Linearization of the Feed-forward Layers in Neural Networks

Lisa Wang
Columbia University
New York, New York
`hw3082@columbia.edu`

Victor Lin
Columbia University
New York, New York
`vl2580@columbia.edu`

Jicheol Ha
Columbia University
New York, New York
`jh4762@columbia.edu`

Abstract

In this work, we propose a functional decomposition of the Gaussian Error Linear Unit (GELU) into two independent mappings, $\Phi(\cdot)$ and $\Psi(\cdot)$. We assume multi-dimensional inputs, specifically linear transformations of the form Wx, where W is a square matrix and x is a column vector. In the biased approach, we use the Taylor series expansion of the standard normal cumulative distribution function to express the GELU as an infinite sum of scaled powers of Wx. Then, by using the Kronecker product, we decompose powers of Wx into linear combinations of the elements of W and x, making explicit the complex nonlinearity that GELU introduces. In the unbiased approach, we train $\Phi(\cdot)$ and $\Psi(\cdot)$ using a Multilayered Perceptron, which effectively addresses nonlinearities in GELU, albeit less explicitly. The biased approach favors mathematical understanding, while the unbiased approach is machine-driven. We show that both can approximate the GELU well. As state-of-the-art transformer-based models use GELU, benefits of a decomposed activation function include faster model convergence.

***K*eywords** Gaussian Error Linear Unit · Kronecker Product · Functional Decomposition · Neural Networks · Activation Function · Matrix Algebra · Taylor Series Approximation · Multilayered Perceptron

1 Introduction

The Gaussian Error Linear Unit (GELU) is a neural network activation function that is used in several deep learning architectures. First introduced by Hendrycks and Gimpel in 2016, GELU is the product of an input x and the probability that x is greater than zero under a standard normal distribution. Mathematically, $\text{GELU}(x) = x \cdot \Phi(x)$, where Φ is the standard normal cumulative distribution function. Unlike traditional activation functions such as the Rectified Linear Unit (RELU) or the hyperbolic tangent function, GELU is smoother around zero, non-monotonic, and continuously differentiable for all inputs, which effectively addresses the vanishing gradient problem observed in ReLU. These properties make it particularly suitable for self-attention mechanisms and transformer architectures, where it has demonstrated superior performance compared to its predecessors. Moreover, its probabilistic nature introduces dilution and dropout that varies with the magnitude of the input signal. This characteristic becomes significant in deep transformer networks, where the interaction between self-attention mechanisms and non-linear activation functions plays a crucial role in model performance. Notable implementations include natural language processing models such as GPT (Brown et al., 2020) and BERT (Devlin et al., 2019), and computer vision models such as Vision transformers (Alexey Dosovitskiy et al., 2010).

2 Problem Statement

We formally define GELU as the product between x and the standard normal cumulative distribution function:

$$\mathrm{GELU}(x) = x \int_{-\infty}^{x} \frac{t}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}\ dt \tag{1}$$

We want to find $\Phi : \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times m}$ and $\Psi : \mathbb{R}^{d} \to \mathbb{R}^{m}$ such that

$$\mathrm{GELU}((Wx)_i) \approx \Phi(W_i)\Psi(x_i)$$

where $(Wx)_i$ is the i^{th} **element** of Wx, W_i the i^{th} **row** of W, and x_i the i^{th} **element** of x for $i \in \{1, ..., d\}$. We assume an element-wise operation of GELU.

3 Biased Estimation

3.1 Mathematical basis for functional decomposition

3.1.1 Taylor series expansion of the standard normal cumulative distribution function

We first express the standard normal cumulative distribution function as a Taylor series. Recall that

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Using this,

$$e^{-\frac{x^2}{2}} = \sum_{n=0}^{\infty} \frac{\left(-\frac{x^2}{2}\right)^n}{n!} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{2^n n!}$$

Let $G(x)$ be the indefinite integral of the standard normal cumulative distribution function. Then,

$$\begin{aligned} G(x) &\coloneqq \frac{1}{\sqrt{2\pi}} \int \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{2^n n!} x\ dx \\ &= \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \int \frac{(-1)^n x^{2n}}{2^n n!} x\ dx \\ &= \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^n n!} \int x^{2n+1}\ dx \\ &= \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+2}}{2^n n!(2n+2)} \end{aligned}$$

Now, let $F(x)$ be the standard normal cumulative distribution function.

