Chapter 2 - Probability: Univariate Models

Giovani Tavares
giovanitavares@outlook.com

University of Sao Paulo — July 15, 2025

Motivation

This document contains a summary of the main topics of the second chapter of the book *Probabilistic Machine Learning: An Introduction* from Kevin P. Murphy.

1 Random Variables (Section 2.2 from the Book)

When modeling a random variable (rv) X, there are some functions and notations to memorize and interpret:

- **Probability Mass Function (pmf):** it is defined for a **discrete** rv and corresponds to the function that maps a value $X = x$ to a probability. It is usually written in lowercase as $0 \leq p(X = x) \leq 1$
- **Cumulative Distribution Function (cdf):** it is defined for both **discrete and continuous** rvs and corresponds to the function that maps a value $X = x$ to a value that computes $p(X \leq x)$
- **Probability Density Function (pdf):** it is defined for **continuous** rvs and corresponds to the derivative of its cdf.

1.1 Probability Density Function (pdf)

Definition 1.1 (Probability Density Function)**.** *Given a continuous random variable X, its probability density function $P(X = x)$ is given by:*

$$p(x) = \frac{d}{dx}P(x) \tag{1}$$

such that its cumulative distribution function is given by (2)

$$P(a \leq x \leq b) = \int_a^b p(x)\, dx \tag{3}$$

Observations:

- The **pdf** is basically the **derivative** of the **cdf**
- The **pdf** is not defined where the **cdf** does not have a derivative

1.2 Quantiles/ Inverse CDF/ Percent Point Function (ppf)

The $quantile$ function (also called the Percent Point Function or simply ppf) is defined as the inverse function of the cdf. This means that given a random variable X's cumulative distibution function $CDF(X \leq x)$, its PPF is given by $PPF(q) = CDF^{-1}(X \leq x_q)$. The output of the PPF function is interpreted as the value $X = x_q$ such that $P(X \leq x_q) = q$.

Let's consider the standard normal random variable $X \sim \mathcal{N}(0, 1)$. Suppose we want to calculate $PPF(0.025) = x_q$, i.e., we want to find x_q such that $CDF(X \leq x_q) = 0.025$. We know that CDF is the integral of the PDF, i.e., the area under the curve representing the probability density of X. Hence, we're looking for find x_q such that $CDF(X \leq x_q) = \int_{-\inf}^{x_q} p(x)\,dx = 0.025$. This means that we're looking for the value in the x-axis for which the integral of the plot adds up to 0.05, which represents 2.5% of the total area under the curve. We usually call the PPF input $\frac{\alpha}{2}$ (*alpha* over two).

For $X \sim \mathcal{N}(0,1)$, $CDF(X \leq x_q) = \int_{-\inf}^{x_q} p(x)\,dx = 0.025 \implies x_q = -1.96$, i.e., $PPF(0.025) = -1.96$, the red value in the plot below. The blue value is inferred because of the normal distribution symmetry. Hence, if we are interpreting the plot from a Baysian perspective, for a standard normal distributed rv, there is a 2.5% chance that the observed value of it will be less than -1.96. On the other hand, the frequentist approach would say that in a long run, 2.5% of the total observations will be of values less or equal to -1.96.

The Normal Distribution's PDF is an even function. This means that $PDF(x) = PDF(-x)$. Hence, $PPF(\frac{\alpha}{2}) = x_q \implies PPF(1 - \frac{\alpha}{2}) = -x_q$.

It is easy to see that the interval $(-1.96, 1.96)$ contains 95% of the standard normal's cumulative probability, which means that 95% of $X \sim \mathcal{N}(0,1)$ will fall within this interval.

We can generalize this result for non-standard normally distributed rvs. If $X \sim \mathcal{N}(\mu, \sigma^2)$, then the 95% interval becomes $(\mu - 1.96\sigma, \mu + 1.96\sigma)$, which is generally rounded to $\mu \pm \sigma$.

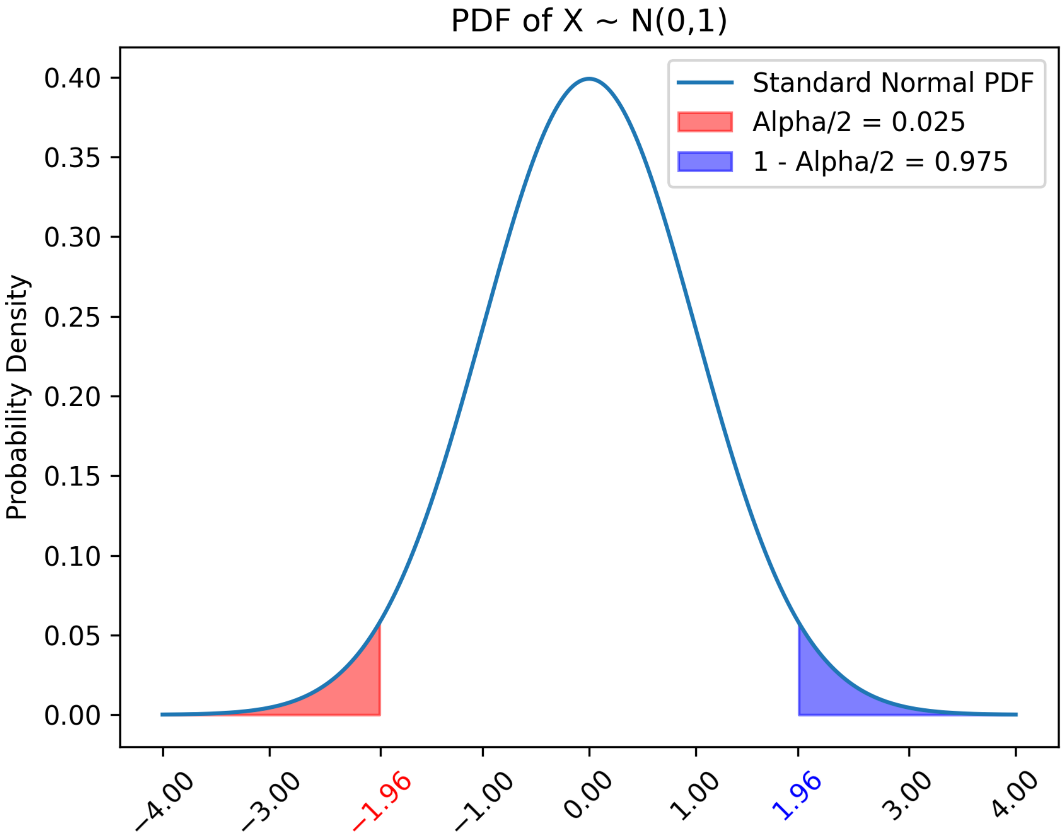

Figure 1: A visualization of the Normal Distribution's symmetric quantiles

1.3 Sets of Related Random Variables

Suppose we have two rvs X and Y. We can define the **joint distribution** $p(x, y) = p(X = x, Y = y)$. If X and Y have finite cardinality, one can represent their joint distribution in a 2D table:

