative of h}} \odot \underbrace{h \odot (1-h)}_{\text{derivative of sigmoid}}$$

$\odot$ - element-wise multiplication

Backpropagation



5. Calculate error for hidden weights

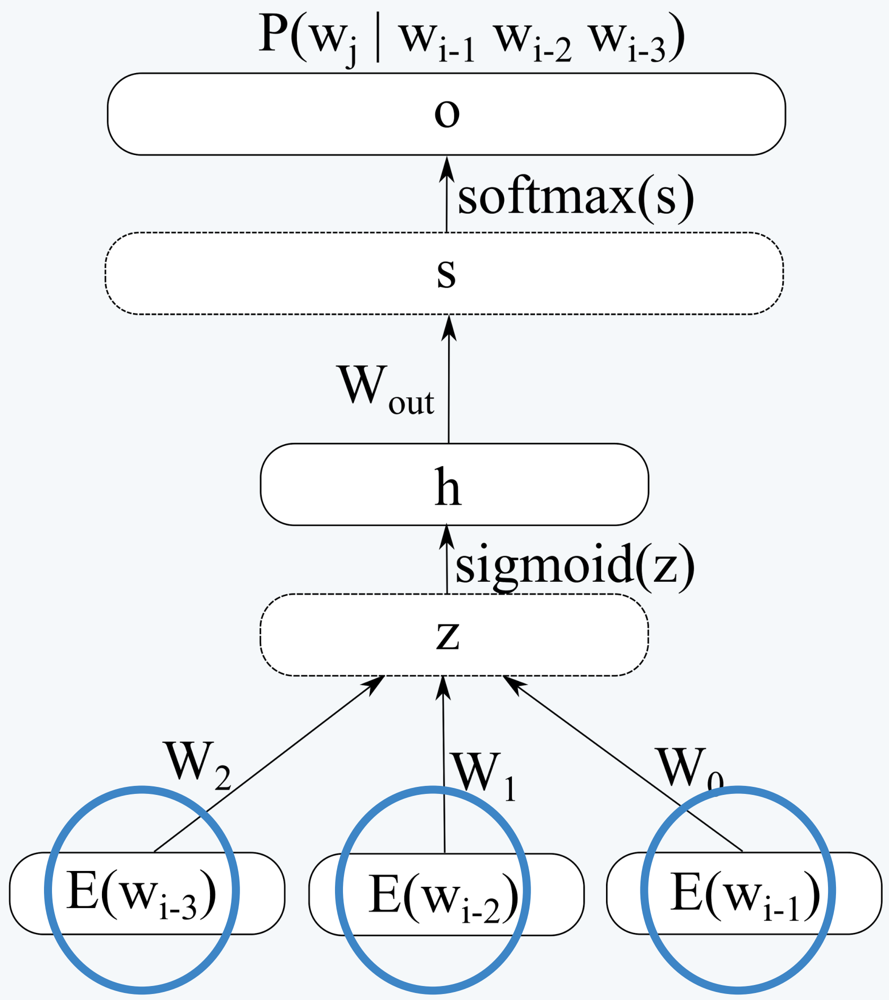
$$z = W_2 \cdot E(w_{i-3}) + W_1 \cdot E(w_{i-2}) + W_0 \cdot E(w_{i-1})$$

$$\frac{\partial L}{\partial W_0} = \frac{\partial L}{\partial z} \cdot E(w_{i-1})^T$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z} \cdot E(w_{i-2})^T$$

