Introduction to Deep Learning

Antonio Valerio Miceli Barone

amiceli@inf.ed.ac.uk

10 March 2020

What is machine learning?

- Big data
  "Data is the New Oil" - Clive Humby, 2006
- Artificial intelligence
  "AI is the New Electricity" - Andrew Ng, 2016

What is machine learning?

- Big data
  "Data is the New Oil" - Clive Humby, 2006
- Artificial intelligence
  "AI is the New Electricity" - Andrew Ng, 2016
- Applications
  - Image classification

 $\rightarrow$ **Cat**

What is machine learning?

- Big data
  "Data is the New Oil" - Clive Humby, 2006
- Artificial intelligence
  "AI is the New Electricity" - Andrew Ng, 2016
- Applications
  - Image classification
     $\rightarrow$ **Cat**
  - Image generation
    Cat $\rightarrow$ 

What is machine learning?

- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...

What is machine learning?

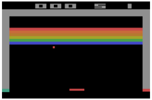
- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    You can't believe what you're looking at because it's so hideous to behold. $\rightarrow$ ☹

What is machine learning?

- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    You can't believe what you're looking at because it's so hideous to behold. → ☹
  - Machine translation
    The cat sat on the mat → **Die Katze saß auf der Matte**

What is machine learning?

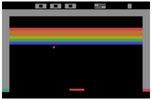
- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    You can't believe what you're looking at because it's so hideous to behold. → ☹
  - Machine translation
    The cat sat on the mat → **Die Katze saß auf der Matte**
  - Game playing



Atari

What is machine learning?

- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    You can't believe what you're looking at because it's so hideous to behold. → ☹
  - Machine translation
    The cat sat on the mat → **Die Katze saß auf der Matte**
  - Game playing

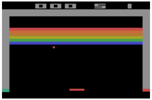


Atari

Go

What is machine learning?

- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    You can't believe what you're looking at because it's so hideous to behold. → ☹
  - Machine translation
    The cat sat on the mat → **Die Katze saß auf der Matte**
  - Game playing



Atari

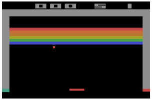
Go

Dota 2

Starcraft II

What is machine learning?

- Applications
  - Automatic speech recognition
  - Text-to-speech
    Siri, Alexa, Google Home, ...
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    You can't believe what you're looking at because it's so hideous to behold. → ☹
  - Machine translation
    The cat sat on the mat → **Die Katze saß auf der Matte**
  - Game playing



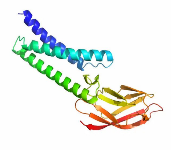
Atari

Go

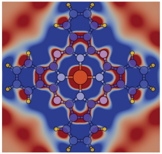
Dota 2

Starcraft II

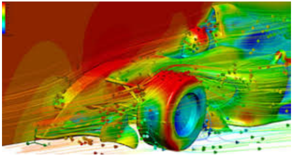
  - Natural sciences and engineering



Protein folding



Quantum physics



Computational fluid dynamics

What is machine learning?

- Machine learning

What is machine learning?

- Machine learning
  - **Automatically create models of the world from data, such that these models can make automatic predictions or decisions**

What is machine learning?

- Machine learning
  - **Automatically create models of the world from data, such that these models can make automatic predictions or decisions**
  - Statistics
    - Use data to understand how the world works

What is machine learning?

- Machine learning
  - **Automatically create models of the world from data, such that these models can make automatic predictions or decisions**
  - Statistics
    - Use data to understand how the world works
  - Artificial intelligence
    - Make machines smart

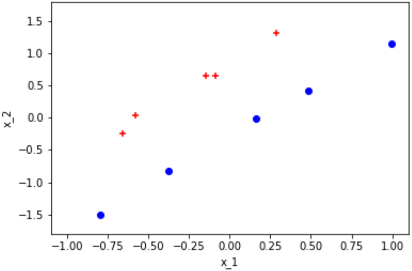
What is machine learning?

- Machine learning
  - **Automatically create models of the world from data, such that these models can make automatic predictions or decisions**
  - Statistics
    - Use data to understand how the world works
  - Artificial intelligence
    - Make machines smart
      "AI is whatever hasn't been done yet" - Larry Tesler

Example

- Training set:

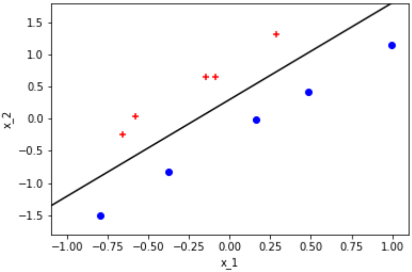
x_1	x_2	y
1.00	1.15	False
-0.58	0.04	True
-0.38	-0.82	False
0.16	-0.01	False
0.29	1.31	True
-0.66	-0.24	True
-0.80	-1.50	False
-0.14	0.66	True
-0.09	0.66	True
0.48	0.41	False



Example

- Training set:

x_1	x_2	y
1.00	1.15	False
-0.58	0.04	True
-0.38	-0.82	False
0.16	-0.01	False
0.29	1.31	True
-0.66	-0.24	True
-0.80	-1.50	False
-0.14	0.66	True
-0.09	0.66	True
0.48	0.41	False

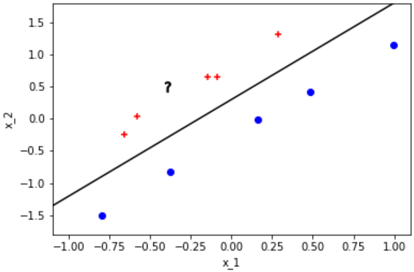


- Find decision boundary

Example

- Training set:

x_1	x_2	y
1.00	1.15	False
-0.58	0.04	True
-0.38	-0.82	False
0.16	-0.01	False
0.29	1.31	True
-0.66	-0.24	True
-0.80	-1.50	False
-0.14	0.66	True
-0.09	0.66	True
0.48	0.41	False



- Find decision boundary
- Test point:

x_1	x_2	y
-0.40	0.50	**?**

Example

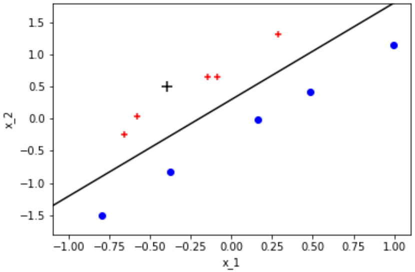
- Training set:

x_1	x_2	y
1.00	1.15	False
-0.58	0.04	True
-0.38	-0.82	False
0.16	-0.01	False
0.29	1.31	True
-0.66	-0.24	True
-0.80	-1.50	False
-0.14	0.66	True
-0.09	0.66	True
0.48	0.41	False

- Find decision boundary
- Test point:

x_1	x_2	y
-0.40	0.50	**True**

- Make a prediction



Supervised learning

- Supervised learning
  - Training set of N **labeled** examples
    $(x^{(1)}, x^{(2)}, \ldots x^{(N)})$
    $(y^{(1)}, y^{(2)}, \ldots y^{(N)})$

Supervised learning

- Supervised learning
  - Training set of N **labeled** examples
    $(x^{(1)}, x^{(2)}, \ldots x^{(N)})$
    $(y^{(1)}, y^{(2)}, \ldots y^{(N)})$
  - Find a parameter θ for a function f that computes y from x
    $y = f_{\theta}(x)$

Supervised learning

- Supervised learning
  - Training set of N **labeled** examples
    $(x^{(1)}, x^{(2)}, \ldots x^{(N)})$
    $(y^{(1)}, y^{(2)}, \ldots y^{(N)})$
  - Find a parameter θ for a function f that computes y from x
    $y = f_\theta(x)$
  - Or a conditional probability distribution
    $p_\theta(y|x)$

Supervised learning

- Supervised learning
  - Training set of N **labeled** examples
    $(x^{(1)}, x^{(2)}, \ldots x^{(N)})$
    $(y^{(1)}, y^{(2)}, \ldots y^{(N)})$
  - Find a parameter θ for a function f that computes y from x
    $y = f_\theta(x)$
  - Or a conditional probability distribution
    $p_\theta(y|x)$
  - Inputs x can be arbitrary (e.g. text, images, speech)
    - But for now we assume $x \in \mathcal{R}^d$

Supervised learning

- Supervised learning
  - Training set of N **labeled** examples
    $(x^{(1)}, x^{(2)}, \ldots x^{(N)})$
    $(y^{(1)}, y^{(2)}, \ldots y^{(N)})$
  - Find a parameter θ for a function f that computes y from x
    $y = f_\theta(x)$
  - Or a conditional probability distribution
    $p_\theta(y|x)$
  - Inputs x can be arbitrary (e.g. text, images, speech)
    - But for now we assume $x \in \mathcal{R}^d$
  - Outputs y
    - Discrete labels: **classification problem**
    - Continuous values: **regression problem**

Supervised learning

- In the previous example
  - Inputs x: 2D vectors
  - Outputs y: binary classes
  - Linear model
    $f_\theta(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0)$
  - Model parameter
    $\theta = (w_1, w_2, b)$

Supervised learning

- In the previous example
  - Inputs x: 2D vectors
  - Outputs y: binary classes
  - Linear model
    $f_\theta(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0)$
  - Model parameter
    $\theta = (w_1, w_2, b)$
- Machine learning: find $\overset{*}{\theta}$ that fits the training data the best
  - We need to operationalize what we mean by "best"

Supervised learning

- In the previous example
  - Inputs x: 2D vectors
  - Outputs y: binary classes
  - Linear model
    $f_\theta(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0)$
  - Model parameter
    $\theta = (w_1, w_2, b)$
- Machine learning: find $\overset{*}{\theta}$ that fits the training data the best
  - We need to operationalize what we mean by "best"
  - Loss function
    $L(y, \hat{y})$
    - Measures how much the predicted label $\hat{y} = f_\theta(x)$ differs from the true label y

Supervised learning

- In the previous example
  - Inputs x: 2D vectors
  - Outputs y: binary classes
  - Linear model
    $f_\theta(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0)$
  - Model parameter
    $\theta = (w_1, w_2, b)$
- Machine learning: find $\overset{*}{\theta}$ that fits the training data the best
  - We need to operationalize what we mean by "best"
  - Loss function
    $L(y, \hat{y})$
    - Measures how much the predicted label $\hat{y} = f_\theta(x)$ differs from the true label y
    - Optimization problem
      $\overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_i L(y^{(i)}, f_\theta(x^{(i)}))$
  - How to train?

