Constraints by Construction

Lecture 5B · Parameterization is part of model design

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

A neural network can output values the model cannot use

V VISUAL

Neural-network output

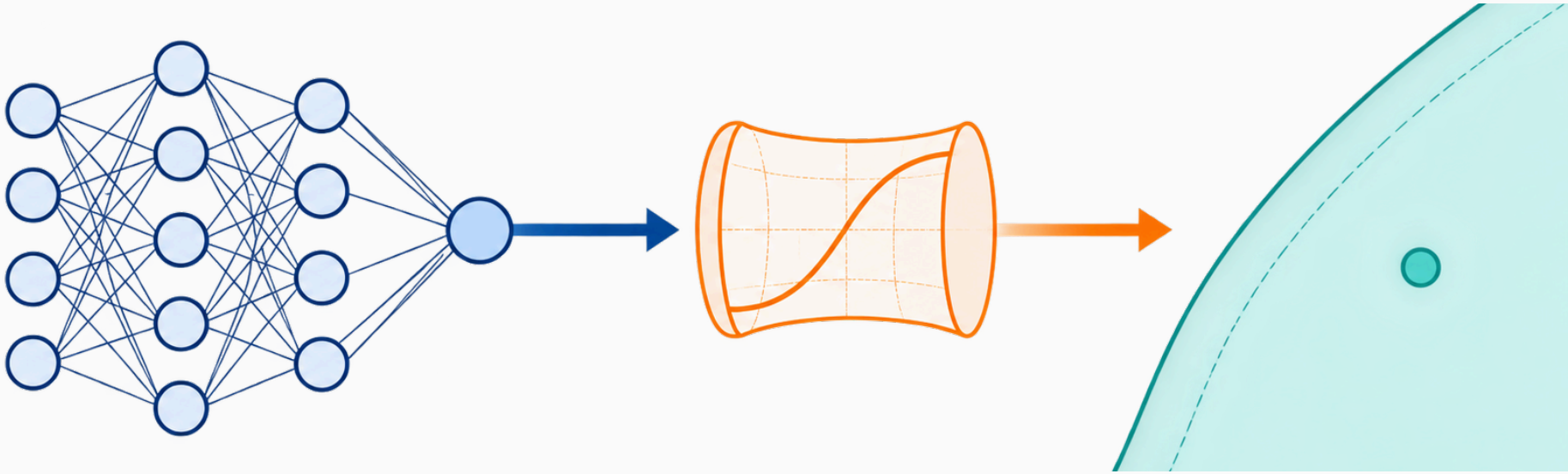
$$z = f_{w(x)} \in \mathbb{R}^d$$

Smooth map

$$\varphi : \mathbb{R}^d \to \mathcal{C}$$

Valid model parameter

$$\theta = \varphi(z) \in \mathcal{C}$$



The network may emit any $z \in \mathbb{R}^d$; φ ensures the model only sees a valid $\theta \in \mathcal{C}$.

Before optimizing: what does scale do?

D DERIVATION

For one Gaussian observation, let $r = y - \mu$ and work inside the valid domain $\sigma > 0$:

$$y \mid x \sim \mathcal{N}(\mu, \sigma^2), \quad \mathcal{L} = \underbrace{\frac{r^2}{2\sigma^2}}_{\text{fit the residual}} + \underbrace{\log \sigma}_{\text{pay for uncertainty}} + C.$$

$\sigma < 0$

Not a valid standard deviation. The probabilistic model is undefined.

$\sigma \to 0$

If $r \neq 0$, the residual term explodes. Tiny scales demand tiny errors.

σ is large

The $\log \sigma$ term grows. The model cannot inflate uncertainty for free.

What will one gradient step produce?

Q QUESTION

Suppose a Gaussian model learns its scale directly. At one update,

$$\sigma_0 = 0.1, \quad \frac{\partial \mathcal{L}}{\partial \sigma} = 5, \quad \eta = 0.1.$$

What is $\sigma_1 = \sigma_0 - \eta \frac{\partial \mathcal{L}}{\partial \sigma}$?