Table 1: Joint Distribution of Two Binary Random Variables X and Y

	$Y = 0$	$Y = 1$
$X = 0$	0.3	0.2
$X = 1$	0.2	0.3

If we are interested in only one of the variables distribution, i.e., we want one of the varibles **marginal**, we can use the follwing rule:

Definition 1.2 (Sum Rule/ Rule of Total Probability)**.** *Given a joint distribution function $p(x, y)$, the marginal $p(x)$ is given by:*

$$p(x) = \sum_{i=1}^{n} p(x, Y = y_i) \tag{4}$$

The marginal for y (i.e., $p(y)$) is defined exactly the same way.

If X and Y are independent variables, which is written as $X \perp Y$, then $p(x, y)$ can be defined as the product of the marginals.

Definition 1.3 (Joint Distribution of Independent Random Variables)**.** *If X and Y are two independent random variables, which is written like $X \perp Y$ then:*

$$\begin{align} p(x, y) = p(x)p(y) \tag{5} \\ \textit{which is a product of two vectors} \tag{6} \end{align}$$

On the other hand, is far more common that variables are indeed dependent on each other, so that the definition above cannot be used to define their joint distribution. Hence, we use the $productrule$ instead:

Definition 1.4 (Product Rule)**.** *If X and Y are two random variables, then the product rule says that their joint distribution is given by:*

$$\begin{align} p(x, y) = p(x)p(y|x) \tag{7} \\ \tag{8} \end{align}$$

We might generalize the joint distribution even further for the cases in which we have a set of random variables and we are interested to study their simultaneous behavior.

Definition 1.5 (Chain Rule of Probability)**.** *Given a set of D random variables denoted by $X_1, X_2, \ldots, X_D$, one can define their joint distribution like the following:*

$$\begin{align} p(x_{1:D}) = p(x)p(x_2|x_1)p(x_3|x_2, x_1) \ldots p(x_D|x_{D-1}, \ldots, x_1) \tag{9} \\ \tag{10} \end{align}$$

A set of random variables $X_1, \ldots, X_D$ is said to be **mutually independent** if the joint distribution for any subset $\{X_1, \ldots, X_m\} \subseteq \{X_1, \ldots, X_D\}$ can be written as a product of the marginals.

Definition 1.6 (Mutual Independence)**.** *A set of D random variables $\{X_1, X_2, \ldots, X_D\}$, is said to be mutually independent iff:*

$$\begin{align} p(x_1, x_2, \ldots, x_m) = p(x_{1:m}) = p(x_1)p(x_2) \ldots p(x_m) \tag{11} \\ \textit{for any } m \leq D \tag{12} \end{align}$$

As previously mentioned, a set of variables is rarely mutually independent, which means that the **chain rule of probability** is usually used to calculate joint distributions of their subsets. Intuitivelly, what this means is that it is very common that one variable influences one-another in a set of rvs. On the other hand, one can often describe X's influence on Y by using another rv Z that medidates the influence such that the X and Y become **conditionally independent**.

Definition 1.7 (Conditional Independence). *X and Y are said to be* ***conditionally independent*** *on Z, which is denoted by $X \perp Y|Z$, iff:*

$$p(X = x, Y = y|Z = z) = p(x|z)p(y|z) \tag{13}$$

1.4 Bayes' Rule

Definition 1.8 (Bayes Rule). *The Bayes' Rule is simply a formula for computing the distribution of a hidden state H given some observed data Y :*

$$p(H = h|Y = y) = \frac{p(Y = y|H = h)p(H = h)}{p(Y = y)} \tag{14}$$

This follows directly from the previously defined product rule (1.4).

There are some important names for the terms of the Bayes' Rule formula:

- $p(H = h|Y = y)$: is called the **posterior distribution**, because it is the distribution of H after observing a certain value of Y.
- $p(Y = y|H = h)$: is called the **likelihood** and is a function of h since y is fixed. It measures how likely it is for $Y = y$ given that $H = h$. Notice this is not a probability distribution because it does not sum up to 1 for all the differen values of h.
- $p(H = h)$: is called the **prior** distribution and is simply the distribution of H before any knowledge about Y is given.
- $p(Y = y)$: is called the **marginal likelihood** and can be calculated using the **sum rule/rule of total probability**, since $p(Y = y) = \sum_{i=1}^{n} p(y, H = h_i)$.

> **Info:** We call **Bayesian Inference** the act of passing from sample data to generalizations (inference) with calculations of certainty given by the *Bayes' Rule*. Hence, when we perform this type of inference, we use a set of observed samples (training set) to tell the shape of the posterior distribution of a variable of interest H as a means to be able to qualify its possible outcomes quantitatively in terms of probabilities given an observation of the world $Y = y$.

1.4.1 The Monty Hall Problem

Question 1

Suppose there is a contestant in a TV show in which there is a hidden prize behind only one of three distinct doors. In the beggining of the show, the contestant must choose one of the three doors and as soon as this choice is made, the host opens one of the remaining two doors **without revealing the prize location**. After such door is opened, the contestant must make a last choice: to keep their first door as the final choice or two switch it by the door that has not been opened by the host. **Which last choice maximixes the probability that the contestant wins the final prize?**

To solve this problem, we first need to model it. Let's say the contestant first selects door 1. Let's define some variables.

- $C = c$ is the event when the contestant selects door c at the end, $i \in \{1, 2, 3\}$
- $H = h$ is the event when the TV host selects door $h \in \{1, 2, 3\}$
- $Y = y$ is the event when the prize is behind door $y \in \{1, 2, 3\}$
- $p(Y = y|H = h)$ is the probability we are interested to maximize. This means we want to the contestant to make a final choice for the door y that is the most likely to have the prize given that the host selected door h.

