Advanced Methods in Text Analytics

Word Representations

Dr. Daniel Ruffinelli - FSS 2025

What are representations? (1)

- Let’s **represent** a hypothetical country called Countryland!
- What **features** should we use?
  - Area size? E.g. 400.000 km^2
  - Population size? E.g. 20M
  - Continent? E.g. Europe
- Let’s group those features as a single 3D **feature vector**
  - Countryland: (400.000; 20M; ‘Central Europe’)
- Features can have different types
  - Real or integer numbers, e.g. population size
  - Categorical, e.g. the continents
- We refer to that vector as a **representation** of ‘Countryland’
- Is that useful for, say, inferring size of the economy?

What are representations? (2)

- How about the following features then?
  - GDP? E.g. 50 billion USD
  - Interest rate? E.g. 2.2%
  - Inflation rate? E.g. 7.2%
- This gives us a different representation of Countryland:
  - (50B; 2.2; 7.2)
- Manually constructing representations: **feature engineering**
  - Often requires expertise in the area
  - E.g. 'government debt to GDP' is a common metric in Economics
- **Feature vectors:** most typical form of input to machine learning (ML) methods

Can we *learn* useful representations?

- Can we **learn useful features**?
  - Then no more feature engineering
  - We may not even need an Economics degree to get useful features
- **Yes!** Deep Learning (DL) methods are useful for this.
  - But ML/DL is mostly engineering, so back to square one?
  - No! These methods are applicable to any domain, e.g. economics, applied mathematics, etc.
- The **learned representations** may be *generally* useful, e.g. word representations
  - Potentially useful for any application with text as input

Word Embeddings

- Learned vector representations of words/tokens
- Can be learned in several different ways
- Classic example: **word2vec**
  - Useful geometrical properties between related word vectors

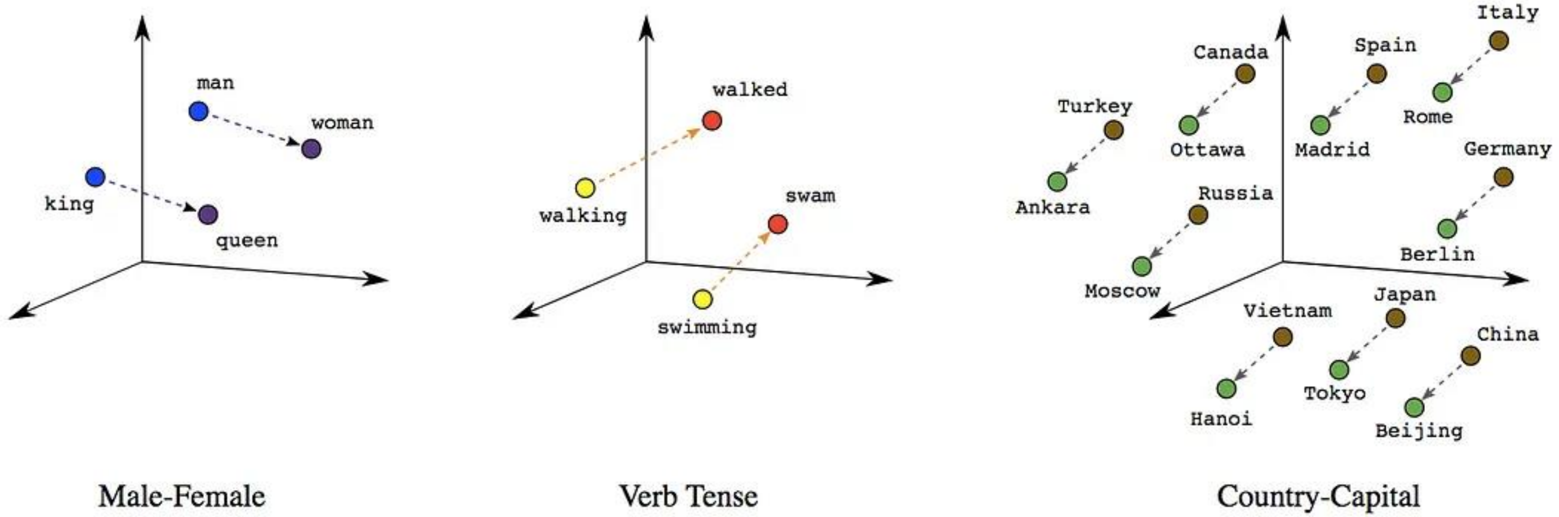


Image source

Challenges

University of Mannheim
School of Business Informatics and Mathematics

- Not straightforward as previous examples suggest!
  - Vector spaces with hundreds or thousands of dimensions
  - Variety in methods for learning word representations
- Thus, many challenges!
  - Here's some examples of those challenges
- **Interpretability:**
  - Learned features often not interpretable (real numbers)
  - Geometric properties of space not straightforward (high-dimensional space)
- **Cost:**
  - Train on terabytes of text
  - Latest LLMs have *trillions* of parameters
  - GPU memory limited
- This **fixed (static) word -> vector mapping seldom used** anymore
  - But still part of architectures, relevant to understand representations

Outline

1. Machine Learning Refresher

2. Deep Learning Basics

3. Word Embeddings

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

Machine Learning Refresher

About this Section

- **Goal:** discuss basic concepts and terms that will be useful for the entire course
- **Outline:**
  1. Basic Machine Learning Concepts
  2. Probability Theory Refresher
  3. Maximum Likelihood Estimation
  4. Summary

What is machine learning? (1)

- Methods for developing algorithms by **learning from data**
  - In contrast to manually coding knowledge we already have
  - Example: flying a helicopter (how many situations must we consider?)
- Umbrella term encompassing **many families of methods**
  - E.g. Inductive Logic Programming, Deep Learning, etc.
- Related (umbrella) terms
  - **Artificial Intelligence:** solutions that show intelligent behavior
  - **Data Mining:** extract value from data (ML usually part of that pipeline)
- Machine learning **in this course**?
  - Mostly **deep learning and related methods**

What is machine learning? (2)

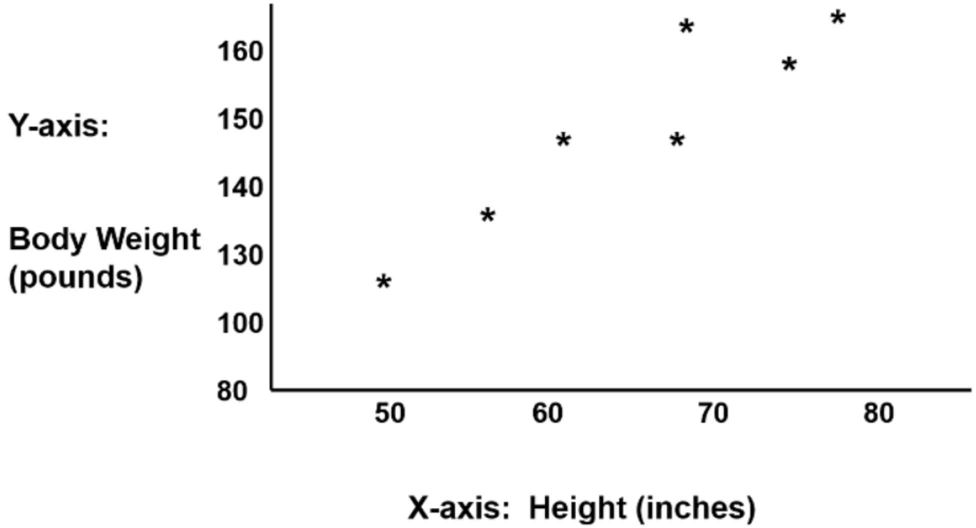
UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- Classic definition:
  - “A computer program is said to learn from **experience** *E* with respect to some class of **tasks** *T* and **performance measure** *P*, if its performance, as measured by *P*, improves with experience *E*.”

    Tom Mitchell, Machine Learning, 1997
- **Experience:** data in some format
  - In our case, text (that needs to be processed so models can "read" it)
- **Tasks:** depends on perspective
  - ML: classification, regression, etc.
  - NLP: machine translation, question answering, etc.
- **Performance measure:** depends on task
  - E.g. F1 score for classification tasks, ROUGE scores for summarization

What is a task?

- **Task:** predict weight of a person (regression, supervised)
- **Data:** How do we represent a person?
  - Let's go with a simple 2D vector: (height, weight)
- We might see a linear relation between input and target
  - **Input:** 1D feature vector (usually denoted by $\mathbf{x}_i$, here: height)
  - **Target:** weight (usually denoted by $\mathbf{y}$, here: real number)



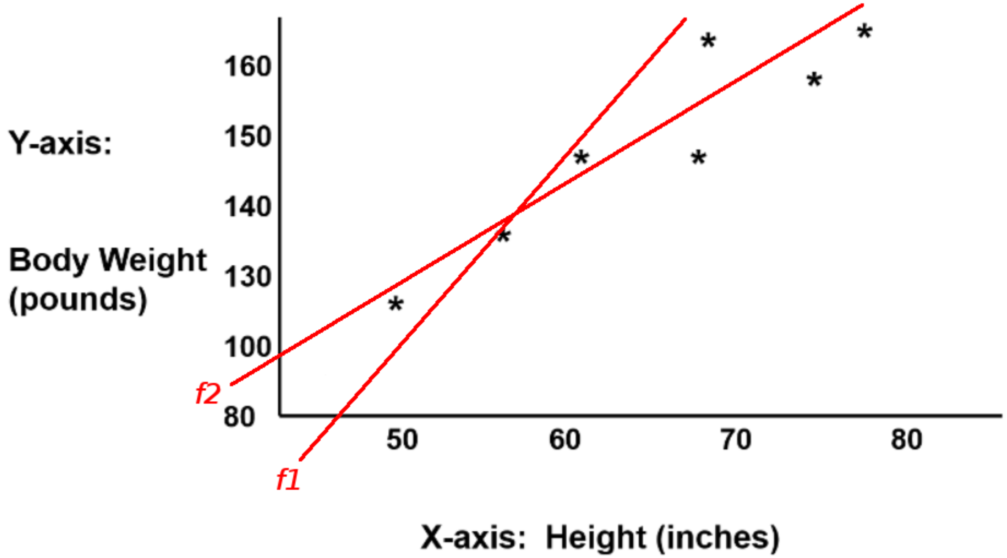
[Example source]

What is a model?

