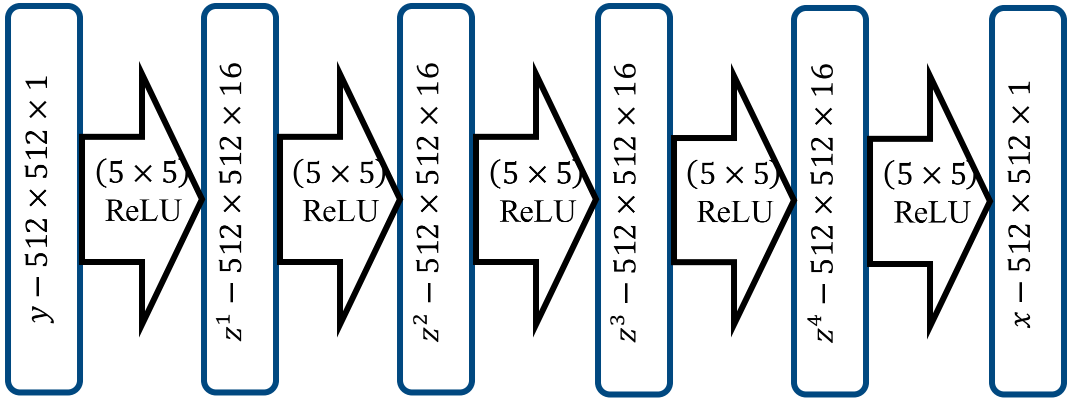
Widely Used DL Techniques

- Dead ReLUs
- Vanishing gradients and skip connections
- ResNet and Residual training
- Batch normalization
- U-Net and merged connections
- Data augmentation

Dead ReLUs



- If input to last ReLU layer is 0,

$$0 \geq w_{(j_1,j_2),i} * z^{4,(j_1,j_2),i} + b$$

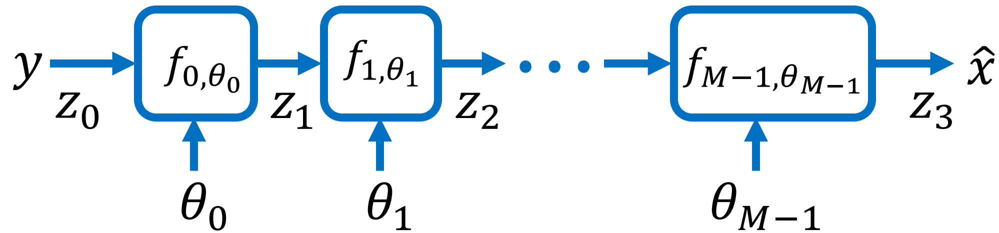
- Then we have that

$$0 = \nabla\sigma\left(w_{(j_1,j_2),i} * z^{4,(j_1,j_2),i} + b\right)$$

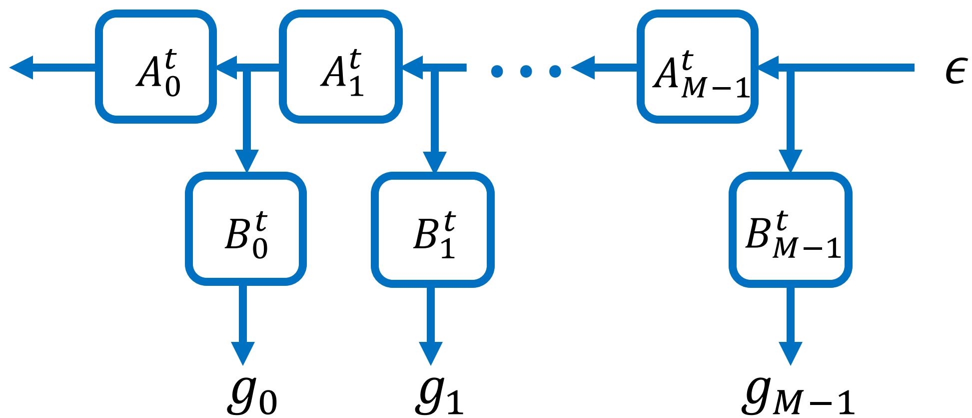
- So all back propagated gradients = 0!
- Solution: Leaky ReLU

Vanishing gradients

- Consider a pipelined DNN



- Associated back-propagation network



Background: SVD and Matrix norms

- The singular value decomposition (SVD) of $A \in \mathbb{R}^{m\times n}$ is:

$$A = U\,\Sigma\,V^t$$

where $U \in \mathbb{R}^{m\times k}$, $\Sigma = \mathrm{di}ag(\sigma_0, \dots, \sigma_{k-1}) \in \mathbb{R}^{k\times k}$, $V \in \mathbb{R}^{n\times k}$, $k = \min(m, n)$.

- The eigen decomposition of the *symmetric matrix* $A \in \mathbb{R}^{n\times n}$ is

$$A = U\,\Sigma\,U^t$$

where $U \in \mathbb{R}^{m\times k}$, $\Sigma = \mathrm{di}ag(\sigma_0, \dots, \sigma_{k-1}) \in \mathbb{R}^{k\times k}$.

- Operator norm of a matrix is

$$||A|| = \max_x \{||Ax||: ||x|| \le 1\}$$

or equivalently

$$||A|| = \max_x \left\{\frac{||Ax||}{||x||}\right\} = \max_i \sigma_i$$

Mathematical Intuition for Vanishing Gradient

- Assume all the systems have the same symmetric adjoint gradient, then

$$A_0 \, A_1 \, A_2 \cdots A_{M-1} = A^M = (U \Sigma U^T)^M = U \Sigma^M U^T$$

  - So then

$$\Sigma^M = diag(\sigma_0^M, \ldots, \sigma_{k-1}^M)$$

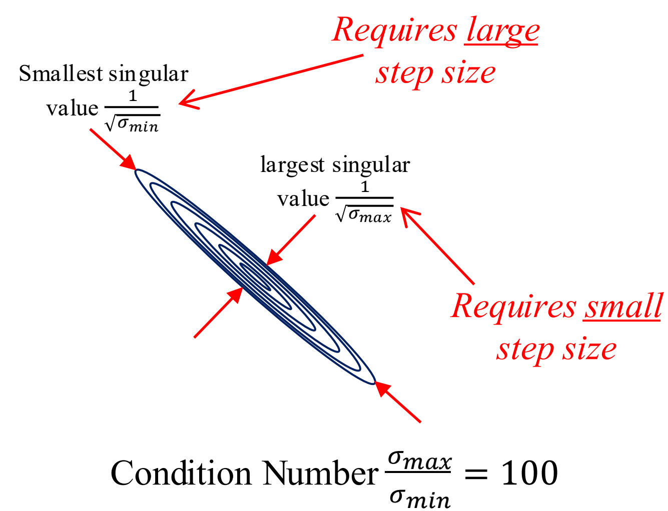
- The conditioning number is

$$\text{Condition number} = \left(\frac{\sigma_{max}}{\sigma_{min}}\right)^M$$

- So the

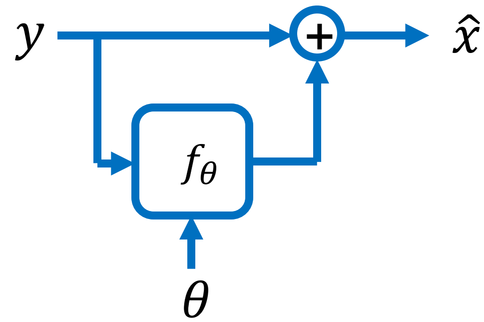
$$\lim_{M \to \infty} (\text{condition number}) = \infty$$