A. 0.6: the step increases uncertainty

B. 0.4: take the absolute value

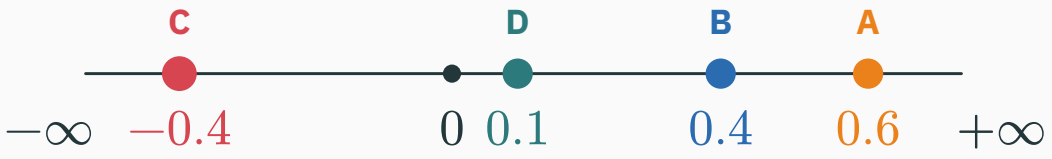
C. -0.4: the step crosses zero

D. 0.1: the model blocks invalid moves

A gradient step can make σ invalid

A ANSWER

$$\sigma_1 = 0.1 - 0.1(5) = -0.4.$$



Correct: C — -0.4

Why: Gradient descent follows the local slope, even when the step makes σ invalid.

What failed?

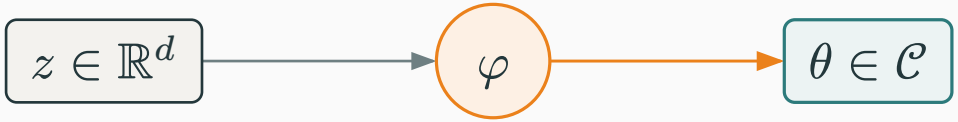
The update followed the usual gradient-descent rule:

$$\text{new} = \text{old} - \eta \times \text{gradient}.$$

The problem is the coordinate: σ is valid only on $(0, \infty)$, but direct optimization treats it as an unconstrained real number.

Correct fix: make invalid values unreachable

D DERIVATION



Positive	$s \in \mathbb{R} \to \sigma = e^s$ or $\sigma = \mathrm{softplus}(s) + \varepsilon, \varepsilon > 0$; either gives $\sigma > 0$
Probability	$z \in \mathbb{R} \to p = \mathrm{sigmoid}(z) \in (0, 1)$
Probability vector (simplex)	$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{z}) : \quad p_k > 0, \quad \sum_k p_k = 1$
Positive-definite covariance	raw vector $\to$ lower-triangular factor $\to \Sigma \succ 0$

$z \in \mathbb{R}^d$: unrestricted network output; d is its length. φ **(phi):** constraint map.
$\mathcal{C}$: allowed parameter set. $\theta = \varphi(z) \in \mathcal{C}$: valid parameter used by the model.

Softplus smoothly turns any real number positive

V VISUAL

Take any real-valued raw coordinate u and define

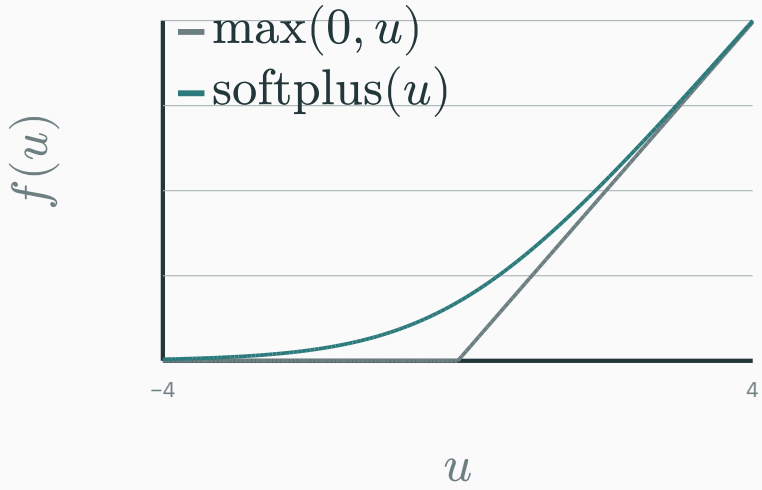
$$\mathrm{softplus}(u) = \log(1 + e^u).$$

Constraint guarantee

$\sigma = \mathrm{softplus}(u) + \varepsilon > \varepsilon$ for every finite u.

Learning signal

Softplus is increasing, and its slope stays below 1.



