Deep Learning

QCon London, March 9th

Machine Learning (Recap)

Machine Learning (Recap)

- What is machine learning?

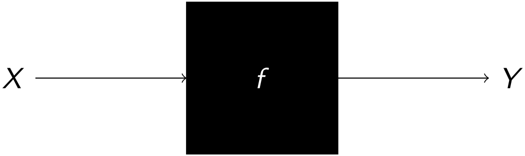
Machine Learning (Recap)

- What is machine learning?

$$X \longrightarrow \qquad\qquad \longrightarrow Y$$

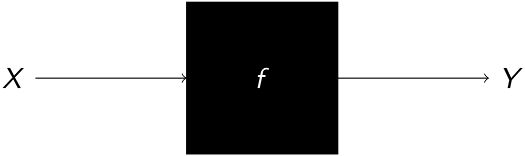
Machine Learning (Recap)

- What is machine learning?



Machine Learning (Recap)

- What is machine learning?



- By knowing a set of data and their targets we can tune f to output what we want.

Machine Learning (Recap)

Machine Learning (Recap)

- We don’t have to define endless rules

Machine Learning (Recap)

- We don’t have to define endless rules
- It can write computational rules nobody of us would ever think of

Machine Learning (Recap)

- We don't have to define endless rules
- It can write computational rules nobody of us would ever think of
- It can often write better computational rules than we could

Machine Learning (Recap)

- We don't have to define endless rules
- It can write computational rules nobody of us would ever think of
- It can often write better computational rules than we could
- It can check more cases than we could

Today's Overview

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent

Today's Overview

- 9:00 Introduction to Deep Learning
    - Short recap on machine learning
    - Build and train a perceptron in numpy
    - Change the code to use PyTorch
    - Gradient Descent
- 10:30 Coffee Break

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch
- 13:00 RNN (continued)

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch
- 13:00 RNN (continued)
- 13:30 Word2Vec

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch
- 13:00 RNN (continued)
- 13:30 Word2Vec
- 14:30 Coffee Break

Today's Overview

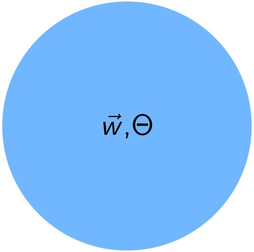
- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch
- 13:00 RNN (continued)
- 13:30 Word2Vec
- 14:30 Coffee Break
- 14:45 Chatbot

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Change the code to use PyTorch
  - Gradient Descent
- 10:30 Coffee Break
- 10:45 Neural Networks
  - Implement a multi-layer perceptron
  - Recurrent neural networks
- 12:00 Lunch
- 13:00 RNN (continued)
- 13:30 Word2Vec
- 14:30 Coffee Break
- 14:45 Chatbot
- 16:00 End of workshop

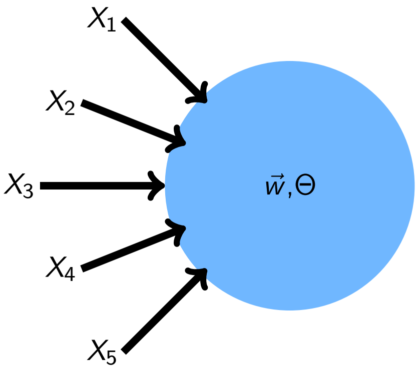
Perceptron

- The perceptron contains a weight ($\vec{w}$) for each input and a threshold (Θ)



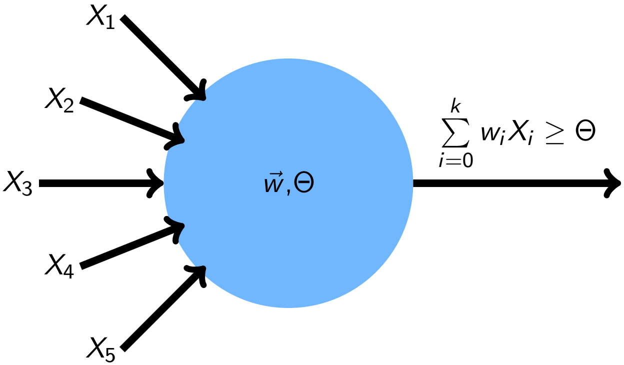
Perceptron

- The perceptron contains a weight ($\vec{w}$) for each input and a threshold (Θ)



Perceptron

- The perceptron contains a weight ($\vec{w}$) for each input and a threshold (Θ)
- Its output can be calculated as $\sum_{i=0}^{k} w_i X_i \geq \Theta$



Perceptron Training

Perceptron Training

- Start with random weights

Perceptron Training

- Start with random weights
- Calculate the perceptron's estimate of the solution with: $\hat{y} = \left(\sum_i w_i X_i \geq 0 \right)$

Perceptron Training

- Start with random weights
- Calculate the perceptron's estimate of the solution with: $\hat{y} = \left(\sum_i w_i X_i \geq 0 \right)$
- Determine the update for the weights using the error and the learning rate (α):
  $\partial w_i = \alpha(y - \hat{y})X_i$

Perceptron Training

- Start with random weights
- Calculate the perceptron's estimate of the solution with: $\hat{y} = \left(\sum_i w_i X_i \geq 0 \right)$
- Determine the update for the weights using the error and the learning rate (α): $\partial w_i = \alpha(y - \hat{y})X_i$
- Calculate new weights as: $w_i \leftarrow w_i + \partial w_i$

Perceptron Training

- Start with random weights
- Calculate the perceptron's estimate of the solution with: $\hat{y} = \left(\sum_i w_i X_i \geq 0 \right)$
- Determine the update for the weights using the error and the learning rate (α): $\partial w_i = \alpha(y - \hat{y})X_i$
- Calculate new weights as: $w_i \leftarrow w_i + \partial w_i$
- Restart the process until the solution is found

