Deep Learning

QCon London, March 9th

Today's Overview

Today's Overview

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

Today's Overview

- 9:00 Introduction to Deep Learning
  - Short recap on machine learning
  - Build and train a perceptron in numpy
  - Gradient Descent
  - Move the code to the GPU using PyTorch
- 10:45 Coffee Break

Today's Overview

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

Today's Overview

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

Today's Overview

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

Today's Overview

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

Why Deep Learning?

Why Deep Learning?

- It can save a lot of time of boooooring feature engineering and writing exceptions

Why Deep Learning?

- It can save a lot of time of boooooring feature engineering and writing exceptions
- It can write computational rules nobody of us would ever think of

Why Deep Learning?

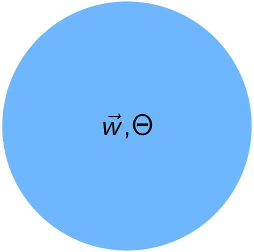
- It can save a lot of time of boooooring feature engineering and writing exceptions
- It can write computational rules nobody of us would ever think of
- It can often write better computational rules than we could

Why Deep Learning?

- It can save a lot of time of boooooring feature engineering and writing exceptions
- 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

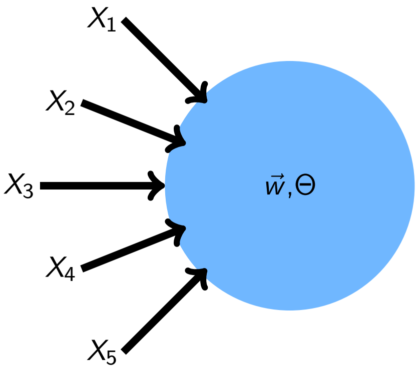
Perceptron

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



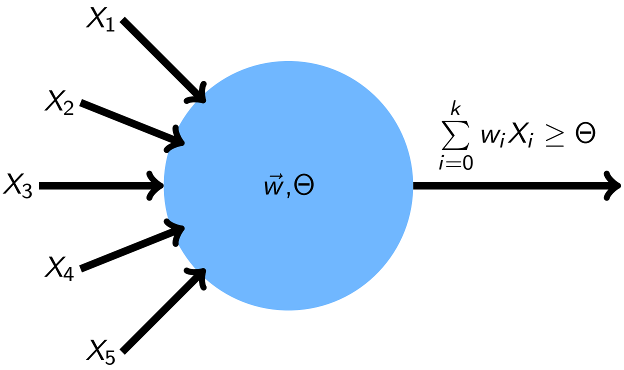
Perceptron

- 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

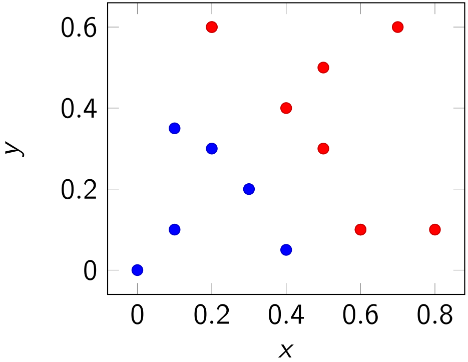
Conclusions

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

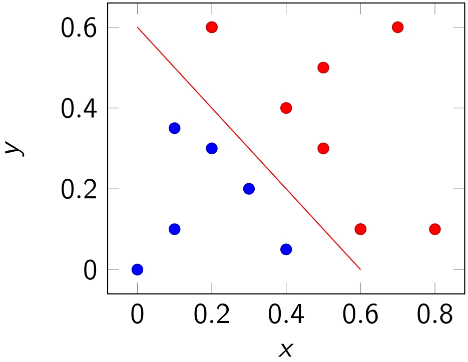
Conclusions

- 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

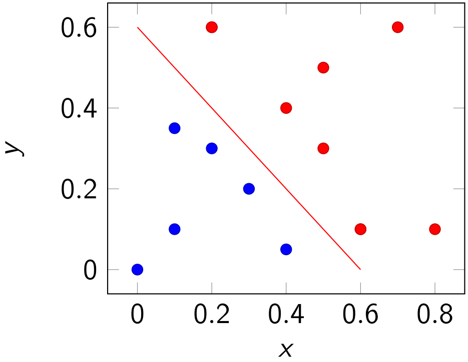


Linear Separability

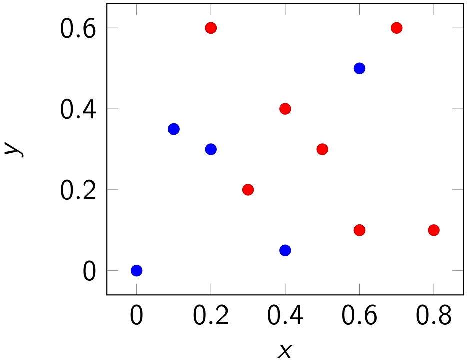


Data are linearly separable -> Can be divided by a plane

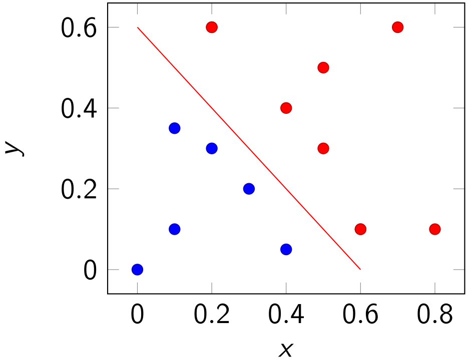
Linear Separability



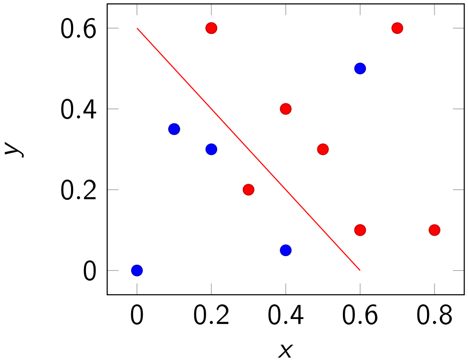
Data are linearly separable -> Can be divided by a plane



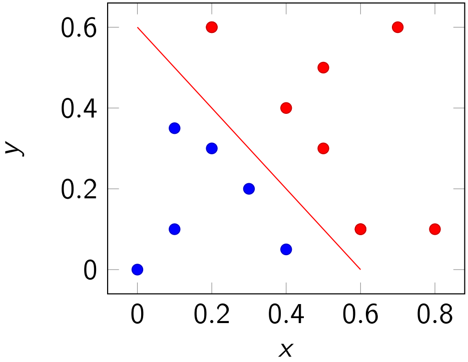
Linear Separability



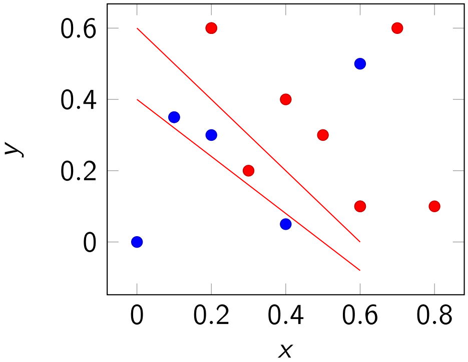
Data are linearly separable -> Can be divided by a plane



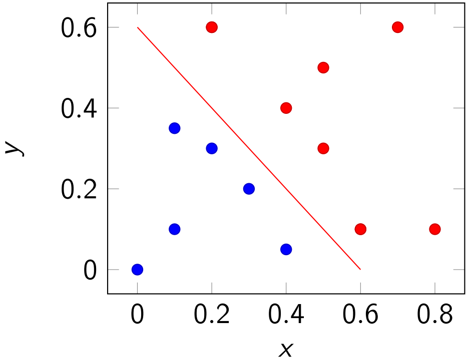
Linear Separability



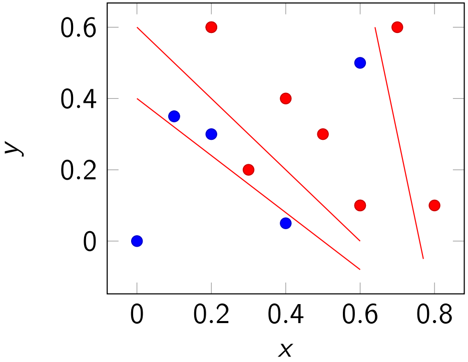
Data are linearly separable -> Can be divided by a plane