$$F(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}} dt$$

Since $F(0) = 0.5$, we see that

$$F(x) = 0.5 + \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{2^n n!(2n+1)}$$

3.1.2 Taylor series expansion of the Gaussian Error Linear Unit

Using the expression from (1), for $x \in \mathbb{R}$,

$$\begin{aligned}\mathrm{GELU}(x) &= xF(x)\\ &= x\left(0.5 + \frac{1}{\sqrt{2\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{2^n n!(2n+1)}\right)\\ &= 0.5x + \frac{1}{\sqrt{2\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+2}}{2^n n!(2n+1)}\end{aligned}$$

Therefore, for each element in Wx, we have

$$\mathrm{GELU}((Wx)_i) = 0.5(Wx)_i + \frac{1}{\sqrt{2\pi}}\sum_{n=0}^{\infty}\frac{(-1)^n((Wx)_i)^{2n+2}}{2^n n!(2n+1)} \tag{2}$$

3.1.3 Multinomial expansion of $(Wx)_i$s

We use the multinomial theorem to decompose the $((Wx)_i)^{2n+2}$ term in (2),

$$\begin{aligned}(Wx_i)^{2n+2} &= (W_i \cdot x)^{2n+2}\\ &= \sum_{j=1}^{d} W_j \cdot x_i \text{ for } j \in \{1, 2, ..., d\}\\ &= \sum_{\substack{k_1+\cdots+k_r=2n+2\\ k_1,k_2,\ldots,k_m \geq 0}} \binom{2n+2}{k_1,\ldots,k_r}\prod_{m=1}^{r}(W_j)^{k_m}\prod_{m=1}^{r}(x_i)^{k_m}\end{aligned}$$

where W_i is the i^{th} row of W and W_j is the j^{th} element of that row, and

$$\binom{2n+2}{k_1,\ldots,k_r} = \frac{(2n+2)!}{k_1!\,k_2!\cdots k_r!}$$

is the multinomial coefficient. Plugging this expression back to (2) completes the function decomposition, whose closed form will be expressed in a Kronecker product in 3.2.2.

3.2 Biased estimation using Kronecker products

3.2.1 General form of approximating GELU using Taylor series

We rewrite the Taylor series approximation of $\mathrm{GELU}(Wx)$ in a general form amenable to decomposition into Φ and Ψ.

Let x'_1 be the first component of Wx. Since the GELU is an element-wise operation, without loss of generality, any decomposition we find for x'_1 can be generalized to all dimensions.

$$x'_1 = (Wx)_1 = W_1 \cdot x$$

where W_1 is the first row of the weight matrix W and x is the input vector.

Plugging x'_1 into (2), the fully expanded GELU approximation using a finite Taylor sum is

$$\mathrm{GELU}(x'_1) \approx \frac{x'_1}{2} + \frac{(x'_1)^2}{\sqrt{2\pi}} - \frac{(x'_1)^4}{6\sqrt{2\pi}} + \frac{(x'_1)^6}{40\sqrt{2\pi}} - \frac{(x'_1)^8}{336\sqrt{2\pi}} + \cdots$$

which has a general form that prepares us for decomposition into Φ and Ψ

$$a_0(W_1x) + a_1(W_1x)^2 + a_2(W_1x)^4 + \cdots \tag{3}$$

where a_n's are the coefficients of (2).

3.2.2 Constructing Φ and Ψ using Kronecker products

We show that each non-linear term $(W_1x)^{2n}$ for $n = 1, 2, ...$ can be decomposed into a dot product between the lifted versions of W and x.

For simplicity, let x be a column vector of dimension $d = 2$. Then, by the multinomial theorem, the first nonlinear term of (3) expands to:

$$\begin{aligned} a_1(W_1x)^2 &= \left(w_1^2x_1^2 + w_1w_2x_1x_2 + w_2w_1x_2x_1 + w_2^2x_2^2\right) \\ &= a_1 \begin{pmatrix} w_1^2 & w_1w_2 & w_2w_1 & w_2^2 \end{pmatrix} \begin{pmatrix} x_1^2 \\ x_1x_2 \\ x_2x_1 \\ x_2^2 \end{pmatrix} \end{aligned} \tag{4}$$

We can equivalently represent (4) as a Kronecker product, a matrix operation that takes two matrices and produces a larger block matrix by multiplying each element of the first matrix by the entire second matrix.

$$a_1 \cdot (W \otimes W) \cdot (x \otimes x)^\intercal$$

where $\otimes$ is the Kronecker product operator.

