

Batched backprop: dense layers on a minibatch

Ten problems on backprop with a minibatch: the dense layer $XW + \mathbf{1}b^\top$ and its gradients $X^\top dY$, $dY^\top \mathbf{1}$ and $dY W^\top$, why a sum over the batch becomes a matrix product, the $1/N$ of a mean loss, activations and masked softmax cross-entropy on a batch, micro-batch accumulation and sequence batches, with worked solutions and the mistakes that transpose, double-count or drop a factor.

BEFORE YOU START

Every earlier page did its derivations one example at a time and then stacked the results. Real training runs a whole minibatch through each layer as one matrix, and the backward pass is written the same way. These ten problems take a dense layer in the layout most array code uses, $XW + \mathbf{1}b^\top$, derive its three gradients from index form, show where the sum over the batch hides inside a matrix product, and then follow the batch dimension through a mean loss, an activation, a masked loss, gradient accumulation and sequences. The five mistakes at the end are the ones that pass a quick look: a weight gradient in the other layout, a $1/N$ applied twice, W where $W^\top$ belongs, an elementwise derivative used as a matrix, and micro-batch gradients summed instead of averaged.

- The conventions are those of the previous pages: a gradient has the shape of the variable it is taken with respect to, $\odot$ is the elementwise product, $\mathbf{1}$ is the all-ones vector, and rows are examples.
- The dense layer: input $X \in \mathbb{R}^{N \times d}$ with row n equal to $x^{(n)\top}$, weights $W \in \mathbb{R}^{d \times m}$, bias $b \in \mathbb{R}^m$, and output $Y = XW + \mathbf{1}b^\top \in \mathbb{R}^{N \times m}$, with row n written $y^{(n)\top}$. This is the layout of most array code, which stores W as inputs by outputs; the one-hidden-layer page used the other layout, $XW_1^\top + \mathbf{1}b_1^\top$ with W_1 outputs by inputs.
- L is a scalar loss computed from Y by the layers above. For any array A in the network, dA is the code-style name for $\nabla_A L$: the array of $\partial L / \partial A \dots$, with the shape of A . Row n of dY is written $dy^{(n)\top}$.
- The backward pass of a layer receives dY from above and returns dW , db , and dX for the layer below.
- A mean loss over the batch is $L = \frac{1}{N} \sum_n \ell_n$, where ℓ_n depends only on example n . Targets are written T to keep Y for the layer's output.

Builds on: Jacobians and the chain rule, One-hidden-layer backprop, the whole backward pass

Problems

1. Give the shapes of X , W , b , Y and of dY , dW , db , dX . Show that row n of Y is the layer applied to example n alone. difficulty ..

2. Compute dW from index form. difficulty ..

3. Compute db . difficulty ..

4. Compute dX . Does it involve a sum over the batch? difficulty ..

5. Write dW as a sum of single-example gradients and show it is $X^\top dY$. How many entries does the Jacobian of Y with respect to W have, and how many are nonzero? difficulty ...

6. Mean loss: $L = \frac{1}{2N} \|Y - T\|_F^2$ with targets $T \in \mathbb{R}^{N \times m}$. Compute dY , dW and db . Where does the $1/N$ enter? difficulty ..

7. Let $H = \tanh(Z)$ elementwise, with $Z \in \mathbb{R}^{N \times m}$. Given dH , compute dZ without forming any Jacobian. difficulty ..

8. Masked softmax cross-entropy: logits $Z \in \mathbb{R}^{N \times k}$, row-wise softmax S , one-hot target rows T , and $\mu \in \{0, 1\}^N$ marking the real rows (padding has $\mu_n = 0$; at least one $\mu_n = 1$). With $\ell_n = -\sum_i T_{ni} \log S_{ni}$, the loss is $L = \sum_n \mu_n \ell_n / \sum_n \mu_n$. Compute dZ , and check the case where every $\mu_n = 1$. difficulty ...