Linear Separability

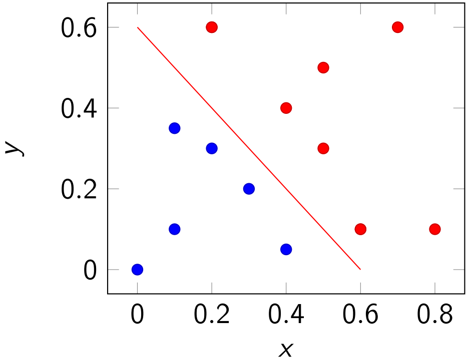


Data are linearly separable -> Can be divided by a plane

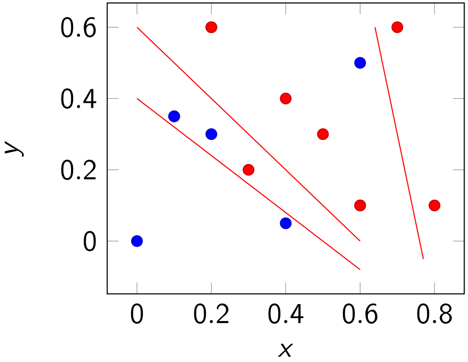


Data is not linearly separable -> Can not be divided by a plane

Linear Separability



Data are 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

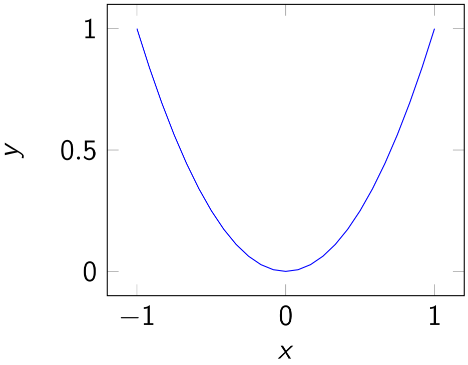
- What we calculated is an activation
  $a = \sum_i w_i X_i$

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.

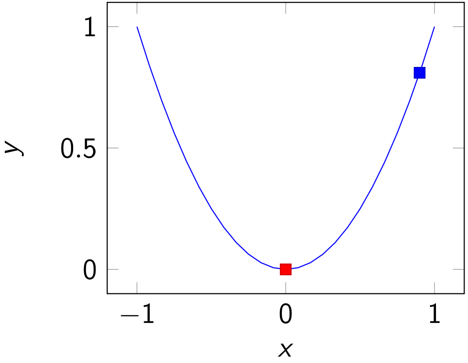
Gradient Descent

- 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: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$



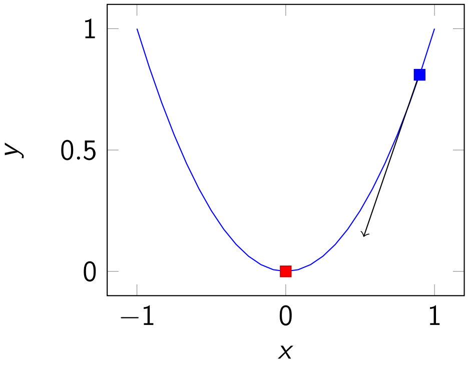
Gradient Descent

- 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: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$



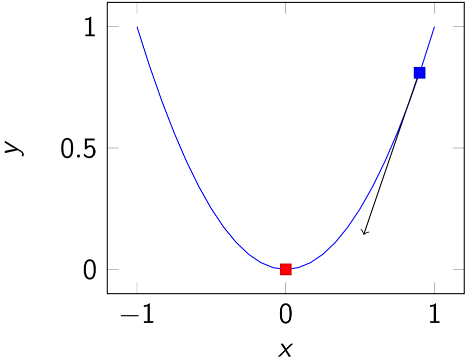
Gradient Descent

- 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: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$
- If we want to decrease $E(w)$ by changing w we need to calculate the gradient of w



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: $E(w) = \frac{1}{2} \sum_{(x,y) \in D} (y - a)^2$
- If we want to decrease $E(w)$ by changing w we need to calculate the gradient of w
- Every deep learning framework can calculate those weights using the chain rule and backpropagation.