For higher-order nonlinear terms ($n \geq 2$) in (3), the general Kronecker form is:

$$a_n(W_1x)^{2n} = a_n \cdot (W^{\otimes 2n}) \cdot (x^{\otimes 2n})^\intercal$$

Thus, for an expanded Taylor series approximation, the decomposed functional estimation for a single output dimension corresponding to GELU(W_1x) is given by

$$a_0(W_1x) + a_n \left(\sum_{i=1}^{d} w_ix_i\right)^{2n} = a_0(W_1x) + a_n \cdot \left(W^{\otimes 2n}\right) \cdot \left(x^{\otimes 2n}\right)^\intercal \text{ for } n \geq 1$$

where w_i is the i^{th} element of W.

Therefore, we propose a functional decomposition of the GELU given by

$$\text{GELU}(W_1x) \approx \Phi(W) \cdot \Psi(x)^\intercal \tag{5}$$

where

$$\Phi(W) = \sum_{n=0}^{N} \Phi_n(W), \quad \text{with} \quad \Phi_0(W) = a_0 \cdot W,\ \Phi_n(W) = a_n \cdot W^{\otimes 2n} \text{ for } n \geq 1,$$

$$\Psi(x) = \sum_{n=0}^{N} \Psi_n(x), \quad \text{with} \quad \Psi_0(x) = x,\ \Psi_n(x) = x^{\otimes 2n} \text{ for } n \geq 1.$$

a_ns are the coefficients in (2), and N is the number of terms in the finite Taylor series.

3.2.3 Code

Link to Colab Notebook: Kronecker Estimator with Validation

Using (5), the code performs the computation for each dimension.

```python
class TaylorSeriesGeluEstimator:
    def __init__(self, n_terms):
        self.n = n_terms
        self.coeffs = self._compute_coeffs()
        self.powers = [1] + [2 * i + 2 for i in range(self.n)]

    def _compute_coeffs(self):
        coeffs = [0.5]
        base = 1 / math.sqrt(2 * math.pi)
        for i in range(self.n):
            c = (-1)**i / (math.factorial(i) * (2**i) * (2 * i + 1))
            coeffs.append(base * c)
        return coeffs

    def _kron_n(self, vec, times):
        out = vec
        for _ in range(times - 1):
            out = np.kron(out, vec)
        return out

    def transform(self, vec, coeffs=None):
        if coeffs is None:
            coeffs = [1.0] * len(self.powers)
        return np.concatenate([c * self._kron_n(vec, p) for c, p in zip(coeffs,
           self.powers)])

    def get_Phi(self, W):
        return np.vstack([self.transform(w, self.coeffs) for w in W])

    def get_Psi(self, x):
        return self.transform(x, coeffs=None)
```

3.2.4 Results

This approximation is biased because the Taylor series approximation is accurate only around $x = 0$, and truncating higher-order terms introduces systematic error away from the expansion point.

In the Colab, we approximate $y = \text{GELU}(Wx)$ using a Taylor series expansion up to five terms combined with Kronecker-lifted features. To ensure the Taylor approximation remains accurate, we constrain Wx to the range $[-3, 3]$, where the maximum Taylor error remains acceptable:

Range of Wx	**Max Error**
$[-1, 1]$	9×10^{-6}
$[-3, 3]$	3.0
$[-5, 5]$	7.8×10^{2}
$[-7, 7]$	2.7×10^{4}

Table 1: Maximum error of Taylor approximation over different Wx ranges.

For the generated input:

$$Wx = [-2.48,\ 1.55,\ 2.23]$$

we construct the lifted features:

$$\Phi(W) \in \mathbb{R}^{3\times 66432}, \quad \Psi(x) \in \mathbb{R}^{66432}$$

and compute the linearized output:

$$y' = \Phi(W) \cdot \Psi(x) = [0.3358,\ 1.4577,\ 2.3061]$$

The comparison between true GELU outputs, Taylor approximations, and the linearized form is shown below:

Quantity	Value
$x' = Wx$	$[-2.48,\ 1.55,\ 2.23]$
$\mathrm{GELU}_{\text{Taylor, 5}}(x')$	$[0.3358,\ 1.4577,\ 2.3061]$
Linearized $y' = \Phi(W) \cdot \Psi(x)$	$[0.3358,\ 1.4577,\ 2.3061]$
$\mathrm{GELU}_{\tanh}(x')$	$[-0.0159,\ 1.4559,\ 2.2016]$

Table 2: Comparison of GELU true values, Taylor approximation, and linearized outputs.