9. Gradient accumulation: split N rows into K micro-batches of $B = N/K$ rows, with $L^{(j)}$ the mean loss over micro-batch j . Show that the full-batch mean loss is $L = \frac{1}{K} \sum_j L^{(j)}$ and give $\nabla_W L$. What changes if the sizes differ? difficulty ...

10. Sequences: $X \in \mathbb{R}^{N \times T \times d}$ holds N sequences of T positions, and the layer acts at every position: $y^{(n,t)} = W^\top x^{(n,t)} + b$, with $x^{(n,t)}$ position t of sequence n . Compute dW and db as matrix products. difficulty ...

Answers

1. $X : N \times d, W : d \times m, b : m, Y : N \times m$, and each dA has the shape of A ; row n of Y is $x^{(n)\top}W + b^\top$
2. $dW = X^\top dY$ ($d \times m$)
3. $db = dY^\top \mathbf{1}$ (m)
4. $dX = dY W^\top$ ($N \times d$)
5. $\sum_n x^{(n)} dy^{(n)\top} = X^\top dY$; the Jacobian of Y with respect to W has $Nm \times dm$ entries, only Ndm of them nonzero, and the matrix product never forms it
6. $dY = \frac{1}{N}(Y - T)$; $dW = \frac{1}{N}X^\top(Y - T)$; $db = \frac{1}{N}(Y - T)^\top \mathbf{1}$
7. $dZ = (1 - H \odot H) \odot dH$ ($N \times m$)
8. $dZ = \frac{1}{\mathbf{1}^\top \mu} \text{diag}(\mu)(S - T)$; with every $\mu_n = 1$ it is $\frac{1}{N}(S - T)$
9. $L = \frac{1}{K} \sum_j L^{(j)}$, so $\nabla_W L = \frac{1}{K} \sum_j \nabla_W L^{(j)}$
10. $dW = \tilde{X}^\top d\tilde{Y}$ and $db = d\tilde{Y}^\top \mathbf{1}$, where $\tilde{X}$ ($NT \times d$) and $d\tilde{Y}$ ($NT \times m$) stack every position of every sequence as a row

Worked solutions

Problem 1

Give the shapes of X , W , b , Y and of dY , dW , db , dX . Show that row n of Y is the layer applied to example n alone.

XW is $(N \times d)(d \times m) = N \times m$.	A product needs the inner dimensions to agree, so W has d rows, one per input feature, and its m columns are the output features.
$\mathbf{1}b^\top$ is $(N \times 1)(1 \times m) = N \times m$, and every row of it is $b^\top$.	$(\mathbf{1}b^\top)_{nj} = 1 \cdot b_j$. This is the matrix form of adding b to every row, which array code does by broadcasting.
Row n of XW is $x^{(n)\top}W$.	Row n of a product is row n of the left factor times the right factor.
dA has the shape of A for every A .	It has one entry $\partial L / \partial A_{..}$ per entry of A , so that $A - \eta dA$ is defined.
$X : N \times d$, $W : d \times m$, $b : m$, $Y : N \times m$, and each dA has the shape of A; row n of Y is $x^{(n)\top}W + b^\top$	Row n uses only row n of X and the shared W and b : the batch is N copies of the single-example layer $y = W^\top x + b$, one per row, sharing the parameters.

Problem 2

Compute dW from index form.

$Y_{nj} = \sum_i X_{ni}W_{ij} + b_j$.	Index form of $Y = XW + \mathbf{1}b^\top$.
W_{ij} appears in Y_{nj} for every n , with coefficient X_{ni} , and in no entry outside column j .	Column j of W builds only column j of Y , and every example uses the same W .
$\frac{\partial L}{\partial W_{ij}} = \sum_n \frac{\partial L}{\partial Y_{nj}} X_{ni} = \sum_n X_{ni} dY_{nj}$.	The chain rule sums over every entry of Y that contains W_{ij} ; by step 2 that is one entry per example.