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z} \cdot E(w_{i-3})^T$$

Backpropagation



6. Calculate error for input vectors

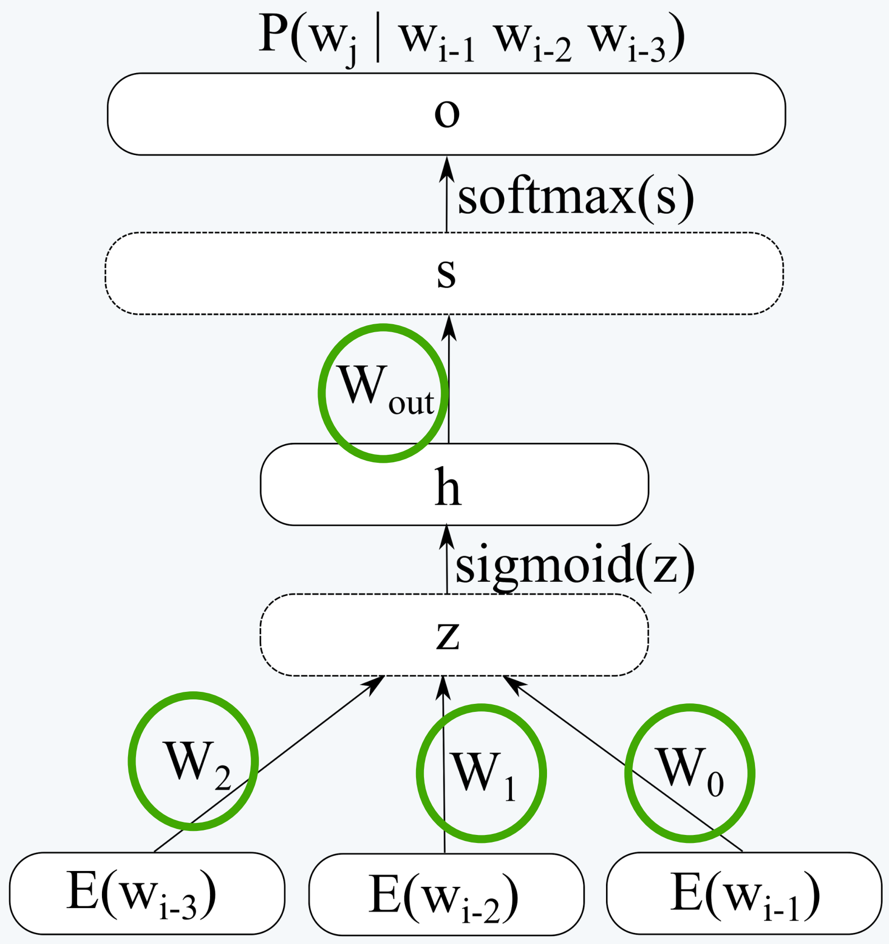
$$z = W_2 \cdot E(w_{i-3}) + W_1 \cdot E(w_{i-2}) + W_0 \cdot E(w_{i-1})$$

$$\frac{\partial L}{\partial E(w_{i-1})} = W_0^T \cdot \frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial E(w_{i-2})} = W_1^T \cdot \frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial E(w_{i-3})} = W_2^T \cdot \frac{\partial L}{\partial z}$$

Backpropagation



7. Update weight matrices

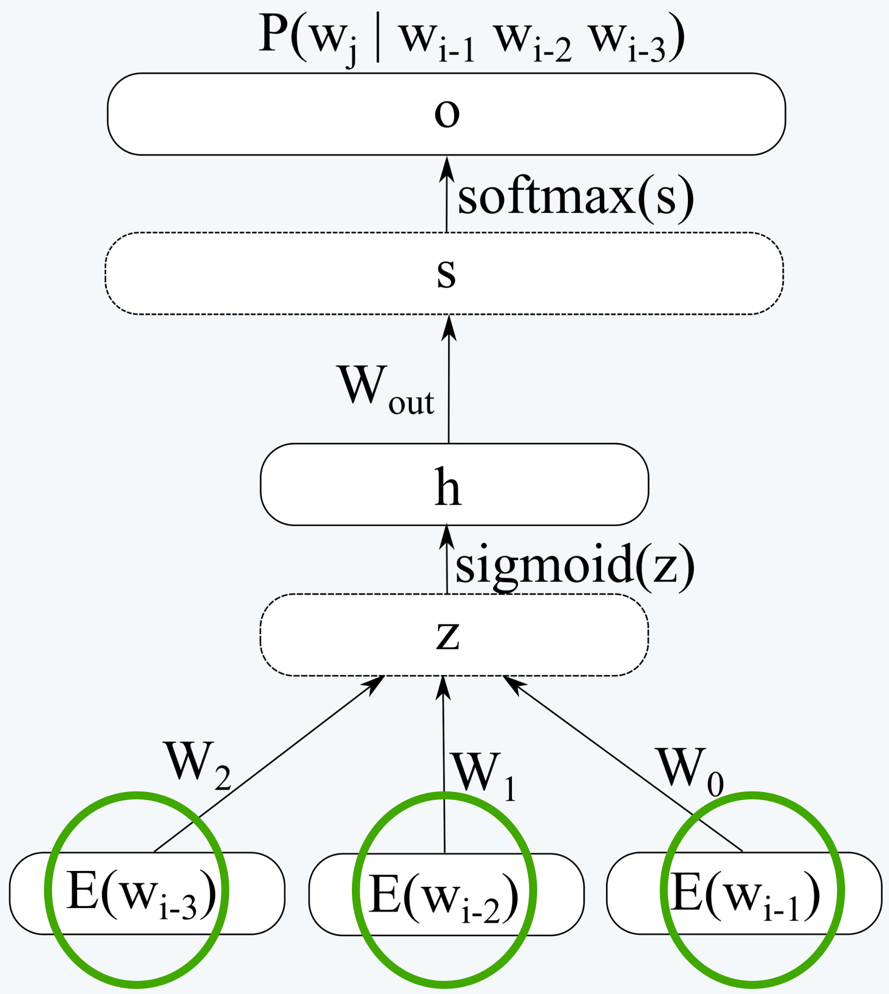
$$W_{out}^{(t)} = W_{out}^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial W_{out}}$$

$$W_0^{(t)} = W_0^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial W_0}$$

$$W_1^{(t)} = W_1^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial W_1}$$

$$W_2^{(t)} = W_2^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial W_2}$$