Perceptron Training

- Start with random weights
- Calculate the perceptron's estimate of the solution with: $\hat{y} = \left(\sum_i w_i X_i \geq 0\right)$
- Determine the update for the weights using the error and the learning rate (α): $\partial w_i = \alpha(y - \hat{y})X_i$
- Calculate new weights as: $w_i \leftarrow w_i + \partial w_i$
- Restart the process until the solution is found
- Let us actually implement this in the first notebook (Perceptron)

Conclusions

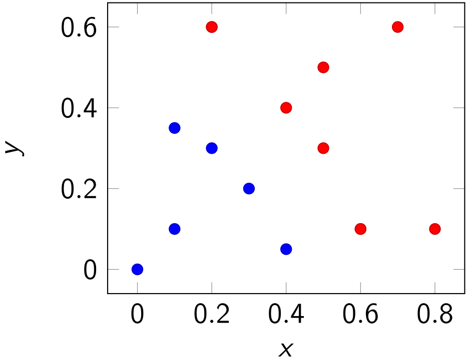
Conclusions

- Congratulations on training your first perceptron! You just taught a computer to learn!

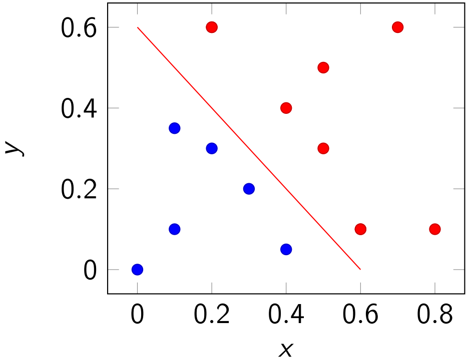
Conclusions

- Congratulations on training your first perceptron! You just taught a computer to learn!
- Why didn't we wait for the final weights, but stop after one run over the dataset?

Linear Separability

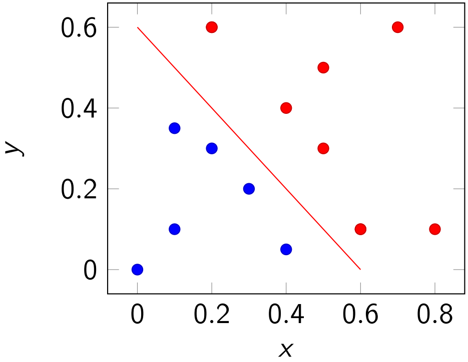


Linear Separability

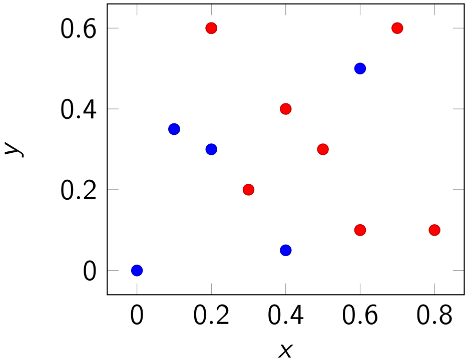


Data is linearly separable -> Can be divided by a plane

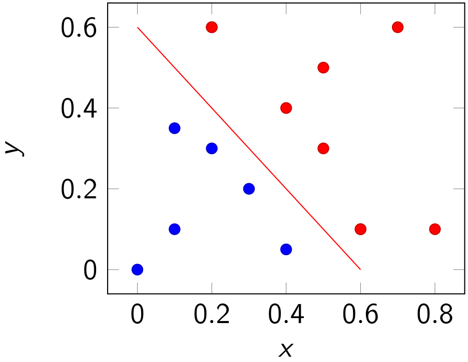
Linear Separability



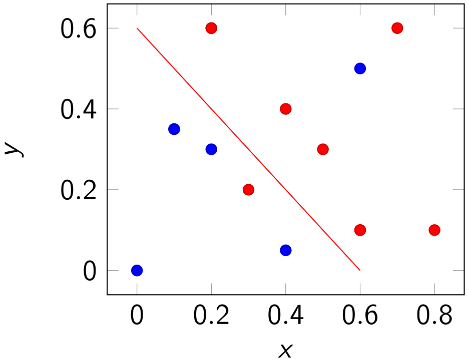
Data is linearly separable -> Can be divided by a plane



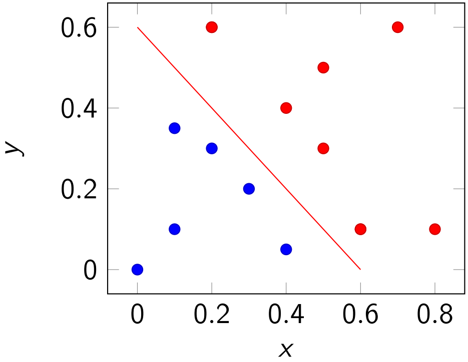
Linear Separability



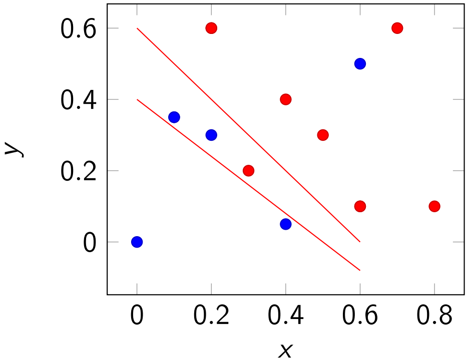
Data is linearly separable -> Can be divided by a plane