The error metrics show that the Φ and Ψ mappings exactly represent the Taylor series approximation:

Metric	Value
RMSE (Taylor vs. Tanh GELU)	0.21182
RMSE (Linearized $\Phi(W) \cdot \Psi(x)$ vs. Taylor GELU)	0.00000

Table 3: Root mean squared errors between the different approximations.

3.3 Approximating GELU by applying Taylor series expansion on CDF

Let $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} = .3989e^{-5z^2}$ be the standard normal density function and

Let $F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt$ be the standard normal cumulative distribution function

We compute a Taylor series expansion:

$$\begin{aligned} G(x) &= \int \frac{1}{\sqrt{2\pi}} e^{\frac{-1}{2}x^2} dx \\ &= \frac{1}{\sqrt{2\pi}} \int \sum_{n=0}^{\infty} \frac{(-1)^n}{n!2^n} x^{2n} dx \\ &= \frac{1}{\sqrt{2\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n}{n!2^n(2n+1)} x^{2n+1} \\ &= \frac{1}{\sqrt{2\pi}} \left(x - \frac{1}{6}x^3 + \frac{1}{40}x^5 - \frac{1}{336}x^7 + \cdots \right) \end{aligned} \tag{6}$$

3.3.1 Approximating using a single reference point

Colab Notebook with Code: Lisa's biased estimation

A common deterministic (and therefore typically biased) approximation is a first-order Taylor expansion. Concretely, for:

$$y = \mathrm{GELU}(Wx) \quad \in \mathrm{R}^d$$

we pick a reference point $\bar{x}$ (or equivalently $\bar{z} = W\bar{x}$) and do

$$\mathrm{GELU}(Wx) \approx \mathrm{GELU}(\bar{z}) + \mathrm{diag}\left(\mathrm{GELU}'(\bar{z})\right) [W(x - \bar{x})]$$

We can write that in the form:

$$\Phi(W)\Psi(x)$$

by letting (for each dimension d):

- $\Phi(W) = \left[\operatorname{diag}\left(\text{GELU}'(\bar{z})\right) W, \text{GELU}(\bar{z})\right]$,
- $\Psi(x) = \begin{bmatrix} x - \bar{x} \\ 1 \end{bmatrix}$

Where the derivative of a point x of the GELU function is:

$$\text{GELU}'(x) = \frac{\tanh\left(\frac{\sqrt{2}\left(\frac{8943x^3}{200000}+x\right)}{\sqrt{\pi}}\right)+1}{2} + \frac{x\left(\frac{26829x^2}{200000}+1\right)\operatorname{sech}^2\left(\frac{\sqrt{2}\left(\frac{8943x^3}{200000}+x\right)}{\sqrt{\pi}}\right)}{\sqrt{2}\sqrt{\pi}}$$

4 Unbiased estimation

4.1 Naive Monte Carlo sampling

4.1.1 Derivation

We propose Naive Monte Carlo sampling to approximate $\text{GELU}(x)$ for $x \in \mathbb{R}$.

$$\begin{aligned} \text{GELU}(x) &= x \cdot \Phi(x) \\ &= x \cdot \mathbb{P}(Z \leq x) \\ &= x \cdot \mathbb{E}\left[\mathbb{1}\{Z \leq x\}\right] \\ &\approx x \cdot \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{z_i \leq x\} \end{aligned}$$

where for $z_i \sim \mathcal{N}(0,1)$, $\mathbb{1}\{Z \leq x\}$ is an indicator function defined as:

$$\mathbb{1}\{z_i \leq x\} = \begin{cases} 1, & z_i \leq x \\ 0, & z_i > x \end{cases}$$

4.1.2 Unbiasedness and consistentency

We show that the estimator $\hat{\text{GELU}}(x) = x \cdot \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{z_i \leq x\}$ is unbiased and consistent. The expectation of the estimator is given as:

$$\begin{aligned} \mathbb{E}\left[\hat{\text{GELU}}(x)\right] &= \mathbb{E}\left[x \cdot \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{z_i \leq x\}\right] \\ &= x \cdot \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}[\mathbb{1}\{z_i \leq x\}] \\ &= x \cdot \frac{1}{N}\sum_{i=1}^{N} \mathbb{P}(z_i \leq x) \\ &= x \cdot \frac{1}{N} \cdot N \cdot \mathbb{P}(Z \leq x) \\ &= x \cdot \mathbb{P}(Z \leq x) \end{aligned}$$

By the weak law of large numbers, as $N \to \infty$, the sample mean converges in probability to the population mean.

$$\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{z_i \le x\} = \mathbb{P}(Z \le x)$$

Thus, the proposed estimator is unbiased and consistent. Below, we provide a pseudo-code to generate a single point estimator.

