

Large Language Models Under the Hood

Part 1: Training Deep Neural Networks

Andrey Kutuzov

University of Oslo

NLDL 2025





Contents

- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities
- 6 Enters Transformer

Tutorial technicalities

Schedule

Schedule

Technicalities

- ▶ Test whether you can access
https://github.com/ltgoslo/nndl_llm_tutorial
- ▶ Find the ip address and the username for your virtual machine (VM) at the spreadsheet below:

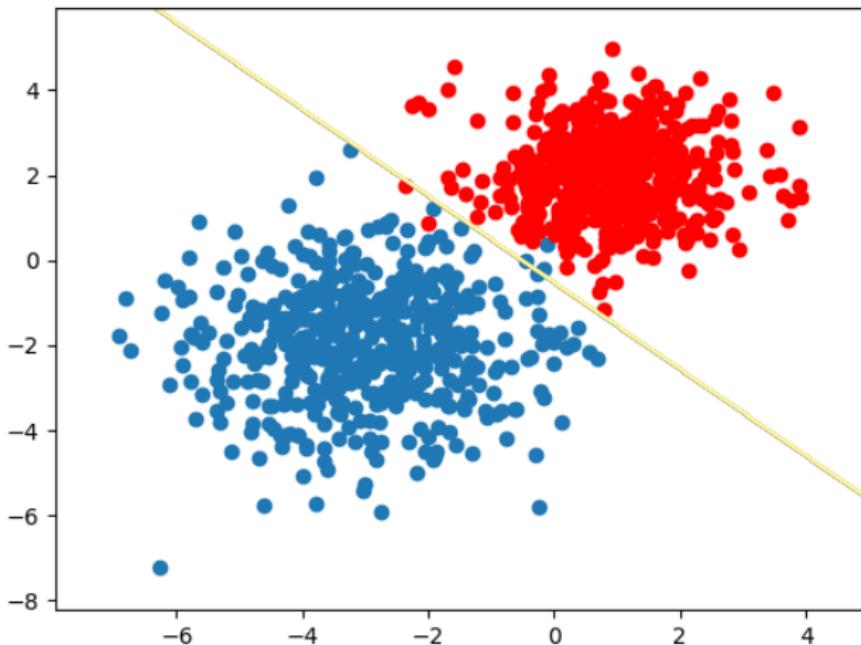
(here goes QR code leading to the spreadsheet)



Contents

- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities
- 6 Enters Transformer

Basics of supervised machine learning



Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$

Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$
 - ▶ for example, e-mail messages.

Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$
 - ▶ for example, e-mail messages.
- ▶ Input 2: corresponding ‘gold’ labels for these instances
 $y_{1:n} = y_1, y_2, \dots, y_n$

Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$
 - ▶ for example, e-mail messages.
- ▶ Input 2: corresponding ‘gold’ labels for these instances
 $y_{1:n} = y_1, y_2, \dots, y_n$
 - ▶ for example, whether the message is spam (1) or not (0).

Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$
 - ▶ for example, e-mail messages.
- ▶ Input 2: corresponding ‘gold’ labels for these instances
 $y_{1:n} = y_1, y_2, \dots, y_n$
 - ▶ for example, whether the message is spam (1) or not (0).
- ▶ The trained models allow to make label predictions for unseen instances.

Basics of supervised machine learning



- ▶ Input 1: a training set of n training instances $x_{1:n} = x_1, x_2, \dots, x_n$
 - ▶ for example, e-mail messages.
- ▶ Input 2: corresponding ‘gold’ labels for these instances
 $y_{1:n} = y_1, y_2, \dots, y_n$
 - ▶ for example, whether the message is spam (1) or not (0).
- ▶ The trained models allow to make label predictions for unseen instances.
- ▶ Generally: some program for mapping instances to labels.

Basics of supervised machine learning

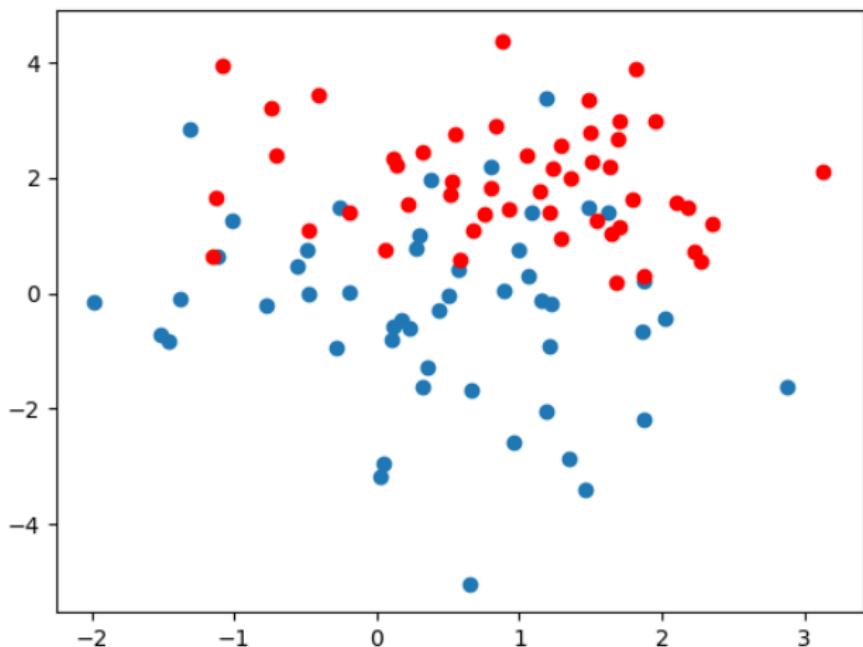


What do you think a good model would look like for this data?

Basics of supervised machine learning



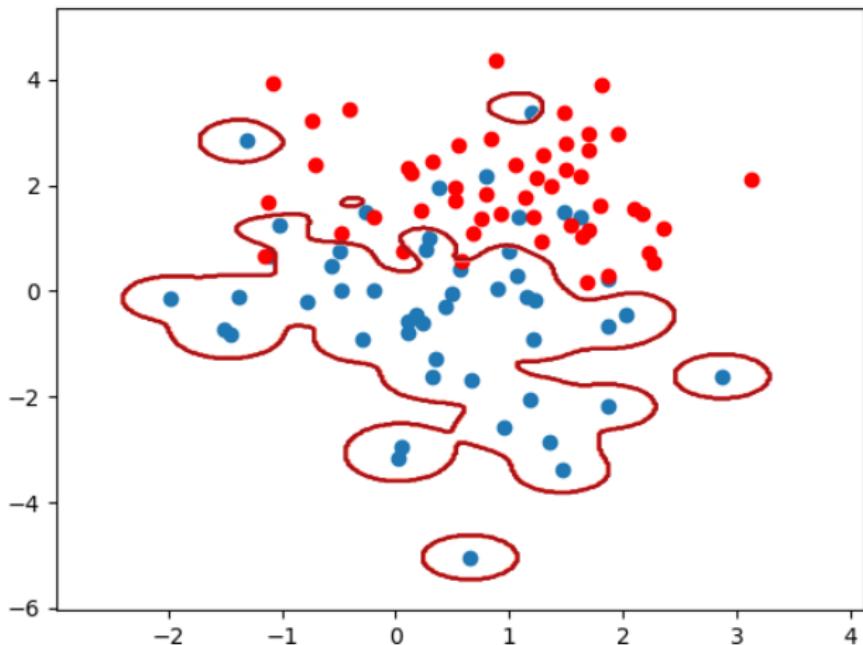
What do you think a good model would look like for this data?



Basics of supervised machine learning



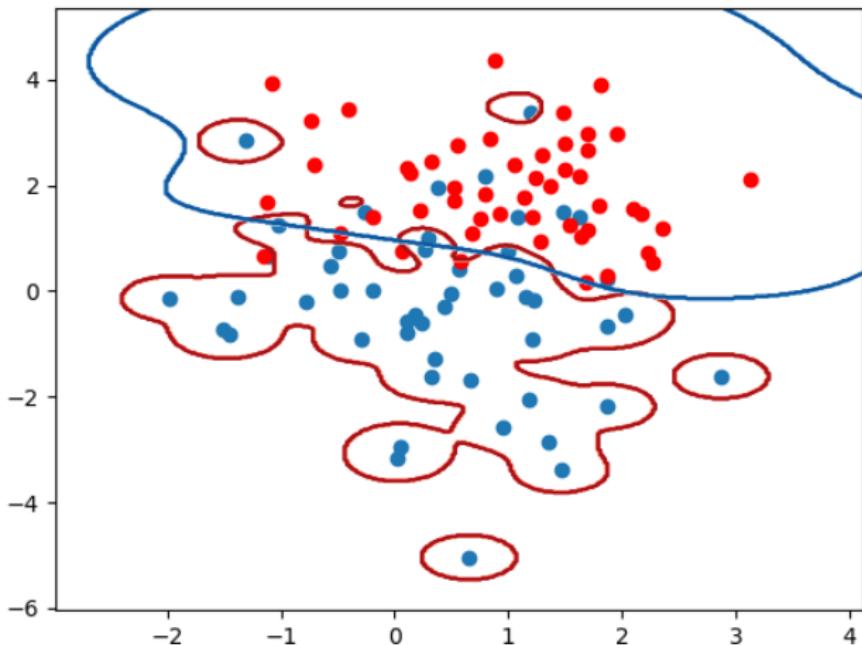
What do you think a good model would look like for this data?



Basics of supervised machine learning



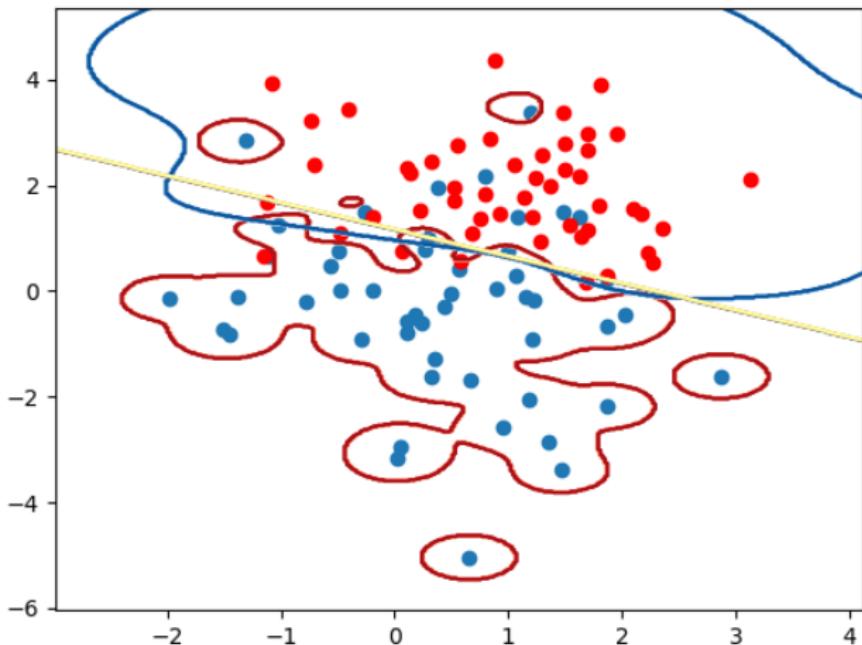
What do you think a good model would look like for this data?