Softplus rounds off the ReLU corner while remaining strictly above zero.

$$u = -4 \to 0.018 \quad u = 0 \to \log 2 \approx 0.693$$
$$u = 3 \to 3.05$$

Can you differentiate softplus?

Q QUESTION

Let $a(u) = 1 + e^u$. Then $\text{softplus}(u) = \log(a(u))$.

1. Inner derivative

$$a'(u) = 0 + e^u = e^u.$$

2. Outer derivative

$$\frac{\partial}{\partial a} \log a = \frac{1}{a}.$$

3. Apply the chain rule

$$\frac{\partial\ \text{softplus}(u)}{\partial u} = \frac{1}{a(u)} \times a'(u)$$

$$= \frac{1}{1 + e^u} \times e^u = \frac{e^u}{1 + e^u}.$$

The derivative of softplus is a sigmoid

D DERIVATION

Divide the numerator and denominator of $\frac{e^u}{1+e^u}$ by e^u.

$$\frac{e^u}{1+e^u} = \frac{\frac{e^u}{e^u}}{\frac{1+e^u}{e^u}}$$

$$= \frac{1}{e^{-u}+1} = \frac{1}{1+e^{-u}}$$

$$= \text{sigmoid}(u).$$

Large negative u

The slope is near 0.

$u = 0$

The slope is exactly $\frac{1}{2}$.

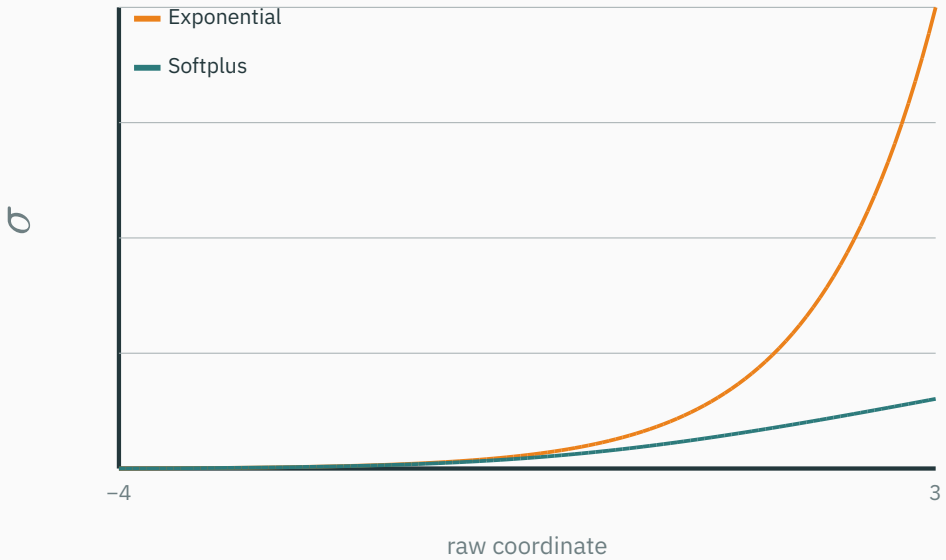
Large positive u

The slope is near 1.

For finite u in exact arithmetic, $0 < \text{sigmoid}(u) < 1$; softplus has no exactly flat branch.

The same raw step is gentler under softplus

V VISUAL



Starting from $\sigma = 10$, take the same raw step $\Delta = +0.1$.

Exponential	$10 \to 11.05$
Softplus	$10 \to 10.10$

Here we use softplus: large predicted scales move gently.

For Gaussian regression, we use softplus

V VISUAL

```
mu, raw_scale = net(x).unbind(-1)  # net(x): [N, 2]
assert y.ndim == 2 and y.shape[-1] == 1
y = y[:, 0]                        # [N, 1] -> [N]
assert y.shape == mu.shape

# Chosen here: positive scale + a small numerical floor
sigma = F.softplus(raw_scale) + 1e-4

# Also valid, but not used here:
# sigma = raw_scale.exp()

loss = -Normal(mu, sigma).log_prob(y).mean()
```

PyTorch `softplus` documentation

Which map?

We use **softplus** from here on. `exp` would also keep σ positive, but it is a different map. Pick one.