Backpropagation



8. Update the input vectors

$$E(w_{i-1})^{(t)} = E(w_{i-1})^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial E(w_{i-1})}$$

$$E(w_{i-2})^{(t)} = E(w_{i-2})^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial E(w_{i-2})}$$

$$E(w_{i-3})^{(t)} = E(w_{i-3})^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial E(w_{i-3})}$$

The general pattern

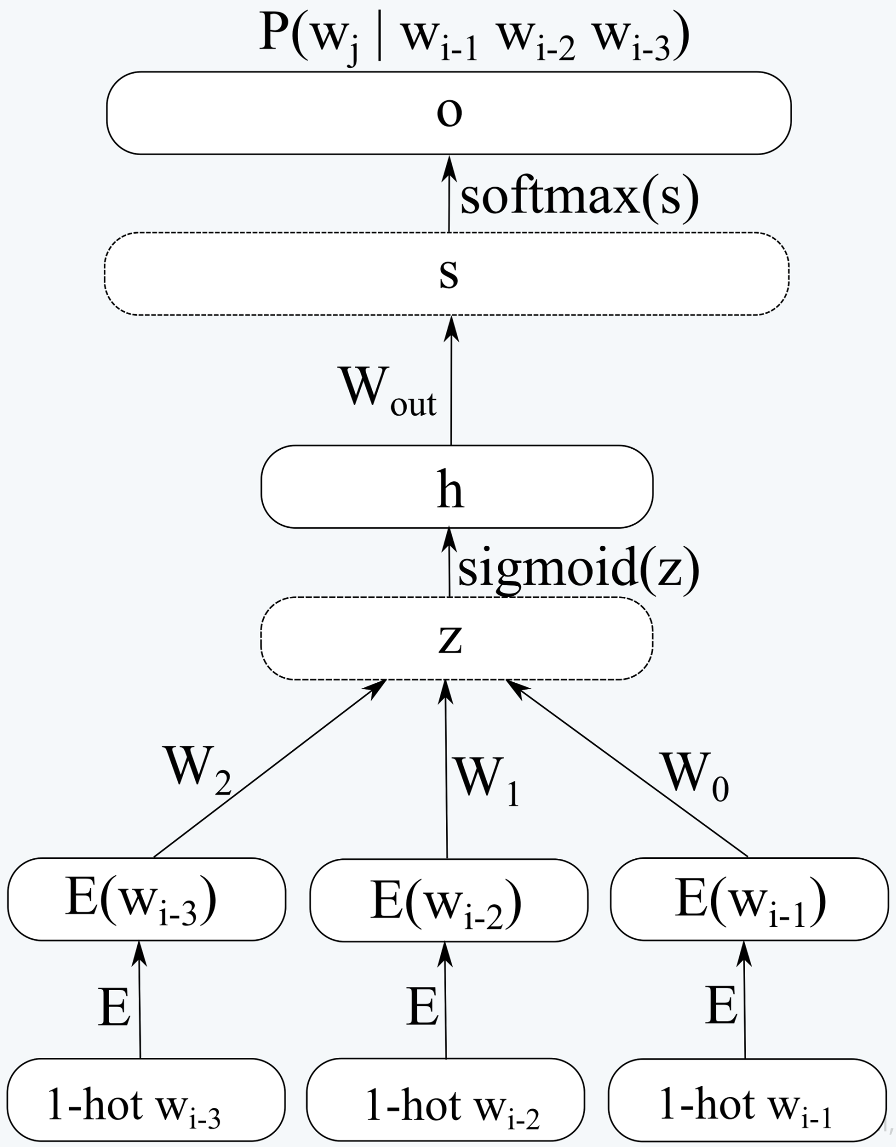
- Error at a hidden layer
  - = error at the next layer,
  - multiplied by the weights between the layers,
  - and element-wise multiplied with the derivative of the activation function

$$\frac{\partial L}{\partial z_k} = (W^T \cdot \frac{\partial L}{\partial z_k}) \odot f'(z_k)$$

- Error on the weights
  - = error at the next layer,
  - multiplied by the values from the previous layer

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_k} \cdot h_{k-1}^T$$

Representation learning



Can think of word representations as just another weight matrix

We update E using the same logic as W_{out}, W_0, W_1, and W_2

Backpropagation tips

- The error matrices have the same dimensionality as the parameter matrices! This can help when debugging.