Basics of supervised machine learning



What do you think a good model would look like for this data?





Linear functions: a popular hypothesis class

Simple linear function

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \quad (1)$$

$$\theta = \mathbf{W}, \mathbf{b} \quad (2)$$

► Function input:

- feature vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$;
- each training instance is represented with d_{in} **features**;
- for example, some properties of the documents.

Simple linear function

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b} \quad (1)$$

$$\theta = \mathbf{W}, \mathbf{b} \quad (2)$$

- ▶ Function input:
 - ▶ feature vector $\mathbf{x} \in \mathbb{R}^{d_{in}}$;
 - ▶ each training instance is represented with d_{in} **features**;
 - ▶ for example, some properties of the documents.
- ▶ Function parameters θ :
 - ▶ **matrix** $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$
 - ▶ d_{out} is the dimensionality of the desired prediction (number of classes)
 - ▶ **bias vector** $\mathbf{b} \in \mathbb{R}^{d_{out}}$
 - ▶ bias 'shifts' the function output to some direction.

Training of a linear classifier

$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

$$\theta = \mathbf{W}, b$$

- ▶ Training is finding the optimal θ .

Training of a linear classifier

$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

$$\theta = \mathbf{W}, b$$

- ▶ Training is finding the optimal θ .
- ▶ 'Optimal' means '*producing predictions \hat{y} closest to the gold labels y on our n training instances*'.

Training of a linear classifier

$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

$$\theta = \mathbf{W}, b$$

- ▶ Training is finding the optimal θ .
- ▶ 'Optimal' means '*producing predictions \hat{y} closest to the gold labels y on our n training instances*'.
- ▶ Ideally, $\hat{y} = y$

$$f(\boldsymbol{x}; \boldsymbol{W}, b) = \boldsymbol{x} \cdot \boldsymbol{W} + b$$

Output of **binary** classification

Binary decision ($d_{out} = 1$):

$$f(\boldsymbol{x}; \boldsymbol{W}, b) = \boldsymbol{x} \cdot \boldsymbol{W} + b$$

Output of **binary** classification

Binary decision ($d_{out} = 1$):

- ▶ ‘*Is this message spam or not?*’

$$f(\boldsymbol{x}; \boldsymbol{W}, b) = \boldsymbol{x} \cdot \boldsymbol{W} + b$$

Output of **binary** classification

Binary decision ($d_{out} = 1$):

- ▶ ‘*Is this message spam or not?*’
- ▶ \boldsymbol{W} is a vector, b is a scalar.

$$f(\boldsymbol{x}; \boldsymbol{W}, b) = \boldsymbol{x} \cdot \boldsymbol{W} + b$$

Output of **binary** classification

Binary decision ($d_{out} = 1$):

- ▶ *'Is this message spam or not?'*
- ▶ \boldsymbol{W} is a vector, b is a scalar.
- ▶ The prediction \hat{y} is also a scalar: either 1 ('yes') or -1 ('no').

$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

Output of **binary** classification

Binary decision ($d_{out} = 1$):

- ▶ ‘Is this message spam or not?’
- ▶ \mathbf{W} is a vector, b is a scalar.
- ▶ The prediction \hat{y} is also a scalar: either 1 ('yes') or -1 ('no').
- ▶ NB: the model can output any number, but we convert all negatives to -1 and all positives to 1 (*sign* function).

$$\theta = (\mathbf{W} \in \mathbb{R}^{d_{in}}, b \in \mathbb{R}^1)$$

Basics of supervised machine learning



$$f(x; W, b) = x \cdot W + b$$

$$\begin{matrix} x \\ \begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline 0 \\ \hline 1 \\ \hline 0 \\ \hline \end{array} \\ \cdot \quad \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 1 & 1 & 1 \\ \hline \end{array} \quad + \quad 0.5 \quad = \text{sign}(1.5) = 1 \end{matrix}$$

Basics of supervised machine learning



$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

Output of **multi-class** classification



$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

Output of **multi-class** classification

Multi-class decision ($d_{out} = k$)



$$f(\mathbf{x}; \mathbf{W}, b) = \mathbf{x} \cdot \mathbf{W} + b$$

Output of **multi-class** classification

Multi-class decision ($d_{out} = k$)

- ▶ ‘*Which languages appear in this tweet?*’

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$$

Output of multi-class classification

Multi-class decision ($d_{out} = k$)

- ▶ ‘Which languages appear in this tweet?’
- ▶ \mathbf{W} is a matrix, \mathbf{b} is a vector of k components.

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$$

Output of **multi-class** classification

Multi-class decision ($d_{out} = k$)

- ▶ ‘*Which languages appear in this tweet?*’
- ▶ \mathbf{W} is a matrix, \mathbf{b} is a vector of k components.
- ▶ The prediction $\hat{\mathbf{y}}$ is also a one-hot vector of k components.

$$f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$$

Output of multi-class classification

Multi-class decision ($d_{out} = k$)

- ▶ ‘Which languages appear in this tweet?’
- ▶ \mathbf{W} is a matrix, \mathbf{b} is a vector of k components.
- ▶ The prediction $\hat{\mathbf{y}}$ is also a one-hot vector of k components.
- ▶ The component corresponding to the correct language has the value of 1, others are zeros, for example:
 $\hat{\mathbf{y}} = [0, 0, 1, 0]$ (for $k = 4$)

$$\theta = (\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}, \mathbf{b} \in \mathbb{R}^{d_{out}})$$

Basics of supervised machine learning



$$f(x; W, b) = x \cdot W + b$$

$$\begin{array}{c} x \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\ \cdot \quad W \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \\ b \\ \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\ + \quad \begin{bmatrix} 2 & 2 & 4 \end{bmatrix} \\ = \quad \text{argmax}(2, 2, 4) \\ = \quad 3 \end{array}$$



Log-linear classification

If we care about how confident is the classifier about each decision:

Log-linear classification

If we care about how confident is the classifier about each decision:

- ▶ Map the predictions to the range of $[0, 1] \dots$

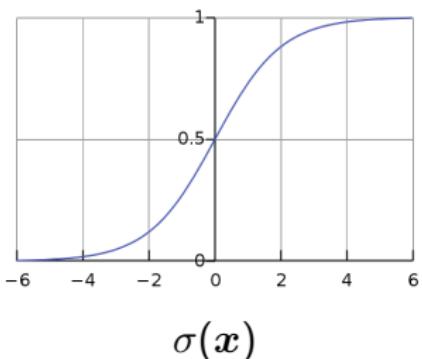
Log-linear classification

If we care about how confident is the classifier about each decision:

- ▶ Map the predictions to the range of $[0, 1] \dots$
- ▶ ...by a **squashing function**, for example, **sigmoid**:

$$\hat{y} = \sigma(f(x)) = \frac{1}{1 + e^{-(f(x))}} \quad (3)$$

- ▶ The result is the **probability** of the prediction!



Basics of supervised machine learning



- ▶ For multi-class cases, log-linear models produce probabilities for all classes, for example:
 $\hat{y} = [0.4, 0.1, 0.9, 0.5]$ (for $k = 4$)

Basics of supervised machine learning

- ▶ For multi-class cases, log-linear models produce probabilities for all classes, for example:

$$\hat{y} = [0.4, 0.1, 0.9, 0.5] \text{ (for } k=4\text{)}$$

- ▶ We choose the **one with the highest score**:

$$\hat{y} = \arg \max_i \hat{y}_{[i]} = \hat{y}_{[2]} \quad (4)$$

Basics of supervised machine learning

- ▶ For multi-class cases, log-linear models produce probabilities for all classes, for example:

$$\hat{y} = [0.4, 0.1, 0.9, 0.5] \text{ (for } k = 4\text{)}$$

- ▶ We choose the **one with the highest score**:

$$\hat{y} = \arg \max_i \hat{y}_{[i]} = \hat{y}_{[2]} \quad (4)$$

- ▶ But often it is more convenient to transform scores into a **probability distribution**, using the **softmax** function:

$$\hat{y} = \text{softmax}(xW + b) \quad (5)$$

$$\hat{y}_{[i]} = \frac{e^{(xW+b)_{[i]}}}{\sum_j e^{(xW+b)_{[j]}}} \quad (6)$$

Basics of supervised machine learning

- ▶ For multi-class cases, log-linear models produce probabilities for all classes, for example:

$$\hat{y} = [0.4, 0.1, 0.9, 0.5] \text{ (for } k=4\text{)}$$

- ▶ We choose the **one with the highest score**:

$$\hat{y} = \arg \max_i \hat{y}_{[i]} = \hat{y}_{[2]} \quad (4)$$

- ▶ But often it is more convenient to transform scores into a **probability distribution**, using the **softmax** function:

$$\hat{y} = \text{softmax}(xW + b) \quad (5)$$

$$\hat{y}_{[i]} = \frac{e^{(xW+b)_{[i]}}}{\sum_j e^{(xW+b)_{[j]}}} \quad (6)$$

- ▶ $\hat{y} = \text{softmax}([0.4, 0.1, 0.9, 0.5]) = [0.22, 0.16, 0.37, 0.25]$
 - ▶ (all scores sum to 1)

Basics of supervised machine learning



$$f(x; W, b) = x \cdot W + b$$

$$\begin{array}{c} x \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \\ \cdot \end{array} \begin{array}{c} W \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \\ + \end{array} \begin{array}{c} b \\ \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \end{array} = \begin{array}{c} \hat{y} \\ \begin{bmatrix} 2 & 2 & 4 \end{bmatrix} \\ \text{softmax} \\ \begin{bmatrix} .1 & .1 & .8 \end{bmatrix} \end{array}$$

(expect this to be used for **language modeling!**)

Basics of supervised machine learning



- The goal of the training is to find the optimal values of parameters in θ .

Basics of supervised machine learning



- ▶ The goal of the training is to find the optimal values of parameters in θ .
- ▶ Formally, it means to minimize the loss $\mathcal{L}(\theta)$ on training or development dataset.

Basics of supervised machine learning



- ▶ The goal of the training is to find the optimal values of parameters in θ .
- ▶ Formally, it means to **minimize the loss** $\mathcal{L}(\theta)$ on training or development dataset.
- ▶ Conceptually, **loss** is a measure of how 'far away' the model predictions \hat{y} are from gold labels y .

Basics of supervised machine learning



- ▶ The goal of the training is to find the optimal values of parameters in θ .
- ▶ Formally, it means to **minimize the loss** $\mathcal{L}(\theta)$ on training or development dataset.
- ▶ Conceptually, **loss** is a measure of how 'far away' the model predictions \hat{y} are from gold labels y .
- ▶ It can be any function $\mathcal{L}(\hat{y}, y)$ returning a scalar value:
 - ▶ for example, $\mathcal{L} = (y - \hat{y})^2$ (**square error**)

Basics of supervised machine learning



- ▶ The goal of the training is to find the optimal values of parameters in θ .
- ▶ Formally, it means to **minimize the loss** $\mathcal{L}(\theta)$ on training or development dataset.
- ▶ Conceptually, **loss** is a measure of how ‘far away’ the model predictions \hat{y} are from gold labels y .
- ▶ It can be any function $\mathcal{L}(\hat{y}, y)$ returning a scalar value:
 - ▶ for example, $\mathcal{L} = (y - \hat{y})^2$ (**square error**)
- ▶ It is averaged over all training instances and gives us estimation of the model ‘fitness’.