- Partial Solutions:
  - ADAM, skip connection, Batch Normalization
  - Normalization, Pre-Normalization

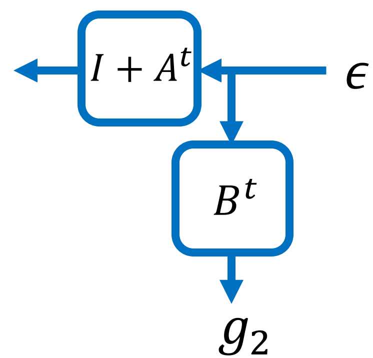


Skipped Connections

- Skip connection concept



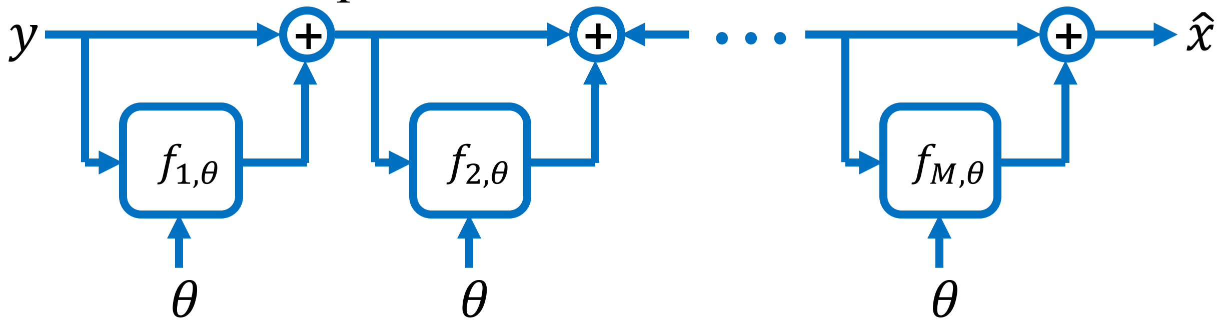
- Adjoint gradient



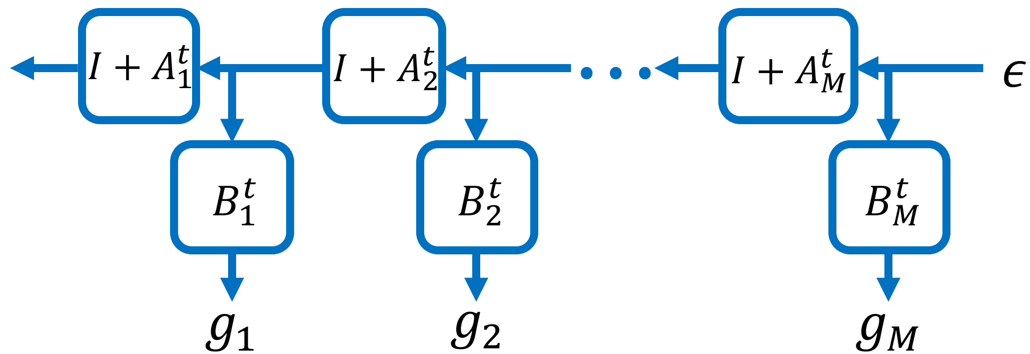
*K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 770-778.

Deep Skipped Connections

- Skip connection concept



- Back propagation



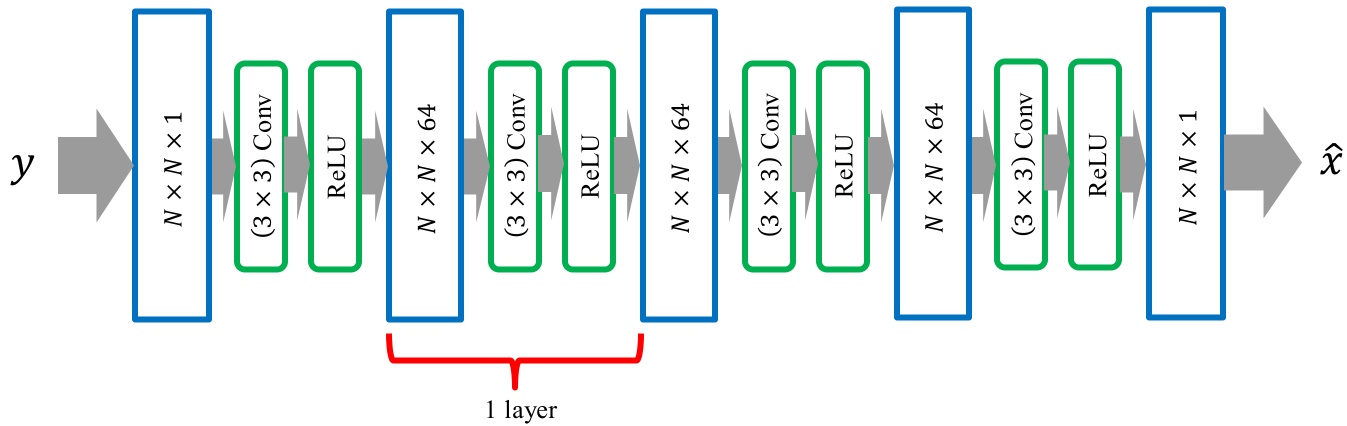
- Adjoint gradient

$$g_1 = B_1^t (I + A_2^t) \cdots (I + A_M^t)\epsilon$$

if $\sigma^*_{(I+A)} \approx 1$, we have that $\|g_0\| \le \sigma^*_{B_0} \|\epsilon\| \Rightarrow$ fixes vanishing gradient

Problem: Slow Training

- Example

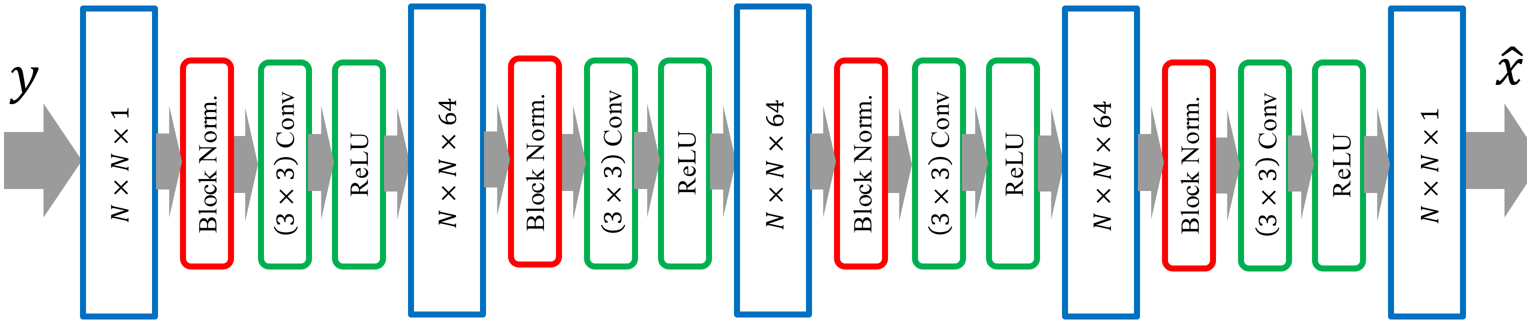


  - Training can be slow in deep layers
  - Internal features can vary rapidly

- Solution: Batch Normalization

Batch Normalization

- Example



- During training:
  - Normalizes sample mean and variance of features for each batch
  - Does not contain trainable parameters
  - Remembers mean and variance estimates learned using "momentum"
- During inference:
  - Uses mean and variance estimates to normalize features
- Issues:
  - Can be very effective, particularly with ResNet training
  - Can add multiplicative "noise" to estimate
  - Best to use when output is small or does not require high relative accuracy
  - It is sometimes useful to turn off parameter adaptation for final training.