What does PyTorch handle?

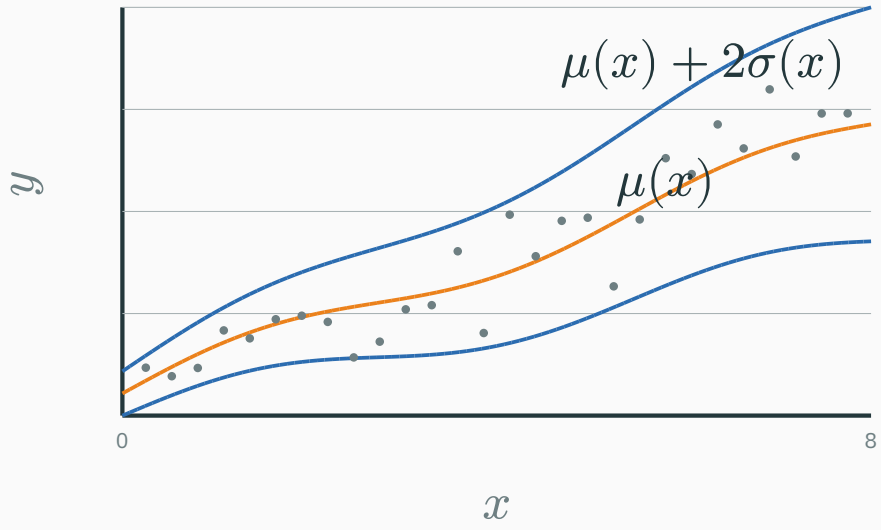
`Normal.log_prob(y)` evaluates the Gaussian likelihood; autograd differentiates through softplus. No hand derivation is needed.

Why assert shapes?

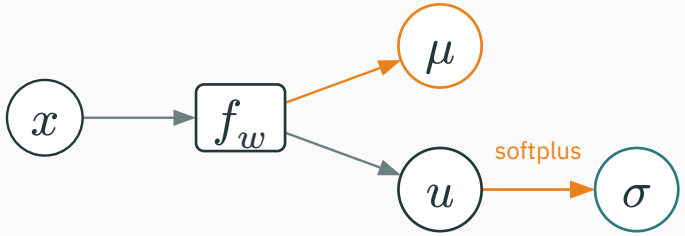
Keep μ and y both `[N]`; otherwise broadcasting can silently produce `[N, N]` log probabilities.

Heteroskedastic regression makes the scale visible

V VISUAL



predicted mean $\mu(x)$ $\mu(x) \pm 2\sigma(x)$ noise widens with x



Two outputs, two meanings

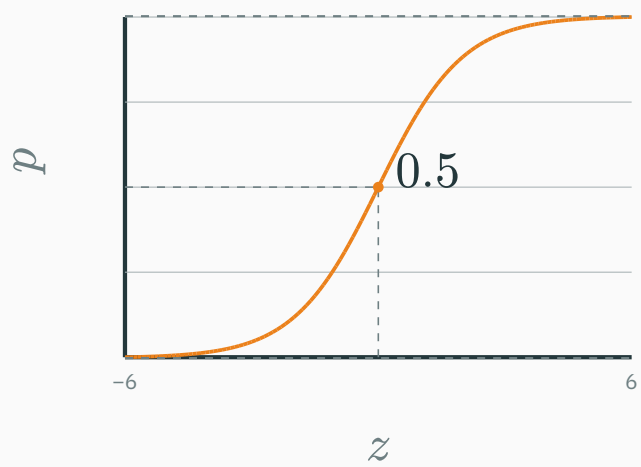
$\mu(x)$ is the conditional mean. Softplus maps the raw $u(x)$ to a valid conditional scale.

Under the Gaussian model, $\sigma(x)$ is the conditional residual, or aleatoric, scale. It does not measure epistemic uncertainty about the network.

Sigmoid maps one logit to a valid probability

V VISUAL

$$p = \text{sigmoid}(z) = \frac{1}{1 + e^{-z}}, \quad z \in \mathbb{R}.$$



z	$p = \text{sigmoid}(z)$
-2	0.119
0	0.500
2	0.881

Negative logits favor class 0; positive logits favor class 1. Running the map backward gives the **log-odds**; let us unpack that scale next.