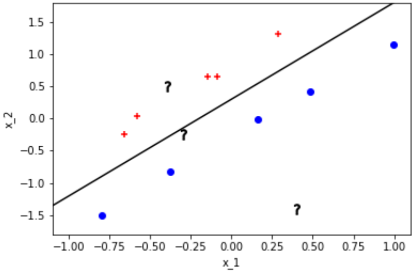
Supervised learning

- In the previous example
  - Inputs x: 2D vectors
  - Outputs y: binary classes
  - Linear model
    $f_\theta(x) = (w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0)$
  - Model parameter
    $\theta = (w_1, w_2, b)$
- Machine learning: find $\overset{*}{\theta}$ that fits the training data the best
  - We need to operationalize what we mean by "best"
  - Loss function
    $L(y, \hat{y})$
    - Measures how much the predicted label $\hat{y} = f_\theta(x)$ differs from the true label y
    - Optimization problem
      $\overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_i L(y^{(i)}, f_\theta(x^{(i)}))$
  - How to train?
    - Make the model probabilistic

Example

- Test set:

x_1	x_2	y
-0.40	0.50	**?**
-0.30	-0.25	**?**
0.4	-1.4	**?**

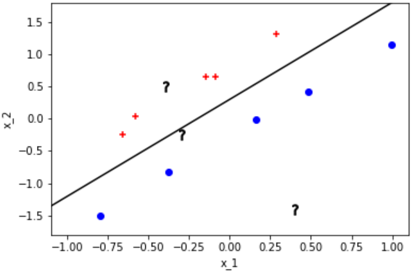


Example

- Test set:

x_1	x_2	y
-0.40	0.50	**?**
-0.30	-0.25	**?**
0.4	-1.4	**?**

- Points close to the decision boundary should have $p \approx 0.5$

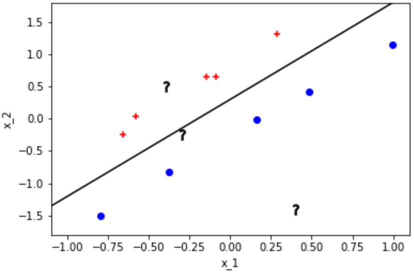


Example

- Test set:

x_1	x_2	y
-0.40	0.50	**?**
-0.30	-0.25	**?**
0.4	-1.4	**?**

- Points close to the decision boundary should have $p \approx 0.5$
- Pre-activation: $z = w^T x + b$ ranges from $-\infty to \infty$, equal to 0 on the boundary

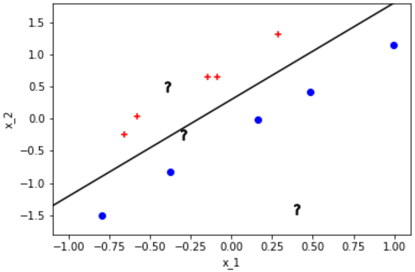


Example

- Test set:

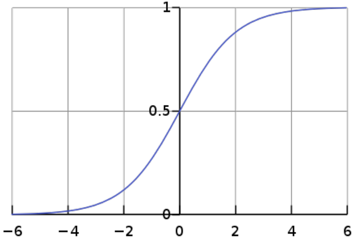
x_1	x_2	y
-0.40	0.50	**?**
-0.30	-0.25	**?**
0.4	-1.4	**?**

- Points close to the decision boundary should have $p \approx 0.5$
- Pre-activation: $z = w^T x + b$ ranges from $-\infty to \infty$, equal to 0 on the boundary
- Rescale to $(0, 1)$ with the **logistic sigmoid** function σ $p(y|z) = \sigma(z) = \frac{1}{1+\exp(-z)}$



Logistic regression

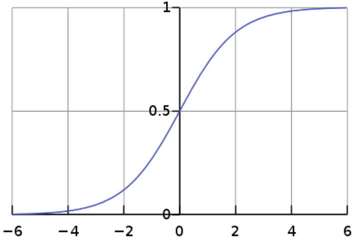
- Pre-activation: $z = w^T x + b$
- Rescale to $(0, 1)$ with the **logistic sigmoid** function σ
$p(y|z) = \sigma(z) = \frac{1}{1+\exp(-z)}$



inary cross-entropy
$$F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$$

Logistic regression

- Pre-activation: $z = w^T x + b$
- Rescale to $(0, 1)$ with the **logistic sigmoid** function σ
  $p(y|z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
- Loss function: negative log-likelihood of the data under the model
  $L(y, p_\theta(.|x)) = -\log p_\theta(y|x)$

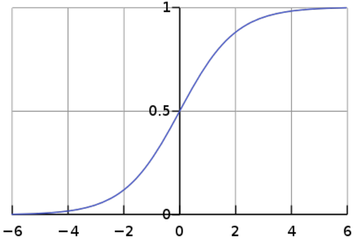


inary cross-entropy

$$F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$$

Logistic regression

- Pre-activation: $z = w^T x + b$
- Rescale to $(0, 1)$ with the **logistic sigmoid** function σ
  $p(y|z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
- Loss function: negative log-likelihood of the data under the model
  $L(y, p_\theta(.|x)) = -\log p_\theta(y|x)$
- Training objective
  $\overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} F(\theta)$
  $F(\theta) = -\frac{1}{N} \sum_i \log p_\theta(y^{(i)}|x^{(i)})$

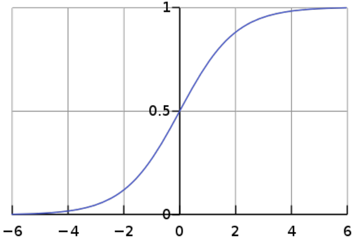


inary cross-entropy

$$F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$$

Logistic regression

- Pre-activation: $z = w^T x + b$
- Rescale to $(0, 1)$ with the **logistic sigmoid** function σ
  $p(y|z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
- Loss function: negative log-likelihood of the data under the model
  $L(y, p_\theta(.|x)) = -\log p_\theta(y|x)$
- Training objective
  $\overset{*}{\theta} = \underset{\theta}{\operatorname{argmin}} F(\theta)$
  $F(\theta) = -\frac{1}{N}\sum_i \log p_\theta(y^{(i)}|x^{(i)})$



Binary cross-entropy

$$F(\theta) = -\frac{1}{N}\sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$$

Logistic regression

- Optimize the binary cross-entropy

$$\overset{*}{\theta} = \underset{\theta}{\text{argmin}} F(\theta)$$

$$F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$$

Logistic regression

- Optimize the binary cross-entropy
  $\overset{*}{\theta} = \underset{\theta}{\text{argmin}} F(\theta)$
  $F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$
- Differentiate $F(\theta)$ with respect to θ
- The gradient must be zero at the minimum
  $\nabla_\theta F(\overset{*}{\theta}) = 0$

Logistic regression

- Optimize the binary cross-entropy
  $\overset{*}{\theta} = \underset{\theta}{\text{argmin}} F(\theta)$
  $F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$
- Differentiate $F(\theta)$ with respect to θ
- The gradient must be zero at the minimum
  $\nabla_\theta F(\overset{*}{\theta}) = 0$
  - Modern deep learning tools (PyTorch, Tensorflow) automate the computation of the gradient

Logistic regression

- Optimize the binary cross-entropy

  $\overset{*}{\theta} = \underset{\theta}{\text{argmin}} F(\theta)$

  $F(\theta) = -\frac{1}{N}\sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$
- Differentiate $F(\theta)$ with respect to θ
- The gradient must be zero at the minimum

  $\nabla_\theta F(\overset{*}{\theta}) = 0$
  - Modern deep learning tools (PyTorch, Tensorflow) automate the computation of the gradient
  - In this simple model there is only one stationary point: the global minimum

Logistic regression

- Optimize the binary cross-entropy

  $\overset{*}{\theta} = \underset{\theta}{\mathrm{argmin}} F(\theta)$

  $F(\theta) = -\frac{1}{N} \sum_i y^{(i)} \log z^{(i)} + (1 - y^{(i)}) \log(1 - z^{(i)})$
- Differentiate $F(\theta)$ with respect to θ
- The gradient must be zero at the minimum

  $\nabla_\theta F(\overset{*}{\theta}) = 0$
  - Modern deep learning tools (PyTorch, Tensorflow) automate the computation of the gradient
  - In this simple model there is only one stationary point: the global minimum
  - For more complicated deep learning models there are many stationary points

Gradient descent

- Solve approximately
  $\nabla_\theta F(\overset{*}{\theta}) = 0$

Gradient descent

- Solve approximately
  $\nabla_\theta F(\overset{*}{\theta}) = 0$
- Full-batch gradient descent
  - $\alpha :=$ learning rate
  - $\theta :=$ initial guess (e.g. random or all zeros)
  - while not converged
    - $\theta := \theta - \alpha \cdot \nabla_\theta F(\theta)$

Gradient descent

- Solve approximately
  $\nabla_\theta F(\overset{*}{\theta}) = 0$
- Full-batch gradient descent
  - $\alpha :=$ learning rate
  - $\theta :=$ initial guess (e.g. random or all zeros)
  - while not converged
    - $\theta := \theta - \alpha \cdot \nabla_\theta F(\theta)$
- If the learning rate is small enough, the algorithm converges close to a local minimum
- A global minimum in this case

Gradient descent

- Full-batch gradient descent
  - Each iteration requires evaluating all the training examples
  - can be quite slow
  - can overfit

Gradient descent

- Full-batch gradient descent
  - Each iteration requires evaluating all the training examples
  - can be quite slow
  - can overfit
- Stochastic gradient descent
- At each iteration, evaluate a proxy objective $\tilde{F}(\theta, B)$ defined only on a small random mini-batch of training examples

Gradient descent

- Full-batch gradient descent
  - Each iteration requires evaluating all the training examples
  - can be quite slow
  - can overfit
- Stochastic gradient descent
- At each iteration, evaluate a proxy objective $\tilde{F}(\theta, B)$ defined only on a small random mini-batch of training examples
- Algorithm:
  - $\alpha :=$ learning rate
  - $N_B :=$ batch size
  - $\theta :=$ initial guess (e.g. random or all zeros)
  - while not converged
    - $B :=$ random subset of training data of size N_B
    - $\theta := \theta - \alpha \cdot \nabla_\theta \tilde{F}(\theta, B)$

Gradient descent

- Full-batch gradient descent
  - Each iteration requires evaluating all the training examples
  - can be quite slow
  - can overfit
- Stochastic gradient descent
- At each iteration, evaluate a proxy objective $\tilde{F}(\theta, B)$ defined only on a small random mini-batch of training examples
- Algorithm:
  - $\alpha :=$ learning rate
  - $N_B :=$ batch size
  - $\theta :=$ initial guess (e.g. random or all zeros)
  - while not converged
    - $B :=$ random subset of training data of size N_B
    - $\theta := \theta - \alpha \cdot \nabla_\theta \tilde{F}(\theta, B)$
- $\nabla_\theta \tilde{F}(\theta, B)$ is an unbiased estimator of $\nabla_\theta F(\theta)$
- Converges with probability 1

Multi-class classification

- What if the output are in more than two classes?