Algorithm 1 Naive Monte Carlo Sampling of GELU

```
1: count ← 0
2: function GELU($x, N$)
3:     for $i$ ← 1 to $N$ do
4:         $z \sim \mathcal{N}(0,1)$                          ▷ generate from a standard normal distribution
5:         if $z \le x$ then
6:             count ← count + 1                          ▷ count the number of successes
7:         end if
8:     end for
9:     return $\left(\frac{\text{count}}{N}\right) \cdot x$                          ▷ return sample mean
10: end function
```

4.2 Unbiased estimation using two neural networks

4.2.1 Decomposing Φ and Ψ as two feature maps

We approximate the GELU feedforward operation by learning feature maps:

$$\mathbf{y} = \text{GELU}(\mathbf{W}\mathbf{x}) \quad \longrightarrow \quad \mathbf{y}' = \Phi(\mathbf{W})\Psi(\mathbf{x})$$

where:

- Ψ: an MLP mapping $\mathbf{x} \in \mathbb{R}^d \to \mathbb{R}^m$,
- Φ: a row-wise MLP mapping $\mathbf{W} \in \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times m}$,

The final prediction $\mathbf{y}'$ is given by a row-wise dot product between $\Phi(\mathbf{W})$ and $\Psi(\mathbf{x})$.

The networks Φ and Ψ are trained jointly by minimizing the mean squared error:

$$\mathbf{L} = \frac{1}{N}\sum_{i=1}^{N} \|\Phi(\mathbf{W}_i)\Psi(\mathbf{x}_i) - \text{GELU}(\mathbf{W}_i\mathbf{x}_i)\|^2$$

4.2.2 Code

Colab Notebook with Code: Unbiased estimation using two neural networks

The code below builds a single computation graph for both Φ and Ψ, allowing efficient gradient updates.

```python
import torch
import torch.nn as nn
import math
from torch.utils.data import Dataset, DataLoader

class GeluDataset(Dataset):
    def __init__(self, N, d):
        self.W = torch.randn(N, d, d)
        self.x = torch.randn(N, d)
        Z = torch.einsum('bij,bj->bi', self.W, self.x)
        self.y = 0.5 * Z * (1 + torch.erf(Z / math.sqrt(2)))
    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        return self.W[idx], self.x[idx], self.y[idx]

class PsiNet(nn.Module):
    def __init__(self, d, m, hidden=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(d, hidden),
            nn.ReLU(),
            nn.Linear(hidden, m)
        )

    def forward(self, x):
        return self.net(x)

class PhiNet(nn.Module):
    def __init__(self, d, m, hidden=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(d, hidden),
            nn.ReLU(),
            nn.Linear(hidden, m)
        )

    def forward(self, W):
        B, D, _ = W.shape
        W_flat = W.view(B * D, D)
        out_flat = self.net(W_flat)
        return out_flat.view(B, D, -1)

class GeluTrainer:
    def __init__(self, d=2, m=16, N=1024, batch_size=64, hidden=64, lr=1e-3):
        self.d = d
        self.m = m
        self.N = N
        self.batch_size = batch_size
        self.dataset = GeluDataset(N, d)
        self.loader = DataLoader(self.dataset, batch_size=batch_size, shuffle=True
            )
        self.Psi = PsiNet(d, m, hidden)
        self.Phi = PhiNet(d, m, hidden)
        self.optimizer = torch.optim.Adam(
            list(self.Psi.parameters()) + list(self.Phi.parameters()), lr=lr
        )
        self.criterion = nn.MSELoss()

    def train(self, num_epochs):
        for epoch in range(1, num_epochs + 1):
            total_loss = 0.0
            for Wb, xb, yb in self.loader:
                self.optimizer.zero_grad()
                x_prime = self.Psi(xb)
                w_prime = self.Phi(Wb)
                y_pred = torch.bmm(x_prime, w_prime.unsqueeze(-1)).squeeze(-1)
                loss = self.criterion(y_pred, yb)
                loss.backward()
                self.optimizer.step()
                total_loss += loss.item() * Wb.size(0)
            mse = total_loss / self.N
            print(f"Epoch {epoch}: MSE = {mse:.4f}")
        return self.Phi, self.Psi
```

4.2.3 Results

Unlike the previous Taylor-based method, whose bias is limited by the fixed polynomial degree, **this approach is unbiased: more data could improve accuracy**.

