0 Instructions (Total 50pts)

Homework is due Monday, March 20, 2024 at 23:59pm Central Time. Please refer to `https://courses.grainger.illinois.edu/cs446/sp2024/homework/hw/index.html` for course policy on homeworks and submission instructions.

1 Neural networks for simple functions (10pts)

In this problem you will be hand designing simple neural networks to model specific functions. In each case assume $x \in \mathbb{R}$ and provide appropriate weights $w_0, w_1, b_0 \in \mathbb{R}^d$ for some d. In other words, each neural network has one input neuron, d hidden neurons, and one output neuron.

1. (1pts) $w_0, w_1 \in \mathbb{R}^2$ such that $f(x) = w_1^T \sigma(w_0 x) = x, \forall x \in \mathbb{R}$ where $\sigma = \text{ReLU}$ (note: $w_0 x$ here is a vector-scalar product: $\mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$, e.g. $[0,1]^T x = [0, x]^T$)

   Answer: $w_0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $w_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

   Grading: 1 point for correct weights, 0 otherwise

   Grading NOTE: some of these problems can have more than one solution, for example in 1.2 they can swap the weights of w_0, w_1

2. (1pts) $w_0, w_1, b_0 \in \mathbb{R}^2$ such that $f(x) = w_1^T \sigma(w_0 x + b_0) = mx + b, \forall x \in \mathbb{R}$ where $\sigma = ReLU$

   Answer: $w_0 = \begin{bmatrix} m \\ -m \end{bmatrix}$, $w_1 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, $b_0 = \begin{bmatrix} b \\ -b \end{bmatrix}$

   Grading: 1 point for correct weights, 0 otherwise

3. (1pts) $w_0, w_1 \in \mathbb{R}$ such that $f(x) = w_1^T \sigma(w_0 x) = b, \forall x \in \mathbb{R}$ where $\sigma = sigmoid$

   Answer: $w_0 = \begin{bmatrix} 0 \end{bmatrix}$, $w_1 = \begin{bmatrix} 2b \end{bmatrix}$

   Grading: 1 point for correct weights, 0 otherwise

4. (2pts) $w_0, w_1, b_0 \in \mathbb{R}^3, \sigma = ReLU$, and $f(x) = w_1^T \sigma(w_0 x + b_0) = \begin{cases} 0 & x \leq -2 \\ 3x + 6 & -2 \leq x \leq 0 \\ -3x + 6 & 0 \leq x \leq 2 \\ 0 & x \geq 2 \end{cases}$

Answer: $w_0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $w_1 = \begin{bmatrix} 3 \\ 3 \\ -6 \end{bmatrix}$, $b_0 = \begin{bmatrix} 2 \\ -2 \\ 0 \end{bmatrix}$

Grading: 2 point for correct weights, 1 point if close, 0 otherwise

5. (1pts) Do there exist $w_0, w_1 \in \mathbb{R}^d$ such that $f(x) = w_1^T \sigma(w_0 x) = x^2, \forall x \in \mathbb{R}$ where $\sigma = ReLU$. Explain why or why not.

   Answer: Impossible, many reasons. Quadratic vs linear functions, i.e., the derivative of f is constant...

   Grading: 1 point for correct answer (NO) with reasonable explanation, 0 otherwise. Look for keywords - e.g., comparison of slopes

6. (2pts) For any $\epsilon > 0$ do there exist fixed finite dimensional $w_0, w_1, b_0 \in \mathbb{R}^d$ such that $|f(x) - x^2| = |w_1^T \sigma(w_0 x + b_0) - x^2| < \epsilon, \forall x$ where $\sigma = ReLU$? Explain why or why not. If yes describe a procedure to construct w_0, w_1, b_0 at a high level.

   Answer: Impossible because x is unbounded. For any fixed ϵ, d, w_0, w_1 the derivative of f is bounded, $||\frac{df}{dx}|| \leq d||w_1||_\infty(||w_0||_\infty + ||b_0||_\infty)$, while the derivative of x^2 is not.

   Grading:

   - 2 for correct answer and explanation (refers to epsilon)
   - -1 for 1 minor error in proof
   - -2 for 2 or more minor errors in proof

7. (2pts) For any $\epsilon > 0$ do there exist fixed finite dimensional w_0, w_1, b_0 such that $|f(x) - x^2| = |w_1^T \sigma(w_0 x + b_0) - x^2| < \epsilon$ where $\sigma = ReLU$ and $x \in [0, 1]$? Explain why or why not. If yes describe a procedure to construct w_0, w_1, b_0 at a high level.