- Family of methods that **we choose**
  - Each comes with a set of **assumptions**
- E.g. based on observed data, we assume linear relation between height and weight
  - Thus, we choose a linear regression model
- **Linear regression:**
  - **Main assumption:** linear relation between inputs and output
  - Formally: learn function *f(x) = y*
  - Here: *f(height) = weight*
  - This model comes from statistics, studied more formally, e.g. more assumptions, ML is more "relaxed", focuses on learning (generalizing)
- How exactly do we **learn** a function?
  - To get there, we need to talk about parameters

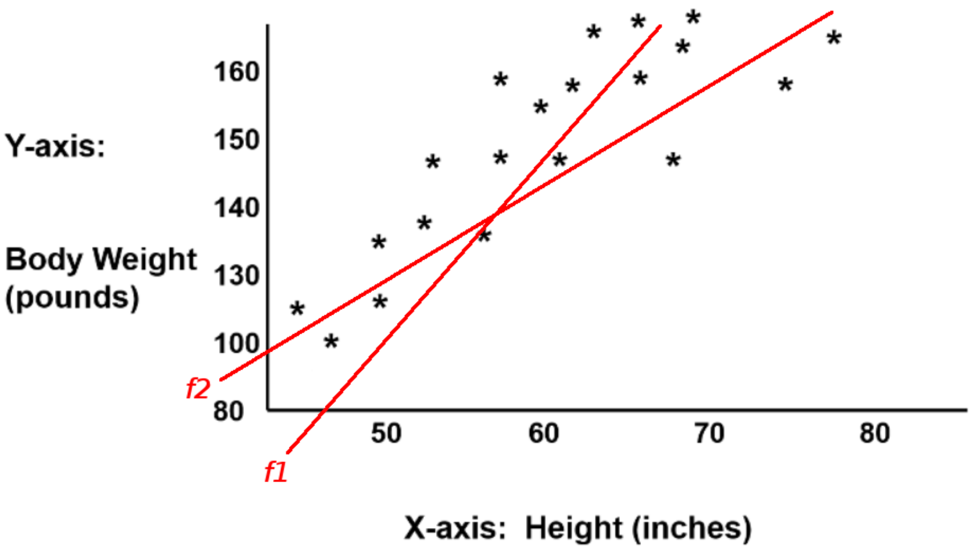
What are parameters?

- We can describe the assumption in linear regression with the **equation of a line**:
  - $y = mx + b$ where m is slope and b is y-intercept
  - In our case: *weight = m height + b*
  - Thus, we have **two parameters: *m, b***
- Each value assignment of m, b: different member of linear regression family
  - Which one **fits** the data better?
  - f_1 or f_2?
  - How do you know? Depends on the goal



What is generalization?

- **Goal of ML:** model should perform well at the task on new data
  - Not just data it was trained on
  - Thus, goal of training a model: **generalize to unseen data**
- For example, which function did you say fits our data best?
  - f_1 or f_2?
- What if we collect more data?
  - Before, f_2 seemed best
  - With more data, f_1 is best
- How do we train for generalization?
  - We **evaluate models on *held-out* evaluation data**, represents unseen data
  - Train and eval data assumed from same distribution

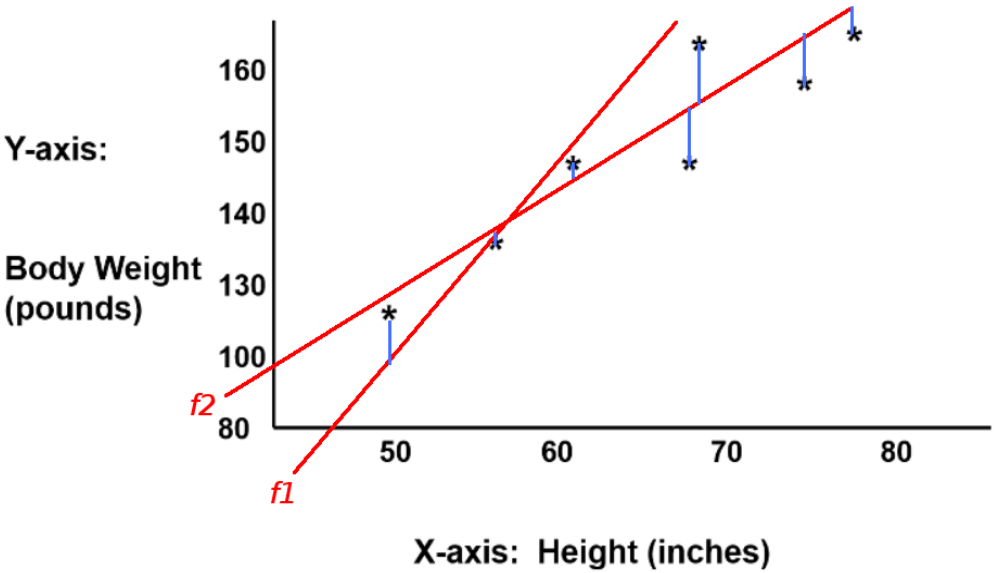


How do we evaluate a model?

- How can we tell which member of the family is more suitable for our task?
  - We need to **select a model** from the many options
- **Performance metric**
  - A **metric we choose to evaluate** how good a **model** is at a task
  - **For example:** mean squared error (MSE)

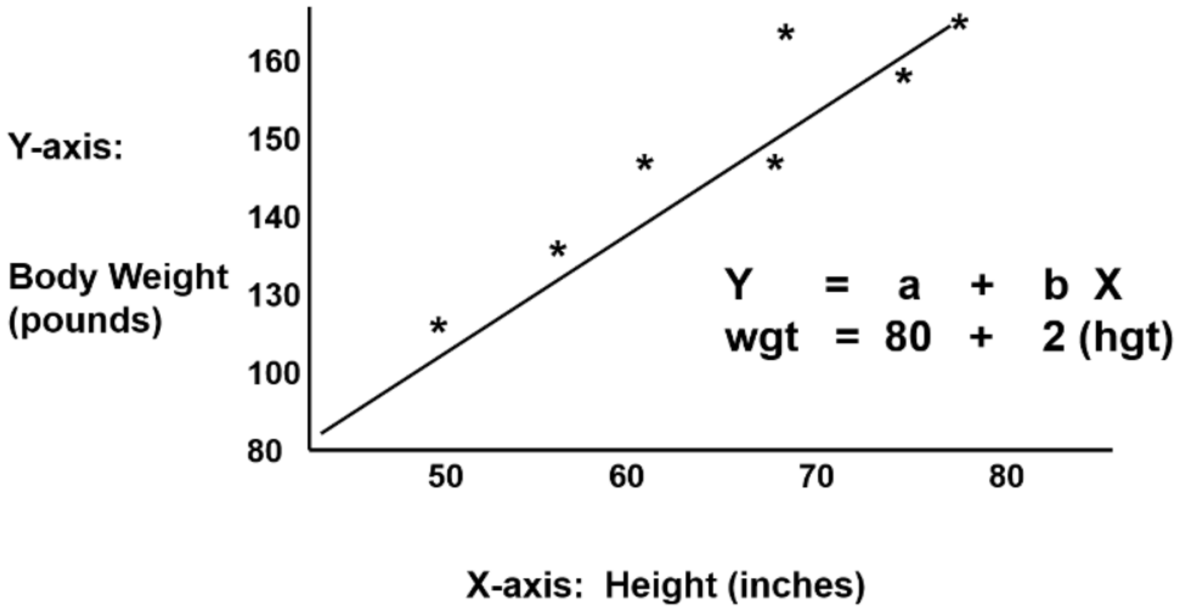
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

where:
- n: # of examples
- Y_i: target values
- Y'_i: predicted values



How do we use our selected model?

- **Model selection:** choosing a value assignment for our parameters
  - E.g. *m = 80, b = 2*
- To make a **prediction**, we make use of our selected model (function)



- Important distinction: inference vs prediction
  - **Inference:** use model to learn about data generation process
  - **Prediction:** predict specific outcomes for new input points

What are *hyper*parameters?

- Other parameters that are **typically not learned**
  - Instead, they are manually set by the user
  - Note that *meta-learning* does exist, e.g. AutoML
- Example in linear regression: using a bias or not
  - No bias: line passes by origin (assumes data is centered)
  - See scikit-learn's documentation on linear regression (fit_intercept)
- How do we choose values for hyperparameters?
  - **Hyperparameter optimization methods**
  - E.g. grid search, random search (known to be better than grid search)
  - Requires familiarity with known tricks/experience/expert knowledge
- Often **involves making assumptions**
  - E.g. this setting worked well in the past
  - E.g. the search space I designed benefits all models equally

What about classification?

- **Logistic regression:** popular model for *classification*
  - **Main assumption:** linear relation between inputs and output
- Essentially: linear regression with a logistic function
  - $y = \sigma(m\,x + b)$ where σ is the logistic (sigmoid) function

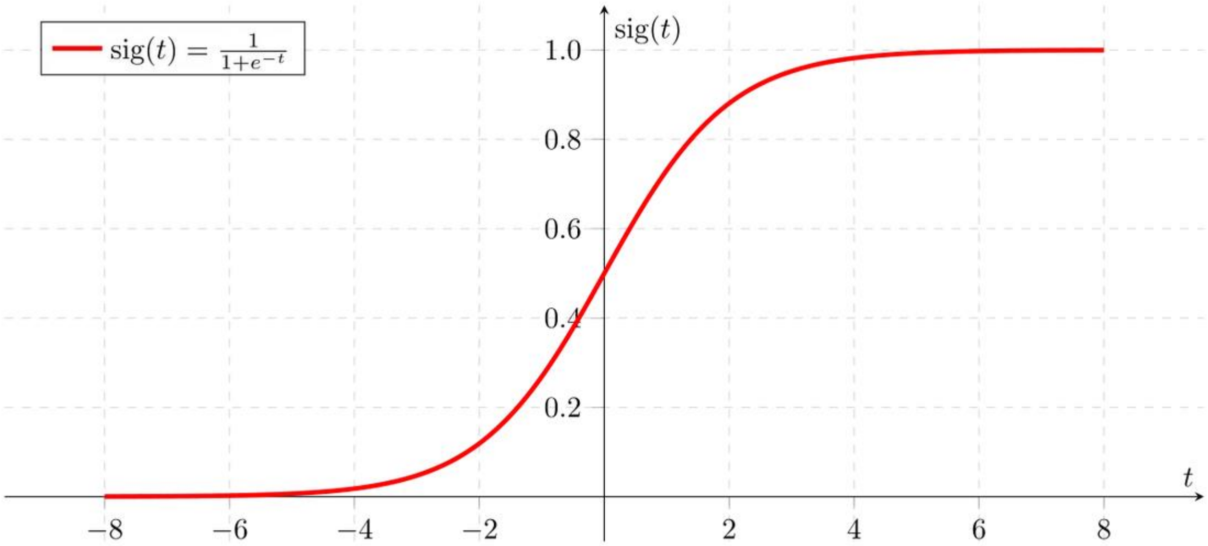


Image source

- Logistic function maps inputs to real interval [0,1]
  - **Can be *interpreted* as a probability distribution**

How do we use logistic regression?