  If W_{out} is a 10x20 matrix, then $\frac{\partial L}{\partial W_{out}}$ is also a 10x20 matrix

- Calculate the errors before updating the weights.
  Otherwise you'll backpropagate with wrong (updated) weights.

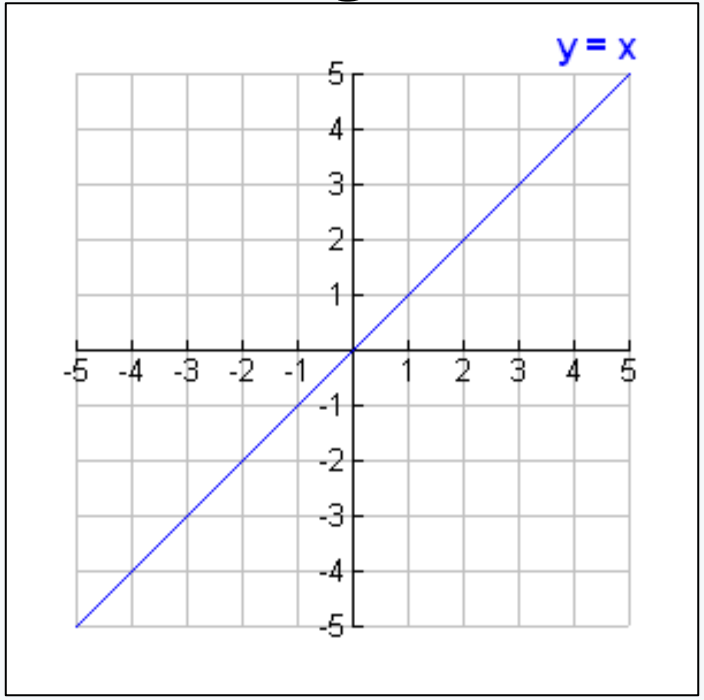
Activation functions

There are different possible activation functions

- It should be differentiable
- We can choose the best one using a development set

Linear

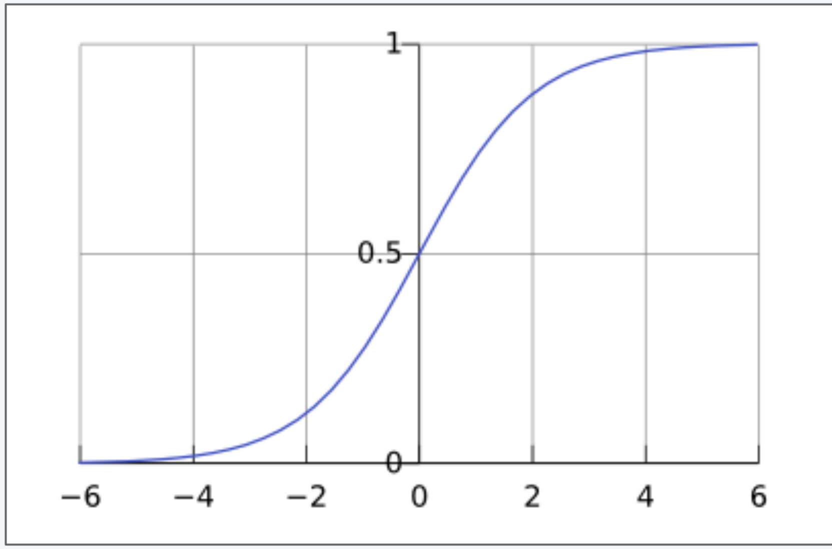
$$y = x$$



Activation functions

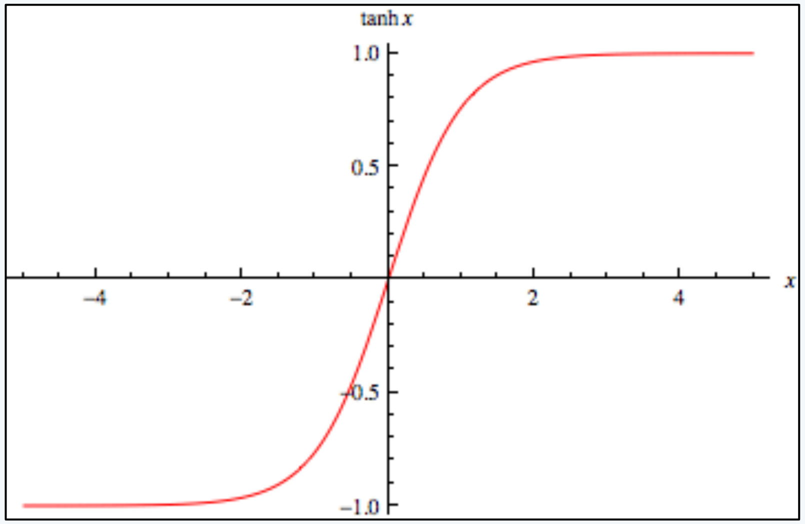
Logistic function (sigmoid)