Suppose $C = 1$. We can skip the calculation for $p(Y = y|H = c)$ and $p(Y = y|H = y)$, because they are certainly undefined and equal to 0, respectively. This leaves us with:

$$\begin{align}
p(Y=1|H=2) &= \frac{p(H=2|Y=1)p(Y=1)}{p(H=2)} = \frac{(1/2)(1/3)}{(1/2)} = 1/3 \tag{15}\\
p(Y=1|H=3) &= \frac{p(H=3|Y=1)p(Y=1)}{p(H=3)} = \frac{(1/2)(1/3)}{(1/2)} = 1/3 \tag{16}\\
p(Y=2|H=3) &= \frac{p(H=3|Y=2)p(Y=2)}{p(H=3)} = \frac{(1)(1/3)}{(1/2)} = 2/3 \tag{17}\\
p(Y=3|H=2) &= \frac{p(H=2|Y=3)p(Y=3)}{p(H=2)} = \frac{(1)(1/3)}{(1/2)} = 2/3 \tag{18}\\
& \tag{19}
\end{align}$$

Above, we see that $p(Y = 2|H = 3)$ and $p(Y = 3|H = 2)$ are the posteriors with the largest values. They are modeling the probability that the prize is behind the door that has not been chosen by either the TV host or by the contestant at the begining, which is $C = 1$. The initial choice of $C = 1$ is irrelevant here and what matters is that changing the choice of the door yelds the maximum probability of winning the prize. **Hence, the contestant should switch their choice after the host opens a door as it will lead to the event in which the probability of winning is the largest.**

1.5 The Univariate Normal Distribution

1.5.1 Cumulative Distribution Function (cdf)

It is the function that tells the probability of a normally distributed random variable $\mathbf{Y}$ to be observed inside a specific interval.

Definition 1.9 (Univariate Gaussian Cumulative Distribution Function (**cdf**)).

$$\begin{align}
p(y_1 \leq Y \leq y_2) &= \int_{y_1}^{y_2} \mathcal{N}(y; \mu, \sigma^2)\, dy \tag{20}\\
&= p(y_2) - p(y_1) \tag{21}
\end{align}$$

Definition 1.10 (Univariate Gaussian Probability Density Function ($\mathbf{pdf}$)).

$$p(Y = y) = \mathcal{N}(y; \mu, \sigma^2) \tag{22}$$

$$= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2} \tag{23}$$

In the $\mathbf{pdf}$ of the univariate gaussian, $\sqrt{2\pi\sigma^2}$ is the normalization constant that ensures that the $\mathbf{cdf}$ equals to 1.

1.5.2 Percent Point Function ($\mathbf{ppf}$)/ Probit Function/ Inverse of $\mathbf{cdf}$/ The quantile

The output of this function tells the observation y for which its cumulative distribution ($\mathbf{cdf}$) is equal to the input z. In other words, $\mathbf{ppf}(\mathbf{z}) = y$ iff $\mathbf{cdf}(\mathbf{y}) = z$.

For example, if $\mathbf{ppf}(\mathbf{0.5}) = y_0$ for a normally distributed rv $Y = y$, then it means that 50% of the observations of Y are less or equal to y_0.

1.6 The Binomial Distribution

It is a distribution that models a sequence of independent binary events. The rv distributed as a $binomial$ basically tells how much of the events in the sequence was a positive.

Definition 1.11 (Binomial Distribution). *Let Y $Bin(\theta, N)$ where N is the sequence length, i.e., the number of bianary events. Hence:*

$$P(Y = y) = \binom{N}{s} \theta^s (1-\theta)^{N-s} \tag{24}$$

$$E(Y) = N\theta \tag{25}$$

$$Var(Y) = N\theta(1-\theta) \tag{26}$$

This is so, because $p(Y = y) = \theta$ and we are dealing with a binary event, so there is a θ probability that the event is positive among N observations.

1.6.1 The Bernoulli Distribution

The Bernoutlli distribution is denoted by $Ber(\theta)$ and it is a special case of the binomial distribution in which the sequence of events hs length 1. Hence:

Definition 1.12 (Bernoulli Distribution). *Let Y $Ber(\theta)$.*

$$p(Y = y) = \begin{cases} \theta & y = 1 \\ 1-\theta & y = 0 \end{cases} = Bin(1, \theta) \tag{27}$$

This is so, because $p(Y = y) = \theta$ and we are dealing with a binary event, so there is a θ probability that the event is positive among N observations.

1.7 The Poisson Distribution

It is a distribution for discrete rv's that account the number of binary events in a fixed period of time under the following assumptions:

- The events in a given period of time are independent of each other
- The average number of positive events in a given period of time is fixed regardless of the beginning of the period

Definition 1.13 (Poisson Distribution). *Let Y $Poisson(\lambda)$, where λ is the average number of positive events in a given period of time. Hence:*

$$p(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!} \tag{28}$$

$$E(Y) = Var(Y) = \lambda \tag{29}$$

The expected value and the variance of a poisson distributed rv are equal to the mean number of positive events in a given period of time, i.e., λ.

The Poisson Distribution closely approximates the Binomial Distributions for large values of N and small values of θ. For example, the distribution of the number of decayed atoms in a set of atoms in a fixed period of time can be assumed to be a Poisson distribution where λ is the average number of atoms that decay in the same period of time. On the other hand, such number N of atoms is very large and the probability of a single one to decay is small, so one can approximate the number s of atoms that decay in the same period of time as a Binomial Distribution.

1.8 The Geometric Distribution

It is a distribution for discrevete rv's that counts the number of Bernoulli trials until the first positive event happens.