   Answer: Yes, consider step functions of the form $m(\sigma(x) - \sigma(x - \delta))$. We can create a linear combination of such step functions to approximate any bounded function.

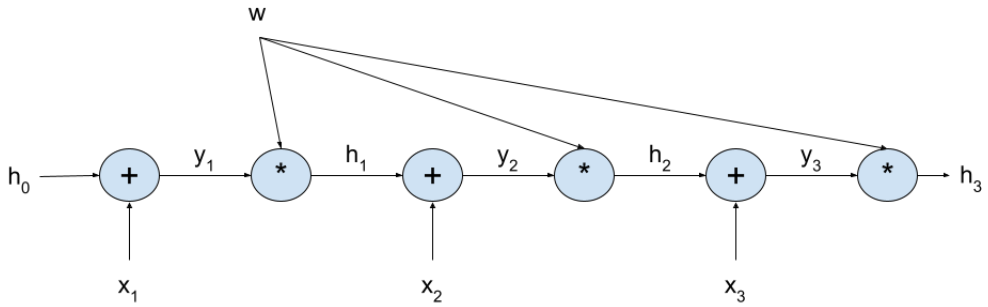
   Grading:

   - 2 for correct answer (YES) and explanation which either describes how we can use step-like functions to approximate each small region, or which refers to a gradient descent loop - note this answer MUST mention how to select the dimension d for full credit.
   - -1 for 1 minor error in proof

- -2 for 2 or more minor errors in proof

2 Backpropagation through time (BPTT) (8pts)

1. (2pts) Consider recurrent model with recurrence $h_t = w(x_t + h_{t-1})$, and $h_0 = 0$. Draw the computation graph for unrolling this RNN for 3 timesteps. You should include 3 inputs x_1, x_2, x_3, the outputs h_0, h_1, h_2, h_3, and refer to intermediate values $y_i = (x_i + h_{i-1})$.



Answer:

Grading:

- -1 if missing some variables
- -1 if incorrect graph

2. (1pts) Compute the forward pass in the above model, i.e., compute $h_0, h_1, h_2, h_3, y_1, y_2, y_3$ as functions of w, x_1, x_2, x_3.

Answer: $y_1 = x_1, h_1 = wx_1, y_2 = wx_1 + x_2, h_2 = w^2x_1 + wx_2, y_3 = w^2x_1 + wx_2 + x_3, h_3 = w^3x_1 + w^2x_2 + wx_3$

Grading:

- -1 if any values are incorrect

3. (2pts) Compute $\frac{\partial h_3}{\partial w}$ in the above model using a backwards pass and your values from the forward pass. HINT: you can directly take the derivative of h_3 from the previous question to check your work, but you must show work for a proper backprop including intermediate partial derivatives.

Answer:

$$\frac{\partial h_3}{\partial w} = y_3 + y_2\frac{\partial h_3}{\partial h_2} + y_1\frac{\partial h_3}{\partial h_1} = y_3 + wy_2 + w^2y_1 = w^2x_1 + wx_2 + x_3 + w^2x_1 + wx_2 + w^2x_1 =$$

$$3w^2x_1 + 2wx_2 + x_3$$

Grading:

- -1 if incorrect value
- -2 if doesn't show work - sum of 3 partial derivative terms (one for each path)

4. (2pts) For sequence $z = [x_1, ..., x_T]$ Consider recurrent model $f(z) = h_T$, where $h_t = w\sigma(x_t + h_{t-1})$, and $h_0 = 0$. Derive $\frac{\partial f}{\partial h_1}$, i.e., $\frac{\partial h_T}{\partial h_1}$. You may use σ' to represent the derivative of nonlinearity σ.

   Answer: $\frac{\partial h_t}{\partial h_{t-1}} = w\sigma'(x_t + h_{t-1}), \frac{\partial h_t}{\partial h_{t-2}} = \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}} = w^2\sigma'(x_t + h_{t-1})\sigma'(x_{t-1} + h_{t-2}) \implies \frac{\partial h_T}{\partial h_1} = w^{T-1}\prod_{i=1}^{T}\sigma'(x_i + h_{i-1})$

   Grading:

   - -1 if incorrect value
   - -1 if answer contains partial derivatives

5. (1pts) Using your previous answer as justification, explain the exploding/vanishing gradient problem in RNNs.