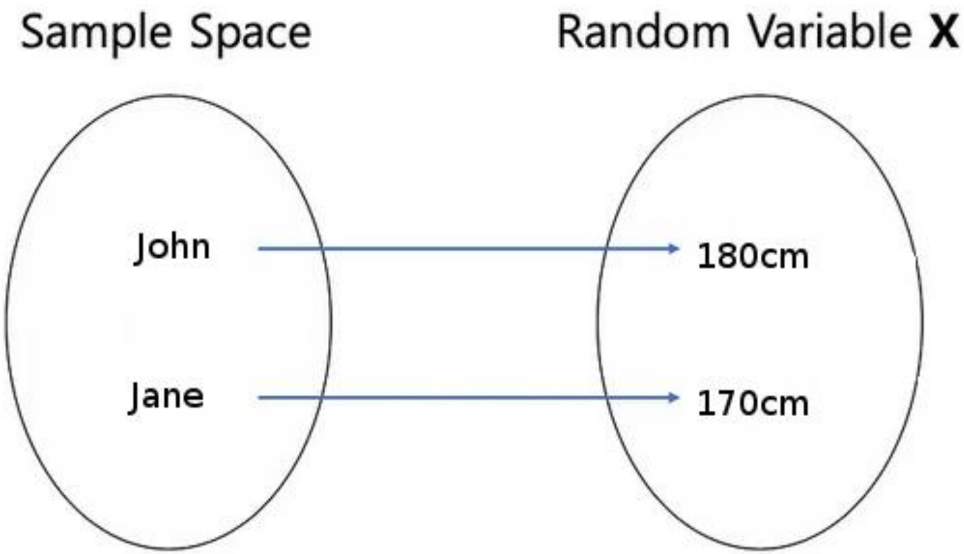
- **Binary** classification:
  - *$P(y = 1) = \sigma(m\ x + b)$ -->* interpreted as probability for positive class
  - *$P(y = 0) = 1 - \sigma(m\ x + b)$ -->* complementary probability for negative class
- That is, we get probabilities for both positive and negative class
  - We can do some *inference*, e.g. how much more likely is one class than the other?
  - But *prediction?*
- **Prediction:** we decide on a threshold
  - *$p(y = 1) >= 0.5$*, we predict positive class
  - *$p(y = 0) < 0.5$*, we predict negative class
- We get the following **decision boundary**
  - *$p(y = 1|x) = p(y = 0|x) = 0.5$*
- Describing tasks and models with **probability theory** is very common!
  - We'll do this often, so let's review the basic concepts

Probability Theory Basics

- This is a **quick refresher** about **probability theory**
- This is later **useful for many things**
  - Describing tasks
  - Describing models, e.g. what language models do
  - More concepts commonly used in NLP research
- Today, some basic intuitions/definitions
  - Random variables
  - Probability distributions
  - Independence of events
  - Sum rule / product rule
  - Joint distributions
  - Conditional probability
  - Conditional independence
  - Bayes rule

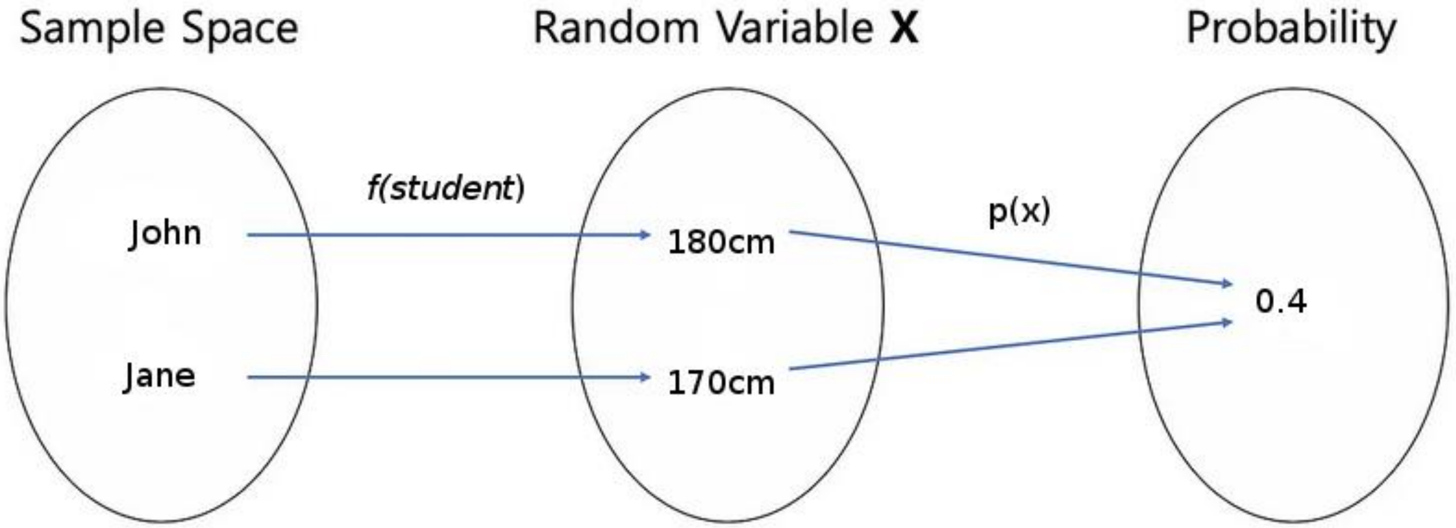
Random Variables

- **Function** between events and something relevant about the events
  - **Usually designed** according to something we want to model
- Example: say we are interested in average height of students
  - **Event:** randomly select a student
  - **Event space:** set of all students the course
- **Random variable X**: *f(student) = height of student*



Probability Distributions (1)

- Essentially a way to describe random variables
- **Random variables:** take values as a result of some random process
- **Distribution of random variable:** how often may each value be observed
- How does a distribution describe this?
  - A(nother) function *p(**X**):* maps set of possible values a random variable can take to interval [0,1]



Probability Distributions (2)

- **Output of *p* for given event *a*:** probability of observing event *a*
  - Written as $p(\mathbf{X}=a)$
  - Example: what is the probability of observing a height of 185cm?
  - $p(\mathbf{X} = 185cm) = 0.1$, i.e. probability is 10%
- **Discrete distributions:** random variable can take finite set of values
  - *p* usually referred to as *probability mass function*
- **Continuous distributions:** r.v. can take infinite set of values
  - *p* usually referred to as *probability density function*
- Examples:
  - Bernoulli (discrete): models output of single binary experiment
  - Gaussian (continuous): normal distribution
- Is **X** in our students' height example discrete or continuous?
  - Depends on how to measure height, as reals or integers

Independence of events

- Events **A** and **B** are **independent** when one event happening does not affect the probability of the other happening
- For example:
  - Event **A**: Germany wins the world cup in 2014
  - Event **B**: Germany loses world cup semifinal
  - Event **C**: The stock market crashed in 2008
  - **Which of these events are dependent/independent?**
- A and B are not independent
  - If Germany loses the semifinal, it cannot win the final
- A and C are independent, as well as B and C
  - Very unlikely the 2008 crash has any impact on the 2014 world cup
- Independence perhaps best understood with **sum rule**

Sum Rule

- Probability of *union* of *n* events happening, e.g. A **OR** B
  - Formally: $p(A \cup B) = p(A) + p(B) - p(A \cap B)$
- For example: r.v. **Y** shows number in die, relevant events are:
  - **A:** "We observe a multiple of 3", i.e. either 3 or 6
  - **B:** "We observe a multiple of 2", i.e. either 2, 4 or 6
- Are **A** and **B** independent? **No**, see overlap in event spaces
- For $p(A \cup B)$ we subtract probability of overlapping events: $p(\mathbf{Y}=6)$
  - $p(A) = 1/6 + 1/6 = 1/3$
  - $p(B) = 1/6 + 1/6 + 1/6 = ½$
  - $p(y = 6) = 1/6$
  - $p(A \cup B) = 1/3 + ½ - 1/6 = 0.66$
- With independent events, last term becomes zero:
  - Event **A**: "We observe an even number", i.e. either 2, 4 or 6
  - Event **B**: "We observe an uneven number", i.e. either 1, 3 or 5

Product Rule

- Probability of conjunction of n events happening, i.e. A **AND** B
- Formally: $p(A,B) = p(A|B)p(B)$
  - $p(A,B)$ = joint distribution of A and B
  - $p(A|B)$ = conditional distribution of A given B
- Side note:
  - All of these rules are *derived* from axioms of probability theory
  - For more, see probability theory literature
- Let's go over joint distributions and conditional distributions

Joint Distributions (1)

- Function $p(X_{1\ldots n})$ from *n* random variables to interval [0,1]
  - Distribution of all possible combinations of those *n* random variables
- Easier to see with joint probability table
- For example, joint distribution of flipping two coins **A** and **B**:

A / B	Heads	Tails
Heads	0.1	0.2
Tails	0.3	0.4

  - Note that all elements in table add up to 1, i.e. a distribution

Joint Distributions (2)

- We can derive many things from a joint distribution
- A very common operation: **marginalization**
  - Given joint distribution of A and B, derive distribution of A or B
- For example, we get distribution of coin A by summing over all possible values of B with the sum rule:

A/B	Heads	Tails	*p(A)*
Heads	0.1	0.2	= 0.3
Tails	0.3	0.4	= 0.7

- New column adds up to 1, i.e. a probability distribution
  - p(A) known as **marginal distribution** of A
- This process is referred to as **marginalizing out** B
  - Normally expressed with sum rule: $p(A) = \sum_b p(A,B = b)$

Conditional Probability

- Probability of **seeing event A knowing event B was already seen**
- Formally, $p(A|B) = p(A,B)/p(B)$
- Intuitively, **space of all possible events shrinks** (for dependent RVs)
  - Imagine we toss a die
  - **A:** "We observe an even number", i.e. either 2, 4 or 6
  - **B:** "We observe a multiple of 3", i.e. either 3 or 6
  - After observing A, event space for B is not {3, 6}, it is {6}
- Note that $p(A|B)$ can be derived from joint distribution $p(A,B)$
  - Marginalize out A to get $p(B)$
  - Then you have everything you need

Conditional Independence

- Two events are **independent given a third event**
  - Observing **A** tells me something about **B**
  - But when observing **C**, **A** no longer says anything about **B**
- For example:
  - Event **A:** knowing a person has good vocabulary
  - Event **B:** knowing a person's height
- Are **A** and **B** independent?
  - No, since children have limited vocabulary
- Now consider event **C**: knowing a person's age
  - Given **C**, **A** and **B** are independent, since height implies age
  - Once we have age, height gives us no new information
  - Thus, **A** and **B** are conditionally independent given **C**

Bayes' Rule

- Apply product rule on conditional probability
- That is, $p(A|B) = p(A,B) / p(B) = p(B|A)\ p(A) / p(B)$
  - Note that we want probability of A given B, i.e. $p(A|B)$
  - And to get it, we use marginal distribution of A, i.e. $p(A)$
  - This may seem **counterintuitive**
- This is typically interpreted like so:
  - $p(A)$ known as **prior probability** distribution of event A
    - Describes **what we know/believe** about event A
  - $p(B|A)$ known as **likelihood**
    - Given event A, what is the likelihood that we get our data
  - $p(A|B)$ known as **posterior probability** of A
    - Interpreted as **updated description of A** after considering what we originally know about A (prior), as well as our data (event B)
- This interpretation gives rise to the *frequentists vs bayesian* debate
- And with this, we are **done with our probability refresher!**

A Probabilistic Perspective of ML

- **Task:** predict outcome of a coin tossed by your (unreliable) friend
  - Many factors: mass distribution, position in fingers, friction, etc.
  - Difficult to *model* physically
- Let's use a probabilistic model
  - Abstracts away complications of physical model
- **Probability:** uncertainty of some *random* event, e.g. coin toss
  - In practice: ***relative frequency of observations*** about the event
  - **Event in this case:** *seeing heads* when tossing the coin
- **Simple Model:** Bernoulli distribution
  - **Intuition:** it models outcome of *single binary* experiment
  - ***Formally:*** $p(k|\Theta) = \Theta^k (1-\Theta)^{1-k}$ where k in {0,1} is possible outcomes
  - **Θ is the only parameter of model:** probability of seeing heads
- How do we get a value for this parameter so we can use our model?
  - We could **estimate this parameter**
  - We could blindly guess, or we could **use data for that estimation**!