$$y = \frac{1}{1 + e^{-z}}$$



Tanh

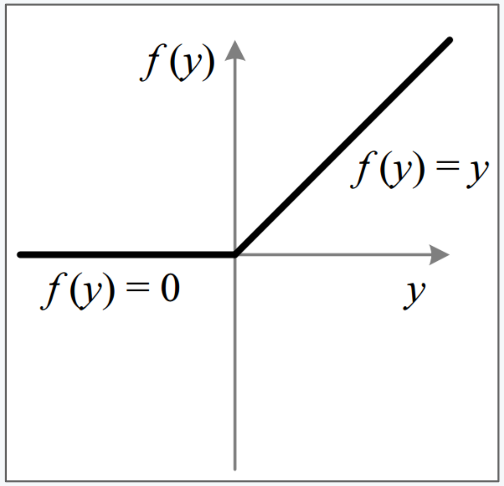
$$y = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Activation functions

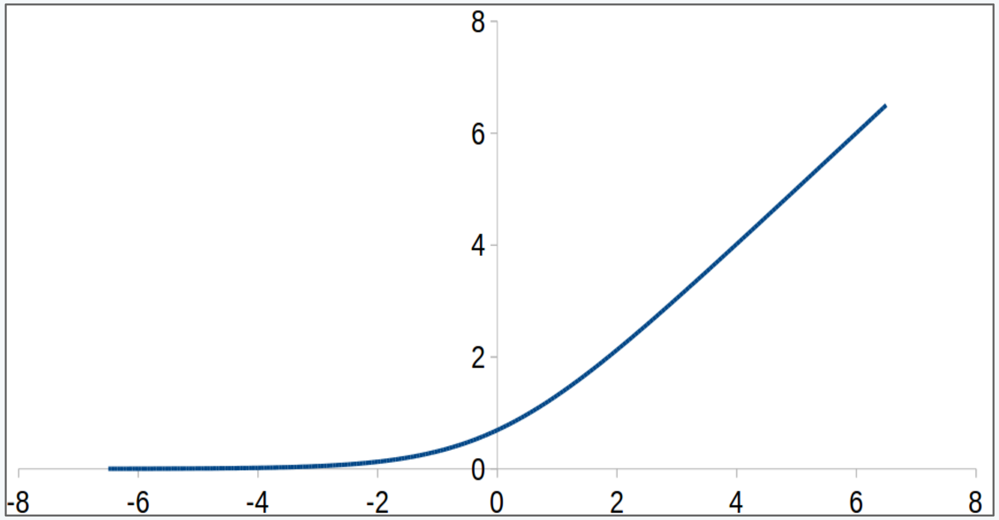
Rectifier (ReLU)