Definition 1.14 (Geometric Distribution). *Let $Y\ Geo(\theta)$, where θ is the probability of sucess.*

$$p(Y = y) = \theta^{y-1}(1-\theta) \tag{30}$$

$$E(Y) = \frac{1}{\theta} \tag{31}$$

$$Var(Y) = \frac{1-p}{p^2} \tag{32}$$

2 Linear Algebra

2.1 Distances

Definition 2.1 (Distance Point-Point). *Let P and Q be two points in $\mathbb{R}^n$. Hence, the distance between P and Q is such that:*

$$\textbf{dist}(P, Q) = ||P - Q||_2 \tag{33}$$

***Scalar form in** $\mathbb{R}^2$**:** If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, then:*

$$\textbf{dist}(P, Q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{34}$$

Definition 2.2 (Distance Point-Plane). *Let P be a points in $\mathbb{R}^n$, Σ be a plane in $\mathbb{R}^n$ with normal vector $\vec{n_\Sigma}$ and Q be any point in Σ. We know the distance from P to Σ is such that:*

$$\textbf{dist}(P, \Sigma) = \frac{|\vec{PQ} \cdot \vec{n_\Sigma}\,|}{||\vec{n_\Sigma}||_2} \tag{35}$$

***Scalar form in** $\mathbb{R}^3$**:** If $P = (x_0, y_0, z_0)$ and the plane Σ has equation $ax + by + cz + d = 0$, then:*

$$\textbf{dist}(P, \Sigma) = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{36}$$

Definition 2.3 (Distance Point-Line). *Let P be a points in $\mathbb{R}^n$, L be a line in $\mathbb{R}^n$ defined by $L : \vec{r}(t) = Q + t\vec{u}$. Then, the distance from P to L is such that:*

$$\textbf{dist}(P, L) = \frac{|\vec{PQ} \times \vec{u}\,|}{||\vec{u}||_2} \tag{37}$$

This is so, because $\vec{PQ} \times \vec{u}$ gives the area of the paralelogram with base equal to $||\vec{u}||_2$ for which the height is exactly the distance between P and L.

***Scalar form in** $\mathbb{R}^2$**:** If $P = (x_0, y_0)$ and the line L has equation $ax + by + c = 0$, then:*

$$\textbf{dist}(P, L) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \tag{38}$$

Definition 2.4 (Distance Line-Line). *Let L_1 be a line in $\mathbb{R}^n$ defined by $L_1 : \vec{r}(t) = P + t\vec{u}$ and L_2 another line in $\mathbb{R}^n$ defined by $L_2 : \vec{r}(t) = Q + t\vec{v}$ such that L_1 and L_2 do not have any intersection. Then, the distance from L_1 to L_2 is such that*

$$\mathbf{dist}(L_1, L_2) = \frac{|\vec{PQ} \cdot (\vec{u} \times \vec{v})\ |}{||(\vec{u} \times \vec{v})||_2} \tag{39}$$

This is so, because the projection of $\vec{PQ}$ onto $\vec{u} \times \vec{v}$, which is a vector perpendicular to both L_1 and L_2 gives the desired distance.

***Scalar form in** $\mathbb{R}^2$: If line L_1 has equation $a_1x+b_1y+c_1 = 0$ and line L_2 has equation $a_2x+b_2y+c_2 = 0$, and the lines are parallel, then:*

$$\mathbf{dist}(L_1, L_2) = \frac{|c_1 - c_2 \cdot \frac{\sqrt{a_1^2+b_1^2}}{\sqrt{a_2^2+b_2^2}}|}{\sqrt{a_1^2 + b_1^2}} \tag{40}$$

Or more simply, if both equations are normalized so that $a_1 = a_2 = a$ and $b_1 = b_2 = b$:

$$\mathbf{dist}(L_1, L_2) = \frac{|c_1 - c_2|}{\sqrt{a^2 + b^2}} \tag{41}$$

2.2 Vector Spaces

Definition 2.5 (Rank of a Matrix). *Let $A \in \mathbb{R}^{m\times n}$ be a matrix. The rank of A is denoted by $\mathbf{rank}(A)$ and is defined as the number of linearly independent rows or columns. This means that the number of linearly independent rows of a matrix is equal to the number of linearly independent columns.*

Definition 2.6 (Range and Nullspace of a Matrix). *Let $A \in \mathbb{R}^{m\times n}$ be a matrix. The **range** of A is denoted by $\mathbf{range}(A)$ and is defined as the vector space generated by its columns, i.e., the range of A is the **span** of its column space. In other words, $\mathbf{range}(A)$ is the vector space with the columns of A as its basis. This means that such vector space will be made of vectors with m dimensions, as m is the number of dimensions of each column of A.*

$$\mathbf{range(A)} = \{\mathbf{v} \in \mathbb{R}^m | \mathbf{Ax} = \mathbf{v}, \mathbf{x} \in \mathbb{R}^n\} \tag{42}$$

*The **nullspace** of A is the set of vectors in $\mathbb{R}^n$ such that $\mathbf{Ax} = \mathbf{0}$.*

$$\mathbf{nullspace(A)} = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{Ax} = \mathbf{0}\} \tag{43}$$

Definition 2.7 (Linear Projection). *Let $A = \{\mathbf{x_1}, \ldots, \mathbf{x_n} | \mathbf{x_i} \in \mathbb{R}^m, i = 1, \ldots n\}$ be a set of $m - dimensional$ vectors. The **linear projection** of a vector $\mathbf{y} \in \mathbb{R}^m$ onto the span of A is defined as the vector $\mathbf{v}$ for which the Euclidean Norm $||y - v||_2$ is the smallest possible. Hence:*

$$\mathbf{proj(y, A)} = argmin_{v \in \mathbf{span(A)}} ||y - v||_2 \tag{44}$$

***Observation:** To find the projection of a vector $\mathbf{y}$ onto the span S of vectors, one first need to find the orthonormal basis for such span using a technic like **Gram-Schmidt** presented in the following subsection. Then, the projection of $\mathbf{y}$ onto S will be the sum of the projection of $\mathbf{y}$ onto each of the vectors of the orthonormal basis.*