Training Models: Parameter Estimation

- Say your friend allows you to observe ten tosses
  - You observe 3 out the 10 are heads
- You can estimate the value of Θ from this data
  - $\Theta = 3/10 = 0.3$ chance of seeing heads in a single toss
- We just counted **relative frequencies**
  - Training example: a single coin toss, i.e. y_i in *{heads, tails}*
  - Training data: 10 coin tosses
  - Relevant observations: we observe heads
- With a value estimated for parameter Θ, we can make predictions!
  - Our model has been trained and is ready to be used **on unseen data**
- Thus, **learning a function** means **estimating its parameters**
  - We don't know the parameters of a model
  - Instead, we have data and use it to estimate those parameters
- Let's look at a common framework for parameter estimation

Maximum Likelihood Estimation (1)

- Common approach for parameter estimation
  - Otherwise known as **MLE**
- Finds model that **maximizes likelihood** of observed data
  - But what does likelihood mean?
- Our model for the coin toss:
  - $p(k|\Theta) = \Theta^k (1-\Theta)^{1-k}$ is a function of k with Θ fixed (how we predict with it)
  - Let's ***instead see it as*** function of Θ with fixed k, i.e. $L(\Theta|k) = \Theta^k (1-\Theta)^{1-k}$
  - That is, a function of parameters given *fixed data* (assumed during training)
- We call $L(\Theta|k)$ the **likelihood function**
  - It is **not a probability distribution**
  - Instead, we use it to find distributions that fit observed data
- How do we use the likelihood function during training?
  - We want distribution with *highest* (max) probability to our data
  - **Assumption:** set of observed data is most likely sample among all data

Maximum Likelihood Estimation (2)

- Formally, we want $\Theta = max_{\Theta} L(\Theta|k)$
  - Equivalently, we want $\Theta = argmax_{\Theta}$ **log** $p(k|\Theta)$
- **First**, we make a common **assumption**: examples are **i.i.d.**
  - This stands for **independent and identically distributed**
  - Examples are independent, belong to same distribution
- **Then**, we apply this assumption to our coin toss setting
  - Our model for a single example: $p(k|\Theta) = \Theta^k (1-\Theta)^{1-k}$
  - The **joint probability of *N* observed examples** is then given by:

$$\begin{aligned} P(D) &= P(y_1, y_2, \ldots, y_N) \\ &= P(y_1)P(y_2)\ldots P(y_N) \\ &= P(head)^k P(tail)^{N-k} \\ &= \theta^k (1-\theta)^{N-k}. \end{aligned}$$

  - Here we used the i.i.d. assumption from line 1 to line 2

Maximum Likelihood Estimation (3)

- **Finally**, to find this maximum value, we:
  - Compute the first derivative
  - Set it to zero
  - Extract a value for Θ

$$\begin{aligned}
\frac{\partial \log P(D)}{\partial \theta} &= \frac{\partial\left(\log \theta^k (1-\theta)^{N-k}\right)}{\partial \theta} \\
&= \frac{\partial\left(k \log \theta + (N-k)\log(1-\theta)\right)}{\partial \theta} \\
&= \frac{k}{\theta} - \frac{N-k}{1-\theta} = 0 \\
&\Rightarrow \hat{\theta} = \frac{k}{N},
\end{aligned}$$

- We obtained relative frequencies, as we had done already!
  - In practice, **MLE often does not have a closed-form solution**
  - Instead, **we rely on numerical methods**
- MLE related to many aspects of ML: cross entropy, MAP, etc.

Empirical Risk Minimization

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- This is a general training approach
- **Main goal:** estimate how a model will work in practice by averaging a **loss function** $L(x_i, y_i)$ on a training set
  - Here x_i is a training example and y_i its corresponding label
- Formally:

$$R_{\mathrm{emp}}(h) = \frac{1}{n}\sum_{i=1}^{n} L(h(x_i), y_i).$$

  - Note the training set is of size n here

- This is the most common training approach in ML/DL/NLP.
  - But what is a loss function, exactly?

What are loss functions?

- Functions that tell us **how a model performs during training**
  - Being a cost/loss, we want to **minimize it**
- Desiderata:
  - **Low loss:** when model prediction matches given label
    - The closer, the lower the loss/cost
  - **High loss:** when model prediction does not match label
    - The farther, the higher the loss/cost
- Example: **log loss**
  - Here *a* is model prediction
  - Note the desired behavior
  - Works with probabilistic models only

$$-\begin{cases}\log a_i, & y_i = 1,\\ \log(1-a_i), & y_i = 0.\end{cases}$$

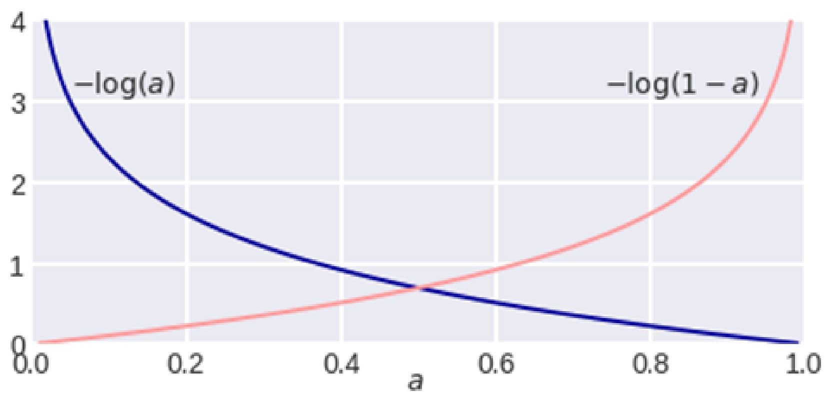


Image source

Let‘s design a training objective

- Combine empirical risk minimization with log loss
  - Assume binary classification
- We have:
  - $J = - 1/N \sum_n [\, y_i \log(p_i) + (1 - y_i) \log (1 - p_i) \,]$
    where N is size of training and p_i is prediction for example i
- Note the following:
  - Labels act as indicator factors
  - Log loss means bad model predictions get punished
  - We negate the entire expression so we can minimize it
- Thus, this is an optimization problem
  - This one can be shown to be a form of maximum likelihood estimation
- **Training objectives:** very important in NLP
  - E.g. masked vs autoregressive language have different, manually designed, training objectives

Summary: ML Refresher

- **Goal:** learn models that perform a task well on unseen data
- **Task:** from some input to some output
  - Regression: real-valued output
  - Classification: categorical output
- **Model:** abstract, often simplified version of a task
  - Comes with set of assumptions, e.g. linearity
- **Evaluation:** given predictions, compute relevant metric
- **Training:** parameter estimation
  - Objective functions (more not covered in *refresher*, e.g. regularization)
- All of these concepts are common in advanced NLP!
  - E.g. LLMs are **models** with trillions of **parameters**
  - Different LLMs **train** on different **objective functions** for different **tasks**
  - **Empirical risk minimization** and **MLE** used during training
  - They are **evaluated** on various **metrics** relevant to various tasks

Deep Learning Basics

About this Section

- **Goal:** discuss basic concepts and terms that will be useful for the entire course
- **Outline:**
  1. Basic Deep Learning Concepts
  2. Backpropagation
  3. Language Models with Feed-forward Neural Networks

Mathematical Notation

- Scalars: a, b
- Vectors: **x**, **y** (by default column vectors, i.e. size is *n x 1*)
- Matrices: **X**, **Y**
- Scalar product: a**x**
- Dot product: $\mathbf{x}^T\mathbf{w}$ (transpose because **x** is column vector)
- Matrix-vector product: **Xy**
- Matrix-matrix product: **XY**

What is an artificial neuron?

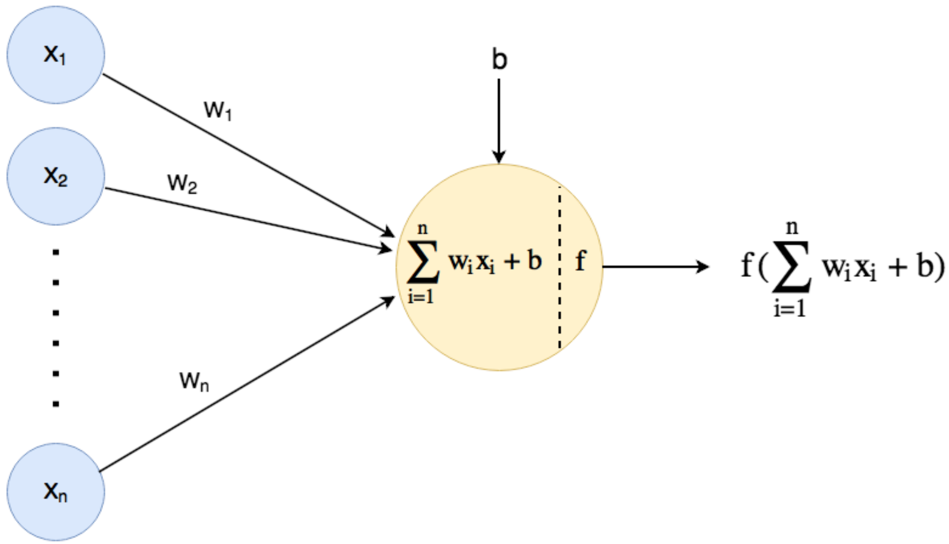
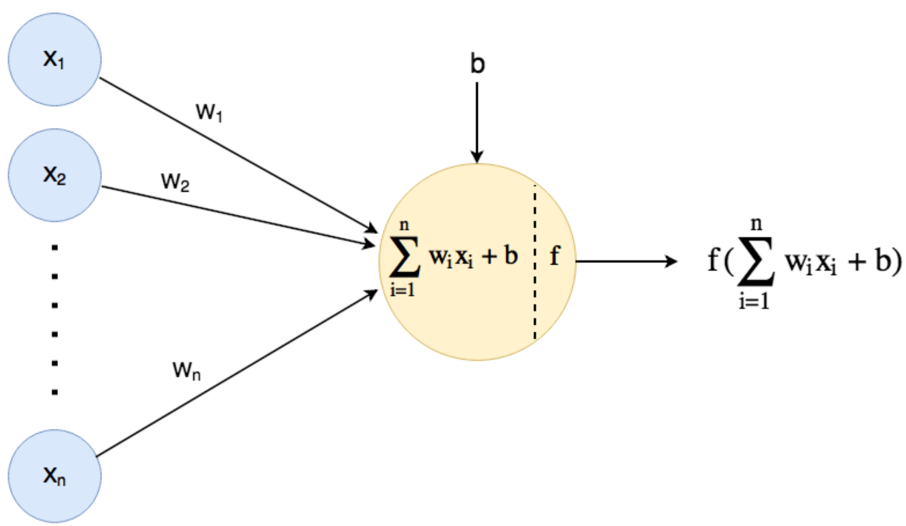


Image source

- Input is n-dimensional vector $\mathbf{x}$
- Features x_i multiplied with weights w_i, their products then added
  - What common operation is this?
  - A bias is optional, but almost always there, adds lots of flexibility to model
- Then an *activation function f* is applied to the result
- Different activation functions determine different artificial neurons