Basics of supervised machine learning



- ▶ The goal of the training is to find the optimal values of parameters in θ .
- ▶ Formally, it means to **minimize the loss** $\mathcal{L}(\theta)$ on training or development dataset.
- ▶ Conceptually, **loss** is a measure of how ‘far away’ the model predictions \hat{y} are from gold labels y .
- ▶ It can be any function $\mathcal{L}(\hat{y}, y)$ returning a scalar value:
 - ▶ for example, $\mathcal{L} = (y - \hat{y})^2$ (**square error**)
- ▶ It is averaged over all training instances and gives us estimation of the model ‘fitness’.
- ▶ $\hat{\theta}$ is the best set of parameters:

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\theta) \quad (7)$$



How do you choose a loss function?

How do you choose a loss function?

1. It depends on your task...

How do you choose a loss function?

1. It depends on your task...

- ▶ regression

How do you choose a loss function?

1. It depends on your task...

- ▶ **regression** - mean absolute error, mean squared error, ...

How do you choose a loss function?

1. It depends on your task...

- ▶ regression - mean absolute error, mean squared error, ...
- ▶ classification

How do you choose a loss function?

1. It depends on your task...

- ▶ **regression** - mean absolute error, mean squared error, ...
- ▶ **classification** - hinge-loss, cross-entropy, ...



How do you choose a loss function?

1. It depends on your task...

- ▶ **regression** - mean absolute error, mean squared error, ...
- ▶ **classification** - hinge-loss, cross-entropy, ...
- ▶ **ranking**

How do you choose a loss function?

1. It depends on your task...

- ▶ **regression** - mean absolute error, mean squared error, ...
- ▶ **classification** - hinge-loss, cross-entropy, ...
- ▶ **ranking** - ranking loss, triplet loss

How do you choose a loss function?

1. It depends on your task...
 - ▶ **regression** - mean absolute error, mean squared error, ...
 - ▶ **classification** - hinge-loss, cross-entropy, ...
 - ▶ **ranking** - ranking loss, triplet loss
2. A mix of **theoretical** and **practical** desires often determine your final choice
 - ▶ For classification, we often use some variant of...

How do you choose a loss function?

1. It depends on your task...
 - ▶ regression - mean absolute error, mean squared error, ...
 - ▶ classification - hinge-loss, cross-entropy, ...
 - ▶ ranking - ranking loss, triplet loss
2. A mix of **theoretical** and **practical** desires often determine your final choice
 - ▶ For classification, we often use some variant of...
 - ▶ ... hinge-loss (**max margin**)

How do you choose a loss function?

1. It depends on your task...
 - ▶ regression - mean absolute error, mean squared error, ...
 - ▶ classification - hinge-loss, cross-entropy, ...
 - ▶ ranking - ranking loss, triplet loss
2. A mix of **theoretical** and **practical** desires often determine your final choice
 - ▶ For classification, we often use some variant of...
 - ▶ ... hinge-loss (max margin)
 - ▶ ... cross-entropy loss
 - ▶ ...and language modeling is secretly classification!



Now we can measure model performance. How can we change our parameters θ to improve?

Optimizing with gradient

- $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.

Optimizing with gradient

- $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- Commonly solved using **gradient** methods:
 1. compute the loss,

Optimizing with gradient

- $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- Commonly solved using **gradient** methods:
 1. compute the loss,
 2. compute gradient of θ parameters with respect to the loss,

Optimizing with gradient

- $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- Commonly solved using **gradient** methods:
 1. compute the loss,
 2. compute gradient of θ parameters with respect to the loss,
 3. (*gradient here is the collection of partial derivatives, one for each parameter of θ*)

Optimizing with gradient

- ▶ $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- ▶ Commonly solved using **gradient** methods:
 1. compute the loss,
 2. compute gradient of θ parameters with respect to the loss,
 3. (*gradient here is the collection of partial derivatives, one for each parameter of θ*)
 4. move the parameters in the opposite direction (to decrease the loss),

Optimizing with gradient

- $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- Commonly solved using **gradient** methods:
 1. compute the loss,
 2. compute gradient of θ parameters with respect to the loss,
 3. (*gradient here is the collection of partial derivatives, one for each parameter of θ*)
 4. move the parameters in the opposite direction (to decrease the loss),
 5. repeat until the optimum is found (the derivative is 0) or until the pre-defined number of iterations (epochs) is achieved.

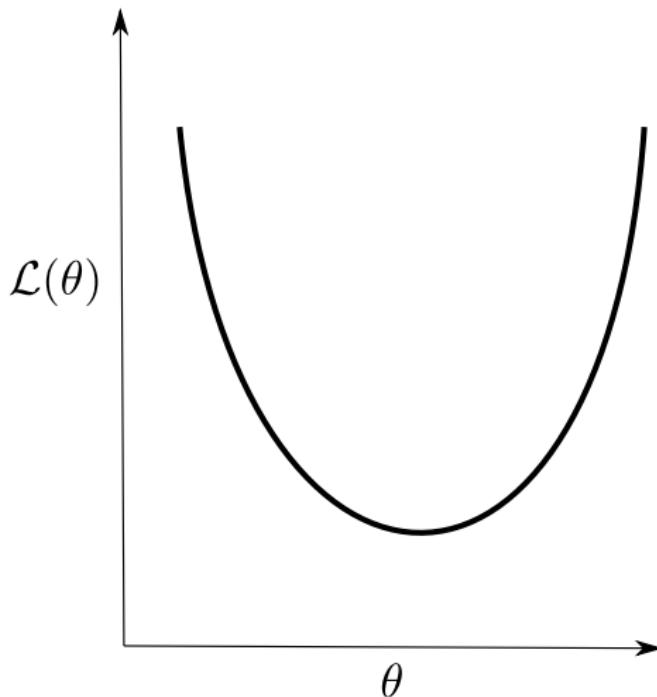
Optimizing with gradient

- ▶ $\hat{\theta} = \arg \min_{\theta} (\mathcal{L}(\theta) + \lambda R(\theta))$ is an **optimization problem**.
- ▶ Commonly solved using **gradient** methods:
 1. compute the loss,
 2. compute gradient of θ parameters with respect to the loss,
 3. (*gradient here is the collection of partial derivatives, one for each parameter of θ*)
 4. move the parameters in the opposite direction (to decrease the loss),
 5. repeat until the optimum is found (the derivative is 0) or until the pre-defined number of iterations (epochs) is achieved.

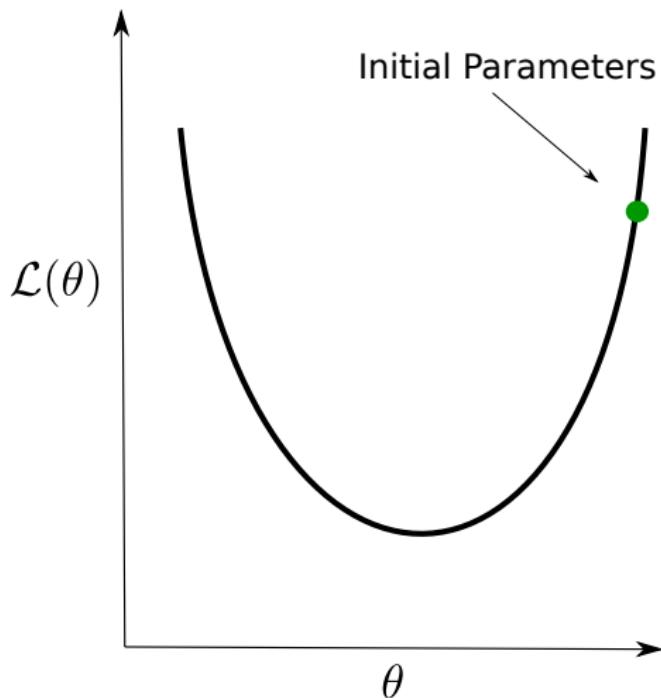
Convexity

- ▶ **Convex functions:** a single optimum point.
- ▶ **Non-convex functions:** multiple optimum points.