$\sum_n X_{ni} dY_{nj} = (X^\top dY)_{ij}.$	$(X^\top)_{in} = X_{ni}$, and a sum over the shared index n is the definition of the matrix product.
$dW = X^\top dY \ (d \times m)$	$(d \times N)(N \times m) = d \times m$, the shape of W . The batch index is the one summed over, which is why N is the inner dimension of the product.

Problem 3

Compute db .

b_j appears in Y_{nj} for every n , with coefficient 1.	$\mathbf{1}b^\top$ copies $b^\top$ into every row (Problem 1, step 2).
$\partial L / \partial b_j = \sum_n dY_{nj}.$	The chain rule sums over the N entries that contain b_j , each with derivative 1.
$db = dY^\top \mathbf{1} \ (m)$	$(m \times N)(N \times 1) = m \times 1$, the shape of b ; multiplying by $\mathbf{1}$ sums each column of dY , which array code writes as a sum over axis 0. A bias added to every row has its gradient summed over every row.

Problem 4

Compute dX . Does it involve a sum over the batch?

X_{ni} appears in Y_{nj} for every j , with coefficient W_{ij} , and in no other row of Y .	Example n feeds only row n : in a dense layer the examples do not interact.
$\frac{\partial L}{\partial X_{ni}} = \sum_j dY_{nj} W_{ij} = (dY W^\top)_{ni}.$	The chain rule sums over the entries of row n that contain X_{ni} , one per output feature; $(W^\top)_{ji} = W_{ij}$.

$$dX = dY W^\top \quad (N \times d)$$

$(N \times m)(m \times d) = N \times d$, the shape of X . The sum is over output features, not over the batch: each example gets its own input gradient, and row n is $(W dy^{(n)})^\top$, the single-example rule for $y = W^\top x + b$.

Problem 5

Write dW as a sum of single-example gradients and show it is $X^\top dY$. How many entries does the Jacobian of Y with respect to W have, and how many are nonzero?

For one example, $y = W^\top x + b$ has $y_j = \sum_i x_i W_{ij} + b_j$, so its gradient with respect to W has entries $x_i dy_j$: it is $x dy^\top$, $d \times m$.

W_{ij} appears only in y_j , with coefficient x_i ; an outer product $ab^\top$ has entries $a_i b_j$.

$$dW = \sum_n x^{(n)} dy^{(n)\top}.$$

Every example uses the same W , so the loss reaches W through all N rows and the contributions add.

For matrices A and C with rows $a^{(n)\top}$ and $c^{(n)\top}$, $\sum_n a^{(n)} c^{(n)\top} = A^\top C$.

$(A^\top C)_{ij} = \sum_n A_{ni} C_{nj} = \sum_n a_i^{(n)} c_j^{(n)}$, the (i, j) entry of the sum of outer products.

Flatten Y to Nm entries and W to dm ; the Jacobian is $Nm \times dm$, and entry $(Y_{nj}, W_{ij'})$ is X_{ni} if $j' = j$ and 0 otherwise.

Y_{nj} depends only on column j of W (Problem 2, step 2), so for each of the Nm outputs only d of the dm weights give a nonzero entry: $Nm \cdot d$ in total.

$\sum_n x^{(n)} dy^{(n)\top} = X^\top dY$; **the Jacobian of Y with respect to W has $Nm \times dm$ entries, only Ndm of them nonzero, and the matrix product never forms it**

Step 3 with $A = X$, $C = dY$. $X^\top dY$ costs Ndm multiply-adds, the same as the forward product XW , while the full Jacobian has Ndm^2 entries: at $N = 512$ and $d = m = 4096$, about 3.5×10^{13} . The matrix product is the vector-Jacobian product of the Jacobians page, done for the whole batch at once.

Problem 6

Mean loss: $L = \frac{1}{2N} \|Y - T\|_F^2$ with targets $T \in \mathbb{R}^{N \times m}$. Compute dY , dW and db . Where does the $1/N$ enter?