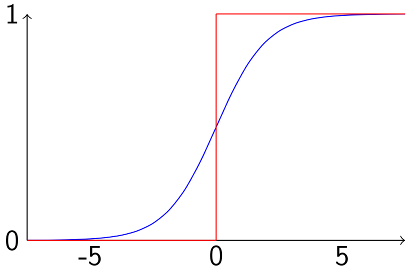
Sigmoid $\rightarrow$ Differentiable Threshold

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

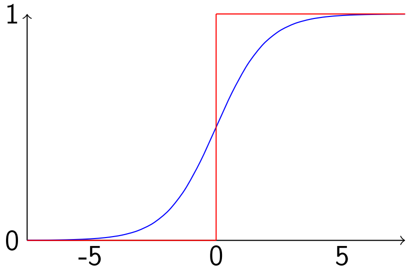
Sigmoid → Differentiable Threshold

- 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 $\rightarrow$ 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
- $\sigma(a) = \frac{1}{1+e^{-a}}$
- The sigmoid allows us to convert the strict classification into a regression over the probability for each point to belong to a certain class



Softmax

Softmax

- To generalize to n-class classification problems we have to extend the function, that calculates the probability

Softmax

- To generalize to n-class classification problems we have to extend the function, that calculates the probability
- Softmax: $p_i = \frac{e^{o_i}}{\sum_i e^{o_i}}$ probabillity for being in class i, given the output o_i

Softmax

- To generalize to n-class classification problems we have to extend the function, that calculates the probability
- Softmax: $p_i = \frac{e^{o_i}}{\sum_i e^{o_i}}$ probabillity for being in class i, given the output o_i
- With this we can define the general loss

Softmax

- To generalize to n-class classification problems we have to extend the function, that calculates the probability
- Softmax: $p_i = \frac{e^{o_i}}{\sum_i e^{o_i}}$ probabillity for being in class i, given the output o_i
- With this we can define the general loss
- Cross-Entropy $E = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{m} y_{i,k} \ln p_{i,k}$

  $y_{i,k}$ 1, if $i = k$ else 0
  $p_{i,k}$ probability for point i belonging to class k

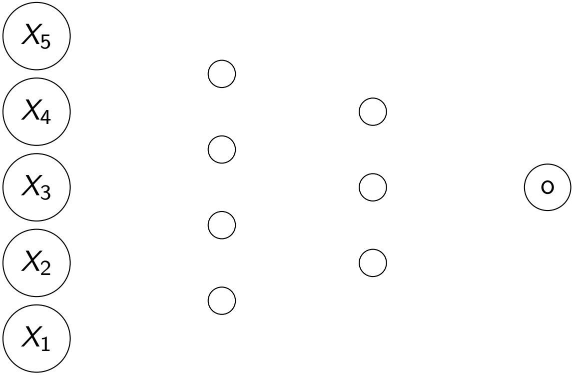
Neural Network

Neural Network

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

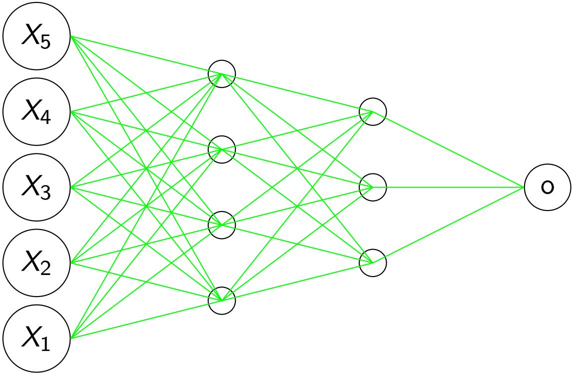
Neural Network

- Similar to the perceptron we have a lot of inputs and output
- However, instead of going from start to end directly we now add hidden units



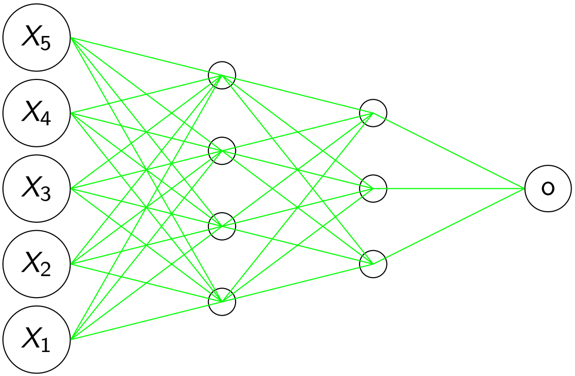
Neural Network

- Similar to the perceptron we have a lot of inputs and output
- However, instead of going from start to end directly we now add hidden units
- For each of the units we now apply the sigmoid function



Neural Network

- Similar to the perceptron we have a lot of inputs and output
- However, instead of going from start to end directly we now add hidden units
- 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 minimise the error as we can calculate the gradients



Optimising Weights

- There are multiple ways to optimize the weights in a neural network.