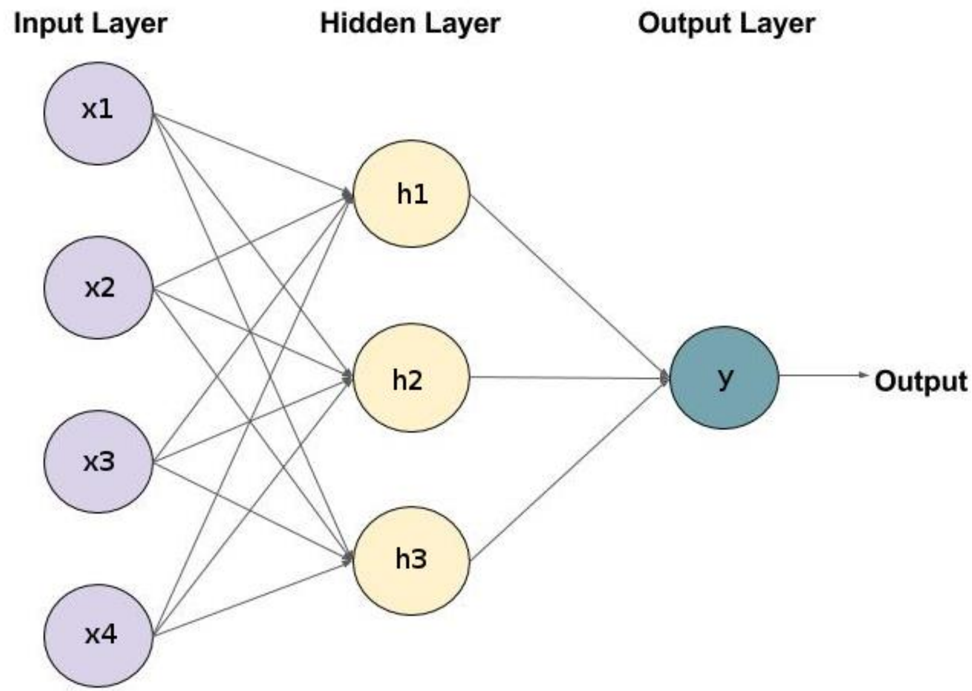
Types of Artificial Neurons

- They are mainly distinguished by their activation functions, e.g.
  - **Linear neuron:** $f(x) = x$
  - **Logistic neuron:** $f(x) = \sigma(x)$ (sigmoid function)
- What model do we obtain with a linear neuron?
  - Linear regression
- And with a logistic neuron?
  - Logistic regression
- More types:
  - **Tanh:** maps to [-1,1]
  - **ReLu:** $f(x) = max(0,x)$
  - Etc.
- By adjusting weights:
  - We learn functions!



Feed-forward Neural Networks

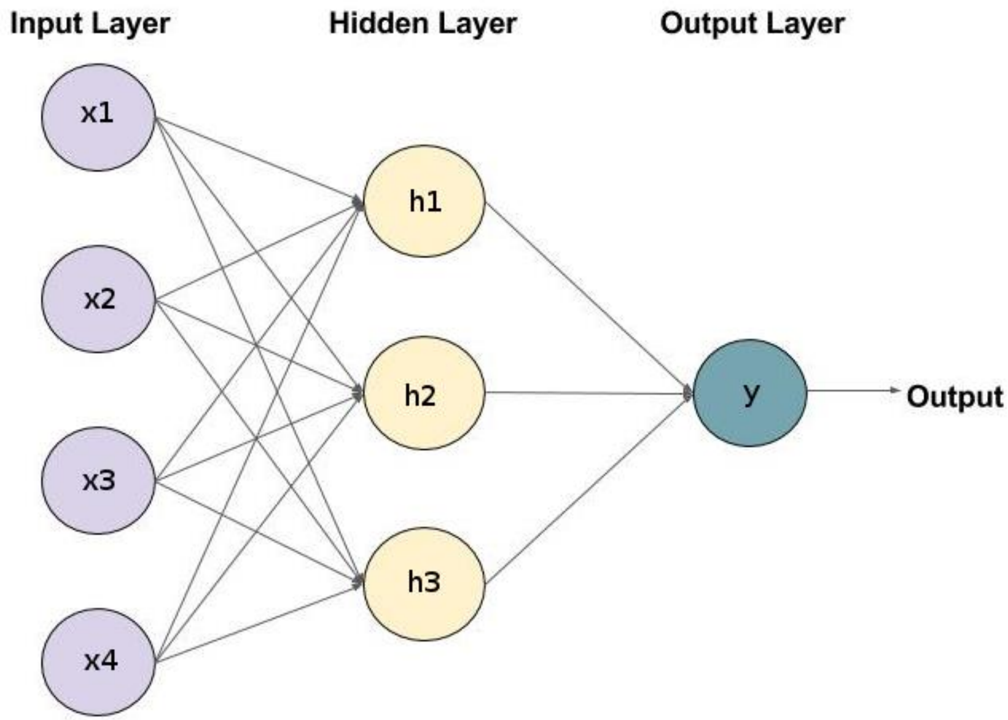
- Let's do some **feature learning**!
  - We use *hidden* layers
  - Input to h_i: ***x* (our data)**
  - h_i: learned values
  - Input to y: ***h* (learned!)**
- If y is linear neuron:
  - Linear regression with learned features
- If y is logistic neuron:
  - Logistic regression with learned features



- Thus, FNNs are a framework for *designing* functions (*algebraic circuits*)
  - Information flows forward (but also backward during training, more later)
  - Main restriction: **no loops! Computations** seen as **directed acyclic graphs**

Operations in FNNs

- Example so far: **fully-connected neural network** (useful intuition)
- What does it compute?
- $\boldsymbol{x}$: $n \times 1$ input vector
- $\boldsymbol{h}$: $m \times 1$ hidden representation
- h_k has n w_i, together: $\boldsymbol{w}_k$
- Let $\boldsymbol{W}_1$ be $n \times m$ matrix
  - m columns, each a $\boldsymbol{w}_i$
- Then $\boldsymbol{h} = \boldsymbol{W}_1^T\boldsymbol{x}$ if h_k linear
  - $\boldsymbol{h} = f(\boldsymbol{W}_1^T\boldsymbol{x})$ w. activ. f
  - $\boldsymbol{h} = f(\boldsymbol{W}_1^T\boldsymbol{x} + \boldsymbol{b})$ w. bias



- Similarly: $y = f(\boldsymbol{W}_2^T\boldsymbol{h} + b)$ where $\boldsymbol{W}_2$ is $m \times 1$, constructed as $\boldsymbol{W}_1$
  - Note network $y = \boldsymbol{W}_3^T\boldsymbol{x} + b$ exists with $\boldsymbol{W}_3 = \boldsymbol{W}_1\boldsymbol{W}_2$, i.e. no hidden layers
  - Why do we always use hidden layers then?

Why use Deep Networks?

- When is a network deep?
  - Not well defined
  - Some say >4 hidden layers
  - Others say >2
- More important: **why?**
- Universality theorem
  - Formalities about FNN limits to approximate functions
  - One takeaway: no depth *needed*

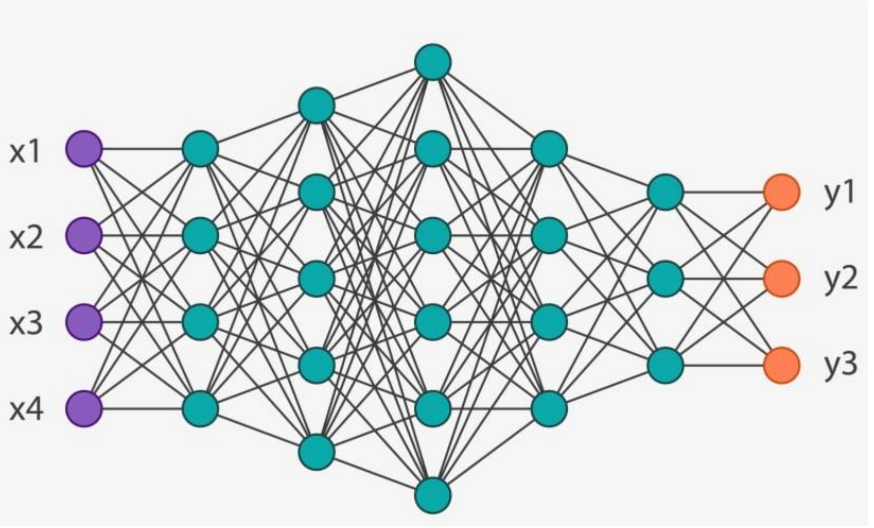


Image source

- Why use depth then? **Hidden representations!**
  - Each hidden layer is a learned feature vector, *hierarchical features*
  - Component for designing networks: number and size of hidden layers
    - Smaller layers force models to learn compact but useful representations
    - More/larger hidden layers: more freedom, more parameters, difficult to train
- More parameters -> stronger model, may overfit, requires lots of data

FNNs for Regression

- Let's design a fully-connected network for regression
- **Input:** n-dimensional $\boldsymbol{x}$
- **Output:** real number y
- So, how many input units?
  - What type of units?
- How many output units?
  - What type of units?
- How many hidden layers?
  - What size/type of units?

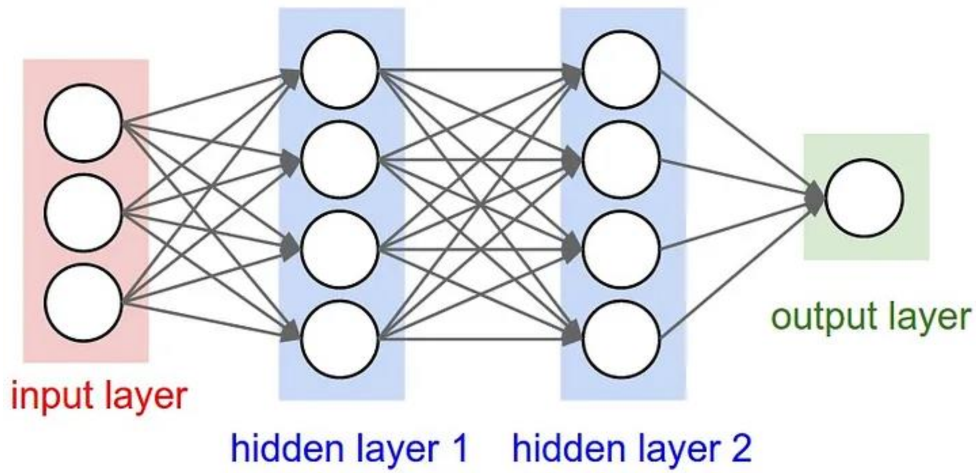


Image source

- How to ensure this number of output units? **Linear layer!**
  - *Project* down to the size that you need
  - Remember: $y = \boldsymbol{W}_k^T \boldsymbol{h} + b$ where $\boldsymbol{W}_k$ is m x n matrix
  - $\boldsymbol{W}_k^T$ projects n dimensional vector to m dimensional vector space
- **Linear transformation** are **everywhere** in Deep Learning!