   Answer: The term w^{T-1} will make the gradient vanish if $w < 1$ and explode if $w > 1$ with large enough T

   Grading:

   - -1 if answer isn't with respect to previous answer
   - OR
   - -1 if doesn't talk about both exploding and vanishing

3 Transformers (7pts)

In self attention we compute the output vector $c = \sum_{i=1}^{n} \alpha_i v_i$ for n value vectors $v_i \in \mathbb{R}^d$ and attention weights $\alpha_i = \frac{exp(k_i^T q)}{\sum_j exp(k_j^T q)}$ computed as a softmax over query vector $q \in \mathbb{R}^d$ and n keys $k_i \in \mathbb{R}^d$.

1. (1pts) Can α_i be infinitely large (or can the denominator be zero)? Can α_i be zero? Explain your answer, including both theoretical and practical/computational considerations.

   Answer: In theory, after normalization all weights will be between 0 and 1. In practice, if all keys are orthogonal to the query we can get division by zero errors, such that $alpha_i$ is not well defined. In theory no individual weight can be 0 (because the exponential is never 0) but in practice under fixed precision you can often get 0 weights.

   Grading:

   - Give full credit as long as mention "precision" in practical setting and answer NO for theoretical.
   - NOTE: some students probably answered an old version of the problem that didn't specify practical and theoretical - give full credit for just theoretical.

2. (1pts) Describe q and k_i such that $\alpha_i \gg \alpha_j$ for some i and all $j \neq i$. What is c in this case? Explain intuitively what such a case represents in the context of attention.

   Answer: q and k_i should be parallel (large dot product) and every other key vector should be orthogonal to q (small dot product). In this case $c \approx v_i$, which we can think of conceptually as a query for the i'th key-value.

   Grading:

   - Give full credit as long as they have the right idea with respect to making one large dot product and other small ones.

3. (1pts) Assuming all key vectors are orthonormal, $k_i^T k_j = 0$ and $||k_i|| = 1$, give a q such that $c \approx \frac{1}{2}(v_1 + v_2)$.

   Answer: $q = a(k_1 + k_2)$ for large a

   Grading:

   - -0.5 if only $q = (k_1 + k_2)$.

4. (2pts) The dot product is one way of measuring similarity between the query and the keys. We can replace it with any similarity, e.g., a kernel. Show that you can rewrite the dot product softmax similarity, $\exp(q^T k_i)$, with the RBF kernel, $\exp(\frac{-||q-k_i||_2^2}{2\sigma})$. In other words, with $\beta_i = \exp(\frac{-||q-k_i||_2^2}{2\sigma}) / \sum_j \exp(\frac{-||q-k_j||_2^2}{2\sigma})$, show that $\beta_i = \alpha_i$. You may assume $\sigma = 1$ and all the keys all have norm 1, $||k_i||_2 = 1$.

   Answer: $||q-k_i||_2^2 = (q-k_i)^T(q-k_i) = q^Tq + k_i^Tk_i + 2q^Tk_i = q^Tq + 1 - 2q^Tk_i \implies \exp(\frac{-||q-k_i||_2^2}{2\sigma}) = \exp(\frac{q^Tk_i}{\sigma})\exp(\frac{-1-q^Tq}{2\sigma}) = \exp(q^Tk_i)\exp(\frac{-1-q^Tq}{2})$**. The second term does not depend on the index and so cancels out in the normalization.**

   Grading:

   - -1 for one minor mistake in proof (for example not simplifying fully to show how constant terms cancel).
   - -2 for two or more minor mistake in proof.

5. (2pts) In general we have one query per element in the sequence of length n. We can rewrite such an operation as $attention(Q, K, V) = softmax(QK^T)V$ for 3 (n, d) matrices. How does computing the output scale with n, the sequence length? Let $P = softmax(QK^T)$ and assume P has rank k and we know its SVD. Show that attention can be computed in $\mathcal{O}(nkd)$ time.

   Answer: $\mathcal{O}(n^2d)$**. Let** $P = U\Sigma X^T \in \mathbb{R}^{n\times n}$ **where** $U \in \mathbb{R}^{n\times n}, \Sigma \in \mathbb{R}^{n\times n}, X \in \mathbb{R}^{n\times n}$**. Since** P **is rank k this means only the first k singular values are non-zero, meaning we can remove the last** $n-k$ **columns of** U, X**.** $P = \hat{U}\hat{\Sigma}\hat{X}^T$**, where** $\hat{U} \in \mathbb{R}^{n\times k}, \hat{\Sigma} \in \mathbb{R}^{k\times k}, \hat{X} \in \mathbb{R}^{n\times k}$**. Finally, computing the product** $PV = \hat{U}\hat{\Sigma}\hat{X}^TV = (\hat{U}(\hat{\Sigma}(\hat{X}^TV))$ **which is** $\mathcal{O}(nkd + k^2d + nkd) = \mathcal{O}(nkd)$

   Grading:

   - -1 for incorrect answer to the first question.
   - -1 for not explaining how the SVD allows for shorter runtime.