$L = \frac{1}{N} \sum_n \ell_n$ with $\ell_n = \frac{1}{2} \|y^{(n)} - t^{(n)}\|^2$, where $t^{(n)\top}$ is row n of T .

The squared Frobenius norm is the sum of the squared entries, which is the sum of the squared row norms.

$$\partial L / \partial Y_{nj} = \frac{1}{N} (Y_{nj} - T_{nj}).$$

Only ℓ_n contains row n of Y , it carries the factor $\frac{1}{N}$, and $\frac{1}{2}u^2$ differentiates to u .

$$dY = \frac{1}{N} (Y - T), N \times m.$$

Step 2 for every entry.

$$dW = X^\top dY \text{ and } db = dY^\top \mathbf{1}.$$

Problems 2 and 3 hold for any dY ; they never used where it came from.

$$dY = \frac{1}{N} (Y - T); dW = \frac{1}{N} X^\top (Y - T);$$

$$db = \frac{1}{N} (Y - T)^\top \mathbf{1}$$

The $\frac{1}{N}$ enters once, in dY , and every later gradient is linear in dY , so it carries the factor without applying it again. dW is the average of the single-example gradients; with a summed loss it would be N times larger.

Problem 7

Let $H = \tanh(Z)$ elementwise, with $Z \in \mathbb{R}^{N \times m}$. Given dH , compute dZ without forming any Jacobian.

H_{nj} depends only on Z_{nj} .

$\tanh$ is applied to each entry on its own, in every row.

$$\partial L / \partial Z_{nj} = dH_{nj} \tanh'(Z_{nj}).$$

Z_{nj} reaches L only through H_{nj} , so the chain rule has one term.

$$\tanh'(u) = 1 - \tanh^2(u), \text{ so } \tanh'(Z_{nj}) = 1 - H_{nj}^2.$$

Differentiate $\tanh = \sinh / \cosh$ with the quotient rule and use $\cosh^2 - \sinh^2 = 1$; the forward pass already saved H .

$$dZ = (1 - H \odot H) \odot dH \quad (N \times m)$$

$1 - H \odot H$ means 1 minus each entry of $H \odot H$. The Jacobian of H with respect to Z is $Nm \times Nm$ and diagonal, with these entries on the diagonal, so multiplying by it is an elementwise product: the batch changes the shape, not the rule.

Problem 8

Masked softmax cross-entropy: logits $Z \in \mathbb{R}^{N \times k}$, row-wise softmax S , one-hot target rows T , and $\mu \in \{0, 1\}^N$ marking the real rows (padding has $\mu_n = 0$; at least one $\mu_n = 1$). With $\ell_n = -\sum_i T_{ni} \log S_{ni}$, the loss is $L = \sum_n \mu_n \ell_n / \sum_n \mu_n$. Compute dZ , and check the case where every $\mu_n = 1$.

Let $M = \sum_n \mu_n = \mathbf{1}^\top \mu$, the number of real rows, so $L = \frac{1}{M} \sum_n \mu_n \ell_n$.

$M \geq 1$ by assumption, and it does not depend on Z , so it is a constant for the derivative.

Row n of Z feeds only ℓ_n , and $\nabla_{z^{(n)}} \ell_n = s^{(n)} - t^{(n)}$, with $z^{(n)\top}, s^{(n)\top}, t^{(n)\top}$ row n of Z, S, T .

The softmax is taken row by row; the gradient is the softmax page's Problem 5 for example n .

Row n of dZ is $\frac{\mu_n}{M} (s^{(n)} - t^{(n)})^\top$.

ℓ_n enters L with coefficient μ_n/M .

Multiplying row n of a matrix by μ_n , for every n , is multiplying it on the left by $\text{diag}(\mu)$.

$$(\text{diag}(\mu)A)_{ni} = \mu_n A_{ni}.$$

$$dZ = \frac{1}{\mathbf{1}^\top \mu} \text{diag}(\mu)(S - T); \text{ with every } \mu_n = 1 \text{ it is } \frac{1}{N}(S - T)$$

With every row real, $\text{diag}(\mu) = I$ and $\mathbf{1}^\top \mu = N$: the softmax page's batched result. Padding rows get a zero gradient whatever their logits, and dividing by the number of real rows, not by N , keeps the loss on the same scale however much padding a batch carries. In code $\text{diag}(\mu)$ is never built: the mask multiplies each row.