*Sergey Ioffe and Christian Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37.

Batch Normalization: Training

- Example for layer with input feature tensor z and output tensor y

Initialize $\mu_I \leftarrow 0;\ \sigma_I^2 \leftarrow 0$

For each b (batch) {

These are noisy

$$\mu_B \leftarrow \frac{1}{K_b} \sum_{k \in S_b} z_k$$

$$\sigma_B^2 \leftarrow \frac{1}{K_b - 1} \sum_{k \in S_b} (z_k - \mu_B)^2$$

For each $k \in S_b$ {

$$\hat{z}_k \leftarrow \frac{z_k - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_k \leftarrow \gamma \hat{z}_k + \beta$$

}

These get saved for later

$$\mu_I \leftarrow \alpha\, \mu_I + (1 - \alpha)\mu_B$$

$$\sigma_I^2 \leftarrow \alpha\, \sigma_I^2 + (1 - \alpha)\sigma_B^2$$

}

Block Normalization: Training

Input parameters:

ϵ – small number to regularize division
α – momentum parameter

Learned parameters:

γ, β – scaling and offset parameters

Estimated parameters:

μ_I, σ_I^2 – learned mean and variance parameters

Parameters saved for inference:

μ_I, σ_I^2 – learned mean and variance parameters
γ, β, ϵ – same as above

Comments:

Time average over $\approx 1 - \frac{1}{\log \alpha}$

Typically, $\alpha \approx 0.99$

β used to ensure that ReLUs are not turned off

Subtle questions: Does back propagation/SGD update the values of μ_B and σ_B^2? Answer: No, these are treated as fixed program variables that are precomputed for each component feature of each batch. But this means that there is essentially a random variation with each batch.

Batch Normalization: Inference

- Example for layer with input feature tensor z and output tensor y

Inputs:

ϵ – same constant as used in training

γ, β – learned from training

μ_I, σ_I^2 – estimated during training

For each input {

$$\hat{z} \leftarrow \frac{z - \mu_I}{\sqrt{\sigma_I^2 + \epsilon}}$$

$$y \leftarrow \gamma \hat{z} + \beta$$

}

Batch Normalization: Inference

Comments:

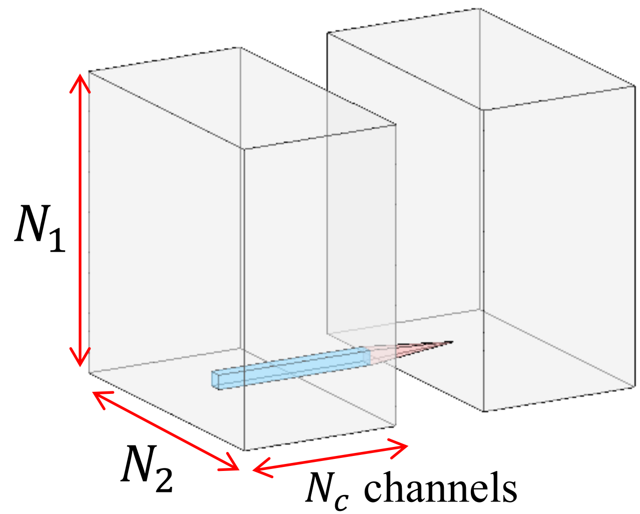
- Typically, only one input for testing
- The noisy values of μ_B and σ_B^2 used in training will typically cause variation in the output of the CNN.
- BN works well in CNNs used for residual estimation.

Warning:

- Don't use BN in CNNs that must have large and precise outputs.
- Or turn off BN adaptation in final training.

Batch Normalization: Important Practical Issues

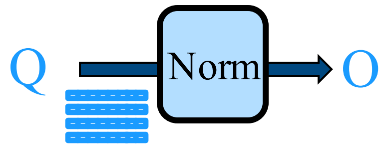
- Typically, μ_I, σ_I^2 are estimated for each channel
  - “**axis**: Integer, the axis that should be normalized (typically the channels axis). For instance, after a Conv2D layer with data_format="channels_first", set axis=1 in BatchNormalization.”



  - Channels typically contain different features for each pixel

Modern Approach: Normalization Layer

- Normalization



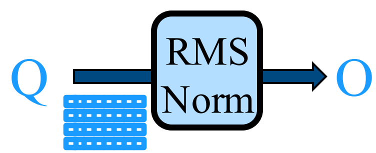
$$\mu_i \leftarrow \frac{1}{d_m}\sum_j Q_{i,j}$$

$$\sigma_i \leftarrow \sqrt{\frac{1}{d_m}\sum_j \left(Q_{i,j}-\mu_i\right)^2}$$

$$O_{i,j} \leftarrow \frac{Q_{i,j}-\mu_i}{\sigma_i}$$

- Important: These are not learned parameters.
- Each row is normalized

- RMS normalization



$$\mu_i \leftarrow \frac{1}{d_m}\sum_j Q_{i,j}$$

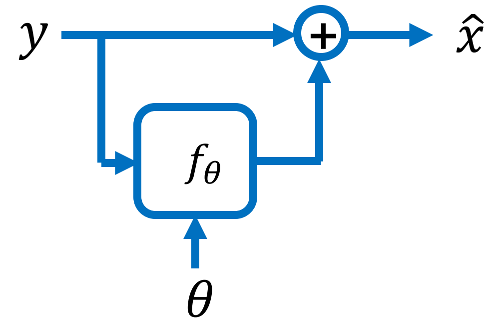
$$\sigma_i \leftarrow \sqrt{\frac{1}{d_m}\sum_j \left(Q_{i,j}-\mu_i\right)^2}$$

$$O_{i,j} \leftarrow \frac{Q_{i,j}}{\sigma_i}$$

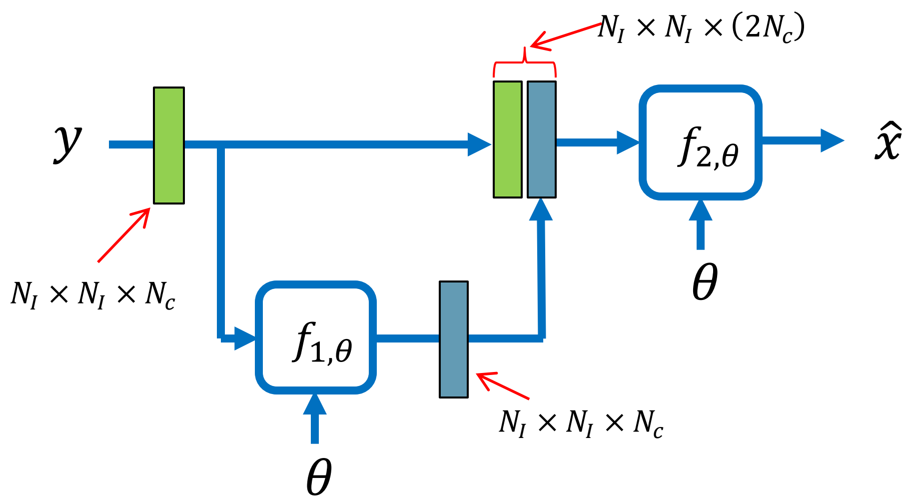
Layer normalization: Jimmy Lei Ba and Jamie Ryan Kiros and Geoffrey E Hinton, “Layer Normalization," arXiv preprint arXiv:1607.06450, 2026, https://arxiv.org/pdf/1607.06450 .
RMS normalization: Zhangyu Shi and Xiao Liu and Jianfei Chen and Jun Zhu, “RMSNorm: Simple Normalization for Transformers,” arXiv preprint arXiv:1910.07467, 2019, https://arxiv.org/pdf/1910.07467 .