4 Resnet (12pts)

In this problem, you will implement a simplified ResNet. You do not need to change arguments which are not mentioned here (but you of course could try and see what happens).

1. (5pts) Implement a class `Block`, which is a building block of ResNet.

   The input to `Block` is of shape (N, C, H, W), where N denotes the batch size, C denotes the number of channels, and H and W are the height and width of each channel. For each data example $\boldsymbol{x}$ with shape (C, H, W), the output of `block` is

   $$\texttt{Block}(\boldsymbol{x}) = \sigma_r(\boldsymbol{x} + f(\boldsymbol{x})),$$

   where σ_r denotes the ReLU activation, and $f(\boldsymbol{x})$ also has shape (C, H, W) and thus can be added to $\boldsymbol{x}$. In detail, f contains the following layers.

   (a) A `Conv2d` with C input channels, C output channels, kernel size 3, stride 1, padding 1, and no bias term.

   (b) A `BatchNorm2d` with C features.

   (c) A ReLU layer.

   (d) Another `Conv2d` with the same arguments as (a) above.

   (e) Another `BatchNorm2d` with C features.

   Because 3×3 kernels and padding 1 are used, the convolutional layers do not change the shape of each channel. Moreover, the number of channels are also kept unchanged. Therefore $f(\boldsymbol{x})$ does have the same shape as $\boldsymbol{x}$.

   Additional instructions are given in doscstrings in `hw3.py`.

   Suggested Library routines: `torch.nn.Conv2d and torch.nn.BatchNorm2d.`

   Remark: Use `bias=False` for the `Conv2d` layers.

2. (2pts) Explain why a `Conv2d` layer does not need a bias term if it is followed by a `BatchNorm2d` layer.

   Answer :

   The `Conv2d` layer outputs a 4d tensor with shape (N, C, H, W). Each channel consists of $N \times H \times W$ elements; denote them by $x_{i,h,w}$. The `BatchNorm2d` layer computes the

mean them

$$\mu = \frac{1}{N \times H \times W} \sum_{i=1}^{n} \sum_{h=1}^{H} \sum_{w=1}^{W} x_{i,h,w},$$

and subtract μ from every $x_{i,h,w}$.

Suppose we add a bias term b to every $x_{i,h,w}$: $x'_{i,h,w} = x_{i,h,w} + b$. Then the mean of $x'_{i,h,w}$ also change by b: $\mu' = \mu + b$. As a result $x_{i,h,w} - \mu = x'_{i,h,w} - \mu'$, and thus we do not need this bias term.

Grading:

- -1 for one minor error in proof.
- -2 for two or more minor error in proof.

3. (5pts) Implement a (shallow) `ResNet` consists of the following parts:
   (a) A `Conv2d` with 1 input channel, C output channels, kernel size 3, stride 2, padding 1, and no bias term.
   (b) A `BatchNorm2d` with C features.
   (c) A ReLU layer.
   (d) A `MaxPool2d` with kernel size 2.
   (e) A `Block` with C channels.
   (f) An `AdaptiveAvgPool2d` which for each channel takes the average of all elements.
   (g) A `Linear` with C inputs and 10 outputs.

   Additional instructions are given in doscstrings in `hw3.py`.

   Suggested Library routines: `torch.nn.Conv2d`, `torch.nn.BatchNorm2d`, `torch.nn.MaxPool2D`, `torch.nn.AdaptiveAvgPool2d` and `torch.nn.Linear`.

   Remark: Use `bias=False` for the `Conv2d` layer.

   Remark: It is not required for you to train the model for the purpose of this problem.