2.2.1 The Gram-Schmidt Process for Orthogornalization

Let $A = \{\mathbf{x_1}, \ldots, \mathbf{x_n} | \mathbf{x_i} \in \mathbb{R}^m, i = 1, \ldots n\}$ be a set of $m - dimensional$ linerarly independent vectors. Let $S = \{\mathbf{v_1}, \ldots, \mathbf{v_n} | \mathbf{v_i} \in \mathbb{R}^m, i = 1, \ldots n\}$ be an orthogonal basis for $\mathbb{R}^n$. The **Gram-Schmidt Process** defines S's vectors as:

$$\mathbf{v_1} = \mathbf{x_1} \tag{45}$$
$$\mathbf{v_2} = \mathbf{x_2} - \mathbf{proj}(\mathbf{x_2}, \mathbf{v_1}) \tag{46}$$
$$\mathbf{v_3} = \mathbf{x_3} - \mathbf{proj}(\mathbf{x_3}, \mathbf{v_1}) - \mathbf{proj}(\mathbf{x_3}, \mathbf{v_2}) \tag{47}$$
$$\vdots \tag{48}$$
$$\mathbf{v_i} = \mathbf{x_i} - \mathbf{proj}(\mathbf{x_i}, \mathbf{v_1}) - \mathbf{proj}(\mathbf{x_i}, \mathbf{v_2}) - \ldots - \mathbf{proj}(\mathbf{x_i}, \mathbf{v_{i-1}}) \tag{49}$$
$$\vdots \tag{50}$$
$$\mathbf{v_n} = \mathbf{x_n} - \sum_{i=1}^{n-1} \mathbf{proj}(\mathbf{x_n}, \mathbf{v_i}) \tag{51}$$

Such that $\mathbf{proj}(\mathbf{x_i}, \mathbf{v_j}) = \frac{\mathbf{x_i} \cdot \mathbf{v_j}}{||\mathbf{v_j}||_2} \mathbf{v_j}$.

2.3 Eigenvalues and Eigenvectors

2.4 What Properties Should Sinusoidal Embeddings (SE) Should Apply?

1. **Periodicity:** the model trained with sinusoidal embeddings should be able to capture the relative positions of tokens effectively, which means that the distances between the embeddings of a pair of clauses should not depend on their absolute position within the longer sentence. This is achieved by making the sinusoidal embeddings functions periodic, which was to be expected from the name of the function.

2. **Unique Representation (Injective Function):** this one is straight forward from the fact that sinusoidal embeddings must represent the positions of tokens within a sentence. This means that sets of tokens in different positions should not be mapped to the same output, otherwise different positions would have the same representation.

3. **Scale Invariance:** sinusoidal embeddings should represent tokens positions consistently regardless of the sequence length. This property is crucial for handling sequences of varying lengths in a transformer model including those that are longer than the ones the model were trained on. Essentialy, the scale invariance property says that the distance of two inputs x_t and x_{t-k} should be similar among different values of t. In other words, $x_t - x_{t-k}$ should not depend on t.

4. **Linearity:** this property is somewhat related to the scale invariance property. For the Sinusoidal Embedding function to be linear is good, because functions having such property are more easily learned by neural networks. Moreover, if the Sinusoidal Embedding Function is linear we also achieve the scale invariance property. This is true, because if such Sinusoidal Embedding function (SE) is linear, for any pair of sets of tokens separated by a distance of k, say X_t, X_{t+k}, there is a linear transformation M such that $M \times SE(X_t) = SE(X_{t+k})$. Hence, in order to represent X_{t+k}, the model must only learn the linear transformation M regardless of the value of k.

2.5 How are Sinusoidal Embeddings Defined?

The author's of the *Attention is All You Need* paper define the Sinusoidal Embeddings Function (SE) like the following.

Definition 2.8 (Sinusoidal Embeddings). *For a position pos in the sequence and a dimension i (where i ranges from 0 to $\frac{d}{2} - 1$), the embedding is given by:*

$$SE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{52}$$
$$SE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{53}$$
$$\tag{54}$$

This function can be vectorially expressed as the following:

$$\mathbf{SE}(pos) = \left[\sin\left(pos \cdot e^{-\frac{2i\ln(10000)}{d}}\right), \cos\left(pos \cdot e^{-\frac{2i\ln(10000)}{d}}\right)\right]_{i=0}^{\frac{d}{2}-1} \tag{55}$$

where:

- pos is the position in the sequence.
- i is the dimension index.
- d is the dimensionality of the embeddings.

Notice that SE is essentialy a function that outputs sin values to the even dimensions of an input x_t and cos for the odd ones with exponentially decreasing frequencies.

In the next section we will use the definition of SE function to verify the validity of the previous 4 properties it should have. Some verification will be done analitically and others will be done using $Python$.

3 Properties Verification

In this section, we will be checking each of the 4 presented properties of the Sinusoidal Embeddings using analytical techniques.

3.1 Peridiocity

This peridiocity of the SE function is straight-forward. Different dimensions i and $i+k$ of an input $x_t \in \mathbb{R}^d$ with position pos will have the same values output by the sin and cos because such functions are periodic.

> **Info:** One question that a reader might have is whether the peridiocity property does not contradict the injectivity one. The answer for that is no. The peridiocity property is observed at a single dimension level, which means $SE_{(pos,2i)}$ that outputs a single value, is be periodic. The injectivity, on the other hand, is observed at the entire embedding level, which means $SE_{(pos)}$, that outputs a vector (embedding) of dimension d is injective.

3.2 Unique Representation (Injective Function)

We are gonna prove that two sequences from different positions are **not** mapped to the same vector/output, using a *proof by contradiction*. Let's assume that two sequences x_{t_1} and x_{t_2}, with different positions t_1 and t_2, respectivelly, are mapped to the same output using the previously SE function. This assumption implies the following **for any dimension i**

$$SE(x_{t_1}) = SE(x_{t_2}) \implies SE(t_1, 2i) = SE(t_2, 2i) \text{ and } SE(t_1, 2i+1) = SE(t_2, 2i+1) \tag{56}$$

$$\implies \sin\left(\frac{t_1}{10000^{\frac{2i}{d}}}\right) = \sin\left(\frac{t_2}{10000^{\frac{2i}{d}}}\right) \text{ and } \cos\left(\frac{t_1}{10000^{\frac{2i}{d}}}\right) = \cos\left(\frac{t_2}{10000^{\frac{2i}{d}}}\right) \tag{57}$$

For the last implication to be true, either the arguments of the sin and cos functions are the exact same or they are separated by $2\pi k$. We know they are not the same, because $t_1 \neq t_2$. Therefore, we're left with the condition:

$$|\frac{t_1}{10000^{\frac{2i}{d}}} - \frac{t_2}{10000^{\frac{2i}{d}}}| = 2\pi k \tag{58}$$

$$\implies |\frac{t_1 - t_2}{10000^{\frac{2i}{d}}}| = 2\pi k \tag{59}$$

$$\implies |t_1 - t_2| = 2\pi k 10000^{\frac{2i}{d}} \tag{60}$$

$$\tag{61}$$

Since t_1 and t_2 are integers that represent the sequences positions, and $10000^{\frac{2i}{d}}$ is a positive real number, the right side of the equation $|t_1 - t_2| = 2\pi k \cdot 10000^{\frac{2i}{d}}$ must also be an integer. However, $2\pi k \cdot 10000^{\frac{2i}{d}}$ is generally not an integer because 2π is an irrational number, which leads us to a contradiction.