$$y = max(0, x)$$

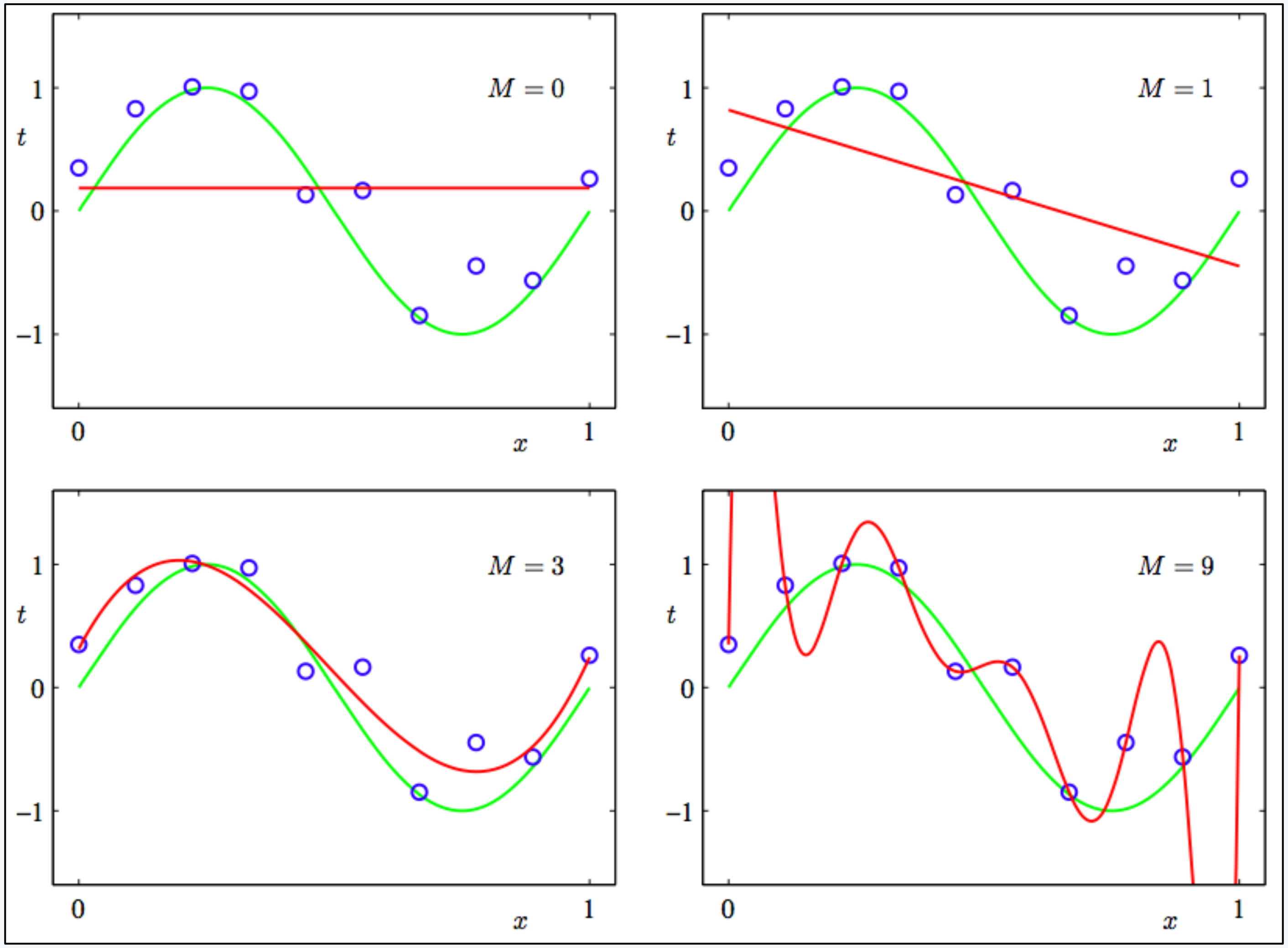


Softplus

$$y = ln(1 + e^x)$$



Overfitting



Regularisation

One way to prevent overfitting is to motivate the model to keep all weight values small

We can add this into our loss function

$$L_k = -\sum_j \hat{y_j} \cdot log(p_j) + \boxed{\frac{\lambda}{2} \cdot \sum_w w^2}$$

The higher the weight values, the larger the loss that we want to minimise

Hyperparameters

Initial weights

- Set randomly to some small values around 0
- Strategy 1: $\beta \cdot \mathcal{N}(0,1)$
- Strategy 2: $Uniform(-r, r)$

$$\text{for tanh: } r = \sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}$$

$$\text{for logistic: } r = 4 \cdot \sqrt{\frac{6}{\text{fan-in} + \text{fan-out}}}$$

Learning rate

- Choose an initial value. Often 0.1 or 0.01
- Decrease the learning rate during training
- Stop training when performance on the development data has not improved

