EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Discussion 6

1. Residual Connection

Neural networks learn by optimization with gradients. The deeper a neural network grows, the more likely it will suffer from either vanishing gradients or exploding gradients problem since downstream layers (ie. earlier layers) will be optimized with either exponentially small or large gradients due to chain rule. Exploding gradients can be resolved using **gradient clipping**, whereas tackling vanishing gradients is more tricky. Here we study how skip connections can help.

(a) One common way to tackle vanishing gradients is by adding residual connections. Fig 1 shows a simplified example of a residual block. Assuming that the weights are affine layer with zero biases, **what's the expression of X_1 in terms of W_1, W_2 and X_0?** $W_i \in \mathbb{R}^{n\times n},\ X_i \in \mathbb{R}^n$.

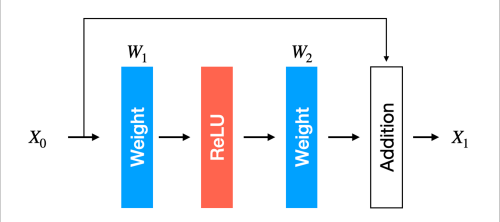


Figure 1: Diagram for Residual Block

Solution:

$$X_1 = W_2\text{ReLU}(W_1X_0) + X_0$$

(b) **Compute $\frac{\partial X_1}{\partial X_0}$.** Based on what you see numerically, **why does residual connection preserves gradient norms better**?

Solution: The Jacobian matrix $\frac{\partial X_1}{\partial X_0}$ is computed using the linearity of differentiation and the chain rule:

$$\frac{\partial X_1}{\partial X_0} = \frac{\partial\left(W_2\text{ReLU}(W_1X_0)\right)}{\partial X_0} + \frac{\partial X_0}{\partial X_0}$$

Let $Z = W_1X_0$. The term $\frac{\partial \text{ReLU}(Z)}{\partial Z}$ is a diagonal matrix, D, where the diagonal elements $D_{i,i}$ are the derivatives of the ReLU function, which are 1 if $Z_i > 0$ and 0 otherwise.

Applying the chain rule:

$$\frac{\partial\left(W_2\text{ReLU}(W_1X_0)\right)}{\partial X_0} = W_2DW_1$$

The final expression for the Jacobian is:

$$\frac{\partial X_1}{\partial X_0} = W_2DW_1 + I$$

where I is the identity matrix.

Numerically, the residual connection preserves gradient norms better because the Jacobian contains the identity I. The presence of I ensures that even if the function term W_2DW_1 is close to the zero matrix, the overall Jacobian is guaranteed to have a magnitude of at least I.

(c) In practice, when transitioning between different layers or blocks within a ResNet architecture, the dimensions of the tensors may change. One common scenario is when the number of channels is doubled and the spatial dimensions are halved, often through a strided convolution. In such cases, the identity shortcut connection (the skip connection that bypasses one or more layers) also needs to change dimensions to match. **Describe how we should modify the expression for X_1 in a residual block to incorporate a skip connection from X_0, when X_0 does not match the shape of X_1?**.

Solution:

One common method to handle dimension change in the identity shortcut is to use a linear projection. This can be achieved by applying a convolutional layer with a kernel size of 1 along the shortcut, and potentially using a stride of 2 if the spatial dimensions are being halved. The convolutional layer will learn a linear projection from the input dimensions to the desired output dimensions, allowing the identity shortcut to be added to the output of the residual block.

The expression for X_1 in terms of W_1, W_2, X_0, and the projection matrix P (representing the convolutional layer along the shortcut) would then be:

$$X_1 = W_2\text{ReLU}(W_1X_0) + PX_0$$

This way, the residual connection still holds, and the network can learn a suitable projection to ensure that the shortcut is meaningful even when the dimensions change. For more details, see the original ResNet paper[1].

2. Understanding Upsampling

Many CNN architectures rely on upsampling (particularly for tasks such as image segmentation). We will show how upsampling via linear interpolation can be done using convolutions and transpose convolutions. Let $\mathbf{x} = [x_1, x_2, x_3]^T$ be a 1D discrete signal. Consider an upsampled signal $\mathbf{y} = [y_1, y_2, y_3, y_4, y_5]^T$ that is a piecewise linear interpolation given by:

$$\mathbf{y} = \left[x_1, \frac{x_1 + x_2}{2}, x_2, \frac{x_2 + x_3}{2}, x_3\right]^T$$

(a) First, define an upsampled signal $\tilde{\mathbf{x}} = [x_1, 0, x_2, 0, x_3]^T$ by inserting zeros between samples. **What is the convolutional kernel $\mathbf{k}$ such that $\mathbf{y}$ is the result of a convolution of $\tilde{\mathbf{x}}$ with $\mathbf{k}$ using stride 1 and zero-padding 1?**

Solution: Note that to make sizes work, we need kernel of size $k = 3$. Set $\mathbf{k} = [k_1, k_2, k_3]^T$ for now. We can solve by considering elements of $\mathbf{y}$:

$$k_2x_1 = x_1$$

[1] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

$$k_1 x_1 + k_3 x_3 = \frac{x_1 + x_2}{2}$$
$$k_2 x_2 = x_2$$
$$k_1 x_2 + k_3 x_3 = \frac{x_2 + x_3}{2}$$
$$k_2 x_3 = x_3$$

From the above equations, we see that $\mathbf{k} = \left[\frac{1}{2}, 1, \frac{1}{2}\right]^T$ recovers the output $\mathbf{y}$.