Multi-class classification

- What if the output are in more than two classes?
- C : number of classes
- output y is one-hot
  $y \in \{0, 1\}^C$ s.t. $\sum_j y_j = 1$

Multi-class classification

- What if the output are in more than two classes?
- C : number of classes
- output y is one-hot
  $y \in \{0, 1\}^C$ s.t. $\sum_j y_j = 1$
- define C linear models

Multi-class classification

- What if the output are in more than two classes?
- C : number of classes
- output y is one-hot
  $y \in \{0, 1\}^C$ s.t. $\sum_j y_j = 1$
- define C linear models
  In matrix form:
  $z = W \cdot x + b$
  where:
  $b \in \mathcal{R}^C$ and $z \in \mathcal{R}^C$ are vectors
  $W \in \mathcal{R}^{d \times C}$ is a matrix

Multi-class classification

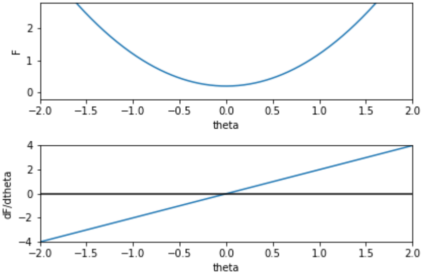
- What if the output are in more than two classes?
- C : number of classes
- output y is one-hot
  $y \in \{0, 1\}^C$ s.t. $\sum_j y_j = 1$
- define C linear models
  In matrix form:
  $z = W \cdot x + b$
  where:
  $b \in \mathcal{R}^C$ and $z \in \mathcal{R}^C$ are vectors
  $W \in \mathcal{R}^{d \times C}$ is a matrix
- Normalize probabilities: softmax
  $p(y_j|z) = \frac{\exp z_j}{\sum_{j'} \exp z_{j'}}$

Multi-class classification

- What if the output are in more than two classes?
- C : number of classes
- output y is one-hot
  $y \in \{0, 1\}^C$ s.t. $\sum_j y_j = 1$
- define C linear models
  In matrix form:
  $z = W \cdot x + b$
  where:
  $b \in \mathcal{R}^C$ and $z \in \mathcal{R}^C$ are vectors
  $W \in \mathcal{R}^{d \times C}$ is a matrix
- Normalize probabilities: softmax
  $p(y_j|z) = \frac{\exp z_j}{\sum_{j'} \exp z_{j'}}$
- Loss function: categorical cross-entropy
  $L(y, p_\theta(.|x)) = -\sum_j \log p_\theta(y_j|x)$

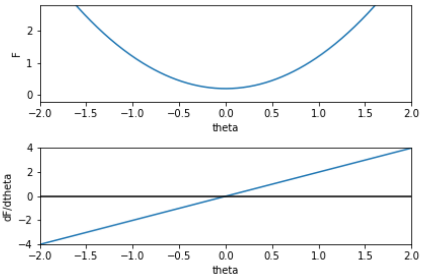
Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$



Gradient descent example

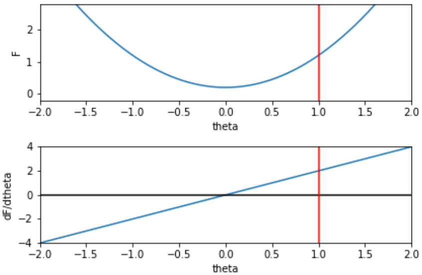
- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$



Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

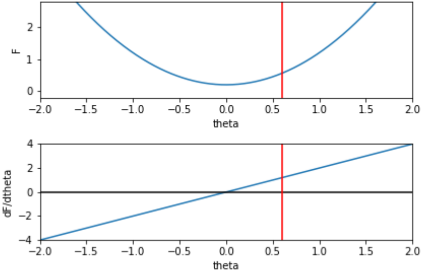
θ	$\nabla_\theta F(\theta)$
1.0	2.0



Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

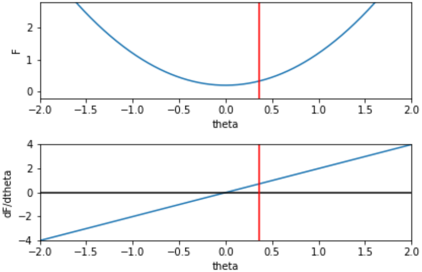
θ	$\nabla_\theta F(\theta)$
1.0	2.0
0.6	1.2



Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

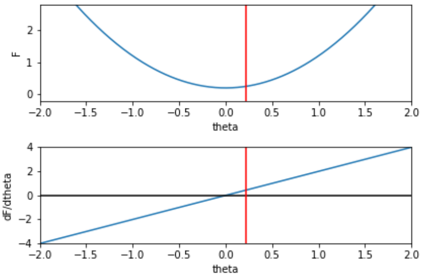
θ	$\nabla_\theta F(\theta)$
1.0	2.0
0.6	1.2
0.36	0.72



Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

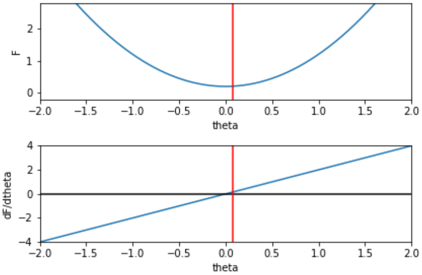
θ	$\nabla_\theta F(\theta)$
1.0	2.0
0.6	1.2
0.36	0.72
0.22	0.43



Gradient descent example

- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

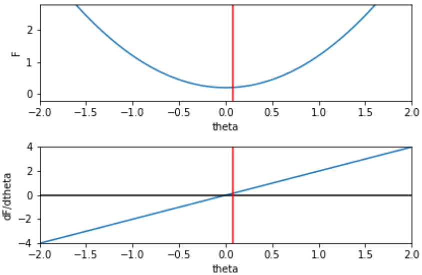
θ	$\nabla_\theta F(\theta)$
1.0	2.0
0.6	1.2
0.36	0.72
0.22	0.43
0.13	0.26



Gradient descent example

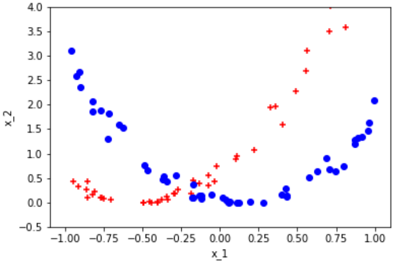
- Example
  $F(\theta) = \theta^2 + 0.2$
  $\nabla_\theta F(\theta) = \theta$
- Gradient descent update
  $\theta := \theta - 0.2 \cdot \nabla_\theta F(\theta)$
- Iterations:

θ	$\nabla_\theta F(\theta)$
1.0	2.0
0.6	1.2
0.36	0.72
0.22	0.43
0.13	0.26
0.08	0.16



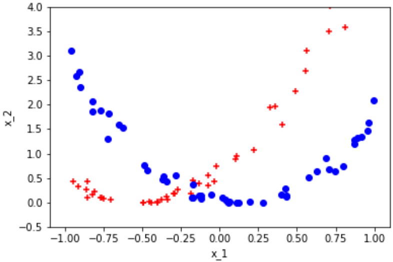
Limitations of linear models

- Sometimes the data is not linearly separable



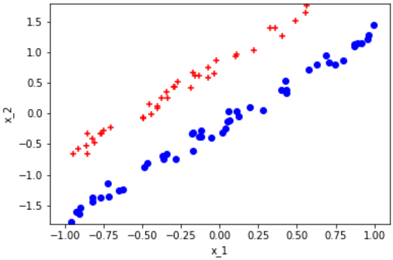
Limitations of linear models

- Sometimes the data is not linearly separable
- **Feature engineering**
  - Find a preprocessing transformation that makes the data linearly separable



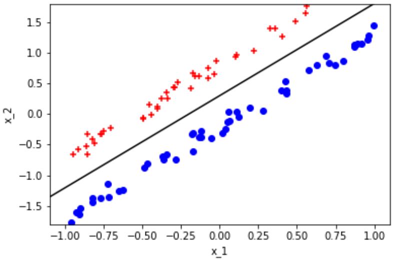
Limitations of linear models

- Sometimes the data is not linearly separable
- **Feature engineering**
  - Find a preprocessing transformation that makes the data linearly separable
  - For example $x_2 := \sqrt{x_2}$



Limitations of linear models

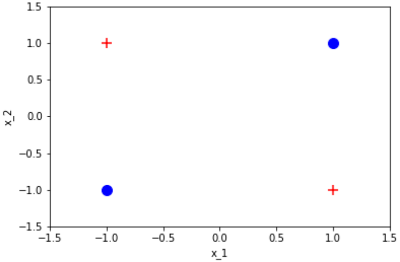
- Sometimes the data is not linearly separable
- **Feature engineering**
  - Find a preprocessing transformation that makes the data linearly separable
  - For example $x_2 := \sqrt{x_2}$
  - Now we can find a decision hyperplane



Limitations of linear models

- XOR problem

x_1	x_2	y
-1.0	-1.0	False
-1.0	1.0	True
1.0	-1.0	True
1.0	1.0	False

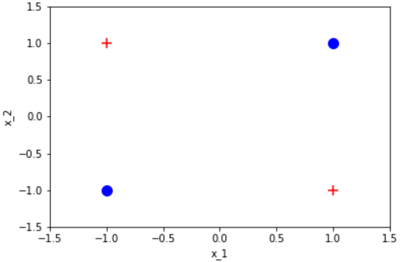


Limitations of linear models

- XOR problem

x_1	x_2	y
-1.0	-1.0	False
-1.0	1.0	True
1.0	-1.0	True
1.0	1.0	False

- Not linearly separable

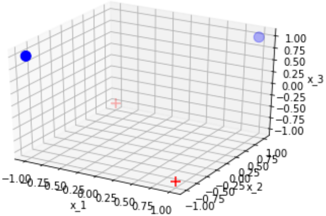


Limitations of linear models

- XOR problem

x_1	x_2	y
-1.0	-1.0	False
-1.0	1.0	True
1.0	-1.0	True
1.0	1.0	False

- Not linearly separable
- But becomes trivial if we define $x_3 = x_1 x_2$

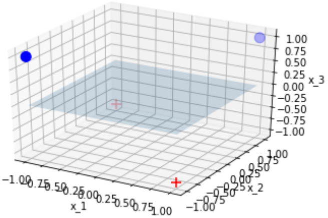


Limitations of linear models

- XOR problem

x_1	x_2	y
-1.0	-1.0	False
-1.0	1.0	True
1.0	-1.0	True
1.0	1.0	False

- Not linearly separable
- But becomes trivial if we define $x_3 = x_1 x_2$
- Solution: ignore x_1 and x_2 and use the sign of x_3