In exact arithmetic, finite logits map into the open interval $(0, 1)$.

Finite precision can saturate sigmoid to exactly 0 or 1 for extreme logits.

Odds compare class 1 with class 0

V VISUAL

$$\text{odds} = \frac{p}{1-p}.$$

In binary classification, p is the probability of class 1 and $1 - p$ is the probability of class 0.

$p = 0.01$	$p = 0.50$	$p = 0.99$
$\text{odds} = \frac{0.01}{0.99}$	$\text{odds} = \frac{0.50}{0.50}$	$\text{odds} = \frac{0.99}{0.01}$
$= \frac{1}{99} \approx 0.010$	$= 1$	$= 99$
Class 0 is strongly favored.	Neither class is favored.	Class 1 is strongly favored.
Odds < 1: class 0	**Odds $= 1$: tie**	**Odds > 1: class 1**

Log-odds stretch probabilities across the real line

V VISUAL

$$z = \operatorname{logit}(p) = \log(\text{odds}) = \log\left(\frac{p}{1-p}\right).$$

Taking the logarithm sends odds below 1 negative, odds of 1 to zero, and odds above 1 positive.

$p = 0.01$	$p = 0.50$	$p = 0.99$
$z = \log\left(\frac{1}{99}\right)$	$z = \log(1)$	$z = \log(99)$
≈ -4.60	$= 0$	$\approx +4.60$
$p \to 0^+ \quad \Rightarrow \quad z \to -\infty$	$p = 0.5 \quad \Rightarrow \quad z = 0$	$p \to 1^- \quad \Rightarrow \quad z \to +\infty$

Inverse maps: $p = \operatorname{sigmoid}(z)$ if and only if $z = \operatorname{logit}(p)$.

Can you rewrite BCE using $\mathrm{softplus}(z)$?

Q QUESTION

Let $y \in \{0, 1\}$, $p = \mathrm{sigmoid}(z)$, and $\ell(z, y) = \mathrm{BCE}(\mathrm{sigmoid}(z), y)$.

$$\ell(z, y) = -y \log p - (1 - y) \log(1 - p).$$

1. Rewrite $\log p$

$$p = \frac{e^z}{1+e^z}$$

$$\log p = z - \log(1 + e^z)$$

2. Rewrite $\log(1 - p)$

$$1 - p = \frac{1}{1+e^z}$$

$$\log(1 - p) = -\log(1 + e^z)$$

3. Substitute and collect the common term

$$\ell(z, y) = -y(z - \log(1 + e^z)) + (1 - y) \log(1 + e^z)$$

$$= -yz + [y + (1 - y)] \log(1 + e^z) = \mathrm{softplus}(z) - yz.$$

A correct formula can still overflow on a computer

D DERIVATION

$$\mathrm{softplus}(z) = \log(1 + e^z)$$ **is mathematically correct.**

A modest input: $z = 2$

$$e^2 \approx 7.389$$

$$\log(1 + e^2) \approx 2.1269.$$

The direct computation creates no unusually large number, so it works.

A large input: $z = 1000$

$$e^{1000} \to \infty \text{ in float32}$$

$$\log(1 + \infty) = \infty.$$

But the true value is approximately 1000, not infinity. The computer overflowed before taking the logarithm.

What we need: the same mathematical function, evaluated in an order that never constructs a huge exponential.

Can we evaluate softplus without a large exponential?

Q QUESTION

Start from $\mathrm{softplus}(z) = \log(1 + e^z)$ and separate the two signs of z.

If $z \geq 0$: move the large term outside

$$\log(1 + e^z)$$
$$= \log\left(e^{z(1+e^{-z})}\right)$$
$$= z + \log(1 + e^{-z}).$$

Now $-z \leq 0$, so the exponential is at most 1.

If $z < 0$: the original form is already safe

$$0 < e^z < 1$$
$$\log(1 + e^z)$$
$$= 0 + \log\left(1 + e^{-|z|}\right).$$

Here $-|z| = z$, so we have not changed the value.