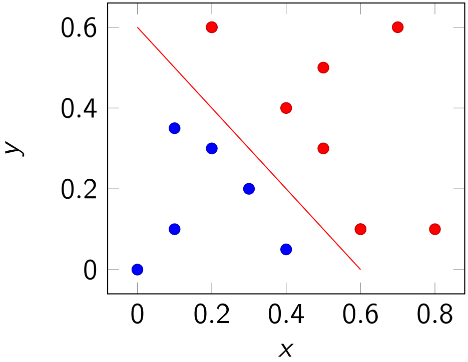
Linear Separability



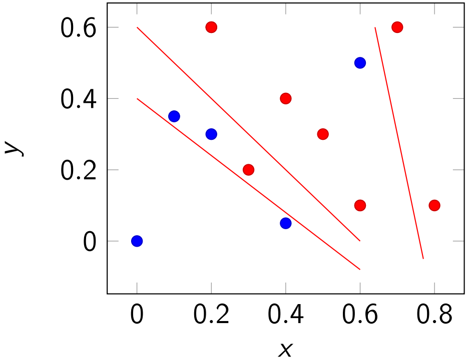
Data is linearly separable -> Can be divided by a plane



Linear Separability

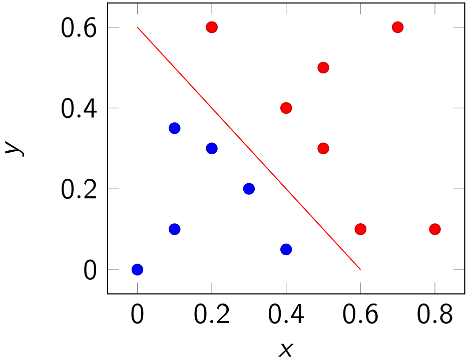


Data is linearly separable -> Can be divided by a plane

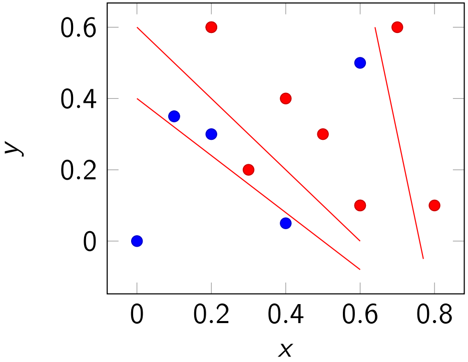


Data is not linearly separable -> Can not be divided by a plane

Linear Separability



Data is linearly separable -> Can be divided by a plane



Data is not linearly separable -> Can not be divided by a plane

The algorithm only converges, if the problem is linearly separable.

Gradient Descent

Gradient Descent

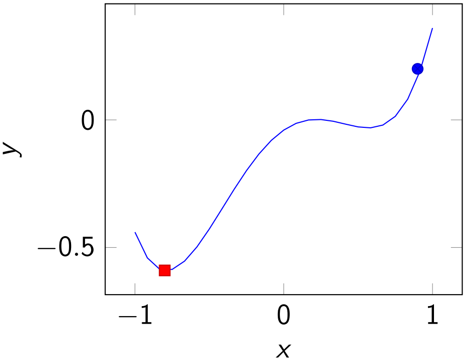
- What we calculated is an activation
  $$a = \sum_i w_i X_i$$

Gradient Descent

- What we calculated is an activation
  $a = \sum_i w_i X_i$
- Let us ignore the threshold and try to get the activation as close to the target value as possible.

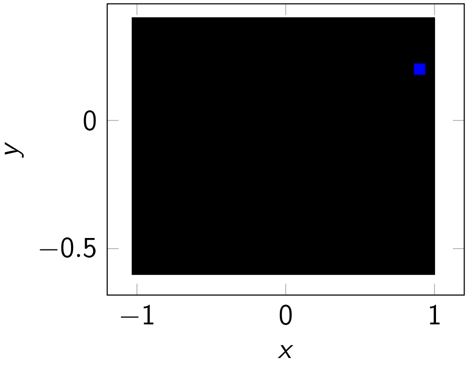
Gradient Descent

- What we calculated is an activation
  $a = \sum_i w_i X_i$
- Let us ignore the threshold and try to get the activation as close to the target value as possible.
- This brings us back to regression with an error: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$



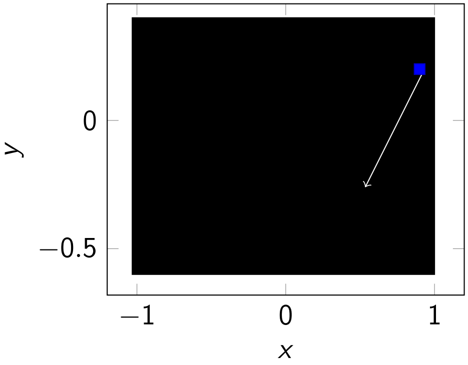
Gradient Descent

- What we calculated is an activation
  $a = \sum_i w_i X_i$
- Let us ignore the threshold and try to get the activation as close to the target value as possible.
- This brings us back to regression with an error: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$
- However, we can only calculate $E(w)$ with the current weights.



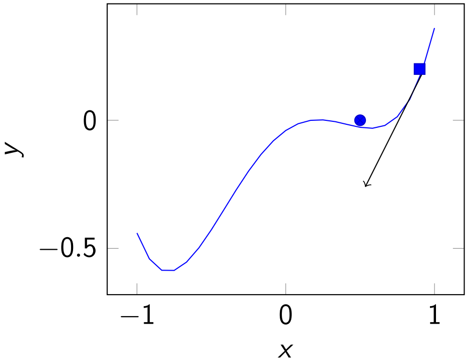
Gradient Descent

- What we calculated is an activation $a = \sum_i w_i X_i$
- Let us ignore the threshold and try to get the activation as close to the target value as possible.
- This brings us back to regression with an error: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$
- However, we can only calculate $E(w)$ with the current weights.
- But we can calculate the gradients for w and then use them to decrease the loss.