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent
- Advanced Methods

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives
  - Randomised Optimisation

Optimising Weights

- There are multiple ways to optimize the weights in a neural network.
- "Simple" Gradient Descent
- Advanced Methods
  - Momentum
  - Higher Order Derivatives
  - Randomised Optimisation
  - Penalty for Complexity

Restriction Bias

Restriction Bias

- Representational power of the algorithm due to the hypothesis considered

Restriction Bias

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan

Restriction Bias

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- Sigmoids $\rightarrow$ non-linear function

Restriction Bias

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?

Restriction Bias

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- 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

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- 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

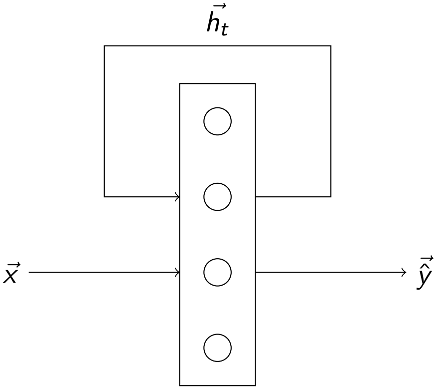
Restriction Bias

- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- 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

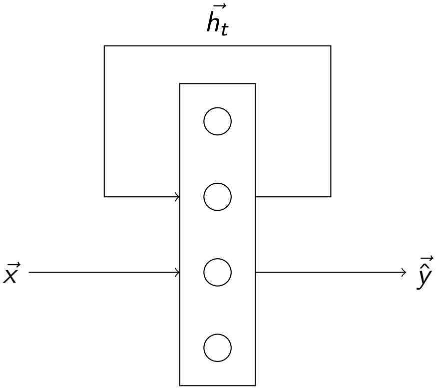
- Representational power of the algorithm due to the hypothesis considered
- Perceptron $\rightarrow$ can only split the data using a plan
- Sigmoids $\rightarrow$ non-linear function
- What can we represent with a network?
  - Boolean Functions: Yes, because we have a network of threshold-like units
  - Continuous Functions: Yes, with one (infinitely wide) layer
  - Arbitrary Functions: Yes, with two (infinitely wide) layers
- What do we do if we want to store information about previous examples in the network

Recurrent Neural Networks



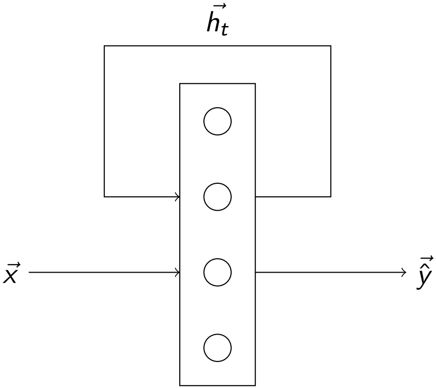
Recurrent Neural Networks

- Recurrent Neural Networks allow one to propagate states through time



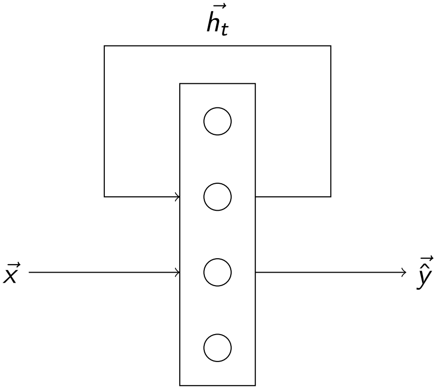
Recurrent Neural Networks

- 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 $y_t = \sigma_y(W_y h_t + b_y)$
- 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.

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 state are stored to the cell state.

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 state are stored to the cell state.
- The forget gate decides how much of the cell state shall be deleted.

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 state 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.

Word2Vec

Word2Vec

- How can we turn text into numbers?

Chatbot

Chatbot

- Let's chat