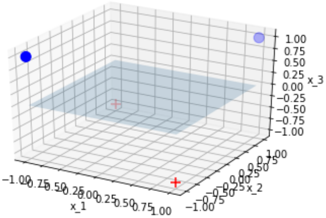


Limitations of linear models

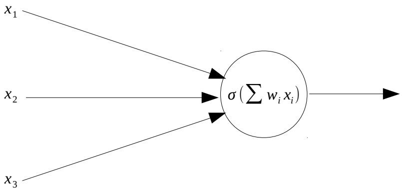
- XOR problem

x_1	x_2	y
-1.0	-1.0	False
-1.0	1.0	True
1.0	-1.0	True
1.0	1.0	False

- Not linearly separable
- But becomes trivial if we define $x_3 = x_1 x_2$
- Solution: ignore x_1 and x_2 and use the sign of x_3
- The XOR problem caused research in neural networks to be abandoned from the 70s to the mid 80s

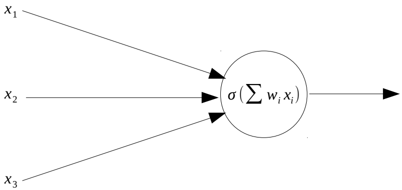


Limitations of linear models



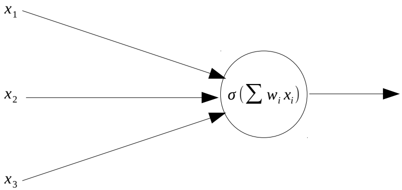
- Feature engineering used to be common until the new deep learning revival of the mid 2010s.
- It can be quite hard and labor-intensive

Limitations of linear models



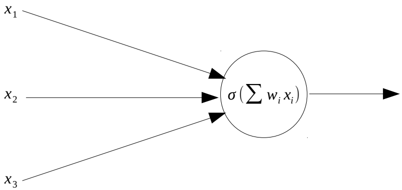
- Feature engineering used to be common until the new deep learning revival of the mid 2010s.
- It can be quite hard and labor-intensive
- Images
  - Edge features
  - Corner feature
  - ...

Limitations of linear models



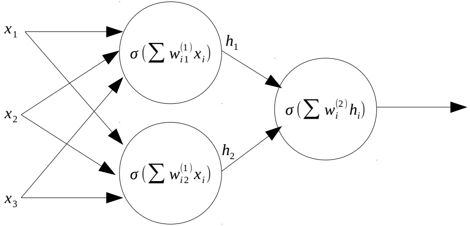
- Feature engineering used to be common until the new deep learning revival of the mid 2010s.
- It can be quite hard and labor-intensive
- Images
  - Edge features
  - Corner feature
  - ...
- Text
  - Word n-grams
  - Character n-grams
  - Part-of-Speech tags
  - Syntactic dependency relations
  - ...

Limitations of linear models



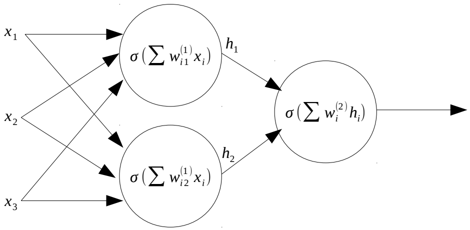
- Feature engineering used to be common until the new deep learning revival of the mid 2010s.
- It can be quite hard and labor-intensive
- Images
  - Edge features
  - Corner feature
  - ...
- Text
  - Word n-grams
  - Character n-grams
  - Part-of-Speech tags
  - Syntactic dependency relations
  - ...
- Can we learn non-linear features automatically?

Multi-layer perceptron



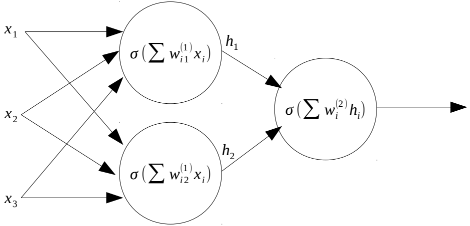
- Simplest deep feed-forward neural network

Multi-layer perceptron



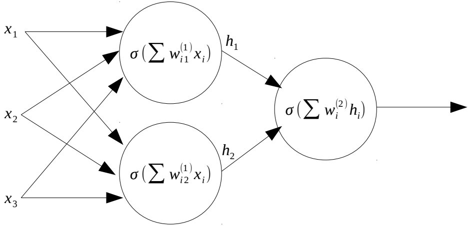
- Simplest deep feed-forward neural network
- Hidden layer(s) computes non-linear features

Multi-layer perceptron



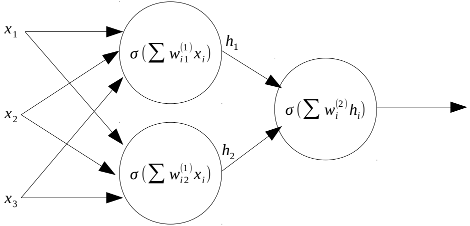
- Simplest deep feed-forward neural network
- Hidden layer(s) computes non-linear features
  - Activation function σ provides non-linearity

Multi-layer perceptron



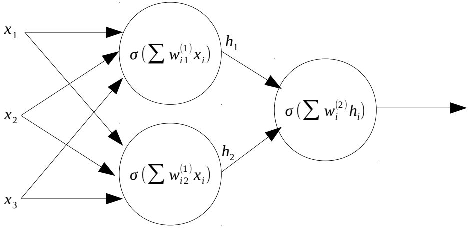
- Simplest deep feed-forward neural network
- Hidden layer(s) computes non-linear features
  - Activation function σ provides non-linearity
  - Deep linear models collapse to shallow linear models

Multi-layer perceptron



- Simplest deep feed-forward neural network
- Hidden layer(s) computes non-linear features
  - Activation function σ provides non-linearity
  - Deep linear models collapse to shallow linear models
- Fully connected

Multi-layer perceptron

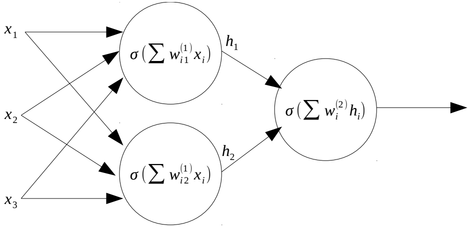


- Equations

$$h^{(0)} = x$$
$$h^{(k)} = \sigma_h(W^{(k)} \cdot h^{(k-1)} + b^{(k)})$$
$$p(y) = \sigma_{out}(W^{out} \cdot h^{(K)} + b^{out})$$

- K hidden layers of width $d^{(k)}$, $W(k) \in \mathcal{R}^{d^{(k-1)} \times d^{(k)}}$

Multi-layer perceptron



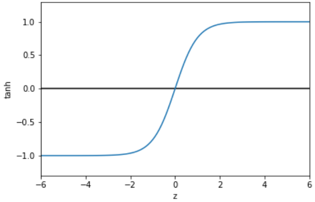
- Equations

$$h^{(0)} = x$$
$$h^{(k)} = \sigma_h(W^{(k)} \cdot h^{(k-1)} + b^{(k)})$$
$$p(y) = \sigma_{out}(W^{out} \cdot h^{(K)} + b^{out})$$

- K hidden layers of width $d^{(k)}$, $W(k) \in \mathcal{R}^{d^{(k-1)} \times d^{(k)}}$
- Usually the output activation function σ_{out} and the hidden activation function σ_h are different

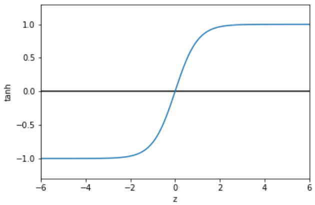
Activation functions

- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$



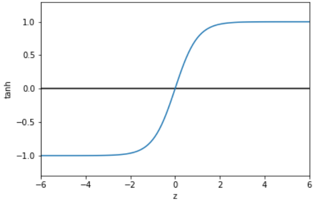
Activation functions

- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$
  - Bounded within $(-1, 1)$
  - Smooth



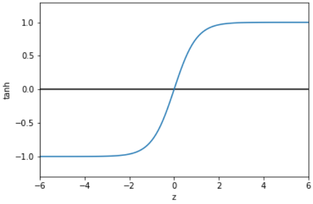
Activation functions

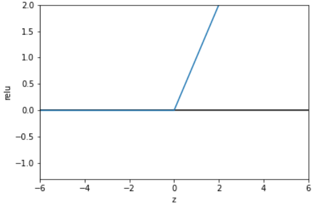
- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$
  - Bounded within $(-1, 1)$
  - Smooth
  - Mostly used in recurrent neural networks



Activation functions

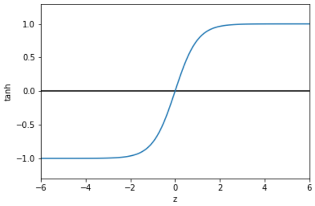
- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$
  - Bounded within $(-1, 1)$
  - Smooth
  - Mostly used in recurrent neural networks
- Rectified Linear Unit
  $\mathrm{relu}(z) = \max(0, z)$

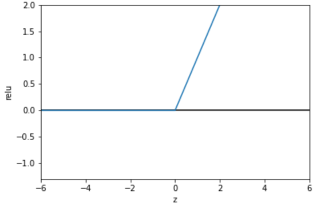




Activation functions

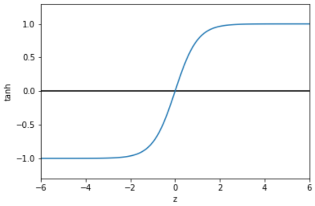
- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$
  - Bounded within $(-1, 1)$
  - Smooth
  - Mostly used in recurrent neural networks
- Rectified Linear Unit
  $\text{relu}(z) = \max(0, z)$
  - Unbounded
  - Piecewise linear
  - Differentiable **almost** everywhere

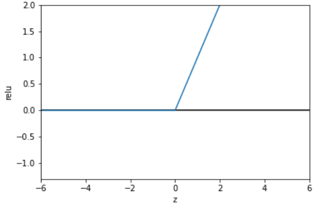




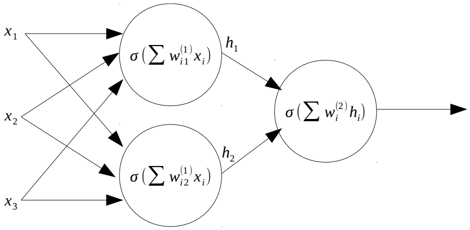
Activation functions

- Hyperbolic tangent
  $\tanh(z) = \frac{\exp(2z)-1}{\exp(2z)+1}$
  - Bounded within $(-1, 1)$
  - Smooth
  - Mostly used in recurrent neural networks
- Rectified Linear Unit
  $\mathrm{relu}(z) = \max(0, z)$
  - Unbounded
  - Piecewise linear
  - Differentiable **almost** everywhere
  - Most common choice in feed-forward neural networks



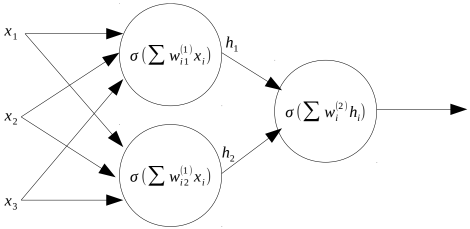


Multi-layer perceptron



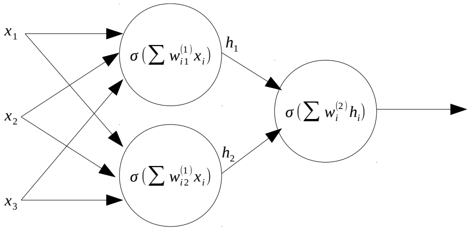
- How many layers, and how big?