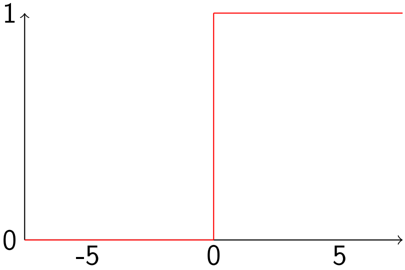


Gradient Descent

- What we calculated is an activation $a = \sum_i w_i X_i$
- Let us ignore the threshold and try to get the activation as close to the target value as possible.
- This brings us back to regression with an error: $E(w) = \frac{1}{2} \sum_{(x,y)\in D} (y - a)^2$
- However, we can only calculate $E(w)$ with the current weights.
- But we can calculate the gradients for w and then use them to decrease the loss.
- Every deep learning framework can calculate those gradients using the chain rule and backpropagation.

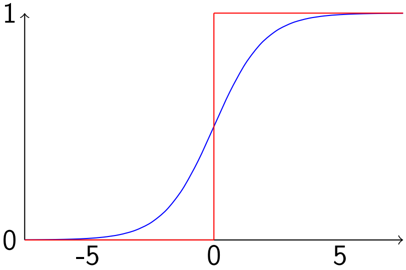


Sigmoid $\rightarrow$ Differentiable Threshold



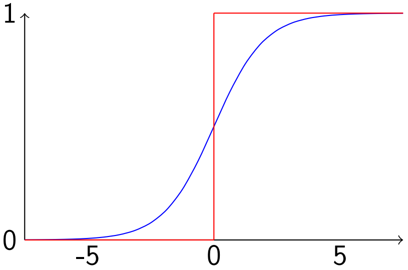
Sigmoid $\rightarrow$ Differentiable Threshold

- Even though the perceptron is great for boolean functions it is not differentiable and has no continuous error function, consequently the gradient cannot be calculated



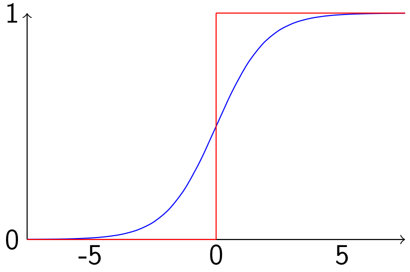
Sigmoid $\rightarrow$ Differentiable Threshold

- Even though the perceptron is great for boolean functions it is not differentiable and has no continuous error function, consequently the gradient cannot be calculated
- $\sigma(a) = \frac{1}{1+e^{-a}}$



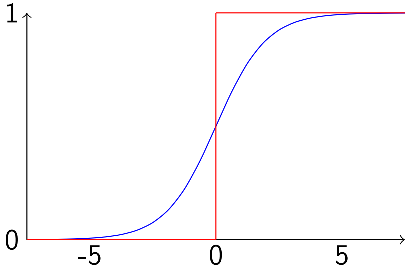
Sigmoid → Differentiable Threshold

- Even though the perceptron is great for boolean functions it is not differentiable and has no continuous error function, consequently the gradient cannot be calculated
- $\sigma(a) = \frac{1}{1+e^{-a}}$
- The sigmoid allows us to convert the strict classification into a regression over the probability for each point to belong to a certain class



Sigmoid → Differentiable Threshold

- Even though the perceptron is great for boolean functions it is not differentiable and has no continuous error function, consequently the gradient cannot be calculated
- $\sigma(a) = \frac{1}{1+e^{-a}}$
- The sigmoid allows us to convert the strict classification into a regression over the probability for each point to belong to a certain class
- Let us use this new knowledge to implement gradient descent in the next notebook.



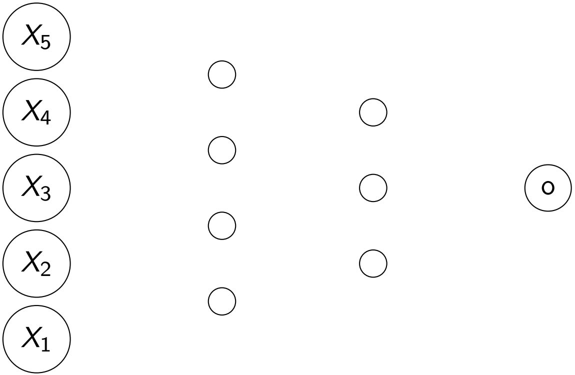
Neural Network

Neural Network

- Similar to the perceptron we have a lot of inputs and an output

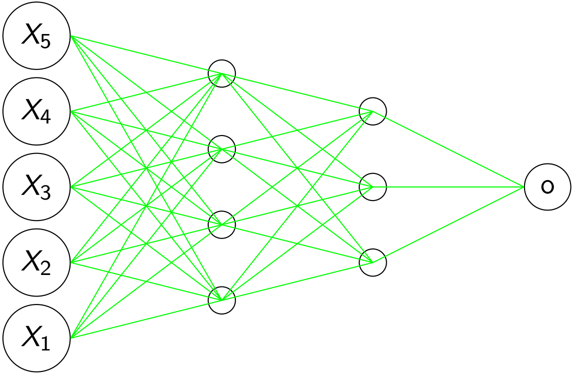
Neural Network

- Similar to the perceptron we have a lot of inputs and an output
- However, instead of going from start to end directly we now add hidden units
- Hereby, every output from the previous layers is added as all inputs