Merge Connections*

- Problem with skip connection
  - Some information may be lost by simply adding y and $f_\theta(y)$
  - Example, $f_\theta(y) = -y$

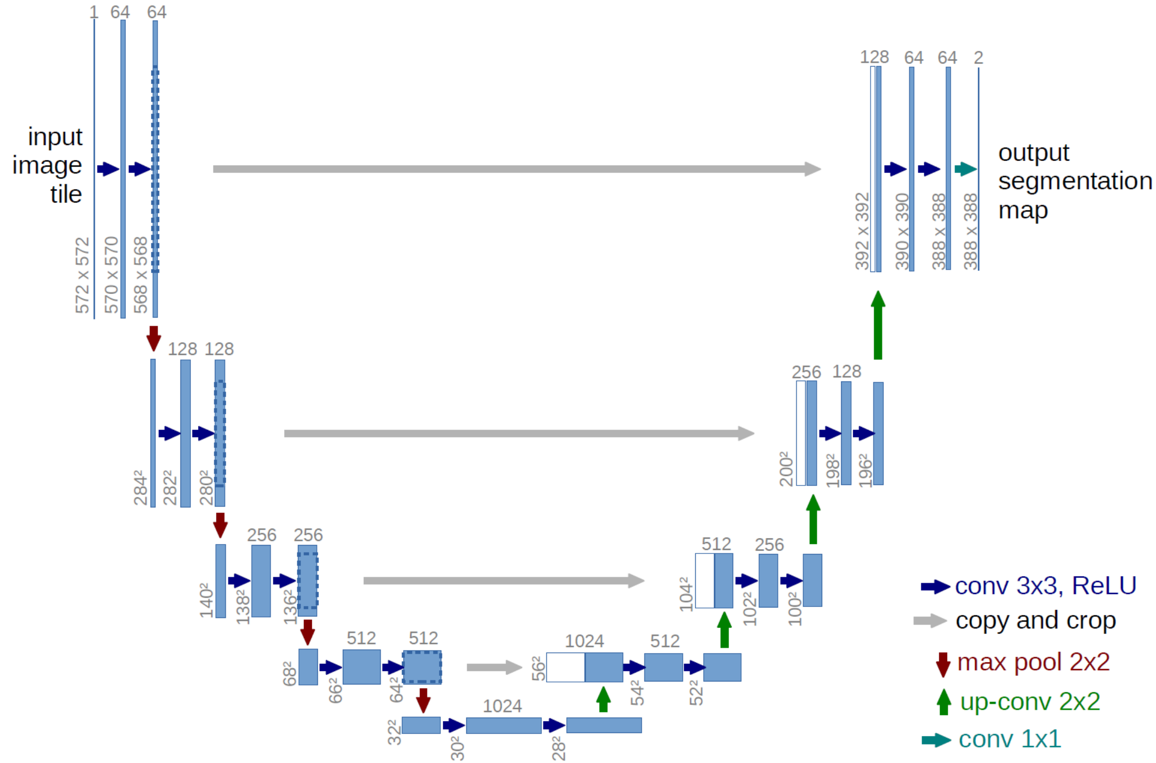


- Merge Connection*
  - Concatenate y and $f_\theta(y)$



*As far as I know, I made up this term.

U-Net*



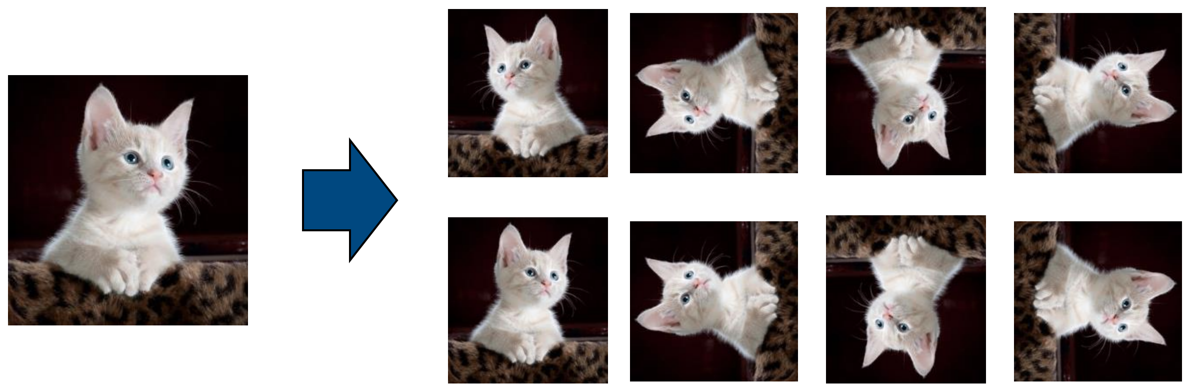
- Can model long distance dependencies
- Much like a wavelet transform, but nonlinear

Figure taken from https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

*Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", Medical Image Computing and Computer-Assisted Intervention (MICCAI 2015), pp. 234-241, 2015.

Data Augmentation

- Problem: Not enough training data
- Solution: Transform the data in different ways



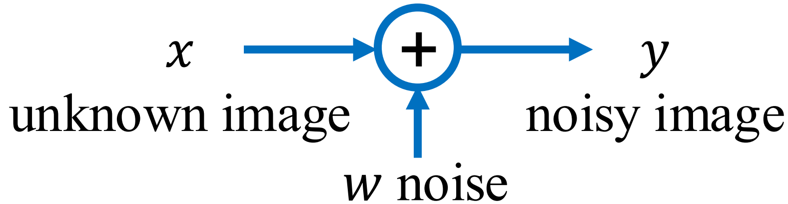
- Can use:
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - Whatever…

Denoising

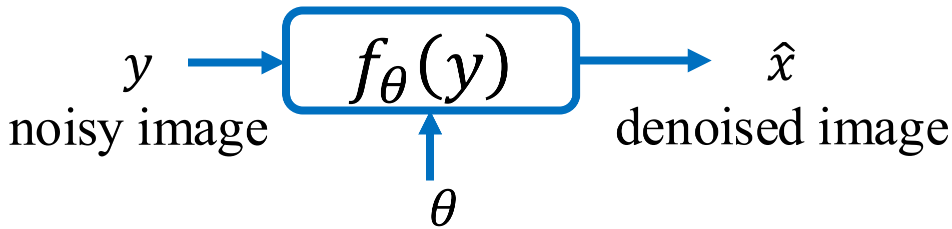
- Predicting noise and Residual training
- DnCNN and ResNet denoiser
- DRUNET denoiser
- Plug-and-Play Denoiser