Problem 9

Gradient accumulation: split N rows into K micro-batches of $B = N/K$ rows, with $L^{(j)}$ the mean loss over micro-batch j . Show that the full-batch mean loss is $L = \frac{1}{K} \sum_j L^{(j)}$ and give $\nabla_W L$. What changes if the sizes differ?

$$L = \frac{1}{N} \sum_n \ell_n = \frac{1}{N} \sum_j \sum_{n \in \text{batch } j} \ell_n.$$

Group the sum by micro-batch; each example is in exactly one.

$$\sum_{n \in \text{batch } j} \ell_n = B L^{(j)}.$$

$L^{(j)}$ is the mean of the B losses in micro-batch j .

$$L = \frac{B}{N} \sum_j L^{(j)} = \frac{1}{K} \sum_j L^{(j)}.$$

$$N = KB.$$

$$\nabla_W L = \frac{1}{K} \sum_j \nabla_W L^{(j)}.$$

The gradient of a sum is the sum of the gradients, and $\frac{1}{K}$ is a constant.

$$L = \frac{1}{K} \sum_j L^{(j)}, \text{ so } \nabla_W L = \frac{1}{K} \sum_j \nabla_W L^{(j)}$$

Accumulating the K micro-batch gradients and dividing once by K reproduces the full-batch gradient, up to rounding. This needs the examples not to interact, which holds for every layer on this page; batch normalisation, which mixes examples, is the exception. With sizes B_j the same steps give weights B_j/N in place of $\frac{1}{K}$.

Problem 10

Sequences: $X \in \mathbb{R}^{N \times T \times d}$ holds N sequences of T positions, and the layer acts at every position: $y^{(n,t)} = W^\top x^{(n,t)} + b$, with $x^{(n,t)}$ position t of sequence n . Compute dW and db as matrix products.

$$dW = \sum_{n,t} x^{(n,t)} dy^{(n,t)\top} \text{ and } db = \sum_{n,t} dy^{(n,t)}.$$

Every position of every sequence uses the same W and b , so each is an example in the sense of Problems 3 and 5, and their contributions add.

Let $\tilde{X} \in \mathbb{R}^{NT \times d}$ have one row $x^{(n,t)\top}$ per pair (n, t) , and $d\tilde{Y} \in \mathbb{R}^{NT \times m}$ the rows $dy^{(n,t)\top}$ in the same order.

Stacking the pairs as rows turns the double sum into a single sum over the NT rows; any order works if both use the same one.

$dW = \tilde{X}^\top d\tilde{Y}$ and $db = d\tilde{Y}^\top \mathbf{1}$, where $\tilde{X}$ ($NT \times d$) and $d\tilde{Y}$ ($NT \times m$) stack every position of every sequence as a row

Problem 5, step 3 and Problem 3 applied to the NT rows: $(d \times NT)(NT \times m) = d \times m$. A position-wise layer treats positions exactly like examples, so a transformer's feed-forward and projection layers see a batch of NT rows. If the loss is a mean over all positions, the factor in $d\tilde{Y}$ is $\frac{1}{NT}$, not $\frac{1}{N}$.

Where this goes wrong

1. Weight gradient in the outputs-by-inputs layout

The one-hidden-layer page computed $\nabla_{W_2} L = \Delta_2^\top H$, a product with the transposed delta on the left, and that pattern is easy to carry from one layer to the next.

$Y = XW + \mathbf{1}b^\top$, W is $d \times m$, dY is $N \times m$

Right so far: the layer and shapes of Problem 1.

“The weight gradient is the upstream gradient, transposed, times the input.”