Multi-layer perceptron



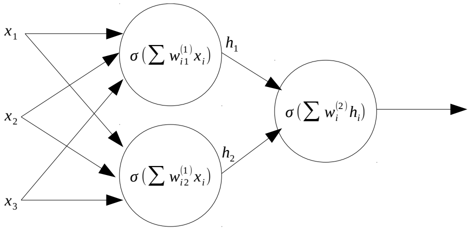
- How many layers, and how big?
  - No hard rule, trial and error

Multi-layer perceptron



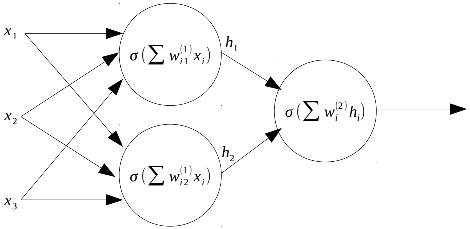
- How many layers, and how big?
  - No hard rule, trial and error
- Do we need more than one hidden layer?

Multi-layer perceptron



- How many layers, and how big?
  - No hard rule, trial and error
- Do we need more than one hidden layer?
  - One **sufficiently large** hidden layer is enough for universal approximation
  - In practice deeper networks generalize much better

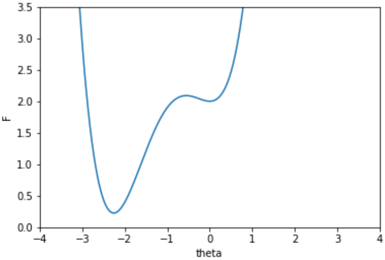
Multi-layer perceptron



- How many layers, and how big?
  - No hard rule, trial and error
- Do we need more than one hidden layer?
  - One **sufficiently large** hidden layer is enough for universal approximation
  - In practice deeper networks generalize much better

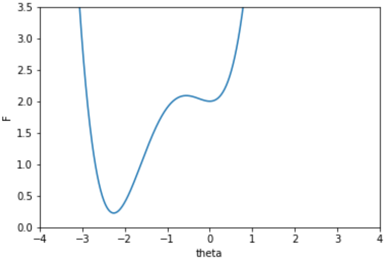
Training

- Stochastic Gradient Descent



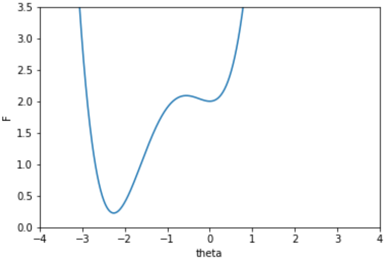
Training

- Stochastic Gradient Descent
- Non-convex optimization
  - Multiple local minima
  - Local maxima
  - Saddle points



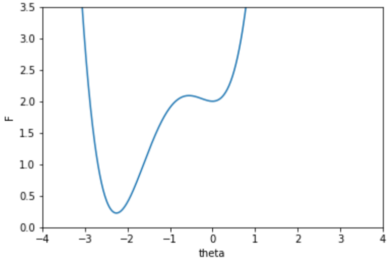
Training

- Stochastic Gradient Descent
- Non-convex optimization
  - Multiple local minima
  - Local maxima
  - Saddle points
- Larger width: **easier** optimization
  - In the 90s it was believed that neural networks were too hard to optimize
  - In the 2010s modern GPUs enabled training wider networks, which converge more easily



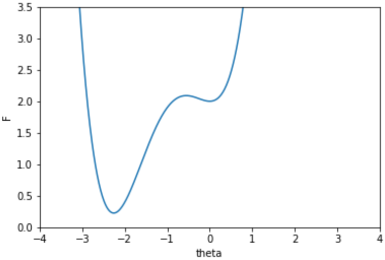
Training

- Stochastic Gradient Descent
- Non-convex optimization
  - Multiple local minima
  - Local maxima
  - Saddle points
- Larger width: **easier** optimization
  - In the 90s it was believed that neural networks were too hard to optimize
  - In the 2010s modern GPUs enabled training wider networks, which converge more easily
- Larger depth: **harder** optimization
  - Vanishing gradients (bad stationary points)
  - Exploding gradients (numerical divergences)



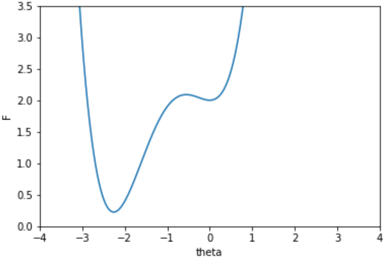
Training

- Deep neural networks can be trained effectively



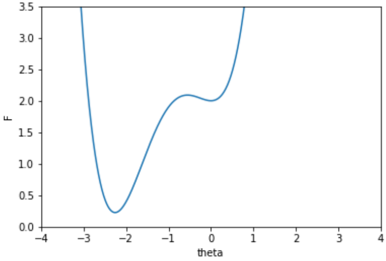
Training

- Deep neural networks can be trained effectively
- as long as we are careful
    - Optimizers
    - Initialization
    - Residual trick
    - Normalization layers
    - . . .

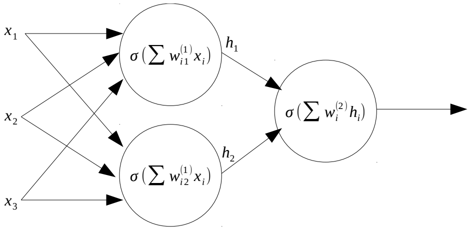


SGD with Momentum

- Add a velocity to the updates
- Algorithm:
  - $\alpha :=$ learning rate
  - $\beta :=$ momentum rate
  - $N_B :=$ batch size
  - $\theta :=$ initial guess (e.g. random or all zeros)
  - $\Delta\theta := 0$
  - while not converged
    - $B :=$ random subset of training data of size N_B
    - $\Delta\theta := -\alpha \cdot \nabla_\theta \tilde{F}(\theta, B) + \beta \cdot \Delta\theta$
    - $\theta := \theta + \Delta\theta$

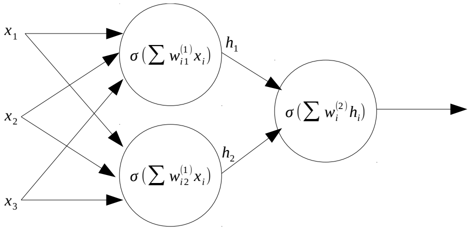


Parameter initalization



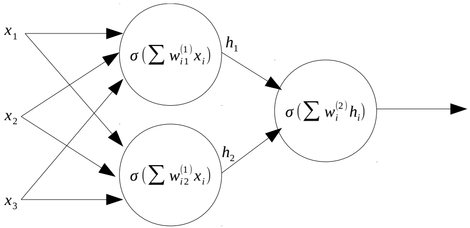
- Can we initialize the parameters to zero?

Parameter initalization



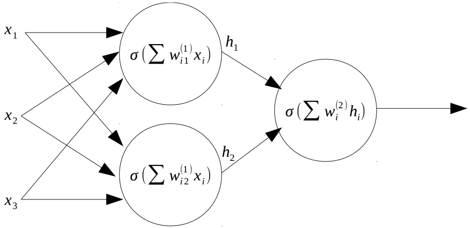
- Can we initialize the parameters to zero?
- No, we need a random initialization

Parameter initalization



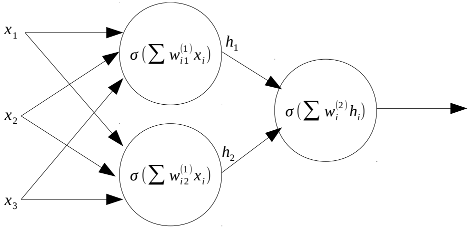
- Can we initialize the parameters to zero?
- No, we need a random initialization
- Glorot / He initializations
  - biases initialized to 0
  - For RELU layers (He)
    $W \sim N(0, 2/d^{(k-1)})$
  - For tanh or output layers (Glorot)
    $W \sim N(0, 1/d^{(k-1)})$

Residual trick



- How to train very very deep networks (hundreds of layers)?

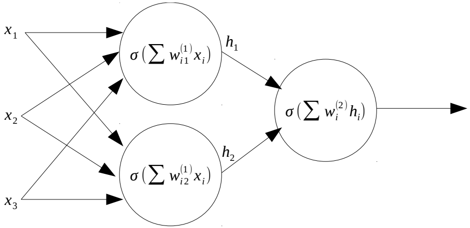
Residual trick



- How to train very very deep networks (hundreds of layers)?
- Add a skip connection to the layers

$$
\begin{aligned}
h^{(0)} &= x \\
h^{(1)} &= \sigma_h(W^{(1)} \cdot h^{(0)} + b^{(1)}) \\
h^{(k)} &= h^{(k-1)} + \sigma_h(W^{(k)} \cdot h^{(k-1)} + b^{(k)}) \\
p(y) &= \sigma_{out}(W^{out} \cdot h^{(K)} + b^{out})
\end{aligned}
$$

Residual trick

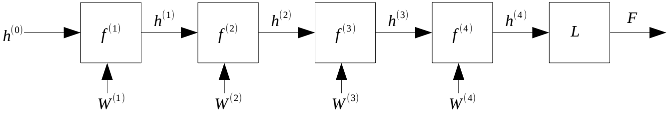


- How to train very very deep networks (hundreds of layers)?
- Add a skip connection to the layers

$$
\begin{aligned}
h^{(0)} &= x \\
h^{(1)} &= \sigma_h(W^{(1)} \cdot h^{(0)} + b^{(1)}) \\
h^{(k)} &= h^{(k-1)} + \sigma_h(W^{(k)} \cdot h^{(k-1)} + b^{(k)}) \\
p(y) &= \sigma_{out}(W^{out} \cdot h^{(K)} + b^{out})
\end{aligned}
$$

- Hidden layers must have the same size

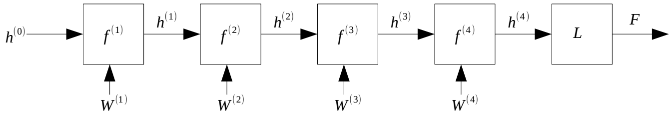
Backpropagation algorithm



- How do we actually compute the gradients of a deep network?

$$h^{(k)} = f^{(k)}(h^{(k-1)}, W^{(k)})$$
$$F = L(h^{(K)})$$

Backpropagation algorithm



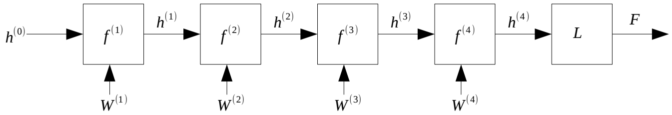
- How do we actually compute the gradients of a deep network?

$$h^{(k)} = f^{(k)}(h^{(k-1)}, W^{(k)})$$
$$F = L(h^{(K)})$$

- Compute the gradient of F w.r.t. the last parameter $W^{(K)}$

$$\frac{\mathrm{d}F}{\mathrm{d}W^{(K)}} = \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}} \cdot \frac{\mathrm{d}h^{(K)}}{\mathrm{d}W^{(K)}}$$
$$\frac{\mathrm{d}h^{(K)}}{\mathrm{d}W^{(K)}} = \frac{\mathrm{d}}{\mathrm{d}W^{(K)}} f^{(K)}(h^{(K-1)}, W^{(K)})$$

Backpropagation algorithm



- How do we actually compute the gradients of a deep network?