FNNs for Classification

- Let's design a fully-connected network for classification
- **Input:** *n*-dimensional **x**
- **Output:** one of *K* classes
- So, how many input units?
  - What type of units?
- How many output units?
  - How do we ensure that?
  - What type of units?
- How to get predictions?
  - **Softmax layer**

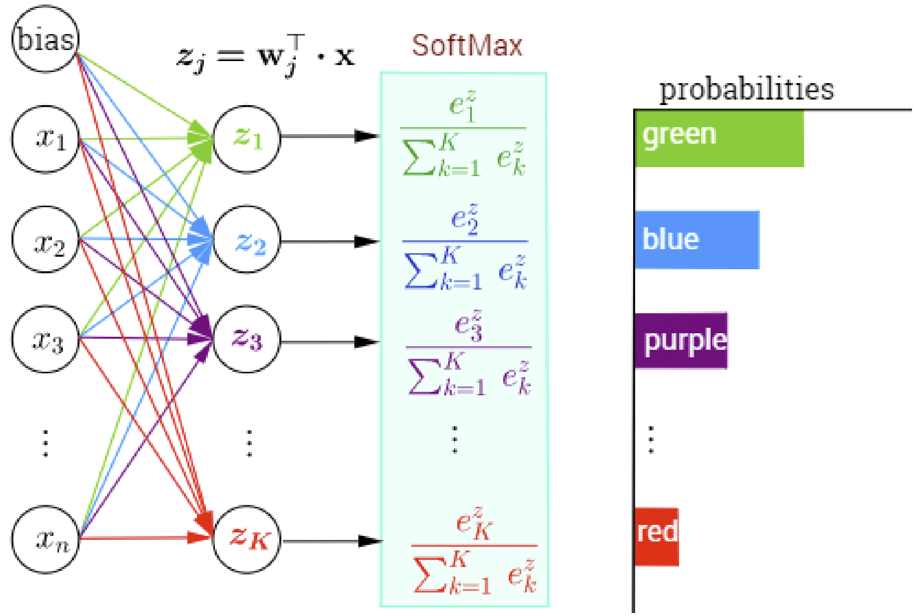


Image source

- Softmax *function*: turns given vector into *probability vector*
  - **x** is a probability vector if $x_i \in [0,1]$ for all i and $\Sigma_i x_i = 1$ (see first tutorial)
- Softmax *layer*: linear layer for projection + softmax function
  - **Input:** vector of logits; **Output:** probability vector for inference/prediction

Information Flow

- "Feed-forward neural networks: information flows forward"
  - Oversimplified! Important information flows backward *during training*
  - This influences design, inspired creation of important components
  - E.g. LSTMs, residual units, etc.
- Training often done with gradient-based optimizers
  - E.g. SGD, Adagrad, etc. (briefly discussed in first tutorial)
  - **Gradient information tells us how to update model parameters!**
- **Forward pass**
  - Compute function we are learning
- **Backward pass**
  - Compute gradient information to learn parameters
- How to compute gradients of functions computed by FNNs? **Backpropagation!**

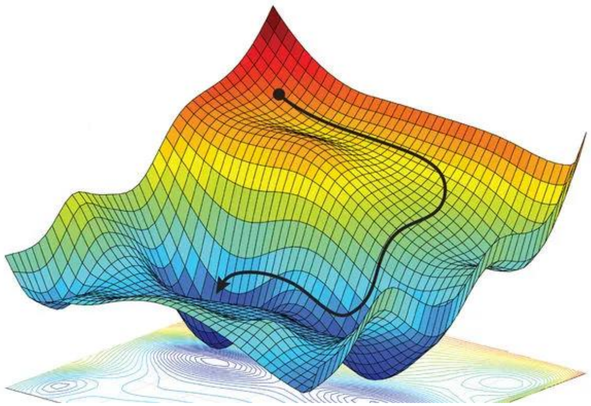


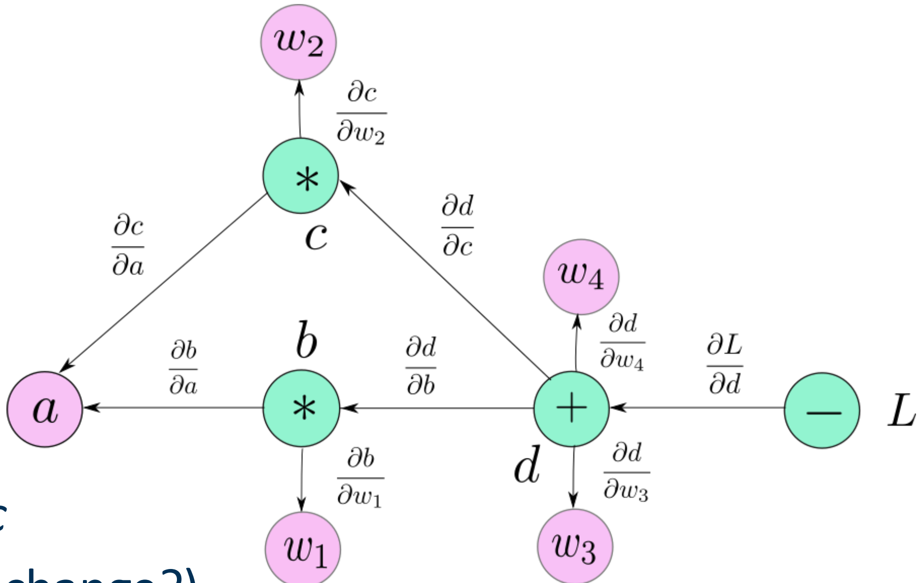
Image source

Backpropagation (1)

- What is a derivative?
  - Let $y = f(h)$
  - dy/dh asks: **how much does *y* change when *h* changes?**
- A gradient is a multidimensional derivative
  - Let $\boldsymbol{h}$ be n-dimensional vector and $y = f(\boldsymbol{h})$ a function from R^n to R
  - $\partial y/\partial \boldsymbol{h} = (\partial y/\partial h_1, \partial y/\partial h_2, \ldots, \partial y/\partial h_n)$ is gradient of y w.r.t. $\boldsymbol{x}$
  - I.e. the gradient of y w.r.t. $\boldsymbol{h}$ is also an n-dimensional (row) vector
  - Geometrically, **gradient vector is direction of the largest change of *y***
- **What rate of change are we interested in during training?**
  - How much does loss L change when model parameters $\boldsymbol{\Theta}$ change?
  - I.e. $\partial L/\partial \boldsymbol{\Theta}$
- What if L is output of a compound parameterized function (FNN)?
  - E.g. $L = f(y)$ where $\boldsymbol{y} = g(\boldsymbol{h}|\boldsymbol{\Theta}_2)$, $\boldsymbol{h} = z(\boldsymbol{x}|\boldsymbol{\Theta}_1)$ (z is 1st layer, g is 2nd, f is loss)
  - Here $\partial \boldsymbol{y}/\partial \boldsymbol{h}$ is a Jacobian ($m \times n$ matrix where m is size of $\boldsymbol{y}$, n size of $\boldsymbol{h}$)
  - Same for $\partial \boldsymbol{h}/\partial \boldsymbol{x}$

Backpropagation (2)

- We **propagate the change/error back** using the chain rule!
- Recall chain rule
  - Let $y = g(h)$ and $h = f(\Theta)$
  - $\partial y/\partial \Theta = \partial y/\partial h \cdot \partial h/\partial \Theta$
- Take this computation graph
  - **Internal nodes:** operations
  - **Leaves:** learnable weights w_i, input a
- Here $L = d - label$, $d = w_3 b + w_4 c$
- We want $\partial L/\partial w_4$ (which rate of change?)
- **Chain rule:** $\partial L/\partial w_4 = \partial L/\partial d \cdot \partial d/\partial w_4$
- Similarly, $\partial L/\partial w_2 = \partial L/\partial d \cdot \partial d/\partial c \cdot \partial c/\partial w_2$ where $c = w_2 a$
  - *Note that* $\partial L/\partial d$ is useful for both $\partial L/\partial w_4$ and $\partial L/\partial w_2$
  - **Reusing gradients is common during backpropagation**



[Image source](#)

Backpropagation (3)

- Perhaps more important, take $\partial L/\partial w_2 = \partial L/\partial d \cdot \partial d/\partial c \cdot \partial c/\partial w_2$
  - Here, gradient $\partial c/\partial w_2$ *is* multiplied by factor $\partial L/\partial d \cdot \partial d/\partial c$
  - $\partial L/\partial d$ comes from *passing gradient* through minus operator in L
  - $\partial d/\partial c$ comes from *passing gradient* through plus operator in d
- Thus, **operations have impact on their "input" gradients!**
  - Important to consider when designing networks
  - E.g. *plus* operator has no effect on gradient
  - *Concat* operator splits gradient into two
- E.g. if either $\partial d/\partial c$ or $\partial L/\partial d$ are close to zero, $\partial L/\partial w_2$ will be small
  - In other words, we get little info about how to change w_2 to improve L
  - Generally, small gradients = slow/difficult to learn
- Many challenges in *training* deep networks, e.g. vanishing gradients
  - Inspired solutions like LSTMs, residual connections (used in transformers)
- **Let's look at a FNN for a practical NLP use case!**

Language Modeling Recap

- Predict the next word:
  - "Every Thursday there is a Schneckenhof …"
- This suggests:
  - **Some words are more likely to appear than others given some context**
- Probabilistically:
  - *p(meeting|"Every Thursday there is a Schneckenhof")*
- **Generally:** $p(w_{n+1}|w_1,w_2,\ldots,w_n)$
  - Conditional probability of w_{n+1} given joint distribution of $w_1,w_2,\ldots,w_n$
  - w_i are words/tokens in some fixed vocabulary V the model can process
- **Such models can predict entire sequences with probability chain rule**
  - $p(w_1,w_2,\ldots,w_n) = p(w_1)p(w_2|w_1)\, p(w_3|w_1,w_2)\ldots\, p(w_n|w_1,w_2,\ldots,w_{n-1})$
- Useful in many NLP settings, e.g. machine translation
  - A good language model should identify more common phrases
  - p(*This sentence makes sense*) < p(*Makes sense this sentence does*)

Language Modeling with FNNs (1)

- Bengio et al. (2003) proposed a FNN that simultaneously learns:
  1. "Distributed representations of words"
  2. "Probability function of word sequences expressed in terms of these representations"
- Size of word vectors (30, 60, 100) much smaller than vocabulary (17K)
- **Intuition: similar words have similar feature vectors**
- Thus, training sentence *The cat is walking in the bedroom* can be generalized to, e.g.:
  - *A dog* is walking in a *room*
  - The cat is *running* in a *room*
  - *A dog* is walking in a bedroom
  - The dog is walking in the *room*
- **How did they implement such a model?**
  - **A feed-forward neural network**
  - Not the first time it was done, but the first time at that scale

Language Modeling with FNNs (2)

- How did they model $p(w_m | w_1, w_2 \ldots, w_{m-1})$?
  1. A mapping C between every word $w_i \in V$ to a m-dimensional vector $\boldsymbol{w}_i$
     Today known as an **embedding layer**.
     Given w_i, get row i in embedding matrix $\boldsymbol{C}$ (V rows, each a vector $\mathbf{w}_\mathrm{i}$)
  1. Function g that maps input sequence of word vectors to a probability vector of size $|V|$, i-th component is probability of i-th word being next
- **Embeddings are parameters that are learned**
- Function g uses these embeddings plus additional parameters $\boldsymbol{\omega}$
  - $p(w_i | w_1, w_2 \ldots, w_{m-1}) = g(i, \boldsymbol{w}_1, \boldsymbol{w}_2 \ldots, \boldsymbol{w}_{m-1} | \boldsymbol{\omega})$ where i is index of word w_i
- Overall, they learned function f, a composition of C and g, with a FNN
  - $f(i, w_1, w_2 \ldots, w_{m-1}) = g(i, C(w_1), C(w_2) \ldots, C(w_{m-1}) | \boldsymbol{\Theta})$
  - All model parameters are $\boldsymbol{\Theta} = (\boldsymbol{C}, \boldsymbol{\omega})$
- It's easy to see what C might look like, but what does g look like?
  - In other words, what did their network architecture look like?

