CS5370 Deep Learning for Vision – Assignment 3

Submitted by:
Vishal Singh Yadav
CS20MTECH01001

1. Forward step: for forward step, the output of a residual block is achieved by applying ReLU on the output after applying $f()$ that transforms the input and adding the input to block in the output.
   For a residual block l,
   $$H_l = ReLU(f_l(H_{l-1}) + H_{l-1})$$
   where $f()$ is the function that transforms the input.

   $f_l(H_{l-1})$ can be calculated as:
   $RL_a = H_l.Wl_a$
   $RL_aA = \sigma RL_a$
   $RL_b = RL_aA.Wl_b$

   Backward step: for backward step, we need to calculate $\frac{\mathrm{d\ out}}{dW_lH}$. We calculate it till we backpropagate the gradients to first layer.
   For residual block l,
   $$\frac{\mathrm{d\ out}}{dW_lH} = (d\ out)(d\ ReLU()) + W_{l+1}H + (W_{l+1}b)\big(d\sigma(RL_{l+1}a)\big)(W_{l+1}a).d(ReLU(H_{l+1})(H_lA)$$

2. Keeping a stride of 1 and using a 3x3 filter without any padding will mean that the image is reduced in size by 2 after each convolution. Using the equations,
   $$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
   and
   $$H_2 = \frac{H_1 - F + 2P}{S} + 1$$
   we get the new size of our image as $(n-2) * (n-2)$ reduced from $n * n$.

   Hence, for each convolution layer we can find it's corresponding support as $9 * 9 \rightarrow 7 * 7 \rightarrow 5 * 5 \rightarrow 3 * 3 \rightarrow 1 * 1$.
   Thus, this network's support at the fourth non-image layer is 9*9 = 81 pixels.

3. Adding more hidden units will decrease bias and increase variance.
   This happens as a larger model tends to have less bias and more variance. By adding more hidden units in the model, we make it larger, decreasing bias and increasing variance.

4. $\sigma(a) = \frac{1}{1+exp(-a)}$
   $tanh(a) = \frac{e^a - e^{(-a)}}{e^a + e^{(-a)}}$

Let $\sigma(a)$ be denoted by $f(x)$. Taking derivative, $f'(x) = f(x)\big(1 - f(x)\big)$.
Similarly, for $tanh(a)$ denoted by $f(x)$, $f'(x) = 1 - f(x)^2$.

For the sigmoid function, the range of outputs is 0 to 1, often used as probabilities. $tanh$ is just rescaling the sigmoid function so that the outputs range from -1 to 1 and add horizontal scaling.

As, $tanh(a) = 2\sigma(2a) - 1$, it is evident that tanh is a linear transformation of $\sigma(a)$. So, by replacing $\sigma(a)$ with $tanh(a)$, the neural network parameters will only differ by a linear transformation, making it possible for the final layer to be changed but still able to calculate the same function.

5. Quadratic error:

$$E(w) = E(w^*) + \frac{1}{2}(w - w^*)^T H(w - w^*) \tag{1}$$

Hessian matrix H, evaluated at w^* has eigenvalue given by

$$Hu_i = \lambda_i u_i \tag{2}$$

Here the eigenvectors u_i, forms a complete orthogonal set so

$$u_i^T u_j = \delta_{ij} \tag{3}$$

We can expand $w - w^*$, as a linear combination of the eigenvectors in the form

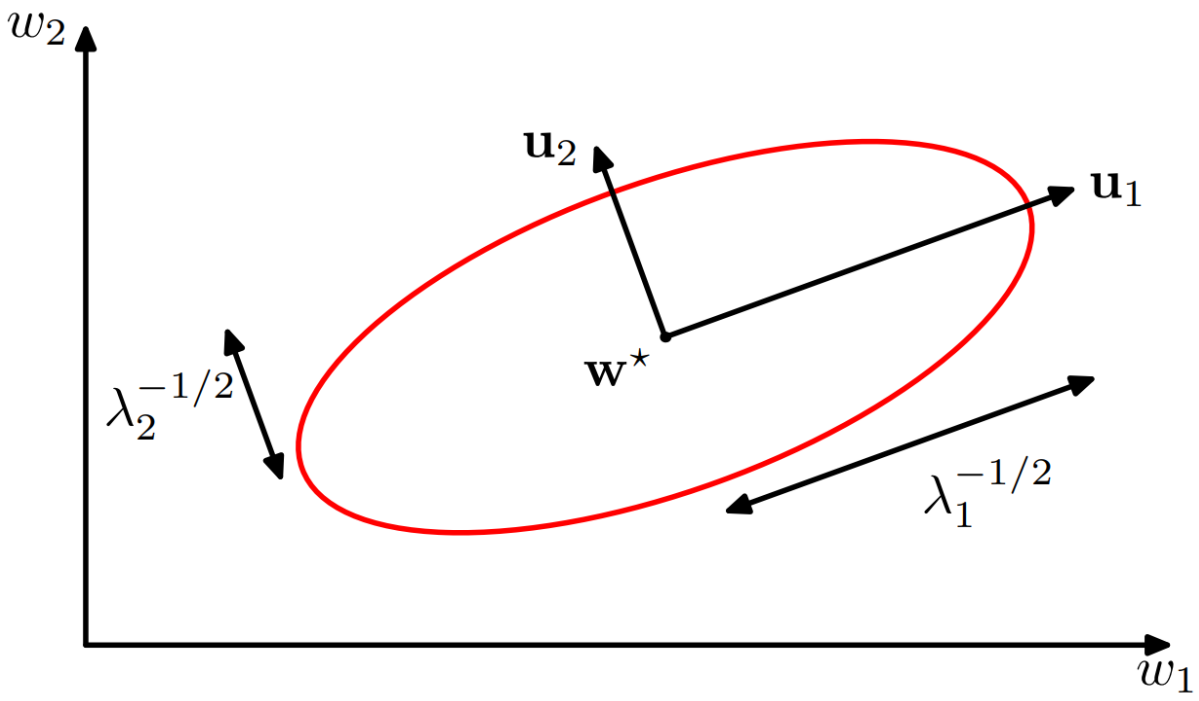
$$w - w^* = \sum_i \alpha_i u_i \tag{4}$$

This can be regarded as a transformation of the coordinate system in which the origin is translated to the point $w^*$and the axes are rotated to align with the eigenvectors.
Substituting (4) in (1) and using (2) and (3), we can rewrite the error function as

$$E(w) = E(w^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2 \tag{5}$$

In the neighborhood of the minimum w^*, the error function can be found out using (5). This will result in contours of constant errors, which are ellipses and whose axes are aligned with the eigenvectors u_i of the Hessian matrix, with lengths that are inversely proportional to the square roots of the corresponding eigenvectors λ_i.

Graphically the constant error around w^* can be shown as



6. We can infer that we have a pre-trained model with a similar domain and type of data from the given statement. Additionally, we have very few instances of new data.

   We will use transfer learning to reuse the pre-trained model for our solution. Since both solutions' datasets are similar, their specialized features (features that the model learns in hidden layers) will remain the same.
   We can keep the feature extraction (i.e., source task generic layers and source task special layers) the same as the previous model and use the limited dataset to train the last classification layer. The classification layer parameters are randomly initialized and trained, keeping all other parts of the network frozen. This way, we can use the pre-trained model for our solution.

   Exact steps:
   - Take previously trained model.
   - Train the classification using new data while ensuring all the model layers except the last are not changed. All changes are to be made only in the last layer.
   - Use the newly trained model.