We trained three models for different numbers of epochs: 3, 10, and 30. The final approximation performance was evaluated on a random test pair $(\mathbf{W}, \mathbf{x})$.

- **3 Epochs:** Φ and Ψ began to approximate GELU, but performance remained coarse. **RMSE = 0.1245**
- **10 Epochs:** Φ and Ψ aligned best with the target function. **RMSE = 0.0822**
- **30 Epochs:** Unexpectedly worse than 10 epochs. **RMSE = 0.0891** — possibly due to overfitting

Despite the low input/output dimension ($d = 2$), the consistently low RMSE (≈ 0.08–0.12) demonstrates that the Φ/Ψ decomposition is a viable proof of concept for approximating GELU.

This suggests that dot-product-based linearization is expressive enough to emulate nonlinear activations and may scale effectively to higher dimensions or more complex transformations, which is explored in the next section.

4.3 Training Φ and Ψ for x with large dimensions

Colab Notebook with Code: Lisa's unbiased estimation

4.3.1 Enhancements on dimensionality

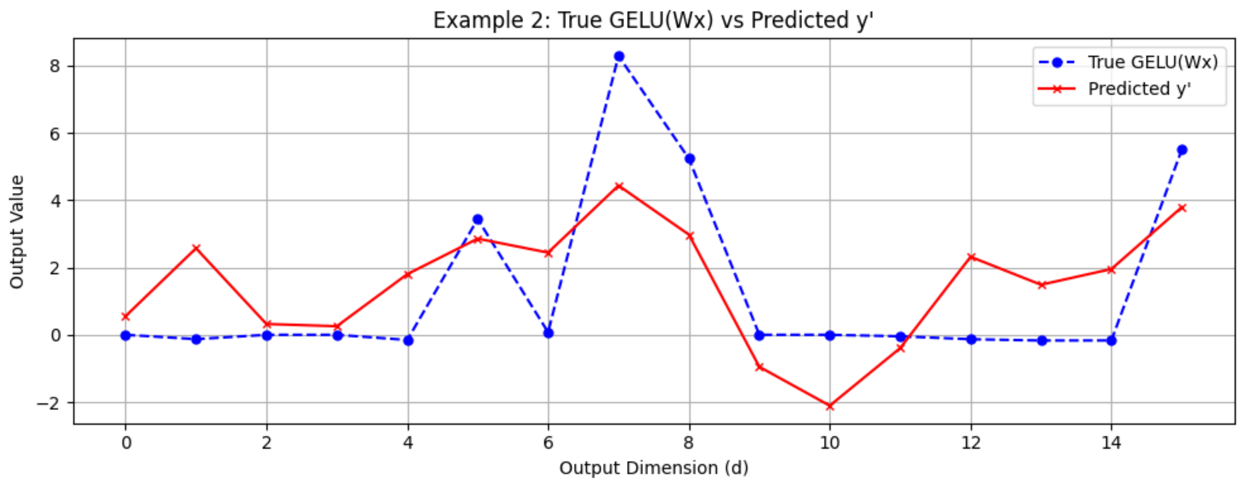
This neural network approach builds upon the previous section, but with an increase of the dimensionality from 2 to 16.