Denoising

- The forward model is $y = x + w$



- The denoising inverse problem



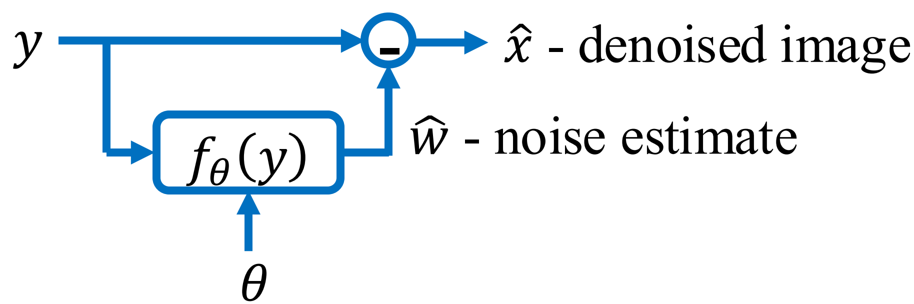
  - Simplest of all inverse problems
  - Typically, $w \sim N(0, \sigma^2 I)$
  - Loss function is $L(\theta) = \frac{1}{K}\sum_{k=0}^{K-1} \|x_k - f_\theta(y_k)\|^2$
  - Maximum likelihood estimate of θ
$$\hat{\theta} = \arg\min_{\theta} L(\theta)$$
  - MMSE estimate of x
$$\hat{x} = f_{\hat{\theta}}(y)$$

Predicting the Noise

- Since the forward model is $y = x + w$, denoised image is given by

$$\begin{aligned}\hat{x} &= E[x|y] \\ &= E[y - w|y] \\ &= y - E[w|y] \\ &= y - f_{\hat{\theta}}(y)\end{aligned}$$

  where $\widehat{w} = f_{\hat{\theta}}(y)$ is an estimate of the noise.



- Intuition:
  - Noise is easier to estimate because it is “smaller”
  - Estimate the noise, and subtract it from the noisy image
  - Same as skipped connection

DnCNN Denoiser

- DnCNN – Deep Neural Convolutional Neural Network
  - Widely used network that first introduced the concept of estimating the noise rather than the image.

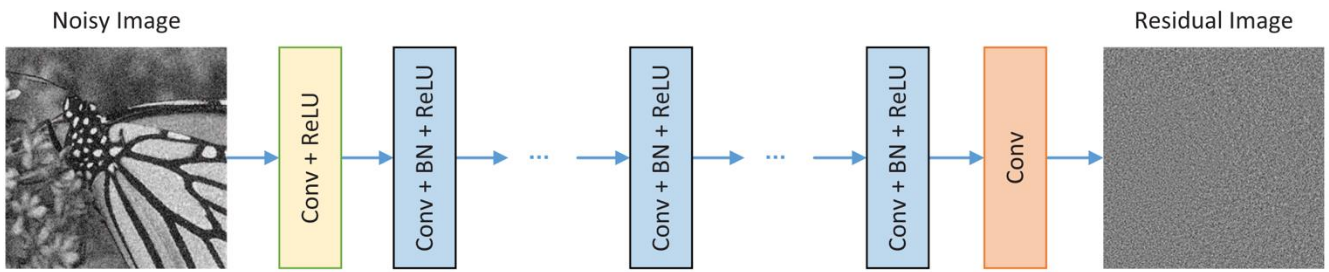


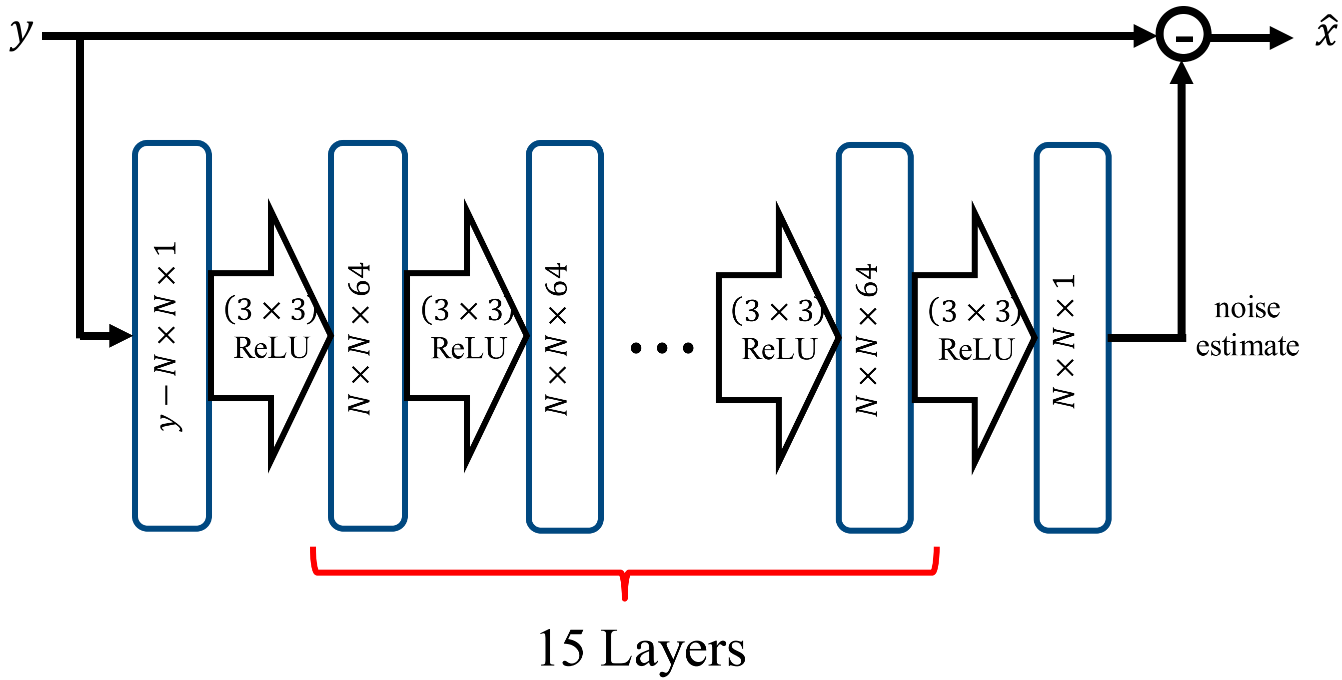
Figure reproduced from *

- Limitations:
  - Fixed resolution limits the long-distance and multi-scale dependencies
  - Must be trained for a specific noise type and variance.

*K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," in *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142-3155, July 2017, doi: 10.1109/TIP.2017.2662206.

Denoising ResNet (DN-ResNet)*

- Typical DN-ResNet denoising network



- Limitation: Dependencies are local

*Haoyu Ren, Mostafa El-khamy, and Jungwon Lee, "DN-ResNet: Efficient Deep Residual Network for Image Denoising", 14th Asian Conference on Computer Vision, Perth, Australia, Dec. 2–6, 2018.

DRUNET Denoiser

- DRUNET
  - Close state-of-the-art denoising network based on U-net architecture.