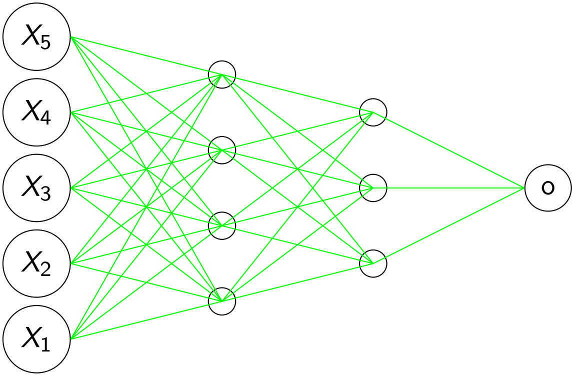
Neural Network

- Similar to the perceptron we have a lot of inputs and an output
- However, instead of going from start to end directly we now add hidden units
- Hereby, every output from the previous layers is added as all inputs
- For each of the units we now apply the sigmoid function



Neural Network

- Similar to the perceptron we have a lot of inputs and an output
- However, instead of going from start to end directly we now add hidden units
- Hereby, every output from the previous layers is added as all inputs
- For each of the units we now apply the sigmoid function
- Since every unit is differentiable, the whole network is differentiable $\rightarrow$ We can use backpropagation to minimize the error as we can calculate the gradients



Restriction Bias

Restriction Bias

- Restriction bias tells us what our model is able to represent

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers
- How do we make sure that we do not learn the arbitrary noise in the dataset, basically remembering it, but instead learn meaningful general rules?

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers
- How do we make sure that we do not learn the arbitrary noise in the dataset, basically remembering it, but instead learn meaningful general rules?
  - Limit the number of neurons

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers
- How do we make sure that we do not learn the arbitrary noise in the dataset, basically remembering it, but instead learn meaningful general rules?
  - Limit the number of neurons
  - Dropout

Restriction Bias

- Restriction bias tells us what our model is able to represent
- Perceptron $\rightarrow$ can only split the data using a hyperplane
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers
- How do we make sure that we do not learn the arbitrary noise in the dataset, basically remembering it, but instead learn meaningful general rules?
  - Limit the number of neurons
  - Dropout
  - Weight Decay

Optimizing Weights

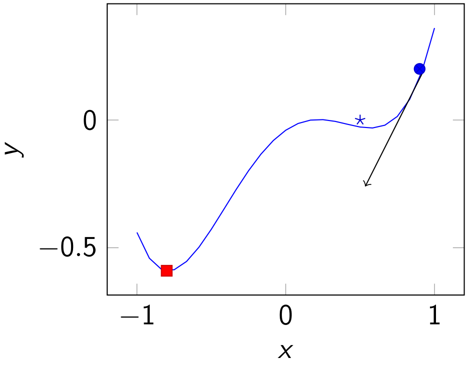
- There are multiple ways to use gradient descent.

Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent

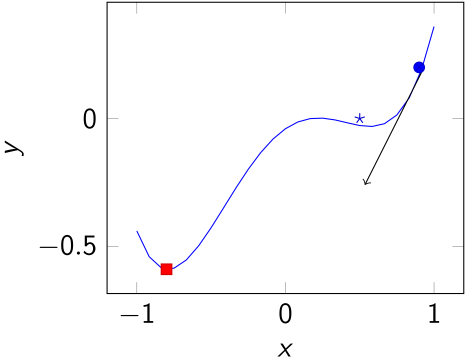
Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent
- Advanced Methods



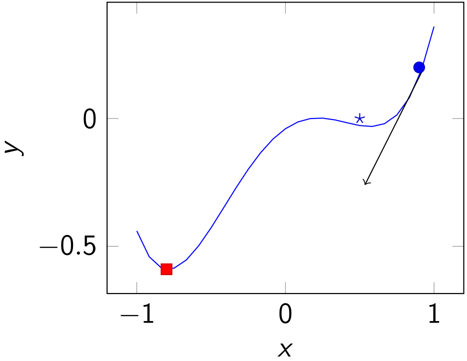
Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum



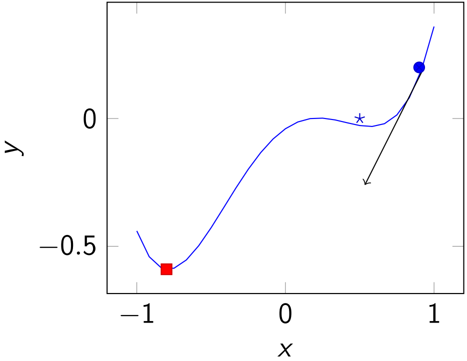
Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives



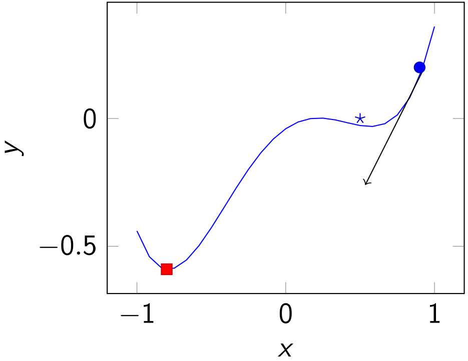
Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives
  - Randomized Optimization

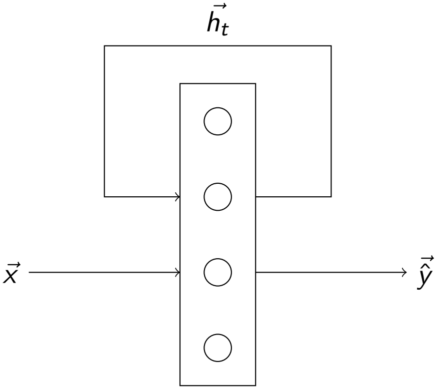


Optimizing Weights

- There are multiple ways to use gradient descent.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives
  - Randomized Optimization
  - Penalty for Complexity