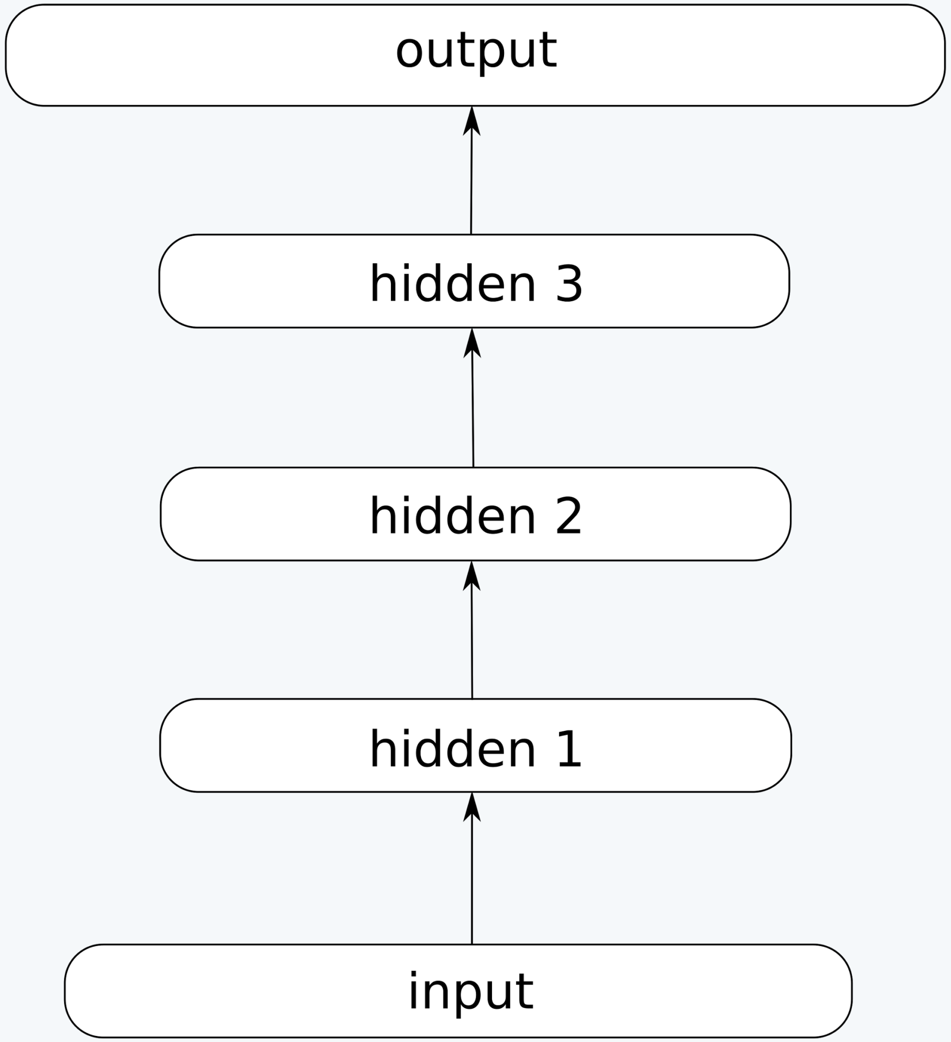
Deep learning

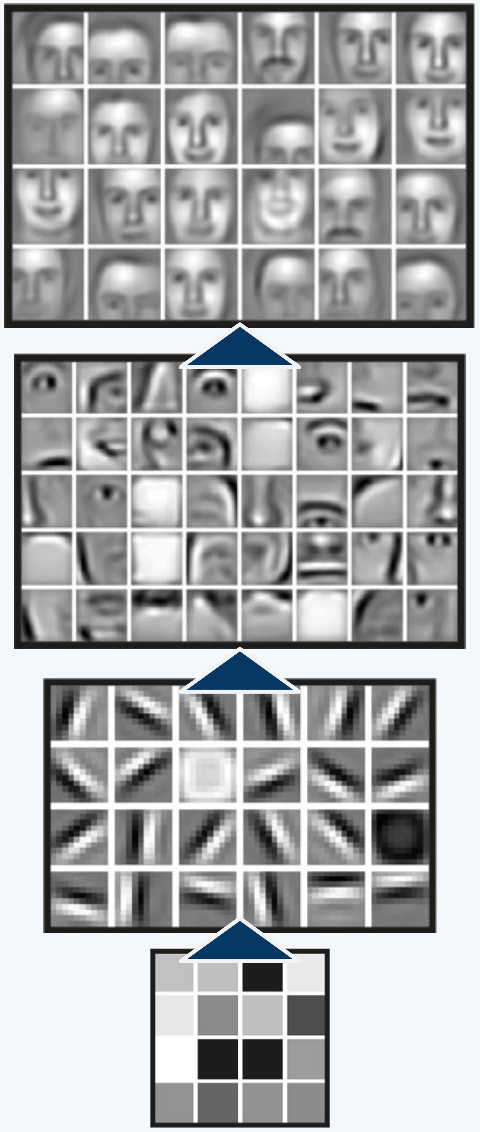
Deep learning - learning with many non-linear layers

Each consecutive layer will learn more complicated patterns of the output from the previous layer

We could add an extra hidden layer into our language model, and it would be a deep network

Deep learning





Neural nets vs traditional models

Negative:

- More unstable, settings need to be just right
- Less explainable (black box)
- Computationally demanding

Positive:

- Less feature engineering (representation learning)
- More expressive and more flexible
- Have been shown to outperform traditional models on many tasks

Gradient checking

A way to check that your gradient descent is implemented correctly. Two ways to calculate the gradient.

Method 1: from your weight updates

$$w^{(t)} = w^{(t-1)} - \alpha \cdot \frac{\partial L}{\partial w} \qquad \frac{\partial L}{\partial w} = \frac{w^{(t-1)} - w^{(t)}}{\alpha}$$

Method 2: actually changing the weight a bit and measuring the change in loss

$$\frac{\partial}{\partial w} L(w) = \lim_{\epsilon \to 0} \frac{L(w+\epsilon) - L(w-\epsilon)}{2\epsilon} \qquad \frac{\partial}{\partial w} L(w) \approx \frac{L(w+\epsilon) - L(w-\epsilon)}{2\epsilon}$$

References

Pattern Recognition and Machine Learning
Christopher Bishop (2007)

Machine Learning: A Probabilistic Perspective

Kevin Murphy (2012)

Machine Learning

Andrew Ng (2012)

https://www.coursera.org/course/ml

Using Neural Networks for Modelling and Representing Natural Languages

Tomas Mikolov (2014)

http://www.coling-2014.org/COLING%202014%20Tutorial-fix%20-%20Tomas%20Mikolov.pdf