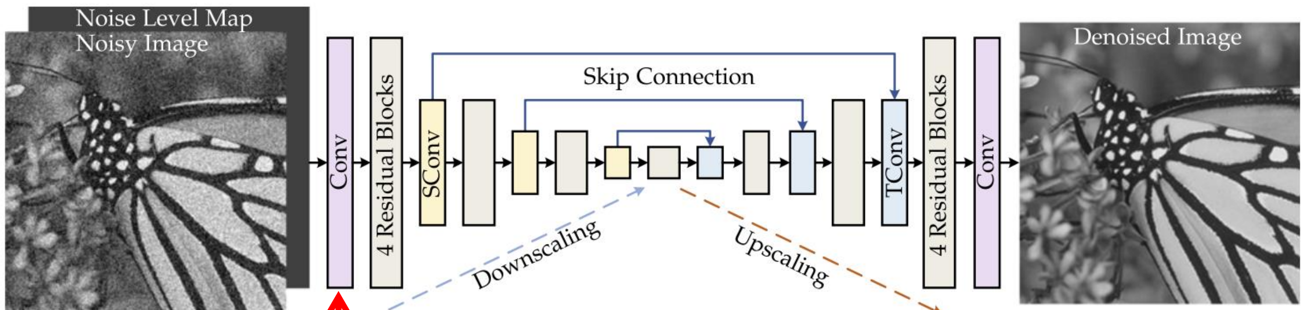


Figure reproduced from *

Insert positional encoding of σ using d layers.

- Advantages:
  - Residual encoding is built into U-Net architecture.
  - Allows for input parameter that controls the amount of noise, σ.
  - Encodes the noise level, σ, as a input channel.

*K. Zhang, Y. Li, W. Zuo, L. Zhang, L. Van Gool and R. Timofte, "Plug-and-Play Image Restoration With Deep Denoiser Prior," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 10, pp. 6360-6376, 1 Oct. 2022, doi: 10.1109/TPAMI.2.

Plug-and-Play Denoiser

Figure reproduced from *

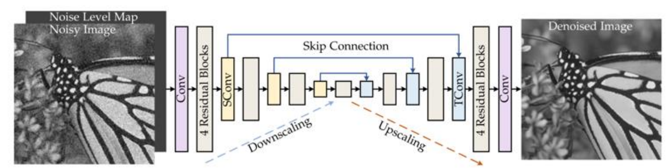
- Plug-and-Play
  - Iterate denoiser with parametric physical model.

Algorithm 1. Plug-and-Play Image Restoration With Deep Denoiser Prior (DPIR)

Input : Deep denoiser prior model, degraded image $\mathbf{y}$, degradation operation $\mathcal{T}$, image noise level σ, σ_k of denoiser prior model at kth iteration for a total of K iterations, trade-off parameter λ.
Output : Restored image $\mathbf{z}_K$.

Initialize $\mathbf{z}_0$ from $\mathbf{y}$, pre-calculate $\alpha_k \triangleq \lambda\sigma^2/\sigma_k^2$.
for $k = 1, 2, \ldots, K$ **do**
  $\mathbf{x}_k = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathcal{T}(\mathbf{x})\|^2 + \alpha_k\|\mathbf{x} - \mathbf{z}_{k-1}\|^2$; // *Solving data subproblem*
  $\mathbf{z}_k = Denoiser(\mathbf{x}_k, \sigma_k)$; // *Denoising with deep DRUNet denoiser and periodical geometric self-ensemble*
end

Proximal map representing a physical model



- Advantages: Combines DNN with physical model.

*Singanallur V. Venkatakrishanan, Charles A. Bouman, and Brendt Wohlberg, "Plug-and-Play Priors for Model Based Reconstruction," IEEE 2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Austin, Texas, USA, December 3-5, 2013.

*K. Zhang, Y. Li, W. Zuo, L. Zhang, L. Van Gool and R. Timofte, "Plug-and-Play Image Restoration With Deep Denoiser Prior," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 44, no. 10, pp. 6360-6376, 1 Oct. 2022, doi: 10.1109/TPAMI.2.

Positional Encoding

- Positional Encoding method and examples
- Encoding denoising parameters
- NeRFs and parametric encoding

Positional Encoding

- Goal: Encode a continuous position, t, as a d dimensional vector.

$$P(t) = [P(t,0), P(t,1), \cdots, P(t,d-1)]$$

- Formula

$$P(t,2i) = \sin\left(\frac{t}{\lambda^i}\right)$$
$$P(t,2i+1) = \cos\left(\frac{t}{\lambda^i}\right)$$

  - where $\lambda = N^{\frac{2}{d}}$
  - Typical values are $d = 512$ and $N = 10{,}000$

- Example: $d = 4$ and $N = 10$

	$i = 0$	$i = 0$	$i = 1$	$i = 1$
$t = 0$	$\sin(0) = 0$	$\cos(0) = 1$	$\sin(0) = 0$	$\cos(0) = 1$
$t = 1$	$\sin(1/1) = 0.84$	$\cos(1/1) = 0.54$	$\sin(1/10) = 0.10$	$\cos(1/10) = 1.0$
$t = 2$	$\sin(2/1) = 0.91$	$\cos(2/1) = -0.42$	$\sin(2/10) = 0.20$	$\cos(2/10) = 0.98$
$t = 3$	$\sin(3/1) = 0.14$	$\cos(3/1) = -0.99$	$\sin(3/10) = 0.30$	$\cos(3/10) = 0.98$

*Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need", Advances in neural information processing systems 30 (2017).

Positional Encoding

- Goal: Encode a continuous position, t, as a d dimensional vector.

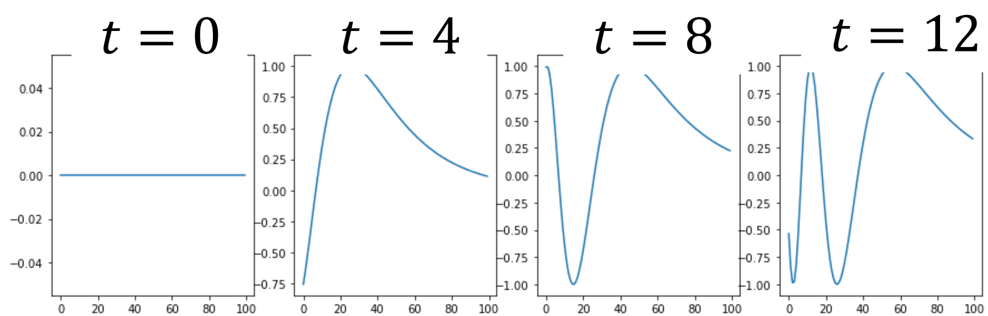
$$P(t) = [P(t,0), P(t,1), \cdots, P(t,d-1)]$$

- Formula

$$P(t,2i) = \sin\left(\frac{t}{\lambda^i}\right)$$
$$P(t,2i+1) = \cos\left(\frac{t}{\lambda^i}\right)$$

  – where $\lambda = N^{\frac{2}{d}}$

- Example: $d = 512$ and $N = 10{,}000$



*Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need", Advances in neural information processing systems 30 (2017).

Positional Encoding: Example

- Example: $d = 512$ and $N = 10{,}000$

ChatGPT 3.5

You
please write numpy code that creates an image with row corresponding to the positional encoding function commonly used in transformers.