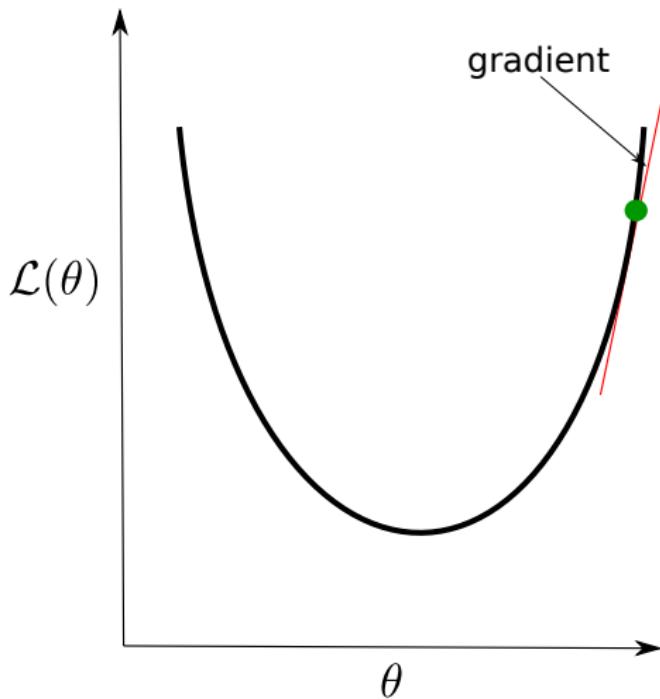
Basics of supervised machine learning



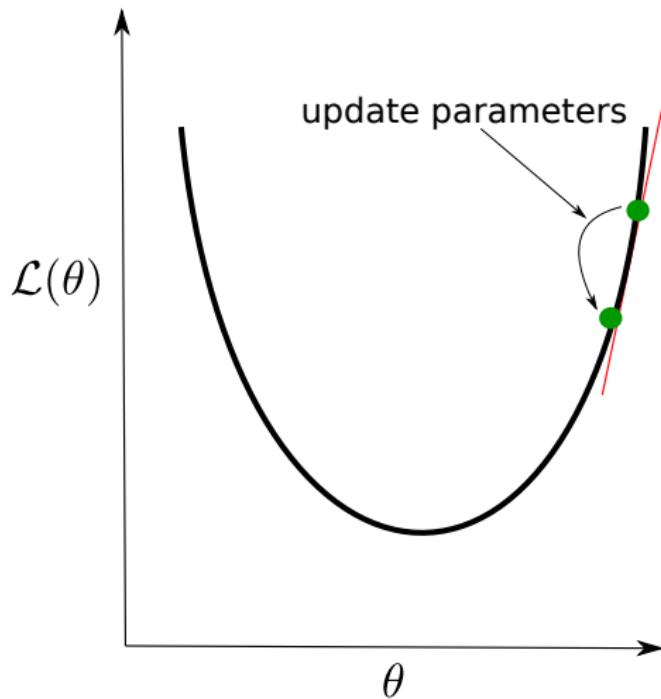
Basics of supervised machine learning



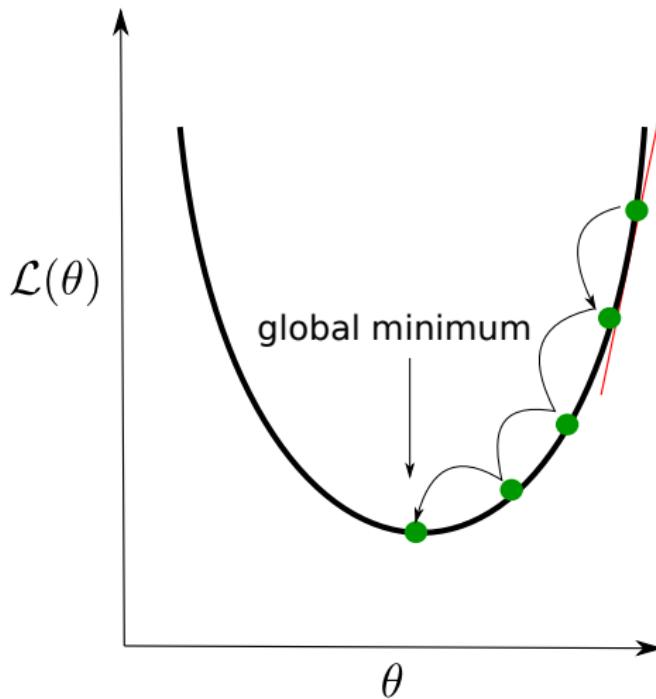
Basics of supervised machine learning



Basics of supervised machine learning



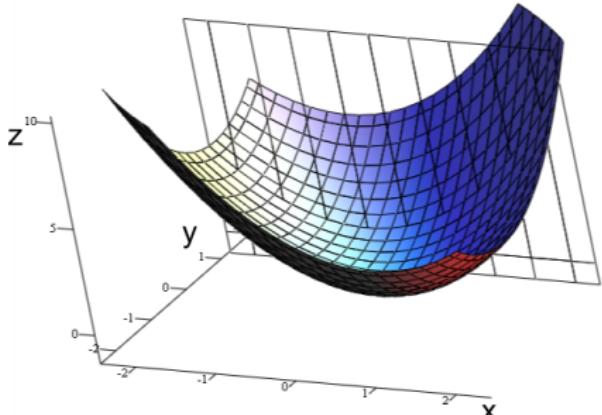
Basics of supervised machine learning



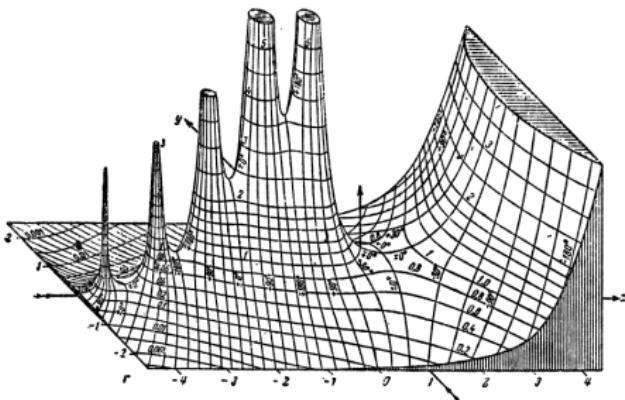
Basics of supervised machine learning



Error surfaces of convex and not-convex functions:

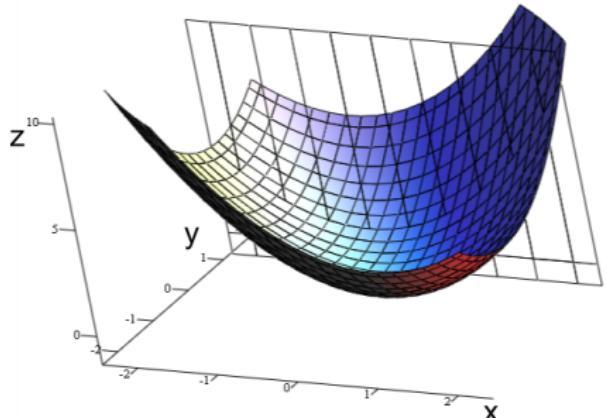


Convex function



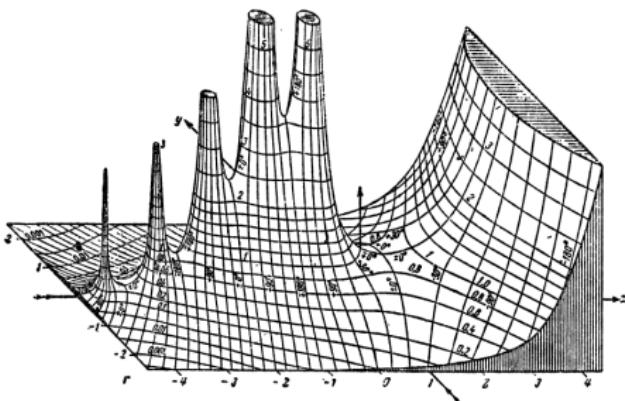
Non-convex function

Error surfaces of convex and not-convex functions:



Convex function

- Convex functions can be easily minimized with gradient methods, reaching the global optimum.
- With non-convex functions, optimization can end up in a local optimum.

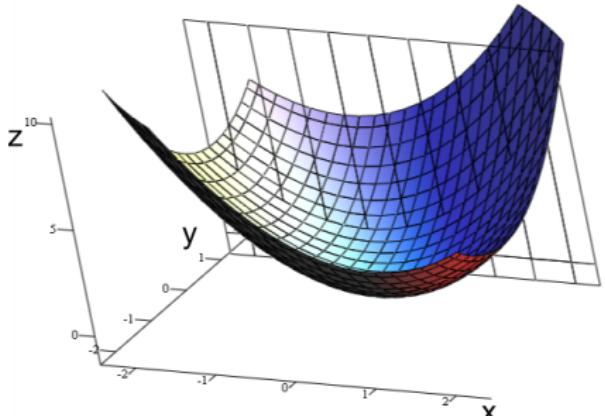


Non-convex function

Basics of supervised machine learning

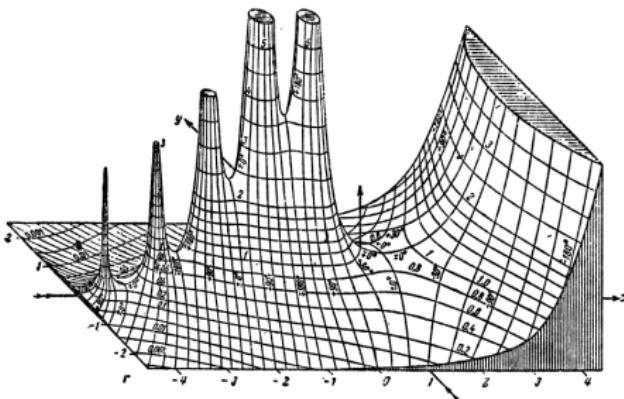


Error surfaces of convex and not-convex functions:



Convex function

- Convex functions can be easily minimized with gradient methods, reaching the global optimum.
- With non-convex functions, optimization can end up in a local optimum.
- Linear and log-linear models as a rule have convex error functions.
- ...unlike non-linear models (e.g., used in 'deep learning').



Non-convex function



Contents

- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities
- 6 Enters Transformer

Advantages and limitations of linear models



- Linear models are efficient and effective.

Advantages and limitations of linear models



- ▶ Linear models are efficient and effective.
- ▶ They are interpretable to a degree: you can find the most important features by looking at the weights

Advantages and limitations of linear models



- ▶ Linear models are efficient and effective.
- ▶ They are interpretable to a degree: you can find the most important features by looking at the weights
- ▶ Can be used on their own (often enough in practice)...
- ▶ ...or as building blocks for non-linear neural classifiers.

Advantages and limitations of linear models



- ▶ Linear models are efficient and effective.
- ▶ They are interpretable to a degree: you can find the most important features by looking at the weights
- ▶ Can be used on their own (often enough in practice)...
- ▶ ...or as building blocks for non-linear neural classifiers.

Unfortunately, linear models can represent only **linear relations** in the data

Advantages and limitations of linear models



- Are there **non-linear functions** that linear models can't deal with?

Advantages and limitations of linear models

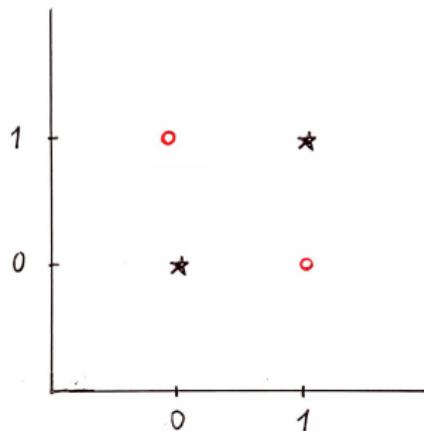


- ▶ Are there **non-linear functions** that linear models can't deal with?
- ▶ Yes, there are.

Advantages and limitations of linear models



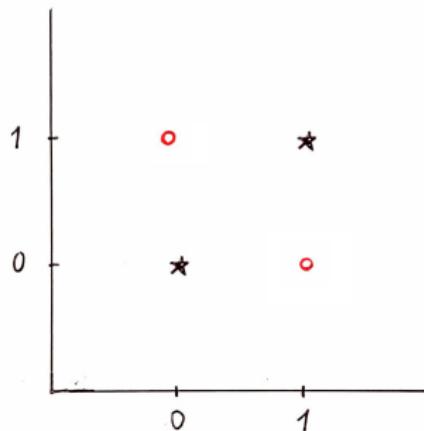
- Are there **non-linear functions** that linear models can't deal with?
- Yes, there are.
- One example is the **XOR** ('excluding OR') function:



Advantages and limitations of linear models



- Are there **non-linear functions** that linear models can't deal with?
- Yes, there are.
- One example is the **XOR** ('excluding OR') function:



It is clearly **not** linearly separable.



Possible solutions

- We can **transform the input** so that it becomes linearly separable.

Possible solutions

- We can **transform the input** so that it becomes linearly separable.
- Linear transformations will not be able to do this.

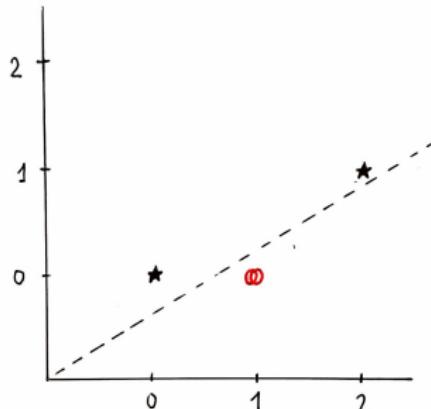
Possible solutions

- We can **transform the input** so that it becomes linearly separable.
- Linear transformations will not be able to do this.
- We need **non-linear transformations**.