5 Image overfitting (13pts)

In this problem, you will be overfitting a neural network to an image. In other words, for a grayscale image M you will train a continuous function $f_\theta(x, y) : \mathbb{R}^2 \to \mathbb{R}$ such that $f(x, y)$ approximates $M_{x,y}$ for all image coordinate (x, y). Your dataset will consist of all coordinate-pixel pairs for a *single image*, and your task will be to experiment with different activation functions. You can run the code in this problem with
`python siren.py -b batch_size -e total_epochs -a [sin, tanh, relu]`

1. Finish the implementation of the MyDataset class - namely implement the __get_item__(self, idx) function which returns the data element corresponding to the given index.
2. Implement the SingleLayer class (used by class Siren), a simple Linear layer followed by an activation. You should initialize the weights of the first layer to be uniform $[\frac{-1}{input\_features}, \frac{1}{input\_features}]$ and every other layer to be uniform$[-\frac{\sqrt{6/hidden_features}}{\omega}, \frac{\sqrt{6/hidden_features}}{\omega}]$. $\omega = 30$ for sin activation and $\omega = 1$ for other activations. In the forward(self, input) function you should multiply the layer output by ω before applying the activation.
3. Set your learning rate appropriately in train()
4. Implement the main loop of the train() function, which trains the Siren model with batches from the dataloader.
5. (9pts) Run experiments with ReLU, Tanh, and Sin activations. For each activation report one successful experiment including:
   (a) total number of epochs
   (b) learning rate
   (c) batch size
   (d) plot a loss curve
   (e) plot model output images, gradients, and laplacians (generated by model_results())

   We consider an experiment "successful" if your loss curve ends relatively flat and the resulting image is mostly clear. Your results should be better for sin than tanh and relu.

   Grading:

   - -1 point if ReLU 3rd plot (laplacian) is not empty
   - -1 point (for each activation) if loss above 0.03 in plot

- -1 point (for each activation) if blurry image (should be at least as clear as ours)
- -1 point (for each activation) if very noisy loss curve

6. (1pts) Provide a mathematical explanation for why the Laplacian looks as it does for ReLU activation.

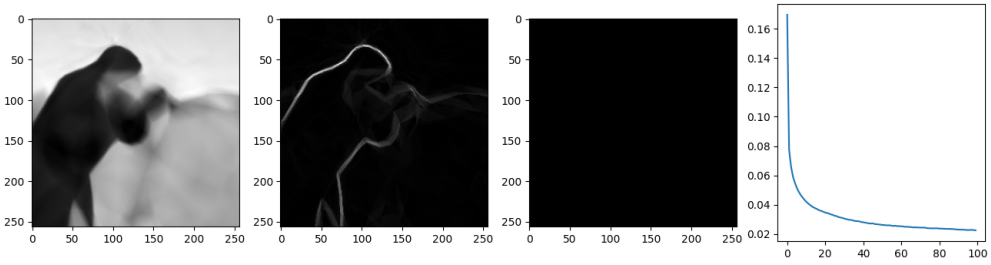
   Answer: enough to say that second derivative of relu is 0 and the laplacian is based on the second derivatives

7. (1pts) When your loss plateaus, what is something you can do to the learning rate to keep improving your model? What is this strategy called?

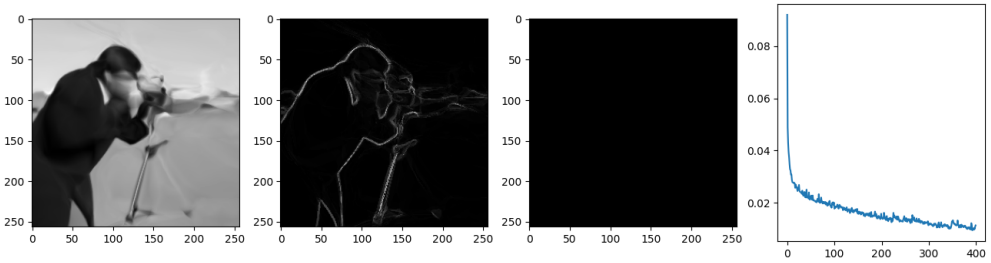
   Answer: you can reduce the learning rate after some number of epochs, called learning rate decay

8. (2pts) Describe what happens if you do not manually set the initial weights? You should report the results for an experiment in this setting for one of the activation functions above.

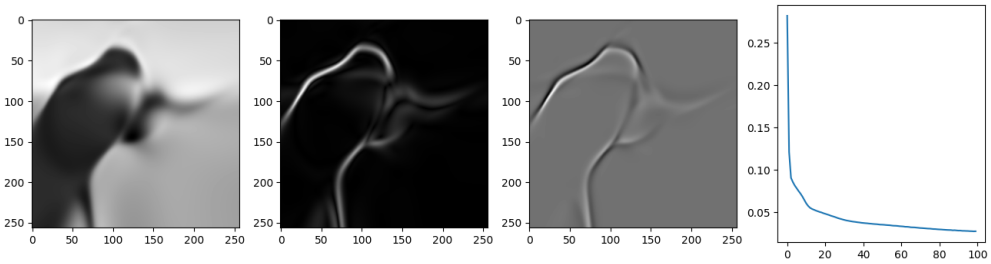
   Answer: see results below - key point is that the loss starts higher and for sin activation it doesn't train at all. Note they only needed to report for one activation.