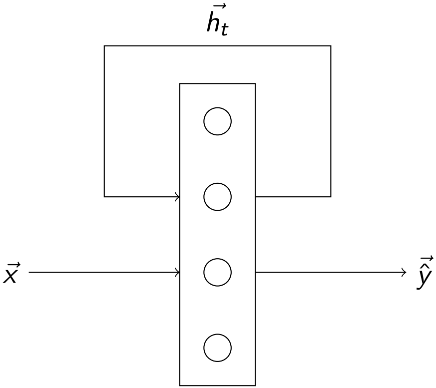


Recurrent Neural Networks



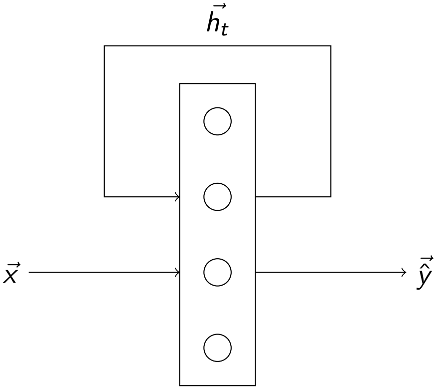
Recurrent Neural Networks

- Recurrent Neural Networks allow one to propagate states through time



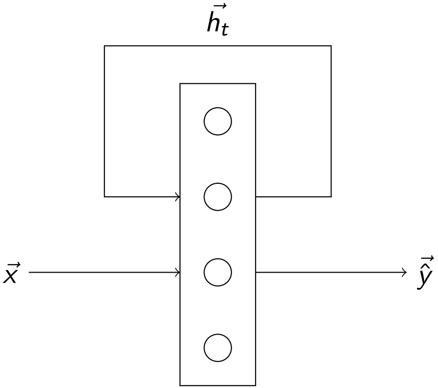
Recurrent Neural Networks

- Recurrent Neural Networks allow one to propagate states through time
- They calculate a hidden state $\vec{h_t}$ which is kept for the next time step, as $h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$



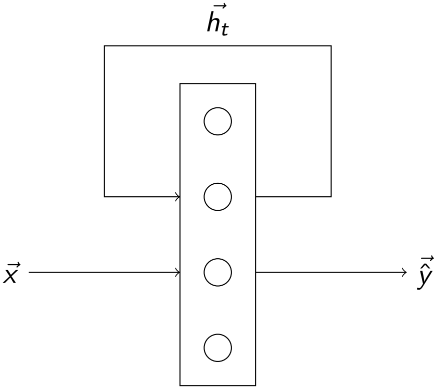
Recurrent Neural Networks

- Recurrent Neural Networks allow one to propagate states through time
- They calculate a hidden state $\vec{h_t}$ which is kept for the next time step, as $h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$
- From this hidden state the current output is calculated as $y_t = \sigma_y(W_y h_t + b_y)$



Recurrent Neural Networks

- Recurrent Neural Networks allow one to propagate states through time
- They calculate a hidden state $\vec{h_t}$ which is kept for the next time step, as $h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$
- From this hidden state the current output is calculated as $y_t = \sigma_y(W_y h_t + b_y)$
- These networks cannot represent long-term dependencies as U_h is always multiplied by itself.



LSTM

LSTM

- LSTM (Long-Short-Term Memory) can store long term dependencies by adding a cell state and utilizing an input, forget and output gates.

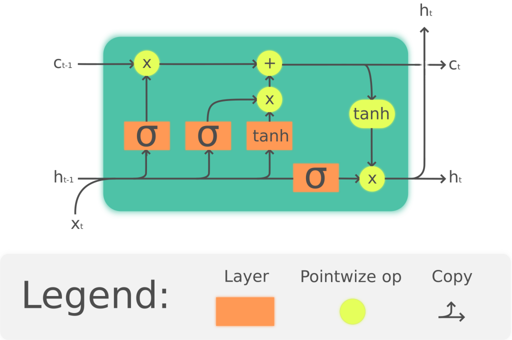


Figure: Courtesy of wikipedia

LSTM

- LSTM (Long-Short-Term Memory) can store long term dependencies by adding a cell state and utilizing an input, forget and output gates.
- The input gate decides how much of the input and hidden states are stored to the cell state.

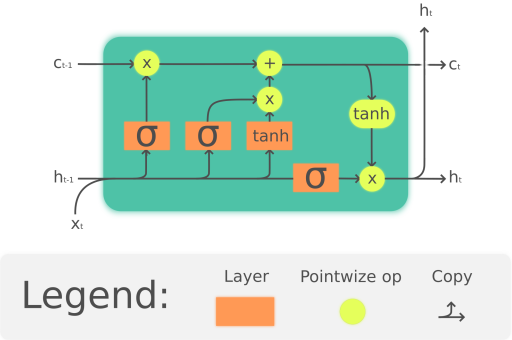


Figure: Courtesy of wikipedia

LSTM

- LSTM (Long-Short-Term Memory) can store long term dependencies by adding a cell state and utilizing an input, forget and output gates.
- The input gate decides how much of the input and hidden states are stored to the cell state.
- The forget gate decides how much of the cell state shall be deleted.

