EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Discussion 7

1. Residual Connection

Neural networks learn by optimization driven by gradients. In deep networks, gradients can vanish or explode as they propagate toward earlier layers through products of layer Jacobians. **Gradient clipping** can limit exploding gradients, whereas tackling vanishing gradients requires different measures. Here we study how skip/residual connections can help.

(a) One common way to help address vanishing gradients is by adding residual connections. Figure 1 shows a simplified residual block. Assume that the two weight layers are linear maps with matrices W_1 and W_2 and no biases. **What is the expression for X_1 in terms of W_1, W_2 and X_0?** $W_1, W_2 \in \mathbb{R}^{n\times n}$ and $X_0, X_1 \in \mathbb{R}^n$.

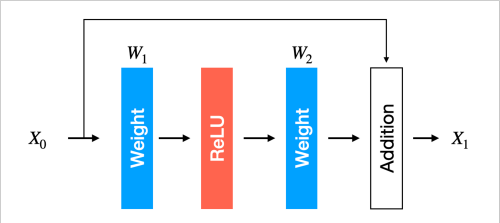


Figure 1: Diagram for Residual Block

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, **why can residual connections help preserve gradient norms**?

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 $Z = W_1X_0$. Away from $Z_i = 0$, the derivative $\frac{\partial\text{ReLU}(Z)}{\partial Z}$ is a diagonal matrix D, with $D_{ii} = 1$ for $Z_i > 0$ and $D_{ii} = 0$ for $Z_i < 0$. At $Z_i = 0$, ReLU is not differentiable; we use the usual backpropagation convention $D_{ii} = 0$.

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:

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

 All Rights Reserved. This may not be publicly shared without explicit permission.

where I is the identity matrix.

The identity term provides a direct path for gradients. When the residual-branch Jacobian W_2DW_1 is small in operator norm, the total Jacobian is close to I, so this block approximately preserves gradient norms. This is not a guarantee for arbitrary weights: for example, if $W_2DW_1 = -I$, the total Jacobian is zero.

(c) In practice, a residual block may change the shape of its input. One common scenario doubles the number of channels and halves the spatial dimensions using a strided convolution. For this part, allow the main branch to change dimensions as well. The shortcut must then match the main branch's output shape. **How should we modify the expression for X_1 to include a skip connection from X_0 when their shapes differ?**

Solution:

One common method is to replace the identity shortcut with a learned linear projection. For a 2D feature map, a 1×1 convolution can change the number of channels, with stride 2 if the spatial dimensions are being halved. Its output shape must match the main branch's output shape.

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:

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

For flattened vectors $X_0 \in \mathbb{R}^n$ and $X_1 \in \mathbb{R}^m$, compatible dimensions are $W_1 \in \mathbb{R}^{h\times n}$, $W_2 \in \mathbb{R}^{m\times h}$, and $P \in \mathbb{R}^{m\times n}$.

Both branches now have the same output shape, so their sum is well-defined. For more details, see the original ResNet paper[1].

2. Regularization and Dropout

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

Let $X \in \mathbb{R}^{n\times d}$ be the data matrix, $\mathbf{y} \in \mathbb{R}^n$ the targets, and $\mathbf{w} \in \mathbb{R}^d$ the weights. Write $\mathbf{f}_j = X_{:,j} \in \mathbb{R}^n$ for the j-th feature column across all samples.

Recall that linear regression optimizes the following learning objective:

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

One way of using **dropout** during SGD is to independently retain each input feature of each sample with probability $p \in (0, 1)$, setting it to zero otherwise, and then perform an SGD step. Here p is the *keep probability*.

The expected dropout training objective is

$$\mathcal{L}_{\text{drop}}(\check{\mathbf{w}}) = \mathbb{E}_R\Big[||\mathbf{y} - (R \odot X)\check{\mathbf{w}}||_2^2\Big] \tag{2}$$

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

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

 All Rights Reserved. This may not be publicly shared without explicit permission.

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

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

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

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

where $\check{\Gamma}$ is diagonal with $\check{\Gamma}_{jj} = \sqrt{\sum_{i=1}^n X_{ij}^2} = \|\mathbf{f}_j\|_2$.

(a) **Suppose $\check{\mathbf{w}}$ minimizes the dropout objective (4). 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 $\mathbf{w}$? How does the choice of p affect Γ?
(Hint: Choose $\mathbf{w} = p\check{\mathbf{w}}$. This is related to scaling predictions at inference time after dropout training. Here training uses unscaled masks. PyTorch instead scales retained activations by $1/p$ during training; its `Dropout` argument is the drop probability $1-p$.)
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)?

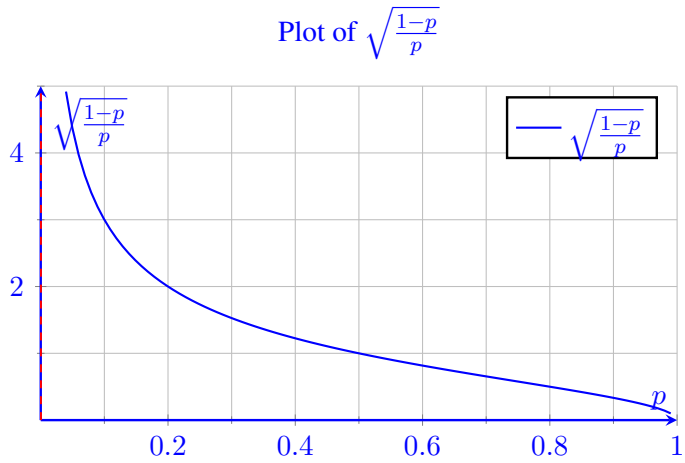
Solution:
The hinted substitution absorbs the p term into $\mathbf{w}$. We then choose Γ accordingly. Since $\check{\mathbf{w}} = \mathbf{w}/p$, we have