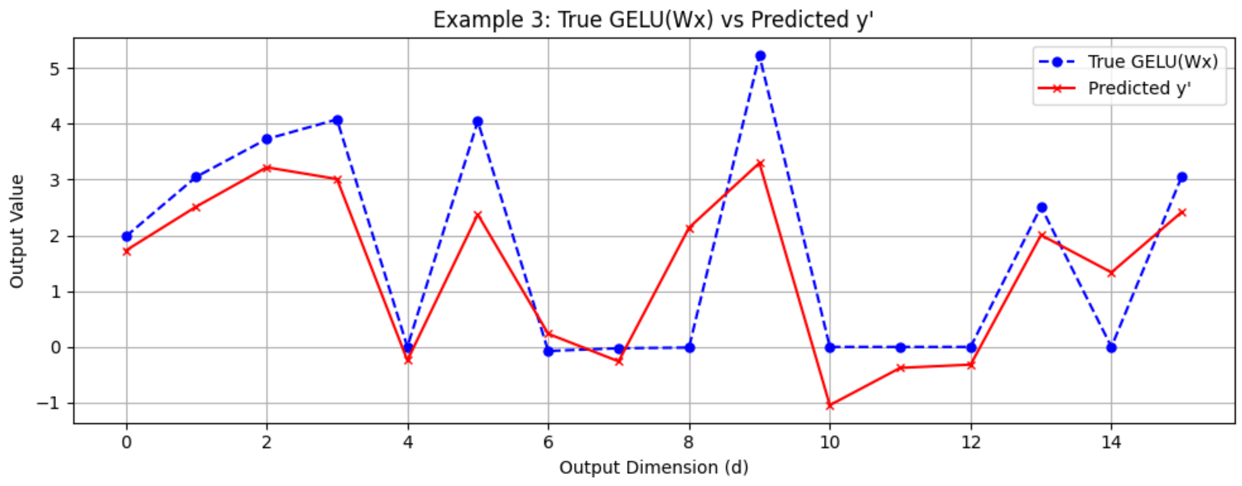
- $\Phi(W)$ and $\Psi(x)$ will be trained to approximate $\text{GELU}(Wx)$
- We'll build both as neural networks:
- $\Psi(x)$ will map the input vector $x \in \mathbb{R}^d$ to a feature space $\mathbb{R}^m$.
- $\Phi(W)$ will map the weight matrix $W \in \mathbb{R}^{d \times d}$ to a projection in $\mathbb{R}^{d \times m}$.

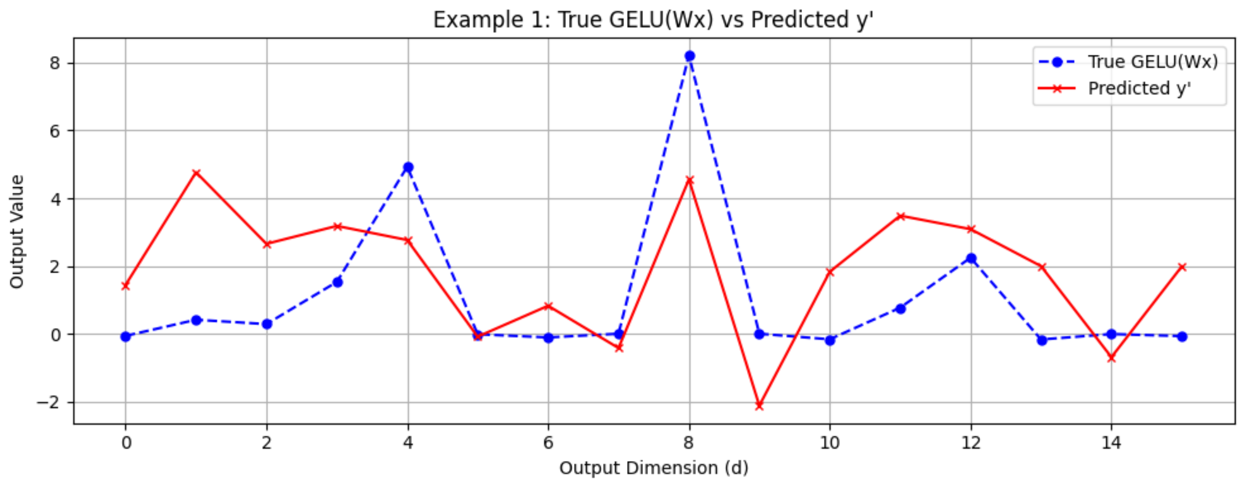
We would like to use backpropagation to train the model to reduce loss, which is the Mean Squared Error (MSE) between the output of the linearized model and the true GELU function.