Language Modeling with FNNs (3)

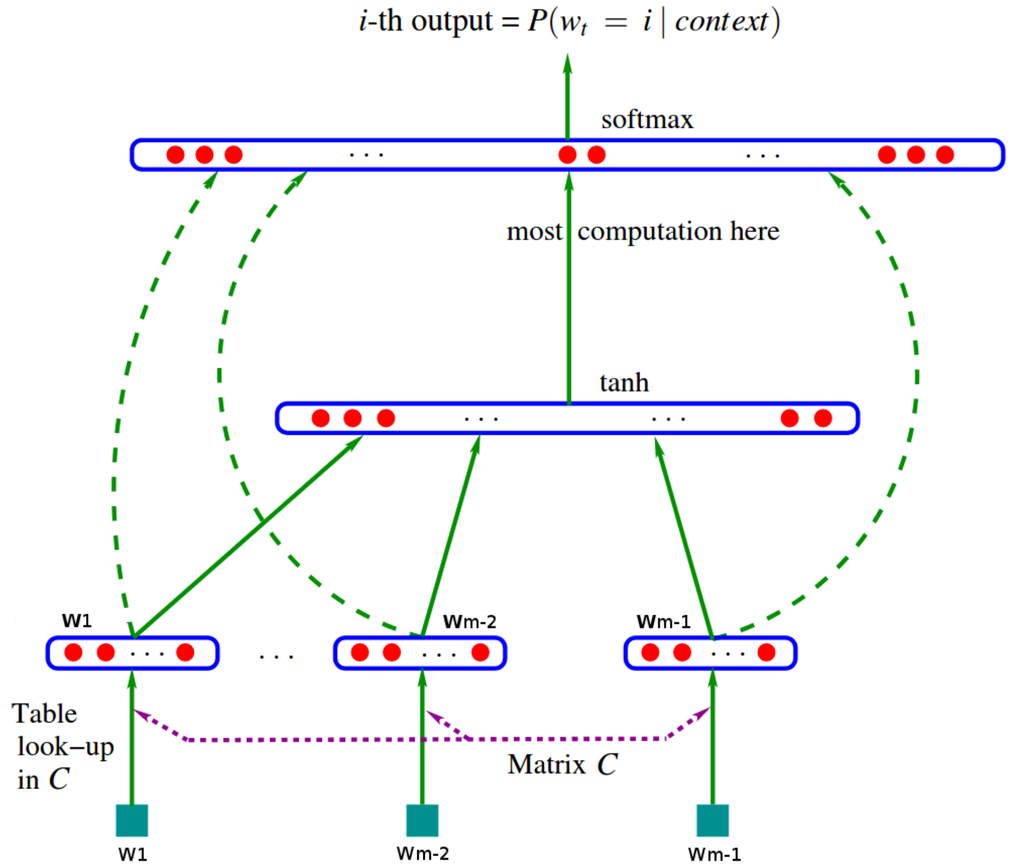
1st: Get word vectors $\boldsymbol{w_i}$

2nd: Concatenate vectors to form $\boldsymbol{x}$

3rd: Hidden layer (matrix ***H*** transform. + *tanh* activ.)

4th: *Optional* linear layer w. matrix ***W*** from input to softmax layer (dashed lines)

5th: Softmax layer (matrix ***U*** transform. + softmax)



Overall: input to softmax

- $y = \boldsymbol{b} + \boldsymbol{Wx} + \boldsymbol{U}tanh(\boldsymbol{d} + \boldsymbol{Hx})$ where ***b,d*** are biases
- Then $\boldsymbol{\Theta} = (\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{W}, \boldsymbol{U}, \boldsymbol{H}, \boldsymbol{C})$

Language Modeling with FNNs (4)

- How did they train?
  - They estimated **Θ** that maximized likelihood of training data (i.e. **MLE**)
- Specifically:
  - $L = 1/T \sum_m \log f(w_1, w_2 \ldots, w_{m-1}; \boldsymbol{\Theta}) + R(\boldsymbol{\Theta})$
  - Here, T is number of words in training corpus
- Note a few things:
  - They average loss over the training set (empirical risk)
  - Penalizing term $R(\boldsymbol{\Theta})$ is regularization (weight decay on ***C, W, H, U***)
- They trained in a **self-supervised** manner
  - I.e. yes, there are labels, but *no manual labeling* required
  - Instead, they used sequences of length *m* in the training corpus
  - For each sequence, first *m-1* words are input, word *m* is target
- **Many of these components still in use today!**
  - Softmax layer, MLE, empirical risk, self-supervision

Summary: DL Basics

- **FNNs:** framework for constructing parameterized real-valued functions
  - We design and learn these functions (algebraic circuits)
- Information flows in both directions on these circuits
  - **Forward pass:** model expressivity
  - **Backward pass:** model "learnability"
- **Backpropagation:** training deep networks, chain rule
- **Important: we can learn useful features with deep models!**
  - Hidden layers interpreted as *learned representations*
  - Learned representations useful in other settings, i.e. transfer learning
  - E.g. style transfer, pre-training word representations
- **Language modeling with FNNs** by Bengio et al. (2003)
  - Established many modeling and training components still used today
- **Later in the course**, other types of FNNs relevant for NLP
  - **RNNs, transformers**

Word Embeddings

About this Section

- **Goal:** introduce concepts and methods about static word embeddings
- **Outline:**
    1. Sparse vs Dense Word Representations
    2. Word2Vec
    3. Properties of Word Vectors
    4. Evaluation of Word Vectors

What do we want to represent? (1)

- Words? I.e. strings of symbols such as *cat* or *dog*?
  - No, those are representations themselves!
  - What do they represent?
- **Meaning!**
  - We map strings such as *cat* or *dog* to concepts (or *senses*)
  - We refer to this mapping as *meaning*, what those words refer to
- **Is meaning a 1-to-1 mapping?**
  - **No**, words can have many meanings (remember ambiguity?)
- How do we, humans, handle word ambiguity? **Context!**
  - “I work at a bank.”
  - What’s more likely? A financial institution or by the river?
  - Again, context. Does the person work in Frankfurt? Are they fishermen?
- So, **we want to represent the relation between words and senses!**

What do we want to represent? (2)

- What do we want from such representations? Ideally usefulness, e.g.
  - Tell us if words have similar meanings, e.g. cat and dog
  - Tell us if words are antonyms, e.g. hot and cold
  - Tell us if they have positive or negative connotations, e.g. happy or sad
- **"Generally, a model of word meaning should allows us to draw inferences to address meaning-related tasks like question answering."** Jurafsky and Martin, 2023
- Perhaps we should represent senses, then, not words!
  - Use something like WordNet, expert-crafted database of senses
- Databases like WordNet have not worked well in practice
  - Missing nuance, "good" is synonym of "proficient" only in some cases
  - New meanings missing, e.g. *genius*, *wicked*, *wizard*
- In practice, **dealing with word representations works better**
  - We focus on their relation to meaning (based on context)
  - We don't commit to a given representation of sense

Sparse Word Representations

- Traditional approach: sparse word vectors
- Example: **one-hot vectors**
  - Given vocabulary *V*, represent each word with vector of size *|V|* using one-hot encoding
- For example, given $V = \{cat, dog, airplane\}$
  - $V_{cat} = [1, 0, 0]$
  - $V_{dog} = [0, 1, 0]$
  - $V_{airplane} = [0, 0, 1]$
- **Problem 1:** *|V|* can be very large
  - Recall Bengio's work from 2003: $|V| = 17K$
- **Problem 2:** No natural notion of similarity from such vectors
  - They are all equally different, e.g. with a dot product
  - But *cat* and *dog* are more similar to each other than to *airplane*
- Generally, **difficult to draw inferences from such representations**
  - Let's look at another approach that addresses these issues

Distributed Representations

- Idea from deep learning community (Geoffrey Hinton in 1986)
- Sparse representations:
  - Each representational component maps to single represented object
  - E.g. each feature in vector corresponds to one word
- **Distributed (or dense) representations:**
  - Many-to-many relation between representational components and represented objects
  - E.g. each word is represented with a set of (distributed) features
- **PROs:** model has freedom to use features “at will“ during learning
- **CONs:** Less interpretable than, e.g. one-hot encoding
  - Given V, we know what word is represented by [0, 1, 0]
  - But what word is this? [0.23, 0.447, 0.02]
- Using **vectors to represent words** also has a **history in linguistics**
  - Osgood et al. (1957) represented words with 3-dimensional vectors
  - Each dimension encoded known useful (non-learned) features

Distributional Semantics (1)

- Idea from linguistics (Joos, 1950; Harris, 1954; Firth, 1957)
- Not the same concept!
  - *Distributed* representations != *distributional* semantics
  - In NLP, the latter can be seen as a special case of the former
- Let's guess what the word *foobar* means
  1. *Foobar* is often played in teams of 7.
  2. A *foobar* match is divided into 3 thirds of 25 minutes each, called runs.
  3. A *foobar* match is won by scoring 10 points.
- **What would you guess the word *foobar* refers to? Why?**
  - A type of food?
  - A sport?
  - A country?
- **Distributional hypothesis:**
  - *"A word is characterized by the company that it keeps."* (Firth, 1957)

Distributional Semantics (2)

- Put differently:
  - We define the meaning of a word by its *distribution* in a language
  - I.e. by its neighboring words, grammatical environment
- **Words whose neighboring words are similar, have similar meanings.**
- We can use this idea to build useful **vector representations** of words!
- For example: **co-occurrence matrices**
  - Given defined *context*, e.g. a document, a sentence, a word window
  - Construct matrix C of size V x V where V is given vocabulary
  - Rows in matrix are words, columns are also words
  - m_{ij} denotes number of times word i appears in same *context* as word j (so, sparse)
- Co-occurrence matrices useful in many applications
  - E.g. latent semantic analysis (LSA) --> singular value decomposition of C
  - Extracts vector representations from (truncated) left–singular vectors
  - Similar count-based methods covered in basic Text Analytics course
- **Word embeddings: distributional hypothesis to learn dense vectors**

Word Embeddings

- **Idea:** to *embed* higher dimensional vectors into lower dimensional vector space such that (some of) the relative properties of the higher dimensional space are kept
  - Hence, **embedding** --> (**relatively) low-dimensional vector representation**
  - Low-dimensional compared to, e.g. $|V|$
  - Size typically between 50 and 1000 dimensions
  - Vectors are dense (distributed) representations
- Historically important approach: **word2vec** (Mikolov et al. 2013)
  - Training based on distributional semantics
  - Done in the context of deep learning (though this model is not deep)
- Intuition:
  - **Instead of counting words for co-occurrence, let's train a classifier to predict whether two words are likely to appear in the same context.**
- Two approaches proposed for learning such a classifier
  - Skip-gram and continuous bag of words (CBOW) (we focus on skip-gram)

The Skip-Gram Algorithm

- Algorithm for learning dense word vectors
- **Basic steps**
  1. Treat target words and neighboring context words as **positive examples**
  2. Randomly sample other words in vocabulary to get **negative examples**
  3. Train a classifier to predict whether two given words appear together
  4. Use learned weights as word embeddings
- **Why negative examples?**
  - In some tasks, negative examples often needed to avoid trivial solutions
  - E.g. push all vectors to same point, thus all vectors similar
  - If training objective is poorly designed, such a solution may "reach objective"
- Let's have a look at:
  1. **The task**
  2. **The model**
  3. **The training objective**

Skip-Gram: The Task

- Say we have the sentence:
  - *… known as [ramen, a Japanese dish that] has recently become…*

    c_1 c_2 w c_3 c_4
- Here, *w* denotes **center word** and c_i denotes **context words**
  - I.e. context window is plus/minus a few words in this case
  - More context --> usually more useful, not always, more later
  - E.g. plus/minus 1 gives us *[a; ???; dish]*
  - Plus/minus 2 gives us *[ramen; a; ???; dish; that]* (assumes no punctuation)
- **Task:** given tuple *(w,c)*, predict if *c* appears in context of *w* or not
  - So, binary classification
- Probabilistically, *P(True|w,c)*
  - Similarly, *P(False|w,c) = 1 - P(True|w,c)*
- How can we model these probabilities?