One expression covers both cases

$$\mathrm{softplus}(z) = \max(z, 0) + \log\left(1 + e^{-|z|}\right).$$

Because $-|z| \leq 0$, the remaining exponential satisfies $e^{-|z|} \leq 1$ and cannot overflow.

Same answer normally; stable answer at extremes

D DERIVATION

Compare float32 outputs

z	Direct formula	Stable route
-1000	0	0
2	2.1269	2.1269
1000	inf	1000

At ordinary inputs the answers agree. At $z = 1000$, only the stable evaluation returns the finite answer.

Check it in PyTorch

```
z = torch.tensor([-1000., 2., 1000.])

naive = torch.log1p(torch.exp(z))
stable = F.softplus(z)

# naive:  tensor([0., 2.1269, inf])
# stable: tensor([0., 2.1269, 1000.])
```

`log1p` improves accuracy when its input is small, but it cannot rescue an `exp(z)` that has already overflowed.

In practice: use `F.softplus`. For binary classification, use the fused BCE-with-logits operation introduced after the gradient derivation.

Is underflow a problem too?

Q QUESTION

$z = +1000$

$$e^{-1000} \to 0$$

$$\mathrm{softplus}(z) = 1000 + \log(1 + 0) = 1000.$$

Only an unrepresentably tiny correction was discarded.

$z = -1000$

$$e^{-1000} \to 0$$

$$\mathrm{softplus}(z) = 0 + \log(1 + 0) = 0.$$

The exact answer is positive, about e^{-1000}, but too small to store.

Forward value: overflow turned a useful finite answer into ∞; underflow here drops only a tiny correction. For a scale, $\sigma = \mathrm{softplus}(u) + \varepsilon$ still gives $\sigma > \varepsilon$.

Learning caveat: $\mathrm{softplus}'(z) = \mathrm{sigmoid}(z)$ also rounds to 0 at $z = -1000$, so an extremely negative raw scale can learn very slowly.

What is the BCE gradient with respect to the logit?

Q QUESTION

Start from the smooth expression we just derived:

$$\ell(z, y) = \mathrm{softplus}(z) - yz.$$

Differentiate with respect to the logit

$$\frac{\partial \ell}{\partial z} = \mathrm{sigmoid}(z) - y = p - y.$$

We used $\mathrm{softplus}'(z) = \mathrm{sigmoid}(z) = p$.

If $p - y < 0$

The loss has a **negative slope** at the current z. Moving z upward makes the loss fall.

If $p - y > 0$

The loss has a **positive slope** at the current z. Moving z downward makes the loss fall.

The crucial next step: gradient descent subtracts this slope.

Which way will the logit move?

Q QUESTION

$$\text{new} \quad z = \text{old} \quad z - \eta(p - y).$$

Target $y = 1$, current probability $p = 0.2$, learning rate $\eta = 0.1$.

One example

$$p - y = 0.2 - 1 = -0.8$$

$$\text{new} \quad z = \text{old} \quad z - 0.1(-0.8)$$

$$= \text{old} \quad z + 0.08.$$

Subtracting a negative slope raises z. A larger logit gives a larger probability p, moving the prediction toward the target $y = 1$.

The same $p - y$ signal trains the whole network

D DERIVATION

Now return to the real network: the optimizer updates θ. For $z = f_\theta(x)$, apply the chain rule:

Error signal $\times$ sensitivity of the logit

$$\nabla_\theta \ell = (p - y)\nabla_\theta z.$$

The product combines the prediction error with each parameter's effect on the logit.

For $z = w^T h + b$

$$\frac{\partial \ell}{\partial w} = (p - y)h$$

$$\frac{\partial \ell}{\partial b} = p - y$$

$$\frac{\partial \ell}{\partial h} = (p - y)w.$$

What the two factors mean

$p - y$ says whether the prediction should move **up or down**, and how strongly.

$\nabla_\theta z$ says how each parameter changes that prediction.