(b) Now, rather than in two steps, we want to derive $\mathbf{y}$ directly from $\mathbf{x}$ in a single transpose convolution. **What is the kernel $\mathbf{k}$ and stride s and padding p such that $\mathbf{y}$ is the result of a transpose convolution of $\mathbf{x}$ with $\mathbf{k}$ using stride s and zero-padding p?**

Solution: The transpose convolution uses the same kernel $\mathbf{k} = \left[\frac{1}{2}, 1, \frac{1}{2}\right]^T$ with stride 2 (to achieve upsampling by a factor of 2) and padding 1.

To verify, we can use the formula for output size of a transpose convolution (using stride s, padding p, kernel size k, and original length n): $n' = s(n-1) + k - 2p$. Substituting in the quantities yields $2(3-1) + 3 - 2(1) = 5$, as desired

3. Regularization and Dropout

This question shows that a form of dropout is equivalent to ridge regularization.

Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 \tag{1}$$

One way of using **dropout** during SGD on the d-dimensional input features $\mathbf{f}_i$ is to keep each feature of each sample at random, with distribution $\sim_{i.i.d}$ $Bernoulli(p)$ (and zeroing it out if not kept), and then performing a SGD step.

It turns out that such dropout makes our learning objective effectively become

$$\mathcal{L}(\check{\mathbf{w}}) = E_{R \sim Bernoulli(p)}\left[||\mathbf{y} - (R \odot X)\check{\mathbf{w}}||_2^2\right] \tag{2}$$

where $\odot$ is the element-wise product and the random binary matrix $R \in \{0,1\}^{n \times d}$ is such that $R_{i,j} \sim_{i.i.d}$ $Bernoulli(p)$. We use $\check{\mathbf{w}}$ to remind you that this is learned by dropout.

Recalling how Tikhonov-regularized (generalized ridge-regression) least-squares problems involve solving:

$$\mathcal{L}(\mathbf{w}) = ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 \tag{3}$$

for some suitable matrix Γ, it turns out we can manipulate (2) to eliminate the expectations and get:

$$\mathcal{L}(\check{\mathbf{w}}) = ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{4}$$

with $\check{\Gamma}$ being a diagonal matrix with $\Gamma_{ii} = \sqrt{\sum_k X_{ki}^2} = \|\mathbf{f}_i\|_2$.

(a) **Suppose we have learned a weight vector $\check{\mathbf{w}}$ using (4) (i.e. with dropout). How do we transform it into some $\mathbf{w}$, such that $\mathbf{w}$ is a solution to the traditionally regularized problem (3)?**

And what would be the Γ matrix for this w? How does the choice of p affect Γ?

(Hint: This is related to how we adjust weights learned using dropout training for using them at inference time. PyTorch by default does this adjustment during training itself, but here, we are doing dropout slightly differently with no adjustments during training.)

The question sounds weird, so here's an imaginary scenario to help make it easier to understand.

Suppose after hours of training on a dataset, we have learned a weight vector $\check{\mathbf{w}}$ using dropout training, and then the boss tells us that we need to actually use Tikhonov-regularized training (but we are allowed to design whatever Γ we choose). How do we find $\mathbf{w}$ without doing training anew? And how do we choose Γ so that the resulting $\mathbf{w}$ actually solves (3)?

Solution:

Choose $\mathbf{w} = p\check{\mathbf{w}}$. I.e., we need to multiply $\check{\mathbf{w}}$ by p.

The idea here is to absorb the p term into $\mathbf{w}$, and then choose Γ accordingly.

By choosing $\mathbf{w} = p\check{\mathbf{w}}$ (and thus $\check{\mathbf{w}} = \frac{1}{p}\mathbf{w}$), we would have

$$\mathcal{L}(w) = ||y - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 \tag{5}$$

$$= ||y - X\mathbf{w}||_2^2 + p(1-p)||\check{\Gamma}\frac{\mathbf{w}}{p}||_2^2 \tag{6}$$

$$= ||y - X\mathbf{w}||_2^2 + ||\sqrt{\frac{(1-p)}{p}}\check{\Gamma}\mathbf{w}||_2^2 \tag{7}$$

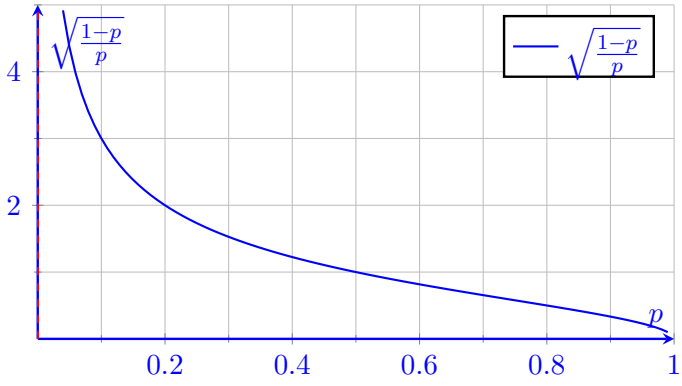
$$= ||y - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 \tag{8}$$