Hence, the initial assumption $SE(x_{t_1}) = SE(x_{t_2})$ must be false, which let's us say that **two sequences x_{t_1} and x_{t_2}, with different positions t_1 and t_2, respectivelly, are not mapped to the same output using the previously SE function**.

3.3 Linearity & Scale Invariance

As previously mentioned, the scale invariance property is a consequence of $SE(pos)$'s linearity. Hence, by proving that $SE(pos)$ is linear we also prove that such function is scale invariant.

We need to find $M \in \mathbb{R}^{d \times d}$ such that $M \times SE(x_t) = SE(x_{t+k})$. We have the following system of equations:

$$\begin{bmatrix} m_{00} & m_{01} & \cdots & m_{0(d-1)} \\ m_{10} & m_{11} & \cdots & m_{1(d-1)} \\ \vdots & \vdots & \ddots & \vdots \\ m_{(d-1)0} & m_{(d-1)1} & \cdots & m_{(d-1)(d-1)} \end{bmatrix} \begin{bmatrix} sin(\omega_0 t) \\ cos(\omega_0 t) \\ \vdots \\ sin(\omega_{(d-2)/2} t) \\ cos(\omega_{(d-2)/2} t) \end{bmatrix} = \begin{bmatrix} sin(\omega_0 (t+k)) \\ cos(\omega_0 (t+k)) \\ \vdots \\ sin(\omega_{(d-2)/2} (t+k)) \\ cos(\omega_{(d-2)/2} (t+k)) \end{bmatrix} \tag{62}$$

$$m_{00} \sin(\omega_0 t) + m_{01} \cos(\omega_0 t) + \cdots + m_{0(d-1)} \cos(\omega_{d-1} t) = \sin(\omega_0 t) \cos(\omega_0 k) + \sin(\omega_0 k) \cos(\omega_0 t) \tag{63}$$

$$m_{10} \sin(\omega_0 t) + m_{11} \cos(\omega_0 t) + \cdots + m_{1(d-1)} \cos(\omega_{d-1} t) = \cos(\omega_0 t) \cos(\omega_0 k) - \sin(\omega_0 k) \sin(\omega_0 t) \tag{64}$$

$$\vdots$$

$$m_{(d-1)0} \sin(\omega_0 t) + m_{(d-1)1} \cos(\omega_0 t) + \cdots + m_{(d-1)(d-1)} \cos(\omega_{(d-2)/2} t) = \cos(\omega_{(d-2)/2} t) \cos(\omega_{(d-2)/2} k) - \sin(\omega_{(d-2)/2} k) \sin \tag{65}$$

This system has a solution:

$$m_{00} = \cos(\omega_0 k) \tag{66}$$
$$m_{01} = \sin(\omega_0 k) \tag{67}$$
$$m_{02} = m_{03} = \ldots = m_{0(d-1)} = 0 \tag{68}$$
$$m_{10} = -\sin(\omega_0 k) \tag{69}$$
$$m_{11} = \cos(\omega_0 k) \tag{70}$$
$$m_{12} = m_{13} = \ldots = m_{1(d-1)} = 0 \tag{71}$$
$$m_{22} = \cos(\omega_1 k) \tag{72}$$
$$m_{23} = \sin(\omega_1 k) \tag{73}$$
$$m_{20} = m_{21} = m_{24} = \ldots = m_{2(d-1)} = 0 \tag{74}$$
$$m_{32} = -\sin(\omega_1 k) \tag{75}$$
$$m_{33} = \cos(\omega_1 k) \tag{76}$$
$$m_{30} = m_{31} = m_{34} = \ldots = m_{3(d-1)} = 0 \tag{77}$$
$$\vdots$$
$$m_{(d-2)(d-2)} = \cos(\omega_{(d-2)/2} k) \tag{78}$$
$$m_{(d-2)(d-1)} = \sin(\omega_{(d-2)/2} k) \tag{79}$$
$$m_{(d-2)0} = m_{(d-2)1} = m_{(d-2)2} = \ldots = m_{(d-2)(d-3)} = 0 \tag{80}$$
$$m_{(d-1)(d-2)} = -\sin(\omega_{(d-2)/2} k) \tag{81}$$
$$m_{(d-1)(d-1)} = \cos(\omega_{(d-2)/2} k) \tag{82}$$
$$m_{(d-1)0} = m_{(d-1)1} = m_{(d-1)2} = \ldots = m_{(d-1)(d-3)} = 0 \tag{83}$$
$$\tag{84}$$

Which lets us define M as:

$$M = \begin{bmatrix} \cos(\omega_0 k) & \sin(\omega_0 k) & 0 & 0 & \cdots & 0 & 0 \\ -\sin(\omega_0 k) & \cos(\omega_0 k) & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos(\omega_1 k) & \sin(\omega_1 k) & \cdots & 0 & 0 \\ 0 & 0 & -\sin(\omega_1 k) & \cos(\omega_1 k) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos(\omega_{(d-2)/2} k) & \sin(\omega_{(d-2)/2} k) \\ 0 & 0 & 0 & 0 & \cdots & -\sin(\omega_{(d-2)/2} k) & \cos(\omega_{(d-2)/2} k) \end{bmatrix} \tag{85}$$

We found a matrix $M \in \mathbb{R}^{d \times d}$ such that $M \times SE(x_t) = SE(x_{t+k})$. Hence, SE is a linear function. Moreover, notice how **M does not depend on t**, only on k. This is what gives us the scale invariability property.

4 Visualizing the Sinuspoidal Embeddings Properties With *Python*

In this section, we will write some $Python$ functions and classes to visualize the 4 cited properties of Sinusoidal Embeddings. Visualization is a great way to grasp such function's behaviour without having to necessarily prove it (even though you can always come back to this post with you're interested in the proofs).