Train with logits

D DERIVATION

Why not form p first?

```
p = torch.sigmoid(z)
loss = -(y * p.log()
         + (1 - y) * torch.log1p(-p)).mean()
```

With $z = 100$ and $y = 0$, float32 sigmoid may round p to 1, so $-\log(1 - p) \to \infty$.

Forming p first can make the naive formula infinite.

Use the fused primitive

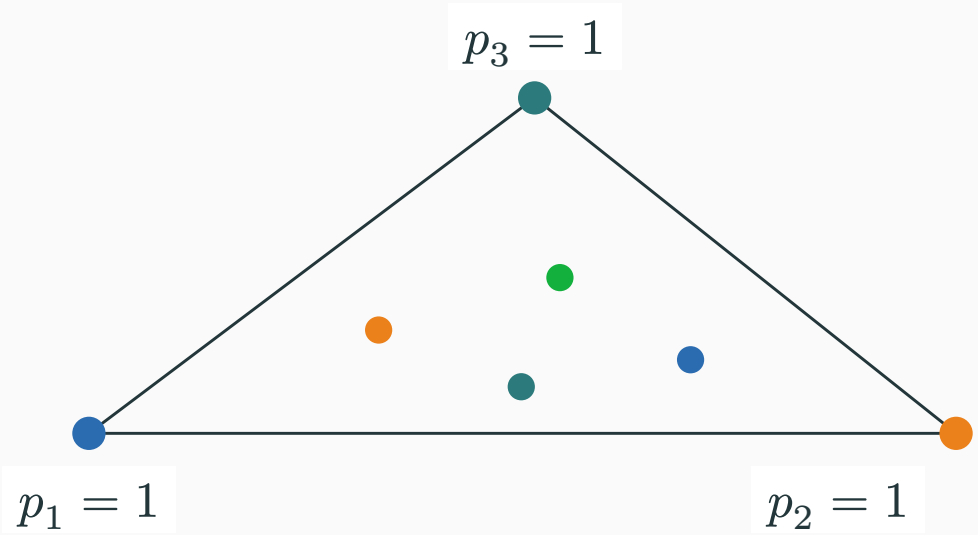
```
assert y.dtype.is_floating_point
assert y.shape == z.shape
assert ((0 <= y) & (y <= 1)).all()

loss = F.binary_cross_entropy_with_logits(z, y)
p = z.sigmoid()  # interpretation/reporting only
```

For $z = 100, y = 0$, the fused loss is approximately 100, not infinity.

PyTorch `BCEWithLogitsLoss`: fused sigmoid and BCE

Softmax maps logits into the simplex

V VISUAL



For $K = 3$, the simplex is this triangle of valid probability vectors. An edge means at least one class has probability 0.

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}, \quad \boldsymbol{z} \in \mathbb{R}^K.$$

The simplex is the set of vectors with $p_k \geq 0$ and $\sum_k p_k = 1$.

In exact arithmetic, finite logits guarantee

$$p_k > 0, \quad \sum_k p_k = 1.$$

Thus softmax lands strictly **inside** the simplex, never exactly on an edge. Boundary distributions are approached as logit differences grow.

What does naive softmax compute at extreme logits?

Q QUESTION

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}.$$

Try $\boldsymbol{z} = (1000, 999, 998)$ and $\boldsymbol{z} = (-1000, -1001, -1002)$ in float32.

Overflow: exponentials become infinite

$$\boldsymbol{z} = (1000, 999, 998)$$

$$\text{float32:} \quad e^{\boldsymbol{z}} \to (\infty, \infty, \infty)$$

$$p_k = \frac{\infty}{\infty} \to \text{NaN}.$$

Underflow: exponentials become zero

$$\boldsymbol{z} = (-1000, -1001, -1002)$$

$$\text{float32:} \quad e^{\boldsymbol{z}} \to (0, 0, 0)$$

$$p_k = \frac{0}{0} \to \text{NaN}.$$

Both vectors have the same logit gaps and should give $\boldsymbol{p} \approx (0.665, 0.245, 0.090)$. The probability distribution exists; the direct floating-point algorithm failed before it could normalize.

Does adding the same constant change softmax?