$$\begin{aligned}
\mathcal{L}_{\text{drop}}(\mathbf{w}/p) &= ||\mathbf{y} - pX\check{\mathbf{w}}||_2^2 + p(1-p)||\check{\Gamma}\check{\mathbf{w}}||_2^2 && (5)\\
&= ||\mathbf{y} - X\mathbf{w}||_2^2 + p(1-p)||\check{\Gamma}\frac{\mathbf{w}}{p}||_2^2 && (6)\\
&= ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\sqrt{\frac{1-p}{p}}\check{\Gamma}\mathbf{w}||_2^2 && (7)\\
&= ||\mathbf{y} - X\mathbf{w}||_2^2 + ||\Gamma\mathbf{w}||_2^2 = \mathcal{L}_{\text{reg}}(\mathbf{w}). && (8)
\end{aligned}$$

Here $\Gamma = \sqrt{\frac{1-p}{p}}\check{\Gamma}$. Since $p > 0$, the map $\mathbf{w} = p\check{\mathbf{w}}$ is invertible, so it maps minimizers of the dropout objective to minimizers of the regularized objective.
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:

- $p = 0.9$: $\Gamma \approx 0.333\check{\Gamma}$
- $p = 0.5$: $\Gamma = 1\check{\Gamma}$
- $p = 0.1$: $\Gamma = 3\check{\Gamma}$
- $p = 0.01$: $\Gamma \approx 9.95\check{\Gamma}$
- $p = 0.001$: $\Gamma \approx 31.61\check{\Gamma}$

 © Faculty teaching EECS 182, Fall 2026. All Rights Reserved. This may not be publicly shared without explicit permission.



(b) Assume all feature columns are nonzero, so the Γ from part (a) is invertible, and choose $\lambda > 0$. We can reparameterize (3) using $\widetilde{X}$ and $\widetilde{\mathbf{w}}$ to obtain a classical ridge-regression objective:

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

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) Recall the distinction between the feature-norm matrix $\check{\Gamma}$ and the regularization matrix Γ:

$$\check{\Gamma} = \text{diag}(\|\mathbf{f}_1\|_2, \ldots, \|\mathbf{f}_d\|_2), \qquad \Gamma = \sqrt{\frac{1-p}{p}}\,\check{\Gamma}.$$

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: From parts (a) and (b), $\tilde{X} = \sqrt{\lambda}\,X\Gamma^{-1} = \sqrt{\lambda p/(1-p)}\,X\check{\Gamma}^{-1}$. Since $\check{\Gamma}_{jj} = \|\mathbf{f}_j\|_2$, the j-th transformed column satisfies

$$\tilde{\mathbf{f}}_j = \sqrt{\frac{\lambda p}{1-p}}\,\frac{\mathbf{f}_j}{\|\mathbf{f}_j\|_2}, \qquad \|\tilde{\mathbf{f}}_j\|_2 = \sqrt{\frac{\lambda p}{1-p}}.$$

 All Rights Reserved. This may not be publicly shared without explicit permission.

Thus all columns of $\tilde{X}$ have the same norm. This is an equivalent ridge reparameterization; dropout does not literally normalize X.

Equal column norms do not imply equal variances without centering. Batch normalization centers and rescales features using batch statistics (with a numerical stabilizer), then typically applies a learned scale and shift. The resemblance is feature rescaling; the two operations are not equivalent.

3. 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 transposed convolutions. Use dilation 1 and no bias throughout. 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: To obtain the required output length, the kernel size must be $K = 3$. Set $\mathbf{k} = [k_1, k_2, k_3]^T$. Using the usual deep-learning cross-correlation convention, we can solve by considering the entries of $\mathbf{y}$:

$$\begin{aligned} k_2 x_1 &= x_1 \\ k_1 x_1 + k_3 x_2 &= \frac{x_1 + x_2}{2} \\ k_2 x_2 &= x_2 \\ k_1 x_2 + k_3 x_3 &= \frac{x_2 + x_3}{2} \\ k_2 x_3 &= x_3 \end{aligned}$$

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

 All Rights Reserved. This may not be publicly shared without explicit permission.

(b) Now, rather than in two steps, we want to derive $\mathbf{y}$ directly from $\mathbf{x}$ in a single transposed convolution. Assume output padding 0. **What kernel $\mathbf{k}$, stride s, and padding parameter p produce $\mathbf{y}$ from $\mathbf{x}$?**

Solution: The transposed convolution uses the same kernel $\mathbf{k} = \left[\frac{1}{2}, 1, \frac{1}{2}\right]^T$ with stride 2 and padding 1. Each x_i contributes x_i at output position $2i - 1$ and $x_i/2$ at each adjacent position. Contributions outside positions 1 through 5 are cropped. Summing these contributions gives exactly $[x_1, (x_1 + x_2)/2, x_2, (x_2 + x_3)/2, x_3]^T$.

To verify, with dilation 1 and output padding 0, the output length of a transposed convolution (using stride s, padding p, kernel size K, and input length n) is $n' = s(n-1) + K - 2p$. Substituting in the quantities yields $2(3-1) + 3 - 2(1) = 5$, as desired.

Contributors:

- Kevin Li.
- Naman Jain.
- Saagar Sanghavi.
- Jerome Quenum.
- Joey Hong.
- Gireeja Ranade.
- Matteo Guarrera.
- Mert Cemri.
- Sizhe Chen.

 All Rights Reserved. This may not be publicly shared without explicit permission.