4.1 Sinusoidal Embedding Definition With Pytorch

The sinusoidal embeddings module will store a multi-embedding tensor with $shape = (max\_pos, embed\_dim)$, where max_pos represents the maximum position we are interested in representing. This way, each of such tensor's row represents a the sinusoidal embedding for a position pos such that $SE(pos), 0 \leq pos < max\_pos$.

```python

import pytorch.nn as nn

class SinusoidalEmbeddings(nn.Module):

        def __init__(self, max_pos:int, embed_dim: int):
                super().__init__()
                # Returns a tensor with shape (time_steps, 1).
                positions = torch.arange(max_pos).unsqueeze(1).float()

                # Creates a tensor with shape (embed_dim //2,). We just need
                # half of the dimensions of the input embeddings to compute all
                # of the sinusoidal embeddings frequencies
                dimensions = torch.arange(start = 0, end = embed_dim, step = 2).float()

                # Compute the frequencies vector
                frequencies = torch.exp(dimensions * -(math.log(10000.0) / embed_dim))

                # Initialize the embeddings tensor with shape (time_steps, embed_dim)
                embeddings = torch.zeros(time_steps, embed_dim, requires_grad=False)

                # Apply sin to even indices (0, 2, 4, ...) of the input embeddings
                embeddings[:, 0::2] = torch.sin(positions * frequencies)

                # Apply cos to odd indices (1, 3, 5, ...) of the input embeddings
                embeddings[:, 1::2] = torch.cos(positions * frequencies)

                self.embeddings = embeddings

        def forward(self, x, t):
                embeds = self.embeddings[t].to(x.device)
                return embeds[:, :, None, None]
```

Listing 1: Sinusoidal Embedding Module Definition

4.2 Sinusoidal Embeddings Periodicity

As previously mentioned, the periodicity of Sinusoidal Embeddings is observed in each of its dimensions. Hence, we need to plot its dimensions values for different positions to see their periodic behavior.

```

max_pos = 100

# Our embeddings will only have 4 dimension
embed_dim = 4
sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim)

# Generate embeddings for a range of time steps
embeddings = sinusoidal_embeddings.embeddings

# Convert embeddings to numpy for plotting
embeddings_np = embeddings.numpy()

# Plot the sunosoidal embeddings for different time steps
plt.figure(figsize=(14, 8))
for i in range(embed_dim):
    plt.plot(embeddings_np[:, i], label=f"Dim {i} (i = {i//2})")

plt.title("SE(pos, 2i)")
plt.xlabel("pos")
plt.ylabel("Value")
plt.legend(loc="upper right", bbox_to_anchor=(1.15, 1))
```

Listing 2: Generating the plot of the embedding's dimensions for in different positions

As noticed in the plot above, the value of the function $SE(pos, 2i)$ defined in 2.8 repeats itself in a frequency that is inversely proportional to i, which is the dimension being represented (an index in the multidimensional embedding). As a consequence, **the higher the dimension of our model's embeddings (previously called d and called $embed\_dim$ in the Sinusoidal Embedding module), the less SE values vary.**

Intuitively, such consequence means that embeddings close to each other (they represent sentences in not very distant positions), will have their differences captured in lower dimensions, because their higher dimensions are likely to be very similar. The exact opposite is true for embeddings that represent sentences that are far away from each other. Let's visualize that by plotting different embeddings' heatmaps.

```

import torch

```

```
# We’ll create embeddings with many dimensions to better see the
# frequency decay effect
max_pos = 100
embed_dim = 128
sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim)

x = torch.zeros(embed_dim)
results = torch.zeros(max_pos, embed_dim)
for pos in range(max_pos):
results[pos] = x + sinusoidal_embeddings.embeddings[pos]

tensor_np = results.numpy()
# Plot the heatmap using matplotlib
plt.figure(figsize=(16, 6))
plt.imshow(tensor_np, aspect=’auto’, cmap=’RdBu’)
plt.colorbar(label=’’)
plt.xlabel(’2i’)
plt.ylabel(’pos’)
plt.title(’SE(pos, 2i)’)
plt.show()
```

Listing 3: Generating the plot of the sinusoidal embeddings for different positions

In the plot above, we see that embeddings with close positions (two close values in the pos axis) have different values of $SE(pos, 2i)$ for very small $2i$ (lower dimensions) while their values for higher dimensions are similar. On the other hand, if we pick two embeddings with very distant pos, they might differ from each other only in higher dimension. What this means is that in order to represent longer sentences (that contain positions very distant from each other), our model needs to have more dimensions. **The longer the input sentences, the higher the model's dimensions need to be.**

4.3 Unique Representativiness (SE is an injective function)

Figure **??** shows us that no two rows are the same because of the exponential decay of the frequencies with the increase of the dimension (increase of $2i$). As each row represents embeddings with different positions, what the figure is essentialy showing us is that two embeddings with different positions will never have the same representation, i.e., SE is injective.

4.4 Linearity & Scale Invariance

As previously mentioned, to be scale invariant, SE must be such that the distance between $SE(pos)$ and $SE(pos+k)$ must be the same as the one between $SE(0)$ and $SE(k)$. In order to check that, we'll calculate the distance between every two sinusoidal embeddings of our module and see the linear property of such distance. Hence, all the rows will have to be subtracted from the first one, from the second one and so on up until the last one.

```

import torch

# We’ll create embeddings with many dimensions to better see the
# frequency decay effect
max_pos = 1000
embed_dim = 1000


sinusoidal_embeddings = SinusoidalEmbeddings(max_pos, embed_dim).
    embeddings

# A single column tensor where each row’s single element contains an
# entire sinusoidal embeddings tensor
T_2 = sinusoidal_embeddings[:, None, :]

# A single row tensor where column's single element contains an
# entire sinusoidal embeddings tensor
T_1 = sinusoidal_embeddings[None, :, :]

# By broadcasting, this operation will save the desired differences
# in the tensor's last dimension
differences = T_2 - T_1

# As the differences were saved in the last dimension, we need
# to calculate the norm with respect to it, which is indicated
# by dim=-1
distances = torch.norm(differences, p=2, dim=-1)

# Plot the resulting 2D tensor as a heatmap
plt.figure(figsize=(8, 6))
plt.imshow(distances.numpy(), aspect="auto", cmap="PuRd")
plt.colorbar(label="L2 Norm (Euclidean Distance)")
plt.xlabel("pos")
plt.ylabel("pos")
plt.title("Euclidean Distances Between Different Position Sinusoidal Embeddings")
plt.show()
```

Listing 4: Generating the plot of the module of the difference between every two different positions sinusoidal embeddings

Figure **??** above shows us how the distance between sinusoidal embeddings of different positions decreases smoothly and linearly for embedidding with $d = 1000$.