The habit that causes the mistake: the rule for $Y = XW^\top + \mathbf{1}b^\top$, where W is outputs by inputs, applied to a layer stored inputs by outputs.

$dW = dY^\top X$ ✗

It is $(m \times N)(N \times d) = m \times d$, the transpose of W 's shape. In $Y = XW$, W_{ij} multiplies X_{ni} into Y_{nj} , so $dW = X^\top dY$ (Problem 2). When $d = m$ the update runs and silently applies the transpose.

2. Dividing by N twice

The bias gradient is often described as “the average of the deltas over the batch”, and with a mean loss that description is true of the result, not of the formula.

$dY = \frac{1}{N}(Y - T)$

Right so far: Problem 6, step 3. The $\frac{1}{N}$ of the mean loss is already inside dY .

“The bias gradient is the average of the upstream gradient over the batch.”

The shortcut that causes the mistake: averaging rows of a dY that already carries the $\frac{1}{N}$.

$$db = \frac{1}{N} dY^\top \mathbf{1} \quad \text{✗}$$

That is $\frac{1}{N^2}(Y - T)^\top \mathbf{1}$, N times too small. The rule is $db = dY^\top \mathbf{1}$, a plain sum (Problem 3), because the factor was applied once at the loss (Problem 6). The bias then learns N times more slowly than the weights, and the gap grows with the batch size.

3. Multiplying the upstream gradient by W instead of $W^\top$

The forward pass multiplies by W , and it is tempting to think the backward pass simply multiplies by it again.

$Y = XW + \mathbf{1}b^\top$, and dX should have the shape of X , $N \times d$

Right so far: Problem 1.

“Backward goes through the same weights, so multiply by W .”

The analogy that causes the mistake: the backward pass uses the same weights, but it runs the map in the opposite direction, from m output features back to d input features, which is what $W^\top$ does.

$$dX = dY W \quad \text{✗}$$

$(N \times m)(d \times m)$ is undefined unless $m = d$. The rule is $dX = dY W^\top$ (Problem 4): $\partial L / \partial X_{ni} = \sum_j dY_{nj} W_{ij}$ sums over W 's second index. For a square W the product runs and sends every gradient back along the wrong weights.

4. Elementwise derivative applied as a matrix product

The chain rule is a product of Jacobians, and in a batch every factor looks like a matrix.

$$H = \tanh(Z), \text{ and } 1 - H \odot H \text{ is } N \times m$$

Right so far: Problem 7, step 3 gives the derivative at every entry.

“The chain rule multiplies the upstream gradient by the local derivative, so multiply the matrices.”

The analogy that causes the mistake: the rule $\nabla_{\text{in}} = J^\top \nabla_{\text{out}}$ for vectors, applied to two $N \times m$ arrays as if the derivative array were the Jacobian.

$$dZ = dH (1 - H \odot H)^\top \mathbf{x}$$

It is $(N \times m)(m \times N) = N \times N$, a matrix over pairs of examples. The Jacobian of an elementwise map is diagonal, so multiplying by it is an elementwise product: $dZ = (1 - H \odot H) \odot dH$ (Problem 7).

5. Summing micro-batch means without dividing by K

Gradient accumulation is usually written as a running sum of gradients, and the division at the end is easy to leave out.

$$L = \frac{1}{K} \sum_j L^{(j)} \text{ with } L^{(j)} \text{ the mean over micro-batch } j$$

Right so far: Problem 9, step 3.

“Accumulate the gradients of the micro-batches; the result is the gradient of the full batch.”

The shortcut that causes the mistake: true when each $L^{(j)}$ is a sum, or already divided by the full N , but here each one is a mean over B rows.

$$\nabla_W L = \sum_j \nabla_W L^{(j)} \mathbf{x}$$

It is K times the full-batch gradient: each $L^{(j)}$ is already a mean, so the sum of K of them must be divided by K (Problem 9). Switching from one batch of N to K micro-batches then multiplies the effective learning rate by K , which looks like a training instability rather than an arithmetic slip.