Q QUESTION

Let c be the same scalar added to every logit.

$$p'_k = \frac{e^{z_k+c}}{\sum_j e^{z_j+c}}.$$

$$p'_k = \frac{e^c e^{z_k}}{e^c \sum_j e^{z_j}}$$

$$= \frac{e^{z_k}}{\sum_j e^{z_j}} = p_k.$$

Only the gaps matter

$$(z_i + c) - (z_j + c) = z_i - z_j.$$

The common offset carries no class information.

Exact consequence

$$(1000, 999, 998) = (2, 1, 0) + 998(1, 1, 1).$$

These vectors have exactly the same softmax in exact arithmetic.

A small example shows shifting preserves probabilities

D DERIVATION

Quantity	Direct: $z = (2, 1, 0)$	Shifted: $r = (0, -1, -2)$
Logits	$(2, 1, 0)$	$(0, -1, -2)$
Exponentials	$(7.389, 2.718, 1.000)$	$(1.000, 0.368, 0.135)$
Sum	11.107	1.503
Probabilities	$(0.665, 0.245, 0.090)$	$(0.665, 0.245, 0.090)$

Why the ratios match

$$e^{z_j - 2} = \frac{e^{z_j}}{e^2}.$$

Every numerator and the denominator were divided by the same positive number e^2, so every ratio is unchanged.

The loss is unchanged too: for class 1, $-\log(p_1) \approx 0.408$ along either route. For $\boldsymbol{z} = (1000, 999, 998)$, subtracting its maximum also gives $\boldsymbol{r} = (0, -1, -2)$; next we work that extreme case through.

The same shift rescues $z = (1000, 999, 998)$

D DERIVATION

Direct fails; shifting first works

Direct float32: $e^{\boldsymbol{z}} \to (\infty, \infty, \infty)$, so $\boldsymbol{p} \to (\mathrm{NaN}, \mathrm{NaN}, \mathrm{NaN})$.

Shifted route:

1. $m = \max_j z_j = 1000$
2. $r_j = z_j - m$, so $\boldsymbol{r} = (0, -1, -2)$
3. $\boldsymbol{w} = e^{\boldsymbol{r}} \approx (1, 0.368, 0.135)$
4. $S = \sum_j w_j \approx 1.503$
5. $\boldsymbol{p} = \frac{\boldsymbol{w}}{S} \approx (0.665, 0.245, 0.090)$.

Why the shifted arithmetic is safe

$r_j = z_j - m \le 0$ for every class.
At least one $r_j = 0$.
Therefore $0 \le e^{r_j} \le 1$.
The denominator contains $e^0 = 1$, so

$$1 \le \sum_j e^{r_j} \le K.$$

No exponential overflows, and the denominator cannot become zero.

Very small probabilities may still round to zero. That is acceptable for reporting, but a training loss should stay in log space.

For training, stay in log space

D DERIVATION

Stable cross-entropy ($m = \max_j z_j$)

$$\ell(\boldsymbol{z}, t) = -\log p_t$$
$$= -z_t + \log\left(\sum_j e^{z_j}\right)$$
$$= -(z_t - m) + \log\left(\sum_j e^{z_j - m}\right).$$

Why log space matters

For $\boldsymbol{z} = (0, 1000)$ and **mathematical class 1**, softmax may round to $(0, 1)$, so $-\log(p_1) = \infty$. The stable expression gives

$$1000 + \log(1 + e^{-1000}) \approx 1000.$$

Use the fused primitives

```
# logits: [N, K]
# target: [N], class indices 0,...,K-1
loss = F.cross_entropy(logits, target)

# only when these values are needed
log_probs = F.log_softmax(logits, dim=-1)
probs = logits.softmax(dim=-1)
```

`cross_entropy` accepts unnormalized logits; `log_softmax` avoids computing `log(softmax(...))` as two unstable operations.

PyTorch `cross_entropy` · PyTorch `log_softmax`

Stable does not make floating-point exact or repair `NaN` or infinite input logits.

A multivariate Gaussian needs a covariance matrix

V VISUAL

$$y \sim \mathcal{N}(\mu, \sigma^2) \quad \rightarrow \quad \boldsymbol{Y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma).$$