Question 2

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

(a) Do this.

(b) Do that.

(c) Do something else.

4.5 Algorithmic issues

In malesuada ullamcorper urna, sed dapibus diam sollicitudin non. Donec elit odio, accumsan ac nisl a, tempor imperdiet eros. Donec porta tortor eu risus consequat, a pharetra tortor tristique. Morbi sit amet laoreet erat. Morbi et luctus diam, quis porta ipsum. Quisque libero dolor, suscipit id facilisis eget, sodales volutpat dolor. Nullam vulputate interdum aliquam. Mauris id convallis erat, ut vehicula neque. Sed auctor nibh et elit fringilla, nec ultricies dui sollicitudin. Vestibulum vestibulum luctus metus venenatis facilisis. Suspendisse iaculis augue at vehicula ornare. Sed vel eros ut velit fermentum porttitor sed sed massa. Fusce venenatis, metus a rutrum sagittis, enim ex maximus velit, id semper nisi velit eu purus.

Algorithm 1: `FastTwoSum`

```
Input: $(a, b)$, two floating-point numbers
Result: $(c, d)$, such that $a + b = c + d$

if $|b| > |a|$ then
    exchange $a$ and $b$ ;
end
$c \leftarrow a + b$ ;
$z \leftarrow c - a$ ;
$d \leftarrow b - z$ ;
return $(c, d)$ ;
```

Fusce varius orci ac magna dapibus porttitor. In tempor leo a neque bibendum sollicitudin. Nulla pretium fermentum nisi, eget sodales magna facilisis eu. Praesent aliquet nulla ut bibendum lacinia. Donec vel mauris vulputate, commodo ligula ut, egestas orci. Suspendisse commodo odio sed hendrerit lobortis. Donec finibus eros erat, vel ornare enim mattis et.

Question 3 *(with optional title)*

In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.

Mauris interdum porttitor fringilla. Proin tincidunt sodales leo at ornare. Donec tempus magna non mauris gravida luctus. Cras vitae arcu vitae mauris eleifend scelerisque. Nam sem sapien, vulputate nec felis eu, blandit convallis risus. Pellentesque sollicitudin venenatis tincidunt. In et ipsum libero. Nullam tempor ligula a massa convallis pellentesque.

5 Implementation

Proin lobortis efficitur dictum. Pellentesque vitae pharetra eros, quis dignissim magna. Sed tellus leo, semper non vestibulum vel, tincidunt eu mi. Aenean pretium ut velit sed facilisis. Ut placerat urna facilisis dolor suscipit vehicula. Ut ut auctor nunc. Nulla non massa eros. Proin rhoncus arcu odio, eu lobortis metus sollicitudin eu. Duis maximus ex dui, id bibendum diam dignissim id. Aliquam quis lorem lorem. Phasellus sagittis aliquet dolor, vulputate cursus dolor convallis vel. Suspendisse eu tellus feugiat, bibendum lectus quis, fermentum nunc. Nunc euismod condimentum magna nec bibendum. Curabitur elementum nibh eu sem cursus, eu aliquam leo rutrum. Sed bibendum augue sit amet pharetra ullamcorper. Aenean congue sit amet tortor vitae feugiat.

In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.

hello.py

```python
#! /usr/bin/python

import sys
sys.stdout.write("Hello World!\n")
```

Fusce eleifend porttitor arcu, id accumsan elit pharetra eget. Mauris luctus velit sit amet est sodales rhoncus. Donec cursus suscipit justo, sed tristique ipsum fermentum nec. Ut tortor ex, ullamcorper varius congue in, efficitur a tellus. Vivamus ut rutrum nisi. Phasellus sit amet enim efficitur, aliquam nulla id, lacinia mauris. Quisque viverra libero ac magna maximus efficitur. Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum mollis eros in tellus fermentum, vitae tristique justo finibus. Sed quis vehicula nibh. Etiam nulla justo, pellentesque id sapien at, semper aliquam arcu. Integer at commodo arcu. Quisque dapibus ut lacus eget vulputate.

Command Line

```
$ chmod +x hello.py
$ ./hello.py

Hello World!
```

Vestibulum sodales orci a nisi interdum tristique. In dictum vehicula dui, eget bibendum purus elementum eu. Pellentesque lobortis mattis mauris, non feugiat dolor vulputate a. Cras porttitor dapibus lacus at pulvinar. Praesent eu nunc et libero porttitor malesuada tempus quis massa. Aenean cursus ipsum a velit ultricies sagittis. Sed non leo ullamcorper, suscipit massa ut, pulvinar erat. Aliquam erat volutpat. Nulla non lacus vitae mi placerat tincidunt et ac diam. Aliquam tincidunt augue sem, ut vestibulum est volutpat eget. Suspendisse potenti. Integer condimentum, risus nec maximus elementum, lacus purus porta arcu, at ultrices diam nisl eget urna. Curabitur sollicitudin diam quis sollicitudin varius. Ut porta erat ornare laoreet euismod. In tincidunt purus dui, nec egestas dui convallis non. In vestibulum ipsum in dictum scelerisque.

Notice: In congue risus leo, in gravida enim viverra id. Donec eros mauris, bibendum vel dui at, tempor commodo augue. In vel lobortis lacus. Nam ornare ullamcorper mauris vel molestie. Maecenas vehicula ornare turpis, vitae fringilla orci consectetur vel. Nam pulvinar justo nec neque egestas tristique. Donec ac dolor at libero congue varius sed vitae lectus. Donec et tristique nulla, sit amet scelerisque orci. Maecenas a vestibulum lectus, vitae gravida nulla. Proin eget volutpat orci. Morbi eu aliquet turpis. Vivamus molestie urna quis tempor tristique. Proin hendrerit sem nec tempor sollicitudin.