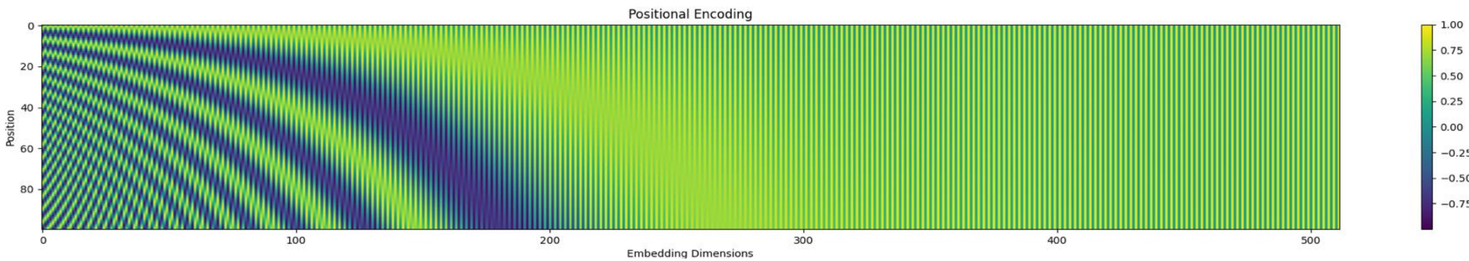
ChatGPT
Certainly! In transformers, positional encoding is commonly used to provide information about the position of tokens in the input sequence. The positional encoding function often used is based on sine and cosine functions. Here's how you can create an image representing positional encoding using numpy:

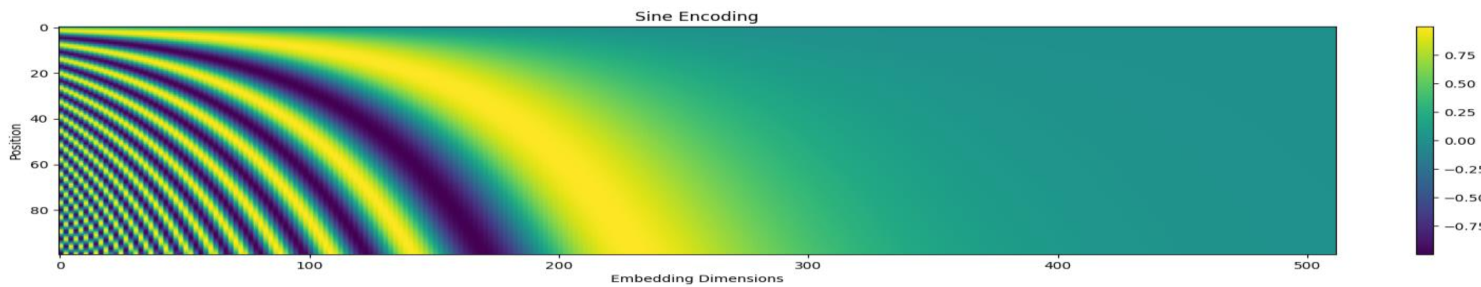
python Copy code
```python
import numpy as np
import matplotlib.pyplot as plt

def positional_encoding(max_len, d_model):
    pos = np.arange(max_len)[:, np.newaxis]
    i = np.arange(d_model)[np.newaxis, :]
    angles = pos / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
    encoding = np.zeros((max_len, d_model))
    encoding[:, 0::2] = np.sin(angles[:, 0::2])
    encoding[:, 1::2] = np.cos(angles[:, 1::2])
    return encoding

def plot_positional_encoding(encoding):
    plt.imshow(encoding, cmap='viridis', aspect='auto')
    plt.xlabel('Embedding Dimensions')
    plt.ylabel('Position')
    plt.title('Positional Encoding')
    plt.colorbar()
    plt.show()

max_len = 100  # Maximum sequence length
d_model = 512  # Embedding dimensions
encoding = positional_encoding(max_len, d_model)
plot_positional_encoding(encoding)
```

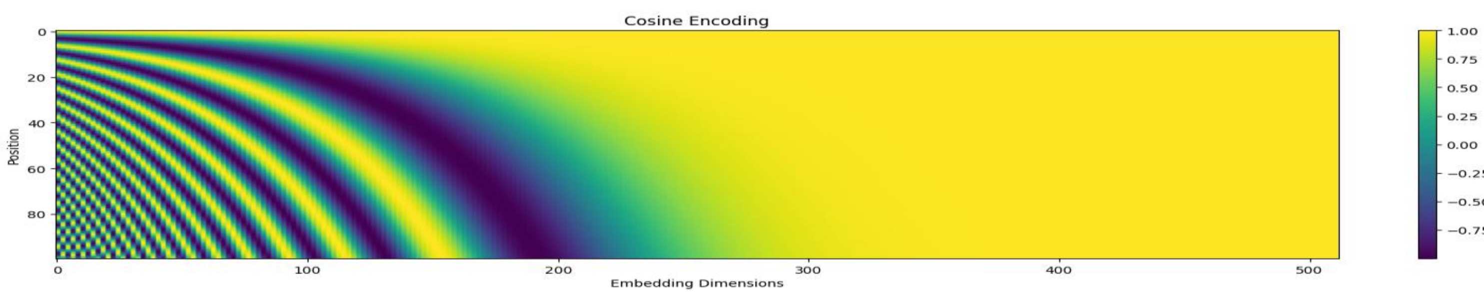
This code will generate an image where the x-axis represents the embedding dimensions and the y-axis represents the position in the sequence. The color intensity represents the values of the positional encoding function. Adjust `max_len` and `d_model` parameters according to your requirements.

Positional Encoding: Example

- Example: $d = 512$ and $N = 10{,}000$

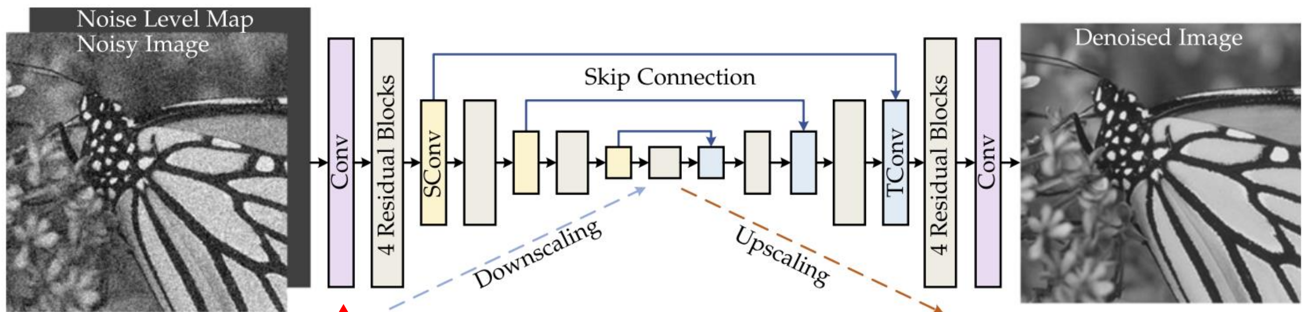






Denoisers with Positional Encoding

- Innovation:
  - Use positional encoding to insert information about noise standard deviation, σ.

