Chessbot

Backpropagation

Hegyi Erik

February 17, 2026

1 Recap

We have defined two different loss functions:

1. **Monte Carlo Loss**

$$\Lambda_i^{MC}(y_i, \hat{y}) = (y_i - \hat{y})^2$$

2. **Bootstrapping Loss**

$$\Lambda_i^{BS}(y_i, y_{i+1}, \hat{y}) = \begin{cases} (y_i - y_{i+1})^2;\ i+1 < s \\ (y_i - \hat{y})^2;\ i+1 = s \end{cases}$$

And we have defined our actual, used loss function as:

$$\Lambda_i(y_i, y_{i+1}, \hat{y}) = \alpha\Lambda_i^{BS}(y_i, y_{i+1}, \hat{y}) + (1-\alpha)\Lambda_i^{MC}(y_i, \hat{y})$$

Where α is a hyperparameter. We have defined the layers as:

$$\mathbf{Y}_i^1 = \mathrm{ReLU}(\mathbf{W}_i^1 \cdot \mathbf{X}_i^1)_{\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_D} + b_i^1) + \eta_i \mathbf{X}_i^1$$

The layer converting the tensor into a vector as:

$$\mathbf{Y}_1^2 = \mathrm{GAP}(\mathbf{X}_1^2) = \frac{1}{\Pi(\mathbf{N})} \begin{pmatrix} \Sigma(\mathbf{X}_{11}^2) \\ \Sigma(\mathbf{X}_{12}^2) \\ \vdots \\ \Sigma(\mathbf{X}_{1C_{l_1}}^2) \end{pmatrix}$$

And the final prediction as:

$$y = \mathbf{Y}_2^2 = \tanh(\mathbf{W}_1^2 \cdot \mathbf{X}_2^2 + \mathbf{B}_2^2)$$

2 Optimizer Setup

For the optimizer, we are going to use **Adam** (Adaptive Moment Estimation) for the optimizer. However, this relies on **SGD** (Stochastic Gradient Descent) as a base, meaning that at first, we are going to define that, and then advance on to Adam.

3 Stochastic Gradient Descent

During the forward pass, for each layer i, we are going to store the input $\mathbf{X}_i$, the output pre-activation ($\mathbf{Z}_i$) and the final output ($\mathbf{Y}_i$).

At the last layer, when we get our final output, we are going to compute the loss using our loss function. Then, we are going to compute the gradient of the loss with respect to the output:

$$\nabla_y \Lambda = \alpha \nabla_y \Lambda^{BS} + (1-\alpha)\nabla_y \Lambda^{MC}$$

Luckily for us, computing the gradients of the bootstrapping and Monte Carlo losses is easy. For bootstrapping:

$$\nabla_y \Lambda^{BS} = 2(y - y_{i+1})$$

And for the Monte Carlo loss:

$$\nabla_y \Lambda^{MC} = 2(y - \hat{y})$$

And if we are on the last step of the game, then

$$\Lambda^{BS} = \Lambda^{MC} \implies \nabla_y \Lambda^{BS} = \nabla_y \Lambda^{MC}$$

Thus, the gradient of the loss function with respect to the output generally is:

$$\nabla_y \Lambda = 2\alpha(y - y_{i+1}) + 2(1-\alpha)(y - \hat{y})$$

Now, let us set $y = \tanh(u)$, where $u = wx + b$. Differentiating the loss function with respect to u, we get that:

$$\nabla_u \Lambda = \nabla_y \Lambda \cdot \frac{dy}{du} = \nabla_y \Lambda \cdot (1 - \tanh^2(u)) = \nabla_y \Lambda (1 - y^2)$$

Now, we need to figure out how much w and b affected the final result. For this, we are going to apply the chain rule once again:

$$\nabla_w \Lambda = \nabla_u \Lambda \cdot \frac{\partial u}{\partial w} = \nabla_u \Lambda x$$

$$\frac{\partial \Lambda}{\partial b} = \nabla_u \Lambda \cdot \frac{\partial u}{\partial b} = \nabla_u \Lambda$$

Now that we have done this, we are going to **backpropagate** by first calculating how much the incoming input affected the loss ($\nabla_x \Lambda$), and then using that as the incoming gradient for the previous layer. Using the chain rule once again, we get that:

$$\nabla_x \Lambda = \nabla_u \Lambda w^T$$

The transpose appears, because in the forward pass, we compute the tensor multiplication between w and x, which means that each component u_i is a weighted sum of the inputs x_j. When differentiating with respect to x, we must sum over all outputs u_i that depend on each input x_j. This reverses the index order of the weight matrix. Let L be the total number of layers, and l be the current layer. Let σ_l be the activation function of layer l. Let u_l be the value before the activation function, and y_l be the value after activation function, such that for the weight w_l and bias b_l of the layer, we can state:

$$u_l = w_l \cdot y_{l-1} + b_l \wedge y_l = \sigma_l(u_l) + \eta y_{l-1}$$

Now, let's define a shorthand notation:

$$\delta^l = \nabla_u^l \Lambda$$

Now, we can derive the gradients for the weight and bias for any arbitrary layer. We know by the chain rule that:

$$\nabla_w \Lambda = \delta^l \cdot \frac{\partial u}{\partial w} = \delta^l y_{l-1}^T$$

$$\nabla_b \Lambda = \delta^l \cdot \frac{\partial u}{\partial b} = \delta^l$$

The transpose appears here, because each weight connects one component of y_{l-1} to on ecomponent of u_l. The gradient thereform forms an outer product between the upstream gradient δ^l and the input vector y_{l-1}. Now, we need to figure out $\nabla_y^{l-1}\Lambda$. The loss depends on y_{l-1} in two ways:

1. Through $u_l = w_l \cdot y_{l-1} + b_l$
2. Through the residual term: ηy_{l-1}

Since we know y_l and u_l, we can calculate $\nabla_y^{l-1}\Lambda$:

$$\nabla_y^{l-1}\Lambda = \nabla_u^l \Lambda \frac{\partial u_l}{\partial y_{l-1}} + \nabla_y^l \Lambda \frac{\partial}{\partial y_{l-1}}(\eta y_{l-1}) = \delta^l w_l^T + \eta \nabla_y^l \Lambda$$

And now we have everything we need to derive the weight and bias gradients. First of all, we need to derive δ^l:

$$\delta^l = \nabla_y^l \Lambda \odot \left[\frac{d\sigma}{du_l}\right]_{u_l}$$

But because

$$\frac{d\sigma}{du_l} = \begin{cases} 1 & u_l > 0 \\ 0 & u_l \leq 0 \end{cases}$$

We can say that:

$$\delta^l = \begin{cases} \nabla_y^l \Lambda & u_l > 0 \\ 0 & u_l \leq 0 \end{cases}$$

And, we have already defined what the weight and bias gradients are going to be:

$$\nabla_w^l \Lambda = \delta^l \cdot \frac{\partial u_l}{\partial w_l} = \delta^l y_{l-1}$$

$$\frac{\partial \Lambda}{\partial b_l} = \delta^l \cdot \frac{\partial u_l}{\partial b_l} = \delta^l$$

Now, there is a special case, where the activation function is not ReLU, but GAP. Here, σ changes, but the logic is the same, thus, if we figure out the value of $\frac{d\sigma}{du_l}$, we can apply the same thought process. Let us remember what the GAP function looked like:

Backpropagation Through Global Average Pooling (GAP)

In the case where the activation function is Global Average Pooling (GAP), the logic of backpropagation remains the same as before: we must compute the derivative of the activation function and apply the chain rule. However, unlike ReLU, GAP is not an elementwise operation.

Let u_l be the input tensor to the GAP layer. Suppose each channel $u_l^{(i)}$ contains M elements. The GAP operation computes the average of each channel:

$$y_l^{(i)} = \sigma(u_l^{(i)}) = \frac{1}{M} \sum_{k=1}^{M} u_{l,k}^{(i)}.$$

Thus, each output component $y_l^{(i)}$ is the mean of the corresponding input channel.

3.1 Derivative of GAP

We now compute the partial derivative of $y_l^{(i)}$ with respect to an arbitrary input element $u_{l,m}^{(j)}$.

There are two cases:

1. If $i \neq j$, then $y_l^{(i)}$ does not depend on $u_{l,m}^{(j)}$, so

$$\frac{\partial y_l^{(i)}}{\partial u_{l,m}^{(j)}} = 0.$$

2. If $i = j$, then

$$y_l^{(i)} = \frac{1}{M} \sum_{k=1}^{M} u_{l,k}^{(i)}.$$

Differentiating with respect to $u_{l,m}^{(i)}$ gives

$$\frac{\partial y_l^{(i)}}{\partial u_{l,m}^{(i)}} = \frac{1}{M}.$$

Therefore, every element in a channel contributes equally to its output average.

3.2 Backpropagation Through GAP

Using the chain rule, we compute the gradient of the loss Λ with respect to the input tensor:

$$\frac{\partial \Lambda}{\partial u_{l,m}^{(i)}} = \frac{\partial \Lambda}{\partial y_l^{(i)}} \cdot \frac{\partial y_l^{(i)}}{\partial u_{l,m}^{(i)}}.$$

Substituting the derivative computed above:

$$\frac{\partial \Lambda}{\partial u_{l,m}^{(i)}} = \frac{1}{M} \frac{\partial \Lambda}{\partial y_l^{(i)}}.$$

3.3 Final Result

The gradient of the loss with respect to each input element of a channel is equal to the upstream gradient divided equally among all M elements:

$$\boxed{\delta_{l,m}^{(i)} = \frac{1}{M} \nabla_{y_l^{(i)}} \Lambda}$$

Thus, during backpropagation, Global Average Pooling distributes the gradient evenly across all spatial positions of each channel.

4 Adaptive Moment Estimation

Adam is a bit more complicated than SGD. We are going to define 3 parameters:

1. β_1
2. β_2
3. λ

And we are going to set $\epsilon = 10^{-8}$.

As a first step, we need the gradients once again. In the case of ReLU, the weight gradient at layer l is:

$$\nabla_w^l \Lambda = \delta^l y_{l-1}$$

And the bias gradient is:

$$\frac{\partial \Lambda}{\partial b} = \delta^l$$

Where $\delta^l = \nabla_y \Lambda (1 - y_l^2) = [2\alpha(y, y_{i+1}) + 2(1-\alpha)(y - \hat{y})](1 - y_l^2)$.
Now, we are going to calculate two values. For the sake of convenience, let O represent the parameter we are updating, which can either be the bias or the weight:

$$p = \beta_1 \cdot p + (1 - \beta_1) \nabla_O^l \Lambda$$
$$q = \beta_2 \cdot q + (1 - \beta_2)(\nabla_O^l \Lambda)^2$$

Now, let us account for bias:

$$\hat{p} = \frac{p}{1 - \beta_1}$$

$$\hat{q} = \frac{q}{1 - \beta_2}$$

And now, let us update the parameter:

$$O = O - \lambda \frac{\hat{p}}{\sqrt{\hat{q}} + \epsilon}$$