Skip-Gram: The Model (1)

- How to get probability *P(True|w,c)*?
  - **Key idea:** based on similarity of corresponding word embeddings
  - I.e. a word *c* is likely to occur near a target word *w* if their corresponding embeddings $\boldsymbol{e}_c$ and $\boldsymbol{e}_w$ are similar
  - Probabilistically, *w* and *c* are independent given their embeddings $\boldsymbol{e}_c$, $\boldsymbol{e}_w$
  - Let $\boldsymbol{\Theta} = [\boldsymbol{e}_{1, \dots,} \boldsymbol{e}_{|V|}]$. We have $P(True|w,c,\boldsymbol{\Theta})$ ($\boldsymbol{\Theta}$ often omitted for brevity)
- How can we represent that similarity?
  - Skip-gram went with **dot product**, i.e. $\boldsymbol{e}_c^T \boldsymbol{e}_w$
- But didn't we need a probability?
  - Ok, then $P(True|w,c) = \sigma(\boldsymbol{e}_c^T \boldsymbol{e}_w)$ where σ is the **sigmoid function**
  - Similarly, $P(False|w,c) = 1 - \sigma(\boldsymbol{e}_c^T \boldsymbol{e}_w) = \sigma(-\boldsymbol{e}_c^T \boldsymbol{e}_w)$
  - The last equality results from a property of the logistic function
- This is the probability of word *c* being in context of word *w*.
  - **What if I want to provide more context?** More context is better, right?

Skip-Gram: The Model (2)

- How do we compute $P(True|w,c_{1:L})$ where L is size of context window?
  - **Assumption: context words are independent of each other**
  - Computing joint distribution of words in context becomes much simpler
- Thus, we have

$$P(True|w,c_{1:L}) = \prod_{i\ in\ L} \sigma(\mathbf{e}_i^T \mathbf{e}_w) = \sum_{i\ in\ L} \log \sigma(\mathbf{e}_i^T \mathbf{e}_w).$$

- **How is this model parameterized?**
  - It actually learns *two* representations for each word
  - One for words as targets, one for words as context (just one vocabulary V)
  - Thus, parameterized by matrices $\boldsymbol{W}$ and $\boldsymbol{C}$, both of size $|V| \times d$ where d is a hyperparameter (embedding size)
  - *In other words,* $\boldsymbol{\Theta} = [\boldsymbol{W}, \boldsymbol{C}]$
- How was this model trained?

Skip-Gram: The Training Objective (1)

UNIVERSITY OF MANNHEIM
School of Business Informatics and Mathematics

- As with Bengio's LM from 2003: **self-supervision**
- Given text corpus (i.e. large sequence of text)
  - Iterate over sequences of size *n = L*2 + 1* (L is size of context window)
  - For each sequence, set center word as target, other words as context
- **For example:**

1. *… [known as ramen, a Japanese] dish that has recently become…*

   c_1 c_2 w c_3 c_4

2. *… known [as ramen, a Japanese dish] that has recently become…*

   c_1 c_2 w c_3 c_4

3. *… known as [ramen, a Japanese dish that] has recently become…*

   c_1 c_2 w c_3 c_4

- Thus, **for sequences 1, 2 and 3, you get positive examples *(w,c)***:

1. (ramen, known), (ramen, as), (ramen, a), (ramen, Japanese)
2. (a, as), (a, ramen), (a, Japanese), (a, dish)
3. (Japanese, ramen), (Japanese, a), (Japanese, dish), (Japanese, that)

Skip-Gram: The Training Objective (2)

- We also need **negative examples!**
- **Given *w*, we randomly sample words from *V* as candidates for *c***
  - *For example:* (ramen, ostrich), (ramen, the), (ramen, Alaska), …
  - They generated k (hyperparameter) negative examples per positive
- Then, given (w,c_{pos}), corresponding $(w,c_{neg\ i})$, and cross entropy (CE), we have:

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

- Generally, they applied empirical risk over all training examples
  - Since this is negative log likelihood (NLL), they did MLE of the training set

Task vs Goal

- **Task:** *P(True|w,c)*
  - What do we use that task for?
  - *Real-world uses?*
- **Goal:** learn word embeddings!
  - Task designed to force model to capture properties (distributional semantics)
  - Original task was different, $P(w_{t+j}|w_t)$
- Other models have tasks and goals
  - E.g. GPT-style LLMs are autoregressive language models
  - BERT-style models are masked language models
- **Sometimes we want to use the model to make predictions**
  - **Other times, we just want their learned weights**

Other Types of Static Embeddings

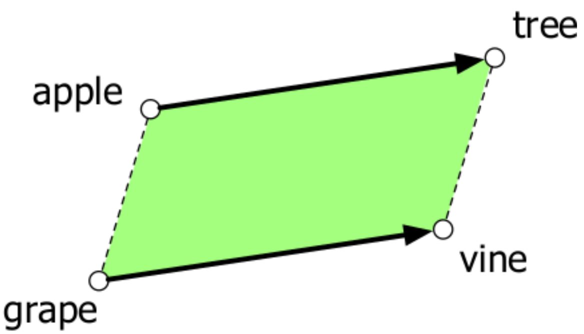
- **Word2vec is useful model for intuition/learning about embeddings**
  - It was also successful and is historically important
  - But there are other ways of learning word embeddings
  - These are often used in practice as well
- **GloVe:** stands for Global Vectors (Pennington et al., 2014)
  - Combines count-based models with methods like word2vec
  - Constructs co-occurrence matrix
  - Learns matrix factorization with least squares objective
  - Covered in basic Text Analytics course
- **fasttext:** addresses problem of unknown words in word2vec
  - That is, new words in test that were not present in training
  - For this, they used representations based on subwords
  - Also covered in basic Text Analytics course
- Static embeddings have some **nice and useful properties!**
  - At the time, they were quite impressive!

Properties of Word Embeddings (1)

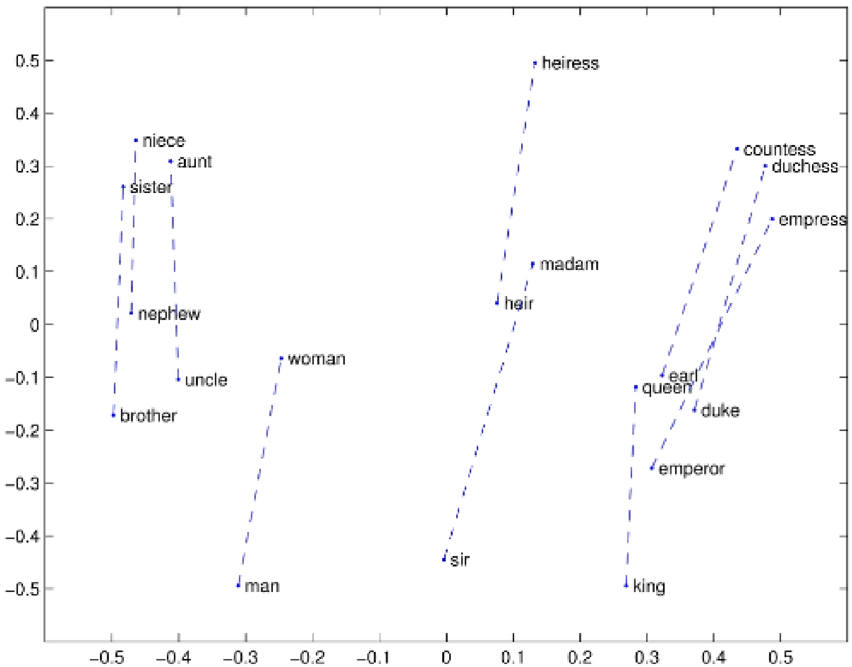
- **Size of context window is important!** (Levy and Goldberg, 2014)
  - It affects the representations we learn!
- **Smaller sizes tend to provide more static embeddings**
  - Depends on immediate neighborhood
  - Similarity tends to be based on same parts of speech
- E.g. what is similar to Hogwarts with small context window?
  - Other fictional schools, e.g. Sunnydale (from Buffy the Vampire Slayer)
- **Larger sizes tend to focus on similarity based on topic**
  - So words with same part of speech not necessarily related
- E.g. what is similar to Hogwarts with larger context window?
  - Dumbledore, Malfoy, half-blood
- **This was observed for skip-gram**
  - Different methods may differ in such properties
  - But it is safe to assume the context window has an impact

Properties of Word Embeddings (2)

- They capture relational similarity
  - Parallelogram model from cognitive science



- Example from GloVe
  - Skip-gram known to capture this
  - It’s not as smooth all the time



Evaluating Word Embeddings

- **Intrinsic evaluation:**
  - Correlation between similarities of representations with expert-given similarity scores
- Examples:
  - WordSim353 (2002): 353 noun pairs, e.g. *(plane, car)*
  - SimLex-999 (2015): more difficult, e.g. *(cup, mug)*
- Another approach: **analogy task** (e.g. *man + woman – king = queen*)
  - Word2vec used this in their original work
- **Most important evaluation: extrinsic!**
  - I.e. using the representations in NLP *downstream tasks* and see if performance improves

Summary: Word Embeddings

- **Vector representations** of words
- **Sparse vs dense** (distributed) representations
  - PROs and CONs to each
  - Different methods to learn them (each a different task)
- **Distributional semantics**
  - Important principle
  - Commonly used to learn word embeddings, language models
  - Spoiler: also used when training LLMs
- Classic way to learn dense word representations: **word2vec**
  - Specifically, the skip-gram task, model and training objective
  - Resulting learned representations have **nice properties**
- **In the future**, other tasks for other goals
  - E.g. masked and autoregressive language models

References

- Natural Language Processing: A Machine Learning Perspective, Zhang et al, 2018
  - Chapters 2, 3, 4 and 13
- Speech and Language Processing, Jurafsky et al., 2023
  - Chapter 3, 6
- Natural Language Processing, Eisenstein, 2018
  - Chapter 14