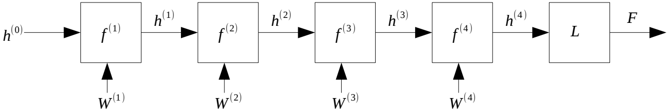
$$h^{(k)} = f^{(k)}(h^{(k-1)}, W^{(k)})$$
$$F = L(h^{(K)})$$

- Compute the gradient of F w.r.t. the last parameter $W^{(K)}$

$$\frac{\mathrm{d}F}{\mathrm{d}W^{(K)}} = \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}} \cdot \frac{\mathrm{d}h^{(K)}}{\mathrm{d}W^{(K)}}$$
$$\frac{\mathrm{d}h^{(K)}}{\mathrm{d}W^{(K)}} = \frac{\mathrm{d}}{\mathrm{d}W^{(K)}} f^{(K)}(h^{(K-1)}, W^{(K)})$$

- Assume that the derviative of any $f^{(k)}$ w.r.t. any of its inputs is efficiently computable given the other input

Backpropagation algorithm



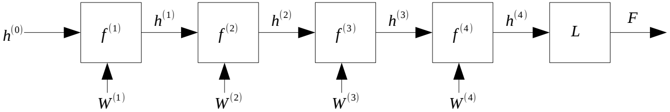
- Compute the gradient of F w.r.t. the second last parameter $W^{(K-1)}$

$$\frac{\mathrm{d}F}{\mathrm{d}W^{(K-1)}} = \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}} \cdot \frac{\mathrm{d}h^{(K)}}{\mathrm{d}h^{(K-1)}} \cdot \frac{\mathrm{d}h^{(K-1)}}{\mathrm{d}W^{(K-1)}}$$

$$\frac{\mathrm{d}h^{(K)}}{\mathrm{d}h^{(K-1)}} = \frac{\mathrm{d}}{\mathrm{d}h^{(K)}} f^{(K)}(h^{(K-1)}, W^{(K)})$$

$$\frac{\mathrm{d}h^{(K-1)}}{\mathrm{d}W^{(K-1)}} = \frac{\mathrm{d}}{\mathrm{d}W^{(K-1)}} f^{(K-1)}(h^{(K-2)}, W^{(K-1)})$$

Backpropagation algorithm



- Compute the gradient of F w.r.t. the second last parameter $W^{(K-1)}$

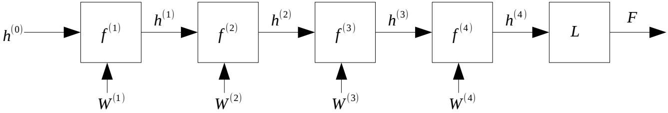
$$\frac{\mathrm{d}F}{\mathrm{d}W^{(K-1)}} = \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}} \cdot \frac{\mathrm{d}h^{(K)}}{\mathrm{d}h^{(K-1)}} \cdot \frac{\mathrm{d}h^{(K-1)}}{\mathrm{d}W^{(K-1)}}$$

$$\frac{\mathrm{d}h^{(K)}}{\mathrm{d}h^{(K-1)}} = \frac{\mathrm{d}}{\mathrm{d}h^{(K)}} f^{(K)}(h^{(K-1)}, W^{(K)})$$

$$\frac{\mathrm{d}h^{(K-1)}}{\mathrm{d}W^{(K-1)}} = \frac{\mathrm{d}}{\mathrm{d}W^{(K-1)}} f^{(K-1)}(h^{(K-2)}, W^{(K-1)})$$

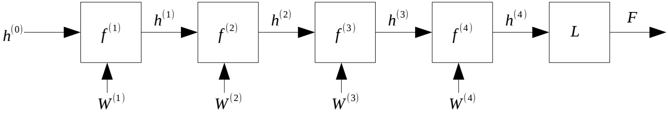
- And so on ...
- Each $\frac{\mathrm{d}F}{\mathrm{d}W^{(k)}}$ can be computed locally from
  - The saved activation: $h^{(k-1)}$
  - The **adjoint**: $\frac{\mathrm{d}L}{\mathrm{d}h^{(K)}} \cdot \frac{\mathrm{d}h^{(K)}}{\mathrm{d}h^{(K-1)}} \cdot \ldots \cdot \frac{\mathrm{d}h^{(k+1)}}{\mathrm{d}h^{(k)}}$

Backpropagation algorithm



- Forward pass:
  - For k in range $[1, K]$
    - Store in memory $h^{(k-1)}$
    - $h^{(k)} := f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Store in memory $h^{(K)}$
  - Return $L(h^{(K)})$

Backpropagation algorithm



- Forward pass:
  - For k in range $[1, K]$
    - Store in memory $h^{(k-1)}$
    - $h^{(k)} := f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Store in memory $h^{(K)}$
  - Return $L(h^{(K)})$

- Backward pass:
  - $h^{(K)}$:= retrive from memory
  - adjoint $:= \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}}$
  - For k in range $[K, 1]$
    - $h^{(k-1)}$:= retrive from memory
    - $\frac{\mathrm{d}F}{\mathrm{d}W^{(k)}} :=$ adjoint $\cdot \frac{\mathrm{d}}{\mathrm{d}W^{(k)}} f^{(k)}(h^{(k-1)}, W^{(k)})$
    - adjoint := adjoint $\cdot \frac{\mathrm{d}}{\mathrm{d}h^{(k-1)}} f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Return $\frac{\mathrm{d}F}{\mathrm{d}W^{(k)}}$ for all k

Backpropagation algorithm

- Reverse-mode automatic differentiation
  - In the general case, the feed-forward neural network is a graph

Backpropagation algorithm

- Reverse-mode automatic differentiation
  - In the general case, the feed-forward neural network is a graph
  - Feed-forward neural networks are always DAGs
  - Recurrent neural networks directed graphs with cycles

Backpropagation algorithm

- Reverse-mode automatic differentiation
  - In the general case, the feed-forward neural network is a graph
  - Feed-forward neural networks are always DAGs
  - Recurrent neural networks directed graphs with cycles
  - For each node, add the **adjoints** coming from all its outgoing edges

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary $\rightarrow$ **NOUN VERB NOUN**
    - One label per word

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary $\rightarrow$ **NOUN VERB NOUN**
    - One label per word
    - E.g. Part-of-speech tagging, Named entity recognition, ...

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. → ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary → **NOUN VERB NOUN**
    - One label per word
    - E.g. Part-of-speech tagging, Named entity recognition, ...
  - Sequence-to-sequence
    The cat sat on the mat → **Die Katze saß auf der Matte**
    - Source and target can be have different length and different vocabularies

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary $\rightarrow$ **NOUN VERB NOUN**
    - One label per word
    - E.g. Part-of-speech tagging, Named entity recognition, ...
  - Sequence-to-sequence
    The cat sat on the mat $\rightarrow$ **Die Katze saß auf der Matte**
    - Source and target can be have different length and different vocabularies
    - E.g. Machine translation, text summarization, dialogue systems, ...

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary $\rightarrow$ **NOUN VERB NOUN**
    - One label per word
    - E.g. Part-of-speech tagging, Named entity recognition, ...
  - Sequence-to-sequence
    The cat sat on the mat $\rightarrow$ **Die Katze saß auf der Matte**
    - Source and target can be have different length and different vocabularies
    - E.g. Machine translation, text summarization, dialogue systems, ...
  - Parsing
    - Use specialized algorithms
    - Can be reduced to sequence-to-sequence

Deep learning for NLP

- Natural language processing tasks
  - Text classification
    The world contains many terrible video game movies. This isn't one of them. $\rightarrow$ ☺
    - One label per sentence
    - E.g. sentiment analysis, topic classification, political polarity classification, ...
  - Text tagging
    John loves Mary $\rightarrow$ **NOUN VERB NOUN**
    - One label per word
    - E.g. Part-of-speech tagging, Named entity recognition, ...
  - Sequence-to-sequence
    The cat sat on the mat $\rightarrow$ **Die Katze saß auf der Matte**
    - Source and target can be have different length and different vocabularies
    - E.g. Machine translation, text summarization, dialogue systems, ...
  - Parsing
    - Use specialized algorithms
    - Can be reduced to sequence-to-sequence
    - Used to be more important in the past

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
  2. Tokenization
  3. Vectorization
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
  2. Tokenization
  3. Vectorization
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
  3. Vectorization
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
  3. Vectorization
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
     - Usually rule-based, ML for some languages without spaces
  3. Vectorization
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
     - Usually rule-based, ML for some languages without spaces
  3. Vectorization
     - Map each token to an integer number in a fixed range
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
     - Usually rule-based, ML for some languages without spaces
  3. Vectorization
     - Map each token to an integer number in a fixed range
     - Each integer corresponds to an **one-hot** vector
  4. Machine learning model
  5. Postprocessing

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
     - Usually rule-based, ML for some languages without spaces
  3. Vectorization
     - Map each token to an integer number in a fixed range
     - Each integer corresponds to an **one-hot** vector
  4. Machine learning model
  5. Postprocessing
     - Task dependent

Deep learning for NLP

- Natural language processing pipeline
  1. Segmentation
     - Divide text into segments (usually sentences)
     - Rule-based heuristics (e.g. recognize newline, ".", and so on)
  2. Tokenization
     - Divide segments into tokens (words)
     - Usually rule-based, ML for some languages without spaces
  3. Vectorization
     - Map each token to an integer number in a fixed range
     - Each integer corresponds to an **one-hot** vector
  4. Machine learning model
  5. Postprocessing
     - Task dependent
     - E.g. devectorization, detokenization, ...