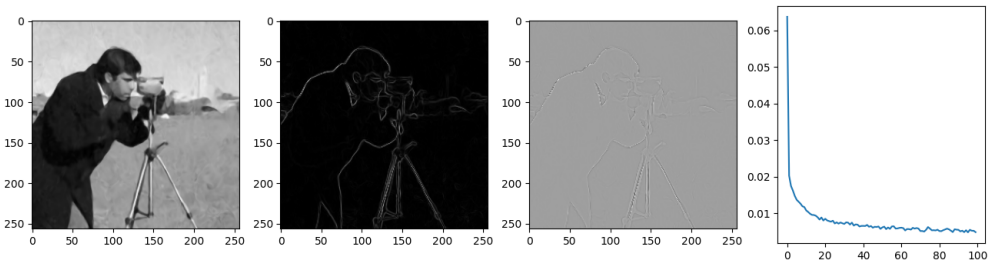
ReLU: lr 0.0001, batch 256



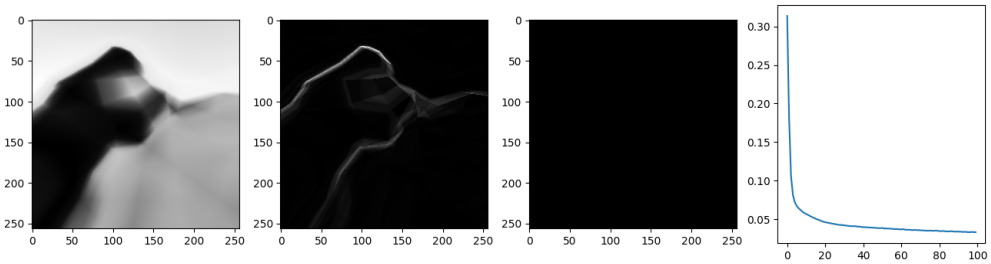
ReLU: lr 0.001, batch 512



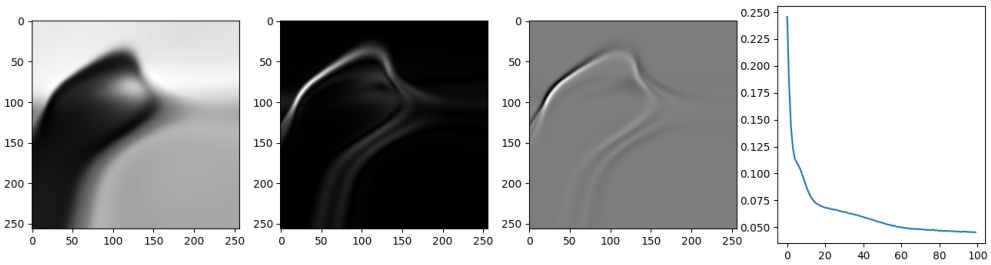
Tanh: lr 0.0001, batch 256



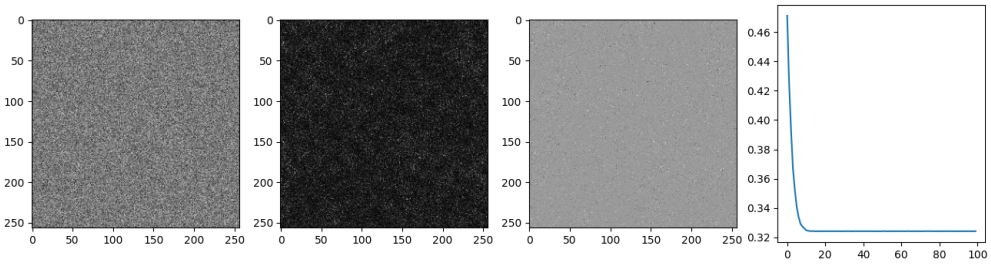
Sin: lr 0.0001, batch 256



ReLU no init: lr 0.0001, batch 256



Tanh no init: lr 0.0001, batch 256



Sin no init: lr 0.0001, batch 256