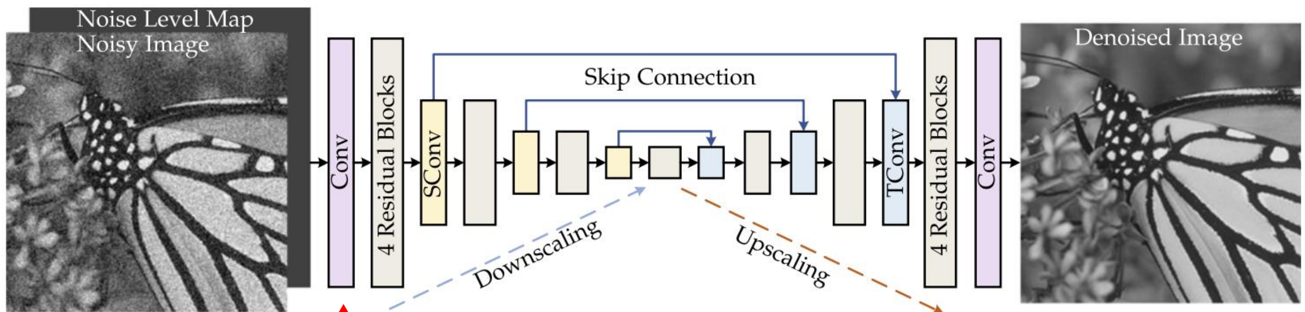


Insert positional encoding of σ using d layers.

* Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising Diffusion Probabilistic Models", Advances in Neural Information Processing Systems, 2020.

Denoisers with Positional Encoding

- Innovation:
  - Use positional encoding to insert information about noise standard deviation, σ.

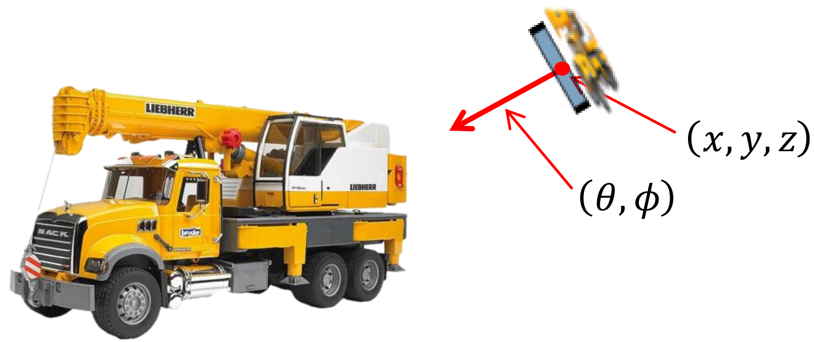


Insert positional encoding
of σ using d layers.

* [Jonathan Ho, Ajay Jain, and Pieter Abbeel, "Denoising Diffusion Probabilistic Models", Advances in Neural Information Processing Systems, 2020.]

The Radiance Field Problem

- The problem:
  - How do you represent a complex scene from any location in space (x, y, z) and orientation (θ, ϕ)?



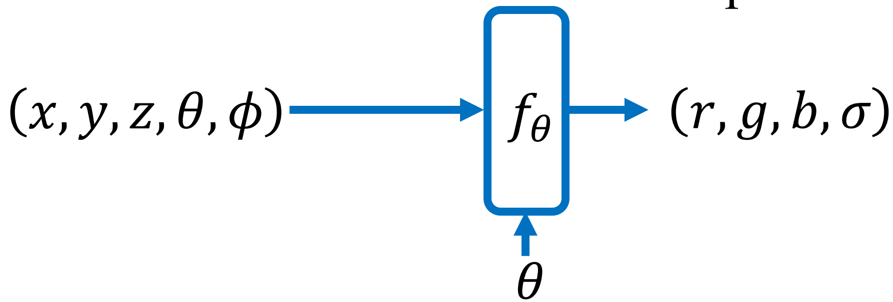
  - This requires that you store a function $r(x, y, z, \theta, \phi)$.
  - If we use single precision floats, this requires:

$$\boxed{1 \text{ PetaByte}} = 1{,}024 \times 1{,}024 \times 1{,}024 \times 360 \times 180 \times (4 \text{ colors}) \times (4\text{Bytes})$$

* website, paper: Ben Mildenhall and Pratul P. Srinivasan and Matthew Tancik and Jonathan T. Barron and Ravi Ramamoorthi and Ren Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020.

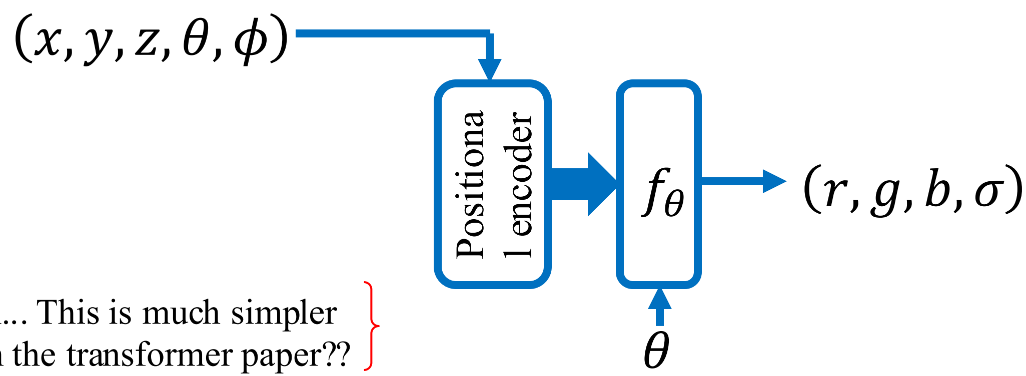
NeRF*: Neural Radiance Field

- The big idea:
  - Train a neural net with *coordinate* inputs.



But this doesn't work so well.

  - Then use a positional encoder!.



But this works great!

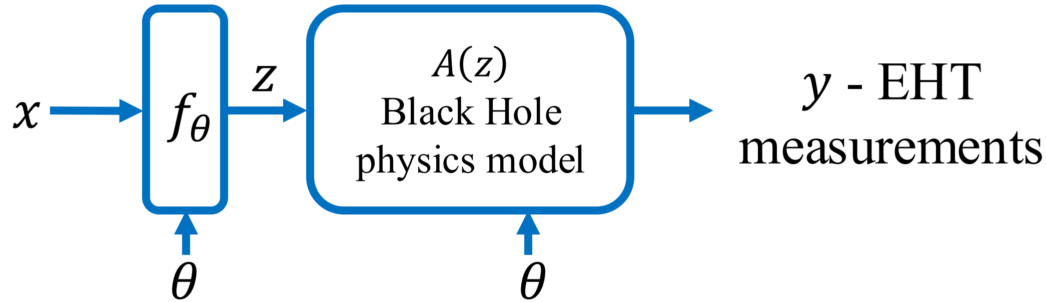
Hmm... This is much simpler than in the transformer paper??

$$\gamma(p) = (\sin(2^0 \pi p), \cos(2^0 \pi p), \cdots, \sin(2^{L-1} \pi p), \cos(2^{L-1} \pi p),)$$

* website, paper: Ben Mildenhall and Pratul P. Srinivasan and Matthew Tancik and Jonathan T. Barron and Ravi Ramamoorthi and Ren Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis", ECCV 2020.

Tomographic Imaging of Black Holes

- Imaging Model:



$$L(\theta) = \sum_x \sum_y \left\| y - A\big(f_\theta(x)\big) \right\|^2$$

- Inverse problem:

$$\theta^* = \arg\min_\theta L(\theta)$$

* paper: Aviad Levis, Andrew A Chael, Katherine L Bouman, Maciek Wielgus, Pratul P Srinivasan, "Orbital polarimetric tomography of a flare near the Sagittarius A* supermassive black hole", Nature Astronomy, 2024.