Possible solutions

- We can transform the input so that it becomes linearly separable.
- Linear transformations will not be able to do this.
- We need non-linear transformations.

For example, $\phi(x_1, x_2) = [x_1 + x_2, x_1 \times x_2]$ maps the instances to another representation and makes the XOR problem linearly separable:



Training mapping functions

- But how to find the **transformation function** ϕ suitable for the task at hand?

Training mapping functions

- ▶ But how to find the **transformation function** ϕ suitable for the task at hand?
- ▶ Idea: leave it for the algorithm to **train a suitable representation mapping function!**

Training mapping functions

- ▶ But how to find the **transformation function** ϕ suitable for the task at hand?
- ▶ Idea: leave it for the algorithm to **train a suitable representation mapping function!**

$$\phi(\mathbf{x}) = g(\mathbf{x}\mathbf{W}' + \mathbf{b}') \quad (8)$$

$$\hat{\mathbf{y}} = \phi(\mathbf{x})\mathbf{W} + \mathbf{b} \quad (9)$$

- ▶ ...where g is a non-linear **activation function**, and \mathbf{W}', \mathbf{b}' are its trainable parameters.

Training mapping functions

- ▶ But how to find the **transformation function** ϕ suitable for the task at hand?
- ▶ Idea: leave it for the algorithm to **train a suitable representation mapping function!**

$$\phi(\mathbf{x}) = g(\mathbf{x}\mathbf{W}' + \mathbf{b}') \quad (8)$$

$$\hat{\mathbf{y}} = \phi(\mathbf{x})\mathbf{W} + \mathbf{b} \quad (9)$$

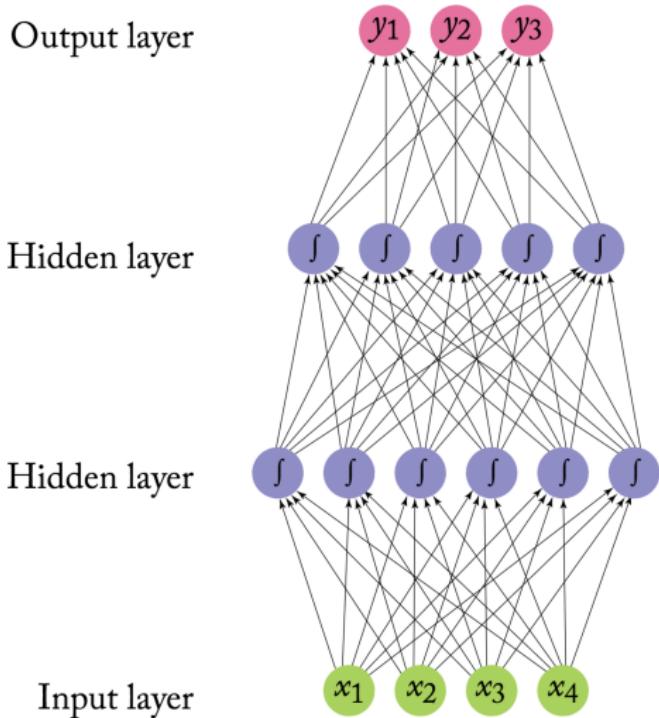
- ▶ ...where g is a non-linear **activation function**, and \mathbf{W}', \mathbf{b}' are its trainable parameters.
- ▶ The equation above defines a simple **multi-layer perceptron (MLP)**: a **neural model**.



Contents

- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities
- 6 Enters Transformer

Going deeply non-linear: multi-layered perceptrons



A simple **feed-forward neural network** (perceptron) with 2 hidden layers.



The nature of perceptrons

- ▶ Input data **goes through successive transformations** at each layer.



The nature of perceptrons

- ▶ Input data goes through successive transformations at each layer.
- ▶ The transformations are linear, but followed with a non-linear activation at each hidden layer.

The nature of perceptrons

- ▶ Input data goes through successive transformations at each layer.
- ▶ The transformations are linear, but followed with a non-linear activation at each hidden layer.
- ▶ At the last layer, the prediction \hat{y} is produced.

The nature of perceptrons

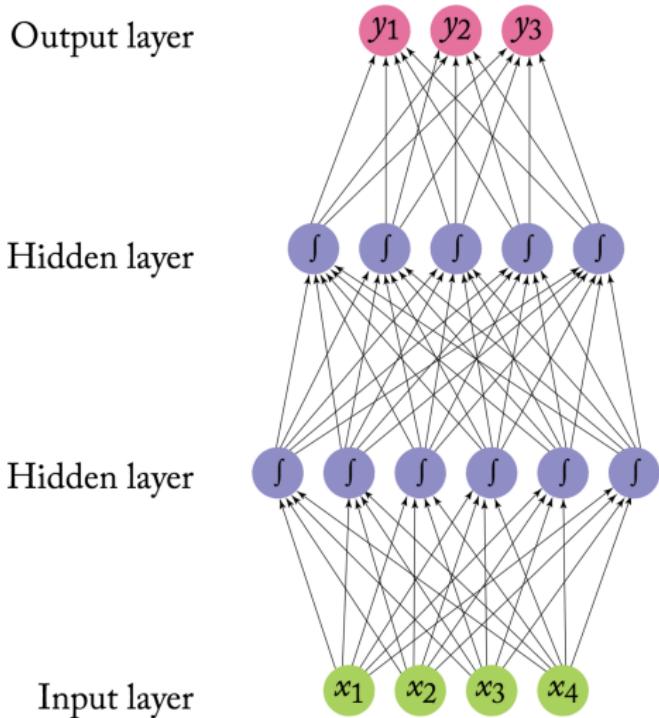
- ▶ Input data goes through successive transformations at each layer.
- ▶ The transformations are linear, but followed with a non-linear activation at each hidden layer.
- ▶ At the last layer, the prediction \hat{y} is produced.
- ▶ Representation functions and the linear classifiers are trained simultaneously.

The nature of perceptrons

- ▶ Input data goes through successive transformations at each layer.
- ▶ The transformations are linear, but followed with a non-linear activation at each hidden layer.
- ▶ At the last layer, the prediction \hat{y} is produced.
- ▶ Representation functions and the linear classifiers are trained simultaneously.

Important: **neural networks with hidden layers can theoretically approximate any function** [Leshno et al., 1993].

Going deeply non-linear: multi-layered perceptrons



A simple **feed-forward neural network** (perceptron) with 2 hidden layers.

'Machines of this character can
behave in a very complicated
manner when the number of units
is large.'

Alan Turing, 'Intelligent Machinery'

[Turing, 1948]



Going deeply non-linear: multi-layered perceptrons



Brain metaphor

- ▶ Why 'artificial **neural** networks'?



Brain metaphor

- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:



Going deeply non-linear: multi-layered perceptrons



Brain metaphor

- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,

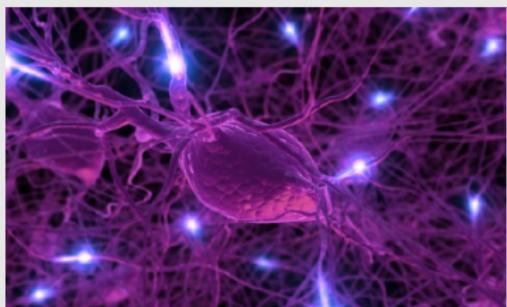


Going deeply non-linear: multi-layered perceptrons



Brain metaphor

- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,



Brain metaphor

- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,
 - ▶ processes (weighs) them differently, depending on their source,



Going deeply non-linear: multi-layered perceptrons



Brain metaphor

- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,
 - ▶ processes (weighs) them differently, depending on their source,
 - ▶ and then passes own electrical signals (**outputs**) to other neurons;



Going deeply non-linear: multi-layered perceptrons



Brain metaphor



- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,
 - ▶ processes (weighs) them differently, depending on their source,
 - ▶ and then passes own electrical signals (**outputs**) to other neurons;
 - ▶ depending on input signals, a neuron can be more or less **activated**,

Going deeply non-linear: multi-layered perceptrons



Brain metaphor



- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,
 - ▶ processes (weighs) them differently, depending on their source,
 - ▶ and then passes own electrical signals (**outputs**) to other neurons;
 - ▶ depending on input signals, a neuron can be more or less **activated**,
 - ▶ ... or completely relaxed ('silent');

Brain metaphor



- ▶ Why 'artificial **neural** networks'?
- ▶ ...because it seems **neurons in human brain act similarly**:
 - ▶ Connected into a large network,
 - ▶ each neuron accepts electrical signals (**inputs**) from other neurons,
 - ▶ processes (weighs) them differently, depending on their source,
 - ▶ and then passes own electrical signals (**outputs**) to other neurons;
 - ▶ depending on input signals, a neuron can be more or less **activated**,
 - ▶ ... or completely relaxed ('silent');
 - ▶ the whole system is **distributed** across many neurons.

...But brain metaphor is not optimal



- ▶ Although the first neural networks were based loosely on knowledge of neural activation at the time, we have long abandoned any real connection to neuroscience.

...But brain metaphor is not optimal



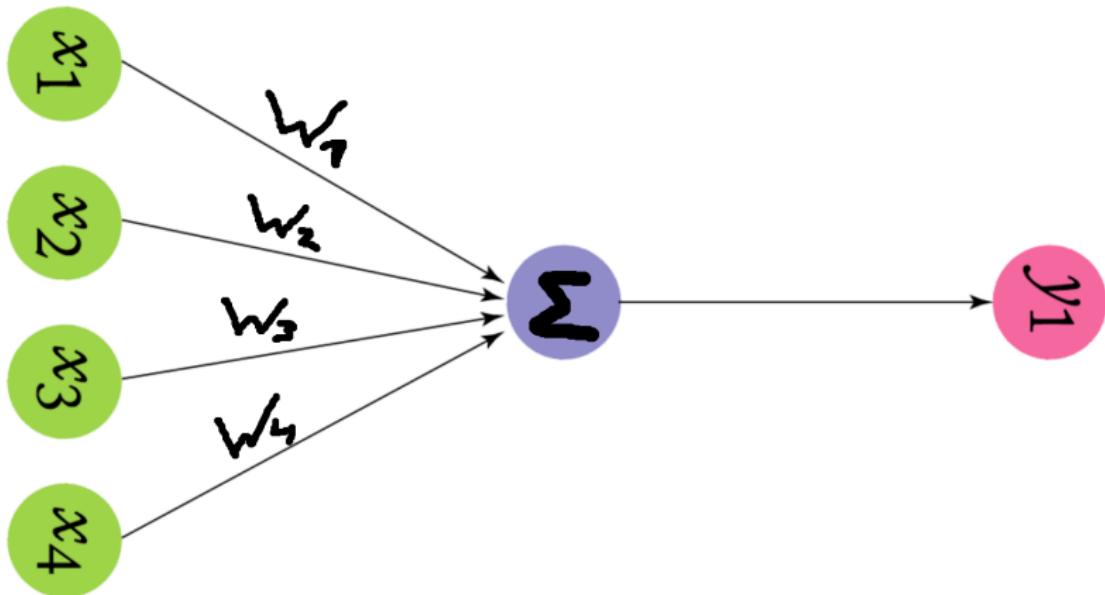
- ▶ Although the first neural networks were based loosely on knowledge of neural activation at the time, we have long abandoned any real connection to neuroscience.
- ▶ Thus, it's **less cool but more convenient** to think about artificial neural networks **in terms of linear algebra concepts**:

...But brain metaphor is not optimal

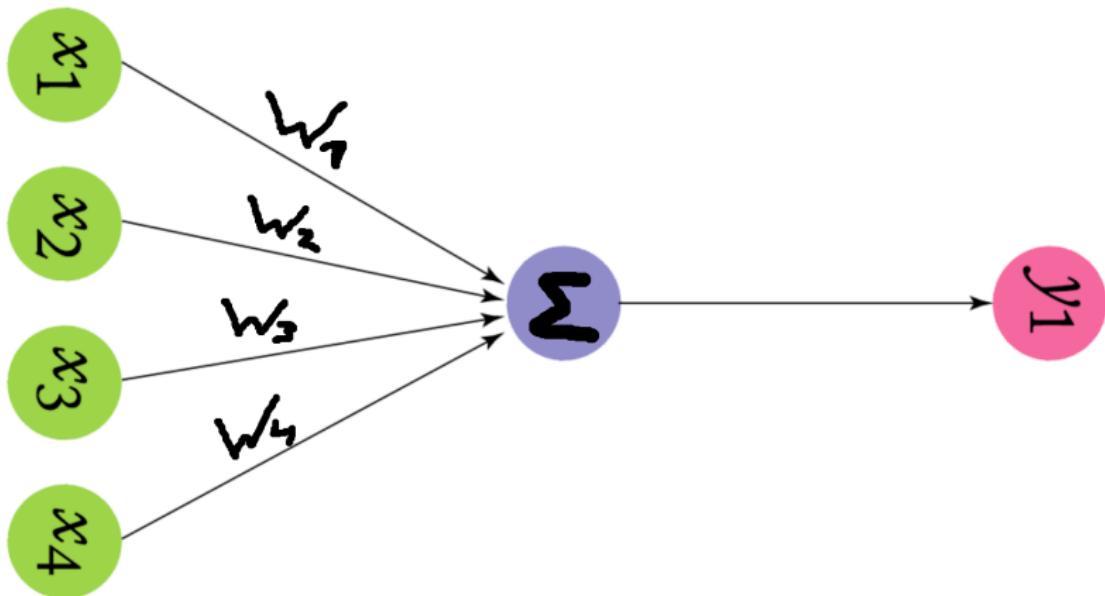


- ▶ Although the first neural networks were based loosely on knowledge of neural activation at the time, we have long abandoned any real connection to neuroscience.
- ▶ Thus, it's **less cool but more convenient** to think about artificial neural networks **in terms of linear algebra concepts**:
 - ▶ vectors,
 - ▶ matrices,
 - ▶ sequential algebraic operations on them.

Going deeply non-linear: multi-layered perceptrons

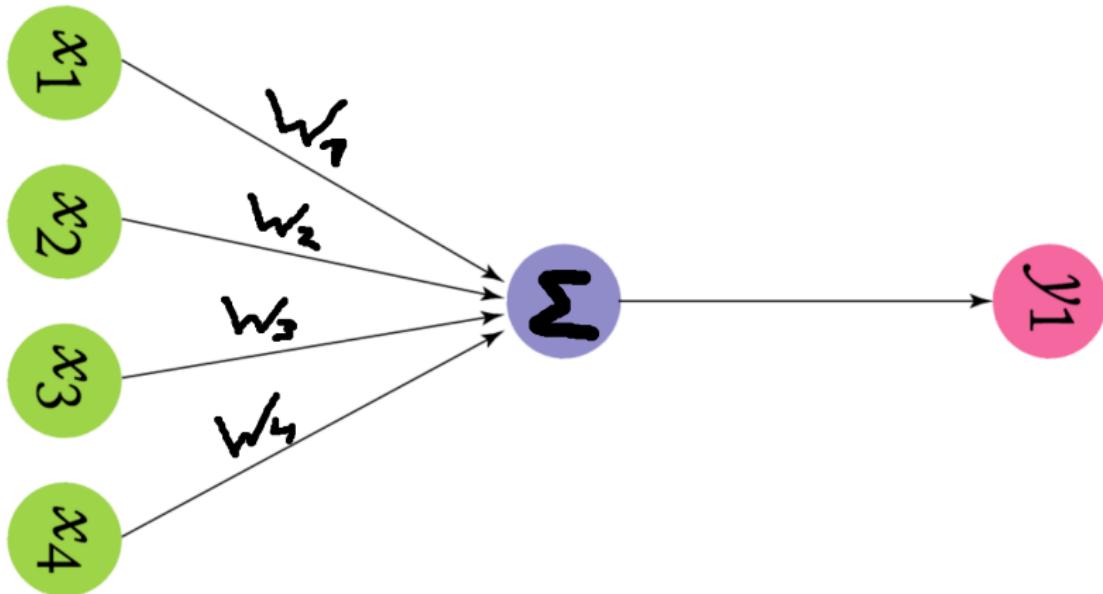


Going deeply non-linear: multi-layered perceptrons



- ▶ In fact this is already a **simple neural network**...
- ▶ with **one neuron Σ** as a computational unit.

Going deeply non-linear: multi-layered perceptrons

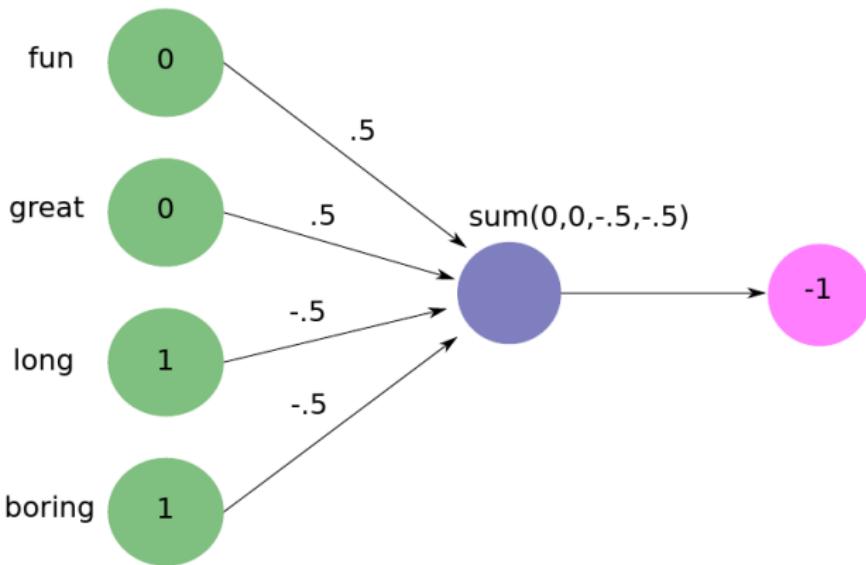


- ▶ In fact this is already a **simple neural network**...
- ▶ with **one neuron Σ as a computational unit**.
- ▶ Σ takes **4 input values** and returns their weighted sum as **output value**.

Going deeply non-linear: multi-layered perceptrons



"It was long and a bit boring"



Sentiment analysis with only one neuron and binary bag-of-words.



Stack of linear classifiers

We can **stack classifiers...**



Stack of linear classifiers

We can **stack classifiers...**

- ▶ by feeding the prediction of one classifier to another one:



Stack of linear classifiers

We can **stack classifiers...**

- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$

Stack of linear classifiers

We can **stack classifiers...**

- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$
- ▶ $\hat{y} = ((xW^0)W^1)W^2$



Stack of linear classifiers

We can **stack classifiers...**

- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$
- ▶ $\hat{y} = ((xW^0)W^1)W^2$
- ▶ ...etc. Can be arbitrarily **deep**

Stack of linear classifiers

We can **stack classifiers...**

- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$
- ▶ $\hat{y} = ((xW^0)W^1)W^2$
- ▶ ...etc. Can be arbitrarily **deep**
- ▶ each subsequent row of neurons (**layer**) is a vector;

Stack of linear classifiers

We can **stack classifiers**...

- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$
- ▶ $\hat{y} = ((xW^0)W^1)W^2$
- ▶ ...etc. Can be arbitrarily **deep**
- ▶ each subsequent row of neurons (**layer**) is a vector;
- ▶ it is multiplied by the next weight matrix (shapes must match).

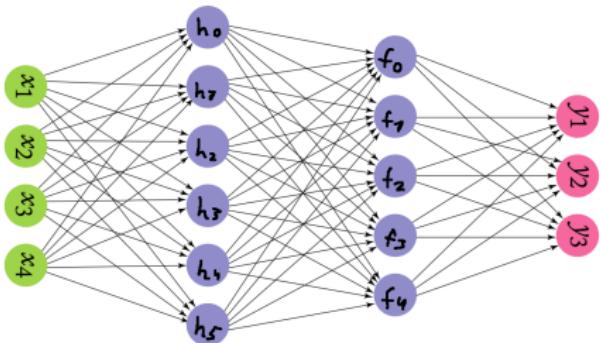
Stack of linear classifiers

We can **stack classifiers**...

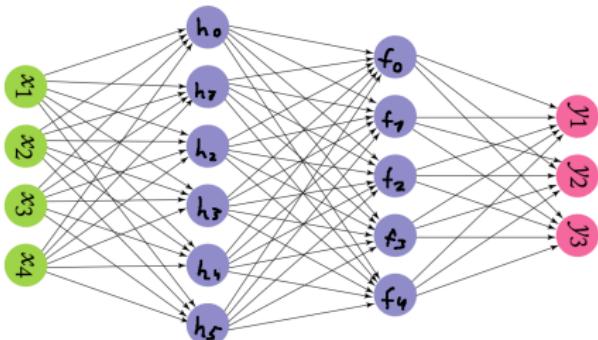
- ▶ by feeding the prediction of one classifier to another one:
- ▶ $\hat{y} = (xW^0)W^1$
- ▶ $\hat{y} = ((xW^0)W^1)W^2$
- ▶ ...etc. Can be arbitrarily **deep**
- ▶ each subsequent row of neurons (**layer**) is a vector;
- ▶ it is multiplied by the next weight matrix (shapes must match).

Can even use **both** multiple neurons and multiple layers.

Going deeply non-linear: multi-layered perceptrons



Going deeply non-linear: multi-layered perceptrons



Looks much like a 'deep neural network'

Stacked linear classifier with multiple computational units at each layer:

$$h = xW^0 \quad (10)$$

$$f = hW^1 \quad (11)$$

$$\hat{y} = fW^2 \quad (12)$$

Going deeply non-linear: multi-layered perceptrons



- ▶ But any stack of linear classifiers is still a linear classifier :-)

Going deeply non-linear: multi-layered perceptrons



- ▶ But any stack of linear classifiers is still a linear classifier :-(
- ▶ Still can't handle XOR and other non-linear problems.



- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities**
- 6 Enters Transformer

Non-linearities



- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.

Non-linearities



- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.
- ▶ They help induce **good input data representations**.

Non-linearities



- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.
- ▶ They help induce **good input data representations**.
- ▶ These representations are processed with a linear classifier in the end.

Non-linearities

- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.
- ▶ They help induce **good input data representations**.
- ▶ These representations are processed with a linear classifier in the end.

Popular activation functions

- ▶ **Sigmoid** (logistic function):

$$\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow [0, 1]$$

Non-linearities

- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.
- ▶ They help induce **good input data representations**.
- ▶ These representations are processed with a linear classifier in the end.

Popular activation functions

- ▶ **Sigmoid** (logistic function):
$$\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow [0, 1]$$
- ▶ **REctified Linear Unit (ReLU)** [Glorot et al., 2011]:

$$relu(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \rightarrow [0, \infty]$$

Non-linearities

- ▶ Differentiable non-linear transformations (**activations**) are the core reason of deep learning's substantial results, including in NLP.
- ▶ They help induce **good input data representations**.
- ▶ These representations are processed with a linear classifier in the end.

Popular activation functions

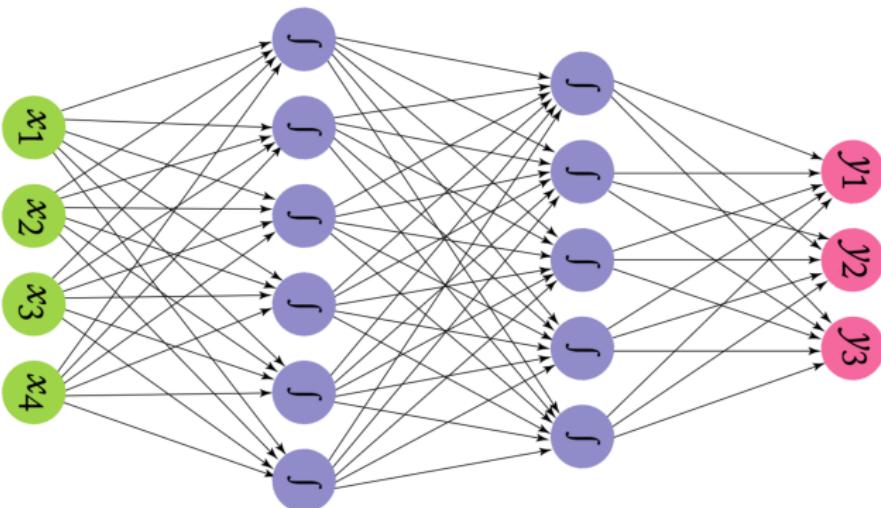
- ▶ **Sigmoid** (logistic function):
 $\sigma(x) = \frac{1}{1+e^{-x}} \rightarrow [0, 1]$
- ▶ **REctified Linear Unit (ReLU)** [Glorot et al., 2011]:

$$relu(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \rightarrow [0, \infty]$$

- ▶ **Hyperbolic tangent**:

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \rightarrow [-1, 1]$$

Non-linearities



The only difference between this **deep neural network** and a stack of linear classifiers: **non-linearities** (\int) between linear transformations.



What exactly is 'deep' in deep neural networks?

- Input data goes through successive transformations ϕ at each layer.



What exactly is 'deep' in deep neural networks?

- ▶ Input data goes through successive transformations ϕ at each layer.
- ▶ The transformations themselves are still linear...



What exactly is 'deep' in deep neural networks?

- ▶ Input data goes through successive transformations ϕ at each layer.
- ▶ The transformations themselves are still linear...
- ▶ ...but followed with some non-linear operation g at each layer.



What exactly is 'deep' in deep neural networks?

- ▶ Input data goes through successive transformations ϕ at each layer.
- ▶ The transformations themselves are still linear...
- ▶ ...but followed with some non-linear operation g at each layer.
- ▶ At the last layer, the prediction \hat{y} is produced.

Non-linearities

What exactly is 'deep' in deep neural networks?

- ▶ Input data goes through successive transformations ϕ at each layer.
- ▶ The transformations themselves are still linear...
- ▶ ...but followed with some non-linear operation g at each layer.
- ▶ At the last layer, the prediction \hat{y} is produced.
- ▶ The whole system is trained end-to-end.

$$\phi(\mathbf{x}) = g(\mathbf{x}\mathbf{W}') \quad (13)$$

$$\hat{y} = \phi(\mathbf{x})\mathbf{W} \quad (14)$$



Feed-forward neural networks

- ▶ ‘**Feed-forward neural network**’ is a more precise name for a multi-layer perceptron with non-linearities.



Feed-forward neural networks

- ▶ ‘**Feed-forward neural network**’ is a more precise name for a multi-layer perceptron with non-linearities.
- ▶ A network in which the neurons/units are connected without cycles.

Feed-forward neural networks

- ▶ ‘**Feed-forward neural network**’ is a more precise name for a multi-layer perceptron with non-linearities.
- ▶ A network in which the neurons/units are connected without cycles.
- ▶ **Outputs from neurons in each layer are passed to neurons in the next layer**, multiplied by the correspondent weights.

Feed-forward neural networks

- ▶ ‘**Feed-forward neural network**’ is a more precise name for a multi-layer perceptron with non-linearities.
- ▶ A network in which the neurons/units are connected without cycles.
- ▶ **Outputs from neurons in each layer are passed to neurons in the next layer**, multiplied by the correspondent weights.
- ▶ A non-linear function is applied element-wise after each layer.

Feed-forward neural networks

- ▶ ‘Feed-forward neural network’ is a more precise name for a multi-layer perceptron with non-linearities.
- ▶ A network in which the neurons/units are connected without cycles.
- ▶ Outputs from neurons in each layer are passed to neurons in the next layer, multiplied by the correspondent weights.
- ▶ A non-linear function is applied element-wise after each layer.
- ▶ No outputs are passed back to previous layers.

Feed-forward neural networks

- ▶ 'Feed-forward neural network' is a more precise name for a multi-layer perceptron with non-linearities.
 - ▶ A network in which the neurons/units are connected without cycles.
 - ▶ Outputs from neurons in each layer are passed to neurons in the next layer, multiplied by the correspondent weights.
 - ▶ A non-linear function is applied element-wise after each layer.
 - ▶ No outputs are passed back to previous layers.
1. The first layer contains **input units**,

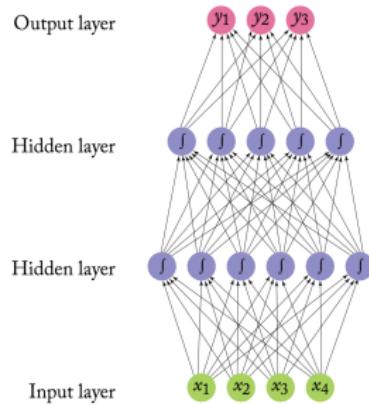
Feed-forward neural networks

- ▶ 'Feed-forward neural network' is a more precise name for a multi-layer perceptron with non-linearities.
 - ▶ A network in which the neurons/units are connected without cycles.
 - ▶ Outputs from neurons in each layer are passed to neurons in the next layer, multiplied by the correspondent weights.
 - ▶ A non-linear function is applied element-wise after each layer.
 - ▶ No outputs are passed back to previous layers.
-
1. The first layer contains input units,
 2. the last layer contains output units,

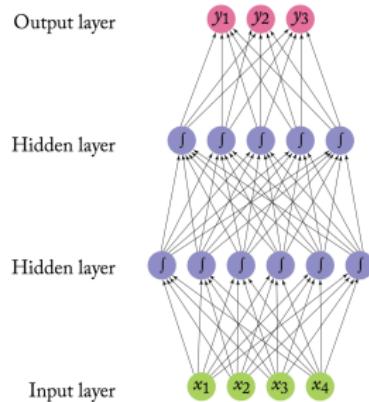
Feed-forward neural networks

- ▶ 'Feed-forward neural network' is a more precise name for a multi-layer perceptron with non-linearities.
 - ▶ A network in which the neurons/units are connected without cycles.
 - ▶ Outputs from neurons in each layer are passed to neurons in the next layer, multiplied by the correspondent weights.
 - ▶ A non-linear function is applied element-wise after each layer.
 - ▶ No outputs are passed back to previous layers.
1. The first layer contains input units,
 2. the last layer contains output units,
 3. all layers in between (hidden layers) contain hidden units, which form hidden representations of the input data.

Non-linearities

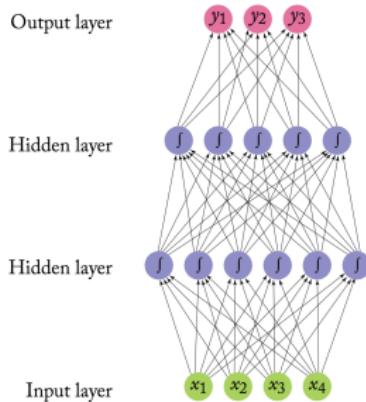


Non-linearities



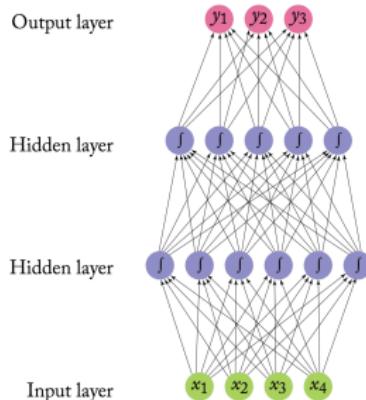
- Feed-forward network with 2 hidden layers;

Non-linearities



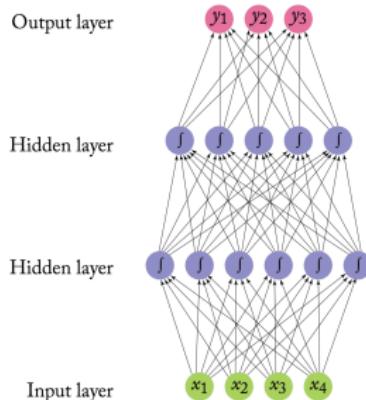
- ▶ Feed-forward network with 2 hidden layers;
- ▶ Before the transition to the I^{n+1} , the output vector of I^n goes through a non-linear function (instead of simple weighted sum).