Word vectorization

- Let V be the size of the vocabulary
- Assign to each token type t an id in $[1, V]$

Word vectorization

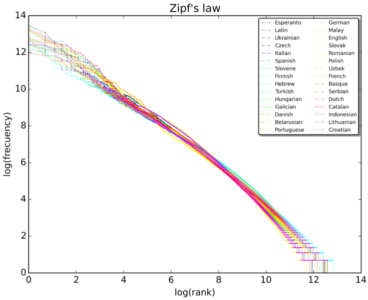
- Let V be the size of the vocabulary
- Assign to each token type t an id in $[1, V]$
  - What is the size of the vocabulary?

Word vectorization

- Let V be the size of the vocabulary
- Assign to each token type t an id in $[1, V]$
  - What is the size of the vocabulary?
  - It is effectively unbounded
    - Names of people, places, companies, ...
    - Number, dates, ...
    - Acronyms, codes, spelling errors, ...

Word vectorization

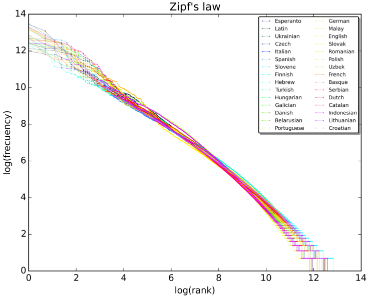
- Let V be the size of the vocabulary
- Assign to each token type t an id in $[1, V]$
  - What is the size of the vocabulary?
  - It is effectively unbounded
    - Names of people, places, companies, ...
    - Number, dates, ...
    - Acronyms, codes, spelling errors, ...
  - A few token types are very common, but most tokens just appear once



From Wikipedia

Word vectorization

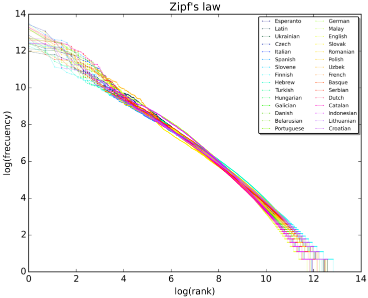
- Either pick a fixed size V (e.g. 10,000 words)



From Wikipedia

Word vectorization

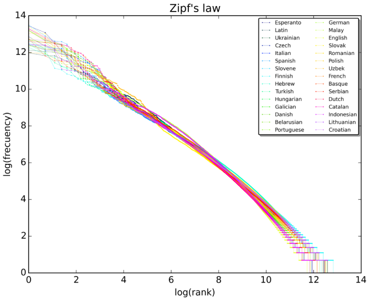
- Either pick a fixed size V (e.g. 10,000 words)
- Or pick a minimum word frequency Q (e.g. 10 times in the training corpus)



From Wikipedia

Word vectorization

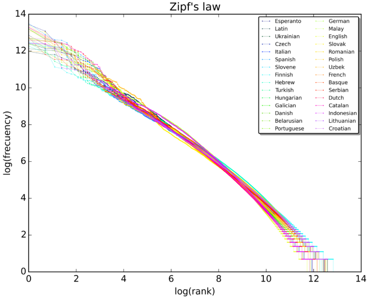
- Either pick a fixed size *V* (e.g. 10,000 words)
- Or pick a minimum word frequency *Q* (e.g. 10 times in the training corpus)
- Replace all other words with a special symbol "**<unk>**"



From Wikipedia

Word vectorization

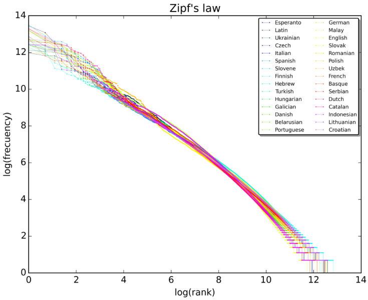
- Input: sequence of M integer token ids $[t_1, t_2, \ldots, t_M]$



From Wikipedia

Word vectorization

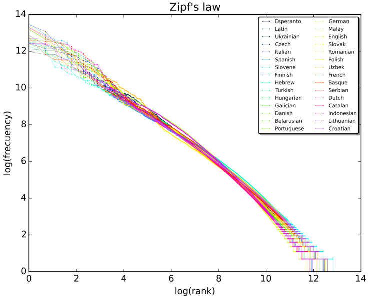
- Input: sequence of M integer token ids $[t_1, t_2, \ldots, t_M]$
- Map to one-hot vectors $e_i \in \{0, 1\}^V$
  $e_{i,j} = 1$ if $t_i = j$, 0 otherwise
- Multiply by an embedding matrix $W \in \mathcal{R}^{d_{emb} \times V}$
  $h_i^{(0)} = W \cdot e_i$



From Wikipedia

Word vectorization

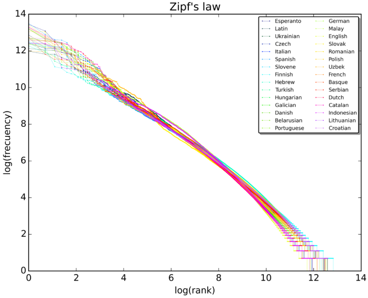
- Input: sequence of M integer token ids $[t_1, t_2, \ldots, t_M]$
- Map to one-hot vectors
  $e_i \in \{0, 1\}^V$
  $e_{i,j} = 1$ if $t_i = j$, 0 otherwise
- Multiply by an embedding matrix
  $W \in \mathcal{R}^{d_{emb} \times V}$
  $h_i^{(0)} = W \cdot e_i$
  - This is equivalent of just selecting the j-th row of W



From Wikipedia

Word vectorization

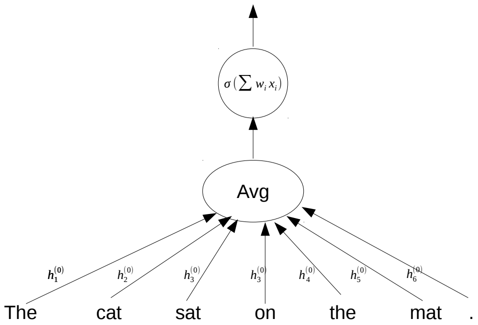
- Input: sequence of M integer token ids $[t_1, t_2, \ldots, t_M]$
- Map to one-hot vectors $e_i \in \{0, 1\}^V$ $e_{i,j} = 1$ if $t_i = j$, 0 otherwise
- Multiply by an embedding matrix $W \in \mathcal{R}^{d_{emb} \times V}$ $h_i^{(0)} = W \cdot e_i$
  - This is equivalent of just selecting the j-th row of W
  - Can be done efficiently with an **Embedding** layer



From Wikipedia

Bag-of-words classification model

- Simple linear model

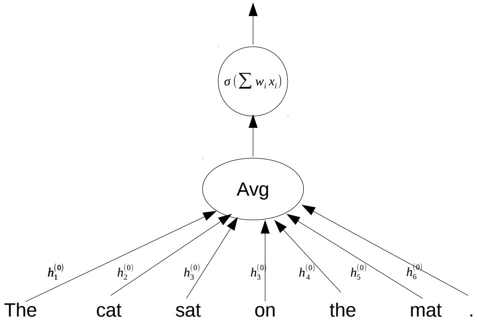


Bag-of-words classification model

- Simple linear model

$$h^{(1)} = \frac{1}{M} \sum_{i=1}^{M} h_i^{(0)}$$

$$p(y) = \text{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$



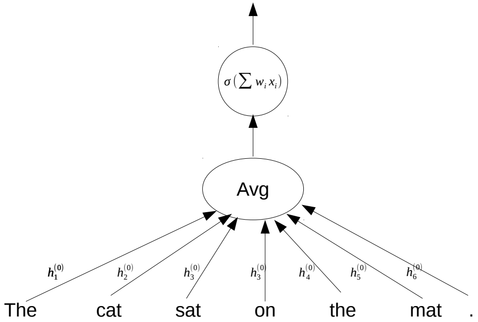
Bag-of-words classification model

- Simple linear model

$$h^{(1)} = \frac{1}{M} \sum_{i=1}^{M} h_i^{(0)}$$

$$p(y) = \text{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

- Often is a strong baseline



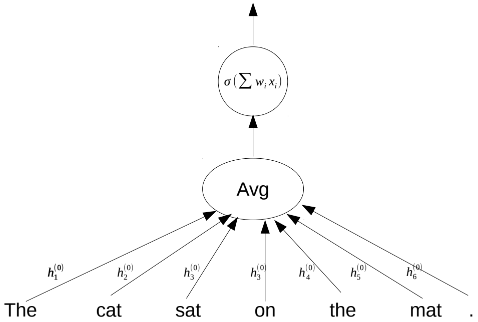
Bag-of-words classification model

- Simple linear model

$$h^{(1)} = \frac{1}{M} \sum_{i=1}^{M} h_i^{(0)}$$

$$p(y) = \mathrm{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

- Often is a strong baseline
- Ignores word order (e.g. "**dog bites man** = **man bites dog**")



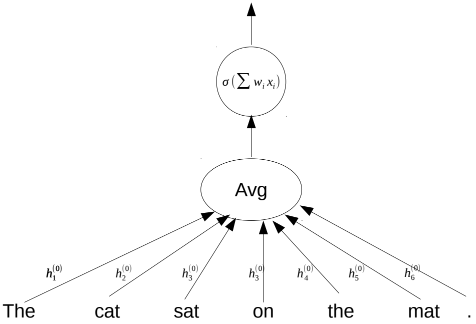
Bag-of-words classification model

- Simple linear model

$$h^{(1)} = \frac{1}{M} \sum_{i=1}^{M} h_i^{(0)}$$

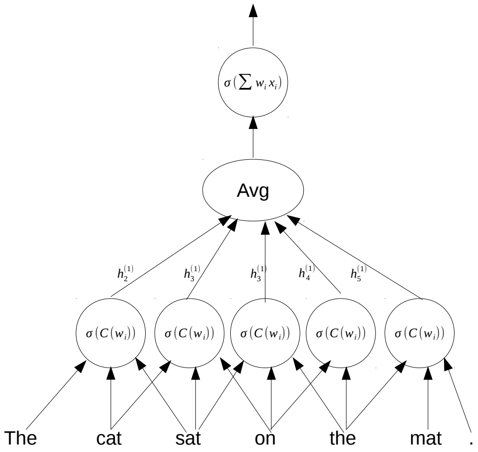
$$p(y) = \text{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

- Often is a strong baseline
- Ignores word order (e.g. "**dog bites man** = **man bites dog**")
- Ignores relations between words



