Decoding GPT: Cooler Architectures

Samy Wu Fung, Daniel McKenzie, Michael Ivanitskiy

2024-01-23

Review: DNNs & SGD

Review: DNNs & SGD

(dense) deep neural networks are a composition of affine transformations with nonlinearities between them:

each layer h_i is defined as

$$h_i(x_i) = \sigma_i(W_i x_i + b_i)$$

and the network as a whole can be written as

$$\texttt{MLP(x)} = h_n(h_{n-1}(\cdots h_2(h_1(x_1))\cdots)) = h_n \circ h_{n-1} \circ \cdots \circ h_2 \circ h_1(x)$$

$$= \bigcirc_{i\in\mathbb{N}_n}^{z=x} h_i(z) = \bigcirc_{i\in\mathbb{N}_n}^{z=x} \sigma_i(W_i x_i + b_i)$$

Parameter Set Definition

the parameter set, denoted θ, includes all the weights W_i and biases b_i for each layer in the network

$$\theta = \{W_i, b_i \mid i \in \mathbb{N}_n\}$$

where $\mathbb{N}_n$ represents the set of indices for each layer in the network.

Loss Function

The "loss function" measures the discrepancy between the predictions of the DNN and the ground truth. For a given pair of input and target output (x, y) sampled from a distribution $\mathcal{D}$, the loss function ℓ is defined as a distance metric between the network's output $\texttt{MLP}(x)$ and the target y:

$$\ell(\texttt{MLP}(x|\theta), y) = \ell(\texttt{MLP}_\theta(x), y) = \text{distance}(\texttt{MLP}_\theta(x), y)$$

this distance metric is usually something like the L_2 norm:

$$\ell(\texttt{MLP}_\theta(x), y) = \|\texttt{MLP}_\theta(x) - y\|_2^2$$

or the cross entropy loss (for classification):

$$\ell(\texttt{MLP}_\theta(x), y) = -\sum_{c \in C} y[c] \log(\texttt{MLP}_\theta(x)[c])$$

Where C is the set of classes (as indices), $y[c] \in \{0, 1\}$ is the true label, and $\texttt{MLP}(x)[c] \in [0, 1]$ is the predicted probability of class c.

Objective: Minimize Expected Loss

Our objective is to find the parameter set θ that minimizes the expected loss over the data distribution $\mathcal{D}$. Formally, this is expressed as:

$$\hat{\theta} = \arg\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} [\ell(\texttt{MLP}_\theta(x), y)]$$

Empirical Risk Minimization

In practice, we don't have access to the entire data distribution $\mathcal{D}$, but only some sample D from $\mathcal{D}$. We approximate the expected loss:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{|D|} \sum_{(x,y)\in D} \ell(\texttt{MLP}_\theta(x), y)$$

Gradient Descent

To minimize the loss, we use gradient descent, which iteratively adjusts the parameters in the direction that most reduces the loss. The parameter update rule in gradient descent is:

$$\theta \leftarrow \theta - \eta \nabla_\theta \ell(\texttt{MLP}_\theta(x), y)$$

where η is the learning rate and ∇_θ denotes the gradient with respect to θ.

Stochastic Gradient Descent (SGD)

In practice, instead of computing the gradient of the loss over the entire dataset, we approximate it using a batch $B \subset D$ of samples from the dataset:

$$\theta \leftarrow \theta - \eta \nabla_\theta \frac{1}{|B|} \sum_{(x,y) \in B} \ell(\texttt{MLP}_\theta(x), y)$$

Backpropagation

Backpropagation is used to compute the gradient of the loss function with respect to each parameter in the network. It's really just the chain rule:

$$\nabla_\theta \ell(\texttt{MLP}_\theta(x), y) = \frac{\partial \ell(\texttt{MLP}_\theta(x), y)}{\partial \theta} = \frac{\partial W_i}{\partial \theta} \cdot \frac{\partial \ell(\texttt{MLP}_\theta(x), y)}{\partial W_i}$$

Why can't we use DNNs for everything?

In the MNIST example, our inputs are 28×28 pixels, and we have 10 classes. Let's say we want to classify real images:

- input space might be 512×512 pixels, 3 color channels
- output space might be 1024 classes (dogs, trees, cars, etc.)
- we probably need a *really* big hidden layer, but maybe something like 1024 will do?

$$512 \times 512 \times 3 \times 1024 + 1024 \times 1024 = 2^9 \times 2^9 \times 3 \times 2^{10} + 2^{10} \times 2^{10}$$

$$= 2^{28} \cdot 3 + 2^{20} \approx 3 \cdot 2.7 \cdot 10^8 + 1.5 \cdot 10^6 \approx 8.1 \cdot 10^8$$

Nowadays, a $\sim$billion parameters isn't that many, but in practice such a network wouldn't learn much of anything. The problem is that we fail to give the network a useful *inductive bias*.

So, we steal the spatial statistics prior:

Pixels that are close together are more likely to be related than pixels that are far apart.

Convolutional neural networks

Understanding Convolutions in Neural Networks

Convolutions are fundamental operations in many neural network architectures, especially in the field of computer vision. They apply a filter or kernel to an input to produce a feature map that captures spatial hierarchies in the data. They're also somewhat inspired by (or at least similar to) the structure of neurons in the visual cortex.

Definition of a Convolution

A convolution involves sliding a kernel (a small matrix) over the input data (like an image) and computing the dot product of the kernel and the overlapping part of the input at each position.

For an input matrix X and a kernel K, the convolution operation $*$ is defined as:

$$(S * K)(i, j) = \sum_m \sum_n X(i + m, j + n) \cdot K(m, n)$$

- Here, (i, j) are the coordinates in the output feature map S.
- m, n iterate over the kernel dimensions.

Applying Convolution in Neural Networks

In the context of a neural network, a convolutional layer applies multiple such kernels to the input. Each kernel produces a separate feature map:

$$S_k = X * K_k$$

- S_k is the feature map resulting from the k-th kernel K_k.
- The network learns the values of the kernel elements during training.

Convolutional Layer

A convolutional layer typically consists of several steps:

1. **Convolution**: Apply kernels to the input to produce feature maps.
2. **Non-linearity**: Apply an activation function, like ReLU, to each feature map.
3. **Pooling (optional)**: Downsample each feature map to reduce dimensionality.

Example: Convolution with Stride and Padding

Convolutions can be modified with 'stride' and 'padding': - **Stride**: Determines how the kernel slides over the input (e.g., skip pixels). - **Padding**: Involves adding extra pixels around the input to control the spatial size of the output feature maps.

The modified convolution formula with stride s and padding p is:

$$(S * K)(i, j) = \sum_m \sum_n X(s \cdot i + m - p, s \cdot j + n - p) \cdot K(m, n)$$