Non-linearities



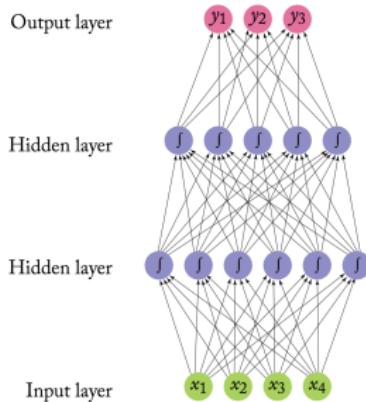
- ▶ Feed-forward network with 2 hidden layers;
- ▶ Before the transition to the I^{n+1} , the output vector of I^n goes through a non-linear function (instead of simple weighted sum).
- ▶ **Fully-connected**: transitions between layers are linear transformations:

Non-linearities



- ▶ Feed-forward network with 2 hidden layers;
- ▶ Before the transition to the I^{n+1} , the output vector of I^n goes through a non-linear function (instead of simple weighted sum).
- ▶ **Fully-connected**: transitions between layers are linear transformations:
 - ▶ each neuron in the layer I^n is connected to each neuron in the layer I^{n+1}

Non-linearities



- ▶ Feed-forward network with 2 hidden layers;
- ▶ Before the transition to the l^{n+1} , the output vector of l^n goes through a non-linear function (instead of simple weighted sum).
- ▶ **Fully-connected**: transitions between layers are linear transformations:
 - ▶ each neuron in the layer l^n is connected to each neuron in the layer l^{n+1}
 - ▶ ... mathematically, a **vector-matrix multiplication** $x^n W^n = x^{n+1}$



Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;



Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .



Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...

Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...
- ▶ ...but gradient-based optimizers still work well in practice.

Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...
- ▶ ...but gradient-based optimizers still work well in practice.
- ▶ NB: one can balance between the number of layers and their sizes (number of neurons).

Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...
- ▶ ...but gradient-based optimizers still work well in practice.
- ▶ NB: one can balance between the number of layers and their sizes (number of neurons).
- ▶ **Different layers learn different hidden representations**
 - ▶ eliminates the need to manually design feature combinations:

Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...
- ▶ ...but gradient-based optimizers still work well in practice.
- ▶ NB: one can balance between the number of layers and their sizes (number of neurons).
- ▶ **Different layers learn different hidden representations**
 - ▶ eliminates the need to manually design feature combinations:
 - ▶ the network is able to learn that '*not*' and '*good*' co-occurrence in a text is a powerful feature in itself...

Network parameters

- ▶ Like in linear classifiers, weights and biases are **parameters of the network**;
- ▶ a.k.a. θ ;
- ▶ the aim of the training is to **learn optimal values for θ** .
- ▶ Unlike in linear classifiers, the error/loss function is usually **not convex**...
- ▶ ...but gradient-based optimizers still work well in practice.
- ▶ NB: one can balance between the number of layers and their sizes (number of neurons).
- ▶ **Different layers learn different hidden representations**
 - ▶ eliminates the need to manually design feature combinations:
 - ▶ the network is able to learn that '*not*' and '*good*' co-occurrence in a text is a powerful feature in itself...
 - ▶ ...and reflect this in its hidden representations.

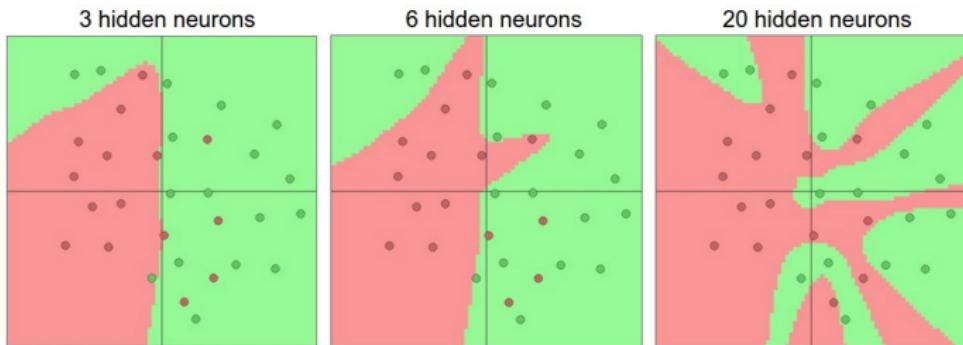
Model size



The deeper and wider the model is, the more capacity (parameters) it has, the more complex functions it can approximate.

Model size

The deeper and wider the model is, the more capacity (parameters) it has, the more complex functions it can approximate.





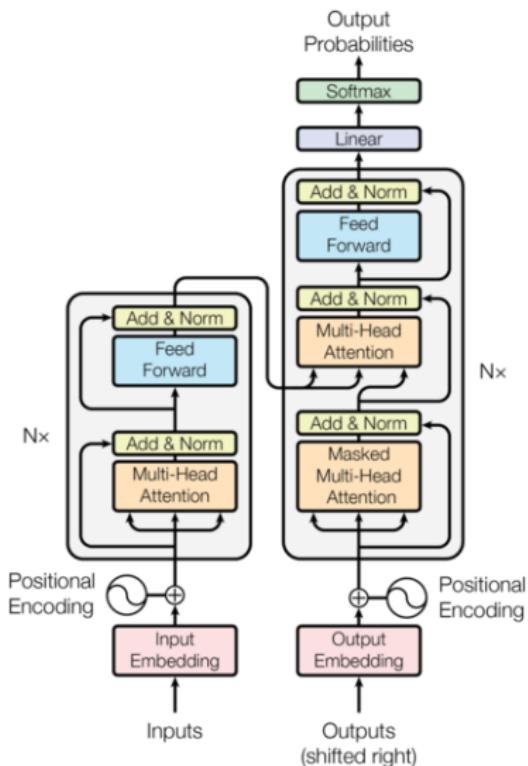
Contents

- 1 Tutorial technicalities
- 2 Basics of supervised machine learning
- 3 Advantages and limitations of linear models
- 4 Going deeply non-linear: multi-layered perceptrons
- 5 Non-linearities
- 6 Enters Transformer

The One

Almost all modern large language models are invariably based in a variation of feed-forward neural network: **Transformer** architecture.

- ▶ BERT, RoBERTa, BART
- ▶ GPT variants
- ▶ Text-to-Text Transfer Transformer (T5)
- ▶ PaLM, BARD, LaMDA, LLaMA bla bla bla
- ▶ Used for text-to-image in DALL-E, Stable Diffusion etc..



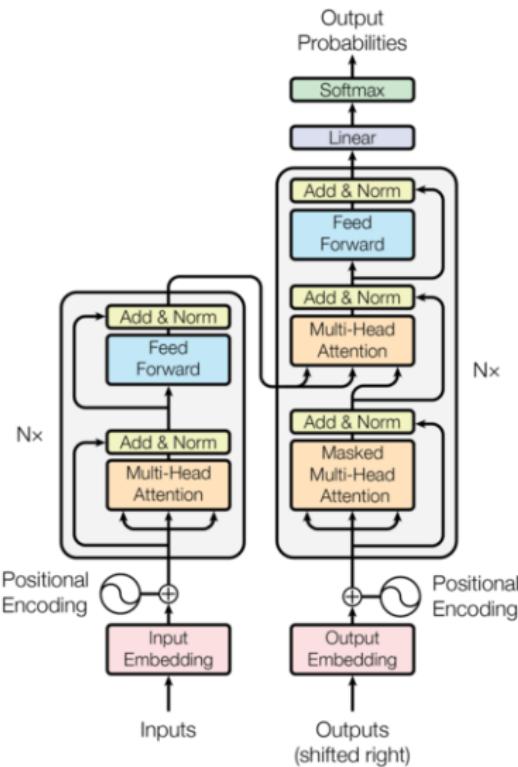
Attention is all you need

“The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder and decoder configuration. The best performing models also connect the encoder and decoder through an **attention mechanism**. We propose a novel, simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.”

– [Vaswani et al., 2017]

– Per 28.02.2023: cited 66,7k times

The Transformer, at a glance



The Transformer, at a glance

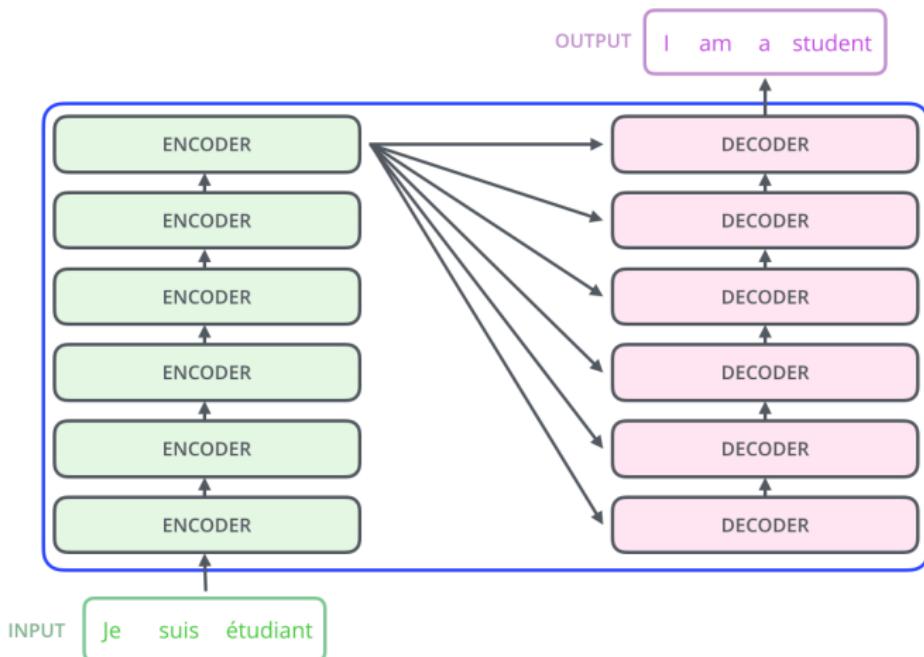


Figure from Jay Alammar's, *The Illustrated Transformer*¹

¹<https://jalammar.github.io/illustrated-transformer/>

Applications of Attention: high level explanation

Three different applications of attention in the Transformer:

1. In the **Encoder-Decoder** way: we use the 'hidden state' from the encoders and do some operation with the state in the decoder
2. **Encoder** Self-Attention: each position in the encoder can attend to previous positions
3. **Decoder** Self-Attention: each position can attend to all previous positions up to that point (autoregressive via masking)

Those are also the **three types of modern large language models (LLMs)**.

So how good is it?

Transformer turned out to be extremely good in machine translation:

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

'A small fraction of the training cost'

[Vaswani et al., 2017]

So how good is it?

Transformer turned out to be extremely good in machine translation:

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

'A small fraction of the training cost'

[Vaswani et al., 2017]

...not long after Transformer became the dominant architecture for language models, NLP models in general, and 'artificial intelligence' overall.

References I

-  Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016).
Layer normalization.
arXiv preprint arXiv:1607.06450.
-  Bahdanau, D., Cho, K., and Bengio, Y. (2015).
Neural machine translation by jointly learning to align and translate.
In Bengio, Y. and LeCun, Y., editors, 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.
-  Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019).
BERT: Pre-training of deep bidirectional transformers for language understanding.
In Burstein, J., Doran, C., and Solorio, T., editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1

References II

(Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota.
Association for Computational Linguistics.

-  Glorot, X., Bordes, A., and Bengio, Y. (2011).
Deep sparse rectifier neural networks.
In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pages 315–323.
-  Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023).
Mistral 7b.
-  Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993).
Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.
Neural Networks, 6(6):861–867.

References III

-  Turing, A. (1948).
Intelligent machinery.
Technical report, National Physical Laboratory.
-  Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
In Advances in Neural Information Processing Systems, pages 5998–6008.