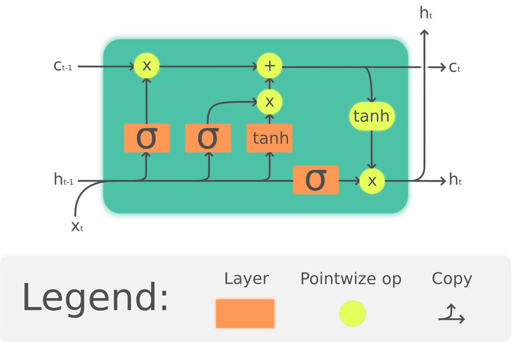


Figure: Courtesy of wikipedia

LSTM

- LSTM (Long-Short-Term Memory) can store long term dependencies by adding a cell state and utilizing an input, forget and output gates.
- The input gate decides how much of the input and hidden states are stored to the cell state.
- The forget gate decides how much of the cell state shall be deleted.
- The output gate decides how much of the cell state is output to the hidden state.

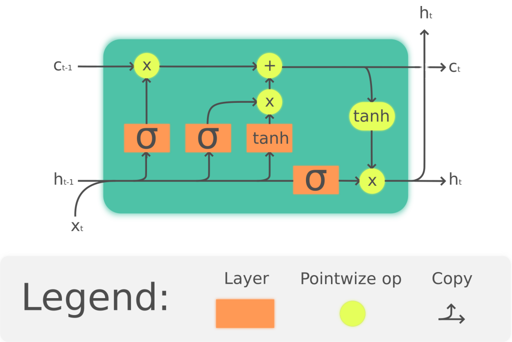


Figure: Courtesy of wikipedia

Word2Vec

Word2Vec

- How can we turn text into numbers?

Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.

Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:

Word2Vec

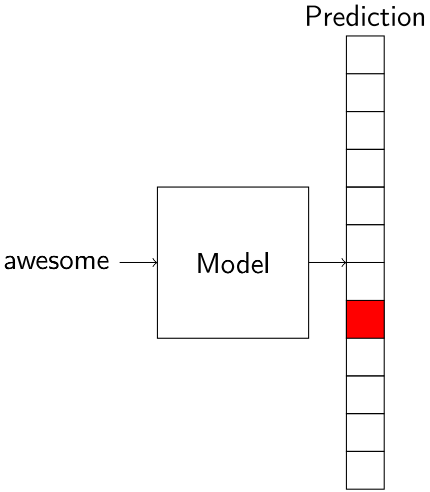
- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _____ food" and let a network predict "awesome"

Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _ _ _ _ food" and let a network predict "awesome"
  - Input "awesome" and let a network predict "Pizza", "is", and "food"

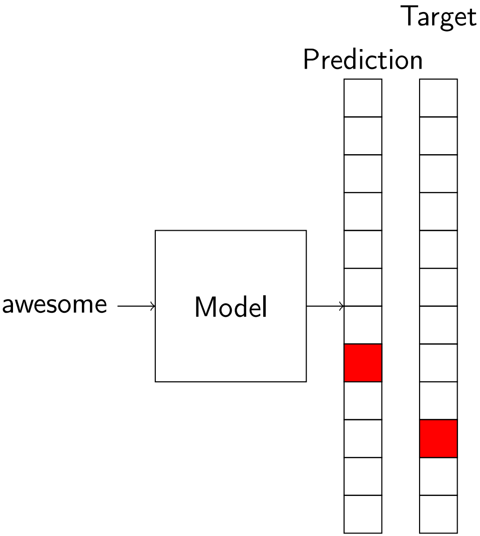
Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _____ food" and let a network predict "awesome"
  - Input "awesome" and let a network predict "Pizza", "is", and "food"



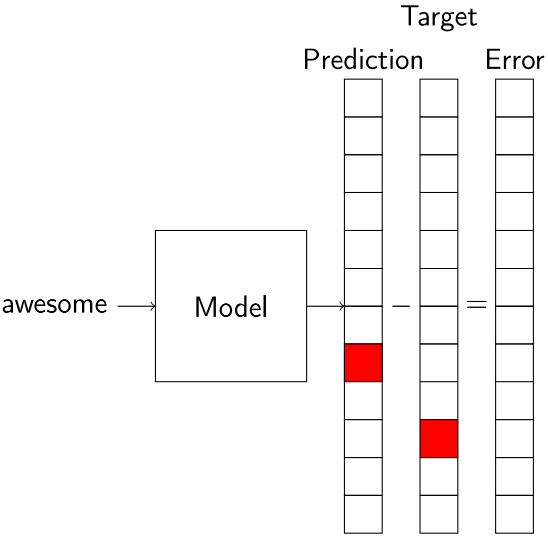
Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
    - Input "Pizza is _____ food" and let a network predict "awesome"
    - Input "awesome" and let a network predict "Pizza", "is", and "food"



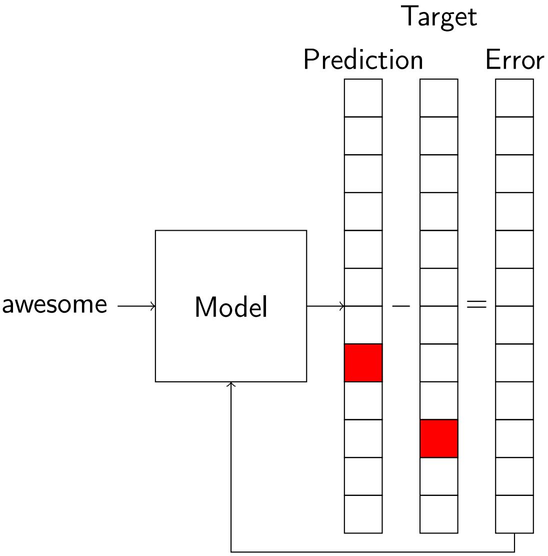
Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _ _ _ _ food" and let a network predict "awesome"
  - Input "awesome" and let a network predict "Pizza", "is", and "food"