Where we chose $\Gamma = \sqrt{\frac{1-p}{p}}\check{\Gamma}$

The factor depends non-linearly on p. Particularly, as p approaches 1, the factor approaches zero, and as p approaches 0, the factor scales as $\frac{1}{\sqrt{p}}$. We can understand this dependence by setting a few different values of p:

- $p = 0.9$: $\Gamma = 0.33\check{\Gamma}$
- $p = 0.5$: $\Gamma = 1\check{\Gamma}$
- $p = 0.1$: $\Gamma = 3.16\check{\Gamma}$
- $p = 0.01$: $\Gamma = 10\check{\Gamma}$
- $p = 0.001$: $\Gamma = 31.6\check{\Gamma}$

Plot of $\sqrt{\frac{1-p}{p}}$



(b) Since Γ in (3) is an invertible matrix, we can define $\widetilde{X}, \tilde{w}$, such that Equation (3) $\mathcal{L}(\widetilde{\mathbf{w}})$ has the same form as classical ridge regression:

$$\mathcal{L}(\widetilde{\mathbf{w}}) = ||\mathbf{y} - \widetilde{X}\widetilde{\mathbf{w}}||_2^2 + \lambda||\widetilde{\mathbf{w}}||_2^2 \tag{9}$$

What are $\widetilde{X}, \tilde{w}$ in terms of the original data matrix X and Γ and w?

Solution: To make the regularization term in (3) look like ridge regression, we'll let $\widetilde{\mathbf{w}} = \lambda^{-1/2}\Gamma\mathbf{w}$, so $\mathbf{w} = \lambda^{1/2}\Gamma^{-1}\widetilde{\mathbf{w}}$. Plugging this into (3) gives us

$$||\mathbf{y} - \lambda^{1/2}X\Gamma^{-1}\tilde{\mathbf{w}}||_2^2 + \lambda||\tilde{\mathbf{w}}||_2^2 \tag{10}$$

From this, we see our modified data matrix should be

$$\tilde{X} = \lambda^{1/2}X\Gamma^{-1}.$$

(c) Using the fact that $\Gamma = \text{diag}(\|\mathbf{f}_1\|, \|\mathbf{f}_2\|, \ldots, \|\mathbf{f}_d\|)$, **what can you say about the norms of the columns of the effective training matrix $\widetilde{X}$? Comment briefly on the similarity between dropout and batch-normalization.**

Solution: Let's say each $\mathbf{f}_j$, $\check{\mathbf{f}}_j$ are the j-th column vector of X and $\tilde{X}$, respectively. Recall that Γ is defined to be a diagonal matrix whose j-th diagonal entry is the norm of the j-th column of X and $\tilde{X} = cX\check{\Gamma}^{-1}$, where c is a rescaling positive constant of that you found in the previous part.

$$\tilde{X} = cX\check{\Gamma}^{-1} = \begin{bmatrix}\check{\mathbf{f}}_1 & \check{\mathbf{f}}_2 & \ldots & \check{\mathbf{f}}_d\end{bmatrix} = c\begin{bmatrix}\mathbf{f}_1 & \mathbf{f}_2 & \ldots & \mathbf{f}_d\end{bmatrix}\begin{bmatrix}\frac{1}{\|\mathbf{f}_1\|_2} & 0 & \ldots & 0 \\ 0 & \frac{1}{\|\mathbf{f}_2\|_2} & \ldots & 0 \\ & & \ddots & \\ 0 & 0 & \ldots & \frac{1}{\|\mathbf{f}_d\|_2}\end{bmatrix}$$

$$= \begin{bmatrix}c\frac{\mathbf{f}_1}{\|\mathbf{f}_1\|_2} & c\frac{\mathbf{f}_2}{\|\mathbf{f}_2\|_2} & \ldots & c\frac{\mathbf{f}_d}{\|\mathbf{f}_d\|_2}\end{bmatrix}$$

Therefore, $\tilde{\mathbf{f}}_j = c\frac{\mathbf{f}_j}{\|\mathbf{f}_j\|_2}$ and $\left\|\tilde{\mathbf{f}}_j\right\|_2 = c$. In other words, dropout makes each column vector of the matrix X constant-norm of c in effect, where $c = \lambda^{1/2}\sqrt{\frac{p}{1-p}}$ is the rescaling constant you found from Parts (a) and (b).

Note that this is similar to batch-normalization's standardization and scaling operations, which makes the column vectors have the same variance.

Contributors:

- Kevin Li .
- Naman Jain.
- Joey Hong.

- Saagar Sanghavi.
- Jerome Quenum.
- Anant Sahai.