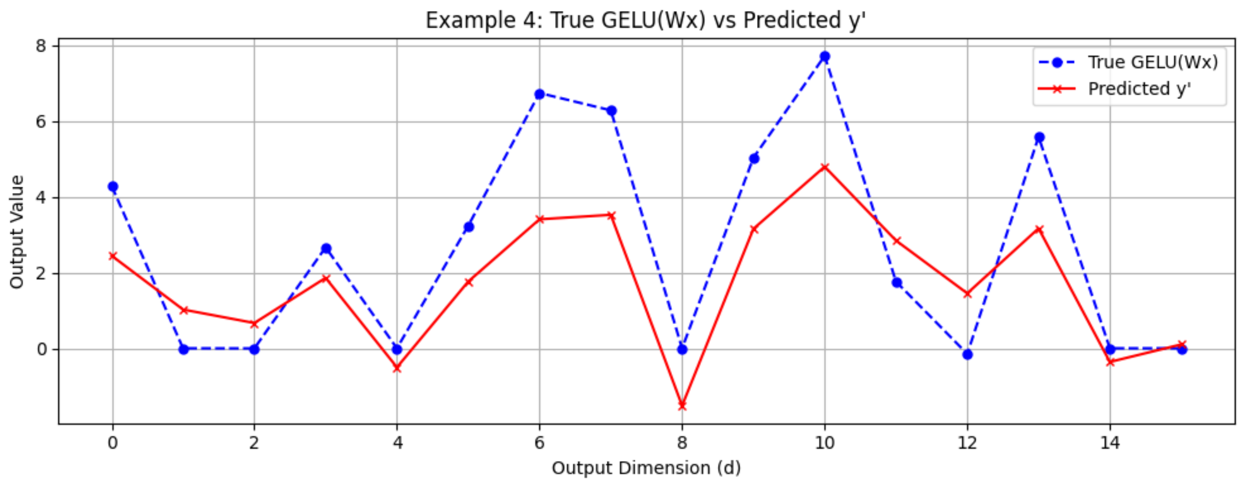
4.3.2 True GELU(Wx) vs. Predicted y'

see example 1-4.


Example 2: True GELU(Wx) vs Predicted y'
True GELU(Wx)
Predicted y'
Output Value
Output Dimension (d)


Example 3: True GELU(Wx) vs Predicted y'
True GELU(Wx)
Predicted y'
Output Value
Output Dimension (d)


Example 1: True GELU(Wx) vs Predicted y'
True GELU(Wx)
Predicted y'
Output Value
Output Dimension (d)



4.3.3 Training loss over steps count

The training loss over 2000 steps is graphed, and we visualized loss curve decreases and gradually converges to the true GELU approximation with an average of ~ 2.013.

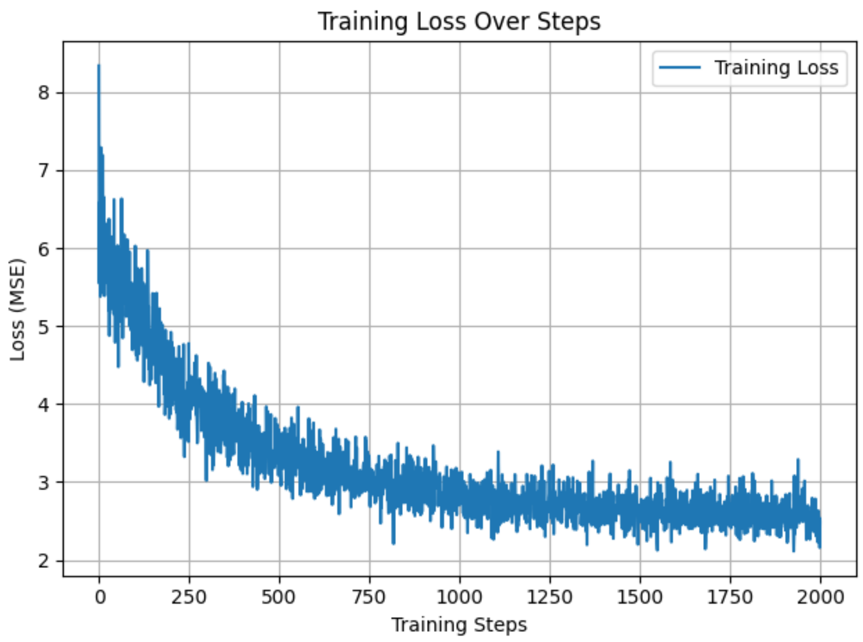


Figure 1: Training Loss over 2000 steps, stabilizes around 2.013

5 Further Exploration and Usages

5.1 Adoption of the two decomposed functions into transformer-based models

As a part of further exploration, and without time constraints, our team would like to explore if decomposed functions have benefits such as accelerating model convergence time. We speculate this on the basis that current state-of-the-art transformer-based natural language processing models use GELU.