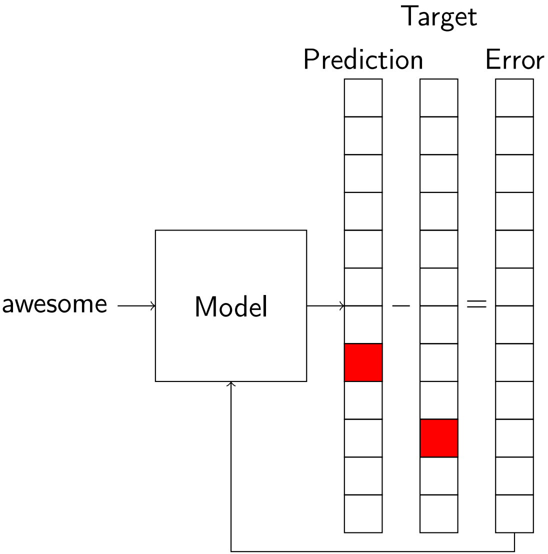
Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _____ food" and let a network predict "awesome"
  - Input "awesome" and let a network predict "Pizza", "is", and "food"



Word2Vec

- How can we turn text into numbers?
- Well let us use a neural network for this.
- Let us have a look at the sentence: "Pizza is awesome food". We can derive two tasks from it:
  - Input "Pizza is _ _ _ _ food" and let a network predict "awesome"
  - Input "awesome" and let a network predict "Pizza", "is", and "food"
- As classical training would update all the weights for each target example (where vocabularies can be million of words), the weights will only be updated for the right targets and a random selection of false targets



Chatbot

Chatbot

- Let's chat

Chatbot

- Let's chat
- The notebook we will use is inspired by the PyTorch chatbot tutorial, but it utilizes AllenNLP.

Encoder

Encoder

- To convert the words to vectors we will use nn.Embedding. These can be trained as part of the neural network.

Encoder

- To convert the words to vectors we will use nn.Embedding. These can be trained as part of the neural network.
- We will then feed these vectors to a bi-directional encoder

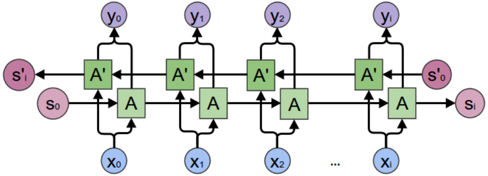


Figure: Image from `https://colah.github.io/posts/2015-09-NN-Types-FP/`

Encoder

- To convert the words to vectors we will use nn.Embedding. These can be trained as part of the neural network.
- We will then feed these vectors to a bi-directional encoder
- This will return output vectors for each time-step

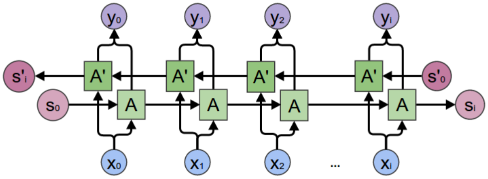


Figure: Image from https://colah.github.io/posts/2015-09-NN-Types-FP/

Attention

Attention

- The decoder is build by a one directional recurrent neural network.

Attention

- The decoder is build by a one directional recurrent neural network.
- The question is now how we input the vectors from the encoder to the decoder.

Attention

- The decoder is build by a one directional recurrent neural network.
- The question is now how we input the vectors from the encoder to the decoder.
- We will let the neural network figure out what is important in each step using attention

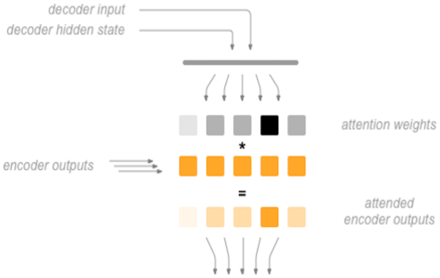


Figure: Courtesy of Sean Robertson

Attention

- The decoder is build by a one directional recurrent neural network.
- The question is now how we input the vectors from the encoder to the decoder.
- We will let the neural network figure out what is important in each step using attention
- The attention mechanism will use the current output of the network and weight the encoder vectors accordingly

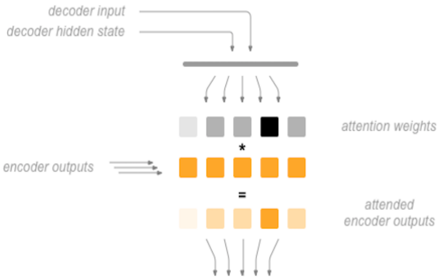


Figure: Courtesy of Sean Robertson

Attention

- The decoder is build by a one directional recurrent neural network.
- The question is now how we input the vectors from the encoder to the decoder.
- We will let the neural network figure out what is important in each step using attention
- The attention mechanism will use the current output of the network and weight the encoder vectors accordingly
- It will then merge the weighted encoder vectors with the current decoder output

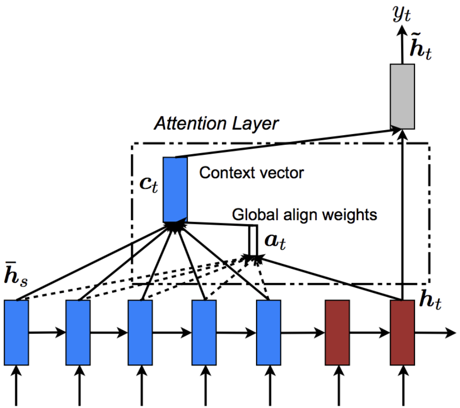


Figure: From arxiv 1508.04025