Convolutional classification model

- Deep neural network



Convolutional classification model

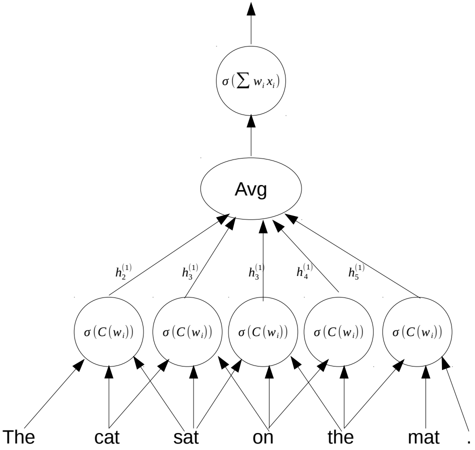
- Deep neural network

$$h_i^{(1)} = \mathrm{ReLU}(\sum_{y=-s}^{s} W_y^{(1)} \cdot h_{y+i}^{(0)} + b^{(1)})$$

$$h^{(2)} = \frac{1}{M'} \sum_{i=1}^{M'} h_i^{(1)}$$

$$p(y) = \mathrm{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

- Kernel size: $2s + 1$ (e.g. 3)



Convolutional classification model

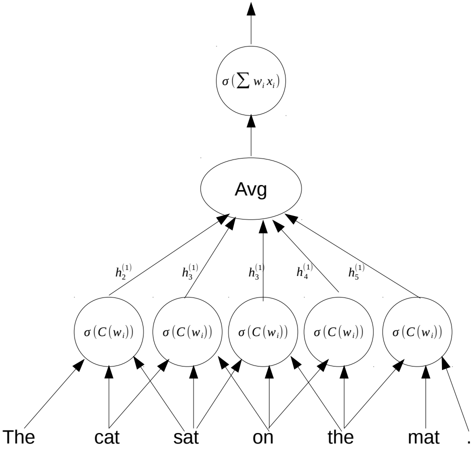
- Deep neural network

$$h_i^{(1)} = \text{ReLU}(\sum_{y=-s}^{s} W_y^{(1)} \cdot h_{y+i}^{(0)} + b^{(1)})$$

$$h^{(2)} = \frac{1}{M'} \sum_{i=1}^{M'} h_i^{(1)}$$

$$p(y) = \text{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

- Kernel size: $2s + 1$ (e.g. 3)
- At each position i the convolution computes a linear layer within a fixed window



Convolutional classification model

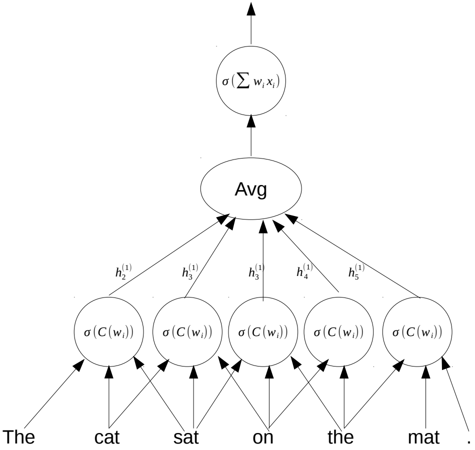
- Deep neural network

$$h_i^{(1)} = \text{ReLU}(\sum_{y=-s}^{s} W_y^{(1)} \cdot h_{y+i}^{(0)} + b^{(1)})$$

$$h^{(2)} = \frac{1}{M'} \sum_{i=1}^{M'} h_i^{(1)}$$

$$p(y) = \text{softmax}(W^{out} \cdot h^{(1)} + b^{out})$$

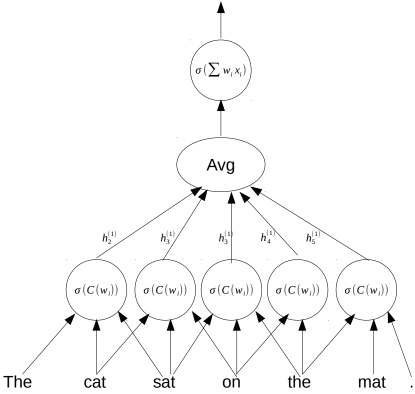
- Kernel size: $2s + 1$ (e.g. 3)
- At each position i the convolution computes a linear layer within a fixed window
- The weight matrix depends on the offset y and is shared over different i



Convolutional classification model

- More concretely, ignoring ReLUs and biases

$$h_2^{(1)} = W_{-1}^{(1)} \cdot \text{"The"} + W_0^{(1)} \cdot \text{"cat"} + W_{+1}^{(1)} \cdot \text{"sat"}$$
$$h_3^{(1)} = W_{-1}^{(1)} \cdot \text{"cat"} + W_0^{(1)} \cdot \text{"sat"} + W_{+1}^{(1)} \cdot \text{"on"}$$
$$h_4^{(1)} = W_{-1}^{(1)} \cdot \text{"sat"} + W_0^{(1)} \cdot \text{"on"} + W_{+1}^{(1)} \cdot \text{"the"}$$
$$\ldots$$

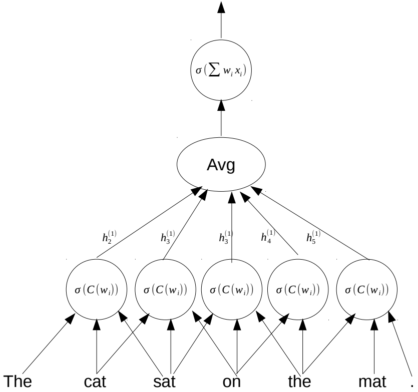


Convolutional classification model

- More concretely, ignoring ReLUs and biases

$$h_2^{(1)} = W_{-1}^{(1)} \cdot \text{"The"} + W_0^{(1)} \cdot \text{"cat"} + W_{+1}^{(1)} \cdot \text{"sat"}$$
$$h_3^{(1)} = W_{-1}^{(1)} \cdot \text{"cat"} + W_0^{(1)} \cdot \text{"sat"} + W_{+1}^{(1)} \cdot \text{"on"}$$
$$h_4^{(1)} = W_{-1}^{(1)} \cdot \text{"sat"} + W_0^{(1)} \cdot \text{"on"} + W_{+1}^{(1)} \cdot \text{"the"}$$
$$\ldots$$

- Captures word n-grams

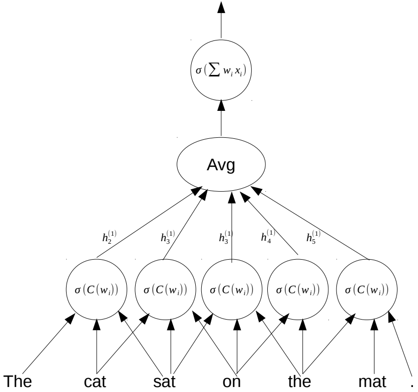


Convolutional classification model

- More concretely, ignoring ReLUs and biases

$$h_2^{(1)} = W_{-1}^{(1)} \cdot \text{"The"} + W_0^{(1)} \cdot \text{"cat"} + W_{+1}^{(1)} \cdot \text{"sat"}$$
$$h_3^{(1)} = W_{-1}^{(1)} \cdot \text{"cat"} + W_0^{(1)} \cdot \text{"sat"} + W_{+1}^{(1)} \cdot \text{"on"}$$
$$h_4^{(1)} = W_{-1}^{(1)} \cdot \text{"sat"} + W_0^{(1)} \cdot \text{"on"} + W_{+1}^{(1)} \cdot \text{"the"}$$
$$\ldots$$

- Captures word n-grams
- We can have multiple layers, of course



Summary

- Supervised machine learning

Summary

- Supervised machine learning
  - Linear classifiers
    - Logistic regression
    - Softmax regression

Summary

- Supervised machine learning
  - Linear classifiers
    - Logistic regression
    - Softmax regression
  - Deep neural networks
    - Fully-connected
    - Convolutional

Summary

- Supervised machine learning
  - Linear classifiers
    - Logistic regression
    - Softmax regression
  - Deep neural networks
    - Fully-connected
    - Convolutional
- Training algorithms
  - Stochastic gradient descent
  - Stochastic gradient descent with momentum
  - Backpropagation to compute gradient

Summary

- Supervised machine learning
  - Linear classifiers
    - Logistic regression
    - Softmax regression
  - Deep neural networks
    - Fully-connected
    - Convolutional
- Training algorithms
  - Stochastic gradient descent
  - Stochastic gradient descent with momentum
  - Backpropagation to compute gradient
- Processing text as input
  - Preprocessing
  - Vectorization

Next part

Next part

- Unsupervised machine learning
  - Language models
  - Unsupervised word embeddings (Glove, FastText)

Next part

- Unsupervised machine learning
  - Language models
  - Unsupervised word embeddings (Glove, FastText)
- Semi-supervised machine learning
  - Fine-tuning
  - Contextual word embeddings (BERT, GPT-2)

Next part

- Unsupervised machine learning
  - Language models
  - Unsupervised word embeddings (Glove, FastText)
- Semi-supervised machine learning
  - Fine-tuning
  - Contextual word embeddings (BERT, GPT-2)
- Model architectures
  - Recurrent neural networks
  - Transformers

Next part

- Unsupervised machine learning
  - Language models
  - Unsupervised word embeddings (Glove, FastText)
- Semi-supervised machine learning
  - Fine-tuning
  - Contextual word embeddings (BERT, GPT-2)
- Model architectures
  - Recurrent neural networks
  - Transformers
- Limitations of deep learning
  - Out-of-domain generalization brittleness
  - Adversarial examples
  - Ethical issues

Resources

- Deep learning frameworks
  - PyTorch `https://pytorch.org/`
  - Tensorflow `https://www.tensorflow.org/`

Resources

- Deep learning frameworks
  - PyTorch `https://pytorch.org/`
  - Tensorflow `https://www.tensorflow.org/`
- NLP toolkits
  - AllenNLP `https://allennlp.org/`
  - Stanford CoreNLP `https://stanfordnlp.github.io/CoreNLP/`
  - NLTK `https://www.nltk.org/`
  - Moses (includes sentence splitters, tokenizers) `https://github.com/moses-smt/mosesdecoder`

Resources

- Deep learning frameworks
  - PyTorch `https://pytorch.org/`
  - Tensorflow `https://www.tensorflow.org/`
- NLP toolkits
  - AllenNLP `https://allennlp.org/`
  - Stanford CoreNLP `https://stanfordnlp.github.io/CoreNLP/`
  - NLTK `https://www.nltk.org/`
  - Moses (includes sentence splitters, tokenizers) `https://github.com/moses-smt/mosesdecoder`
- Repositories
  - awesome-nlp `https://github.com/keon/awesome-nlp`
    - Reading material
    - Tools
    - Datasets

Exam

- Write a paper (2-3 pages) on the course
  - Either summarize all the course or focus on one specific topic

Exam

- Write a paper (2-3 pages) on the course
  - Either summarize all the course or focus on one specific topic
- **OR**
- Write a short proposal (2-3 pages) for a project on Deep Learning for NLP that will be completed at the end of the next part

Exam

- Write a paper (2-3 pages) on the course
  - Either summarize all the course or focus on one specific topic
- **OR**
- Write a short proposal (2-3 pages) for a project on Deep Learning for NLP that will be completed at the end of the next part
- Submit before the start of the next part

Thanks for your attention

Thanks for your attention
Course material: `https://github.com/Avmb/IntroDeepLearning`