Deep Learning for Natural Language Processing (without Magic)

Richard Socher, Christopher Manning (2013)

http://nlp.stanford.edu/courses/NAACL2013/

Extra materials

One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Philipp Koehn, Tony Robinson

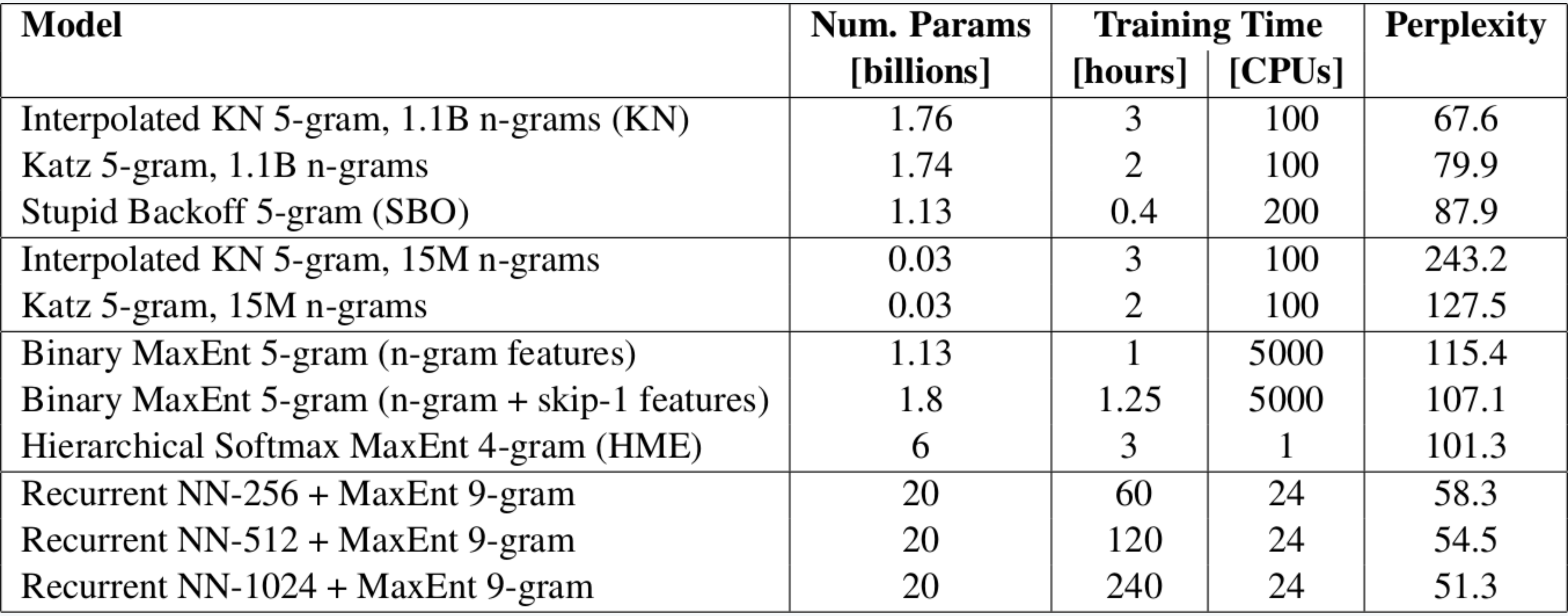
Google, University of Edinburgh, Cantab Research Ltd

2014

Dataset

- A new benchmark dataset for LM evaluation
- Cleaned and tokenised
- 0.8 billion words for training
- 160K words used for testing
- Discarding words with $\text{count} < 3$
- Vocabulary $V = 793471$
- Using <S>, </S> and <UNK> tokens

Results



Model	Num. Params [billions]	Training Time [hours]	Training Time [CPUs]	Perplexity
Interpolated KN 5-gram, 1.1B n-grams (KN)	1.76	3	100	67.6
Katz 5-gram, 1.1B n-grams	1.74	2	100	79.9
Stupid Backoff 5-gram (SBO)	1.13	0.4	200	87.9
Interpolated KN 5-gram, 15M n-grams	0.03	3	100	243.2
Katz 5-gram, 15M n-grams	0.03	2	100	127.5
Binary MaxEnt 5-gram (n-gram features)	1.13	1	5000	115.4
Binary MaxEnt 5-gram (n-gram + skip-1 features)	1.8	1.25	5000	107.1
Hierarchical Softmax MaxEnt 4-gram (HME)	6	3	1	101.3
Recurrent NN-256 + MaxEnt 9-gram	20	60	24	58.3
Recurrent NN-512 + MaxEnt 9-gram	20	120	24	54.5
Recurrent NN-1024 + MaxEnt 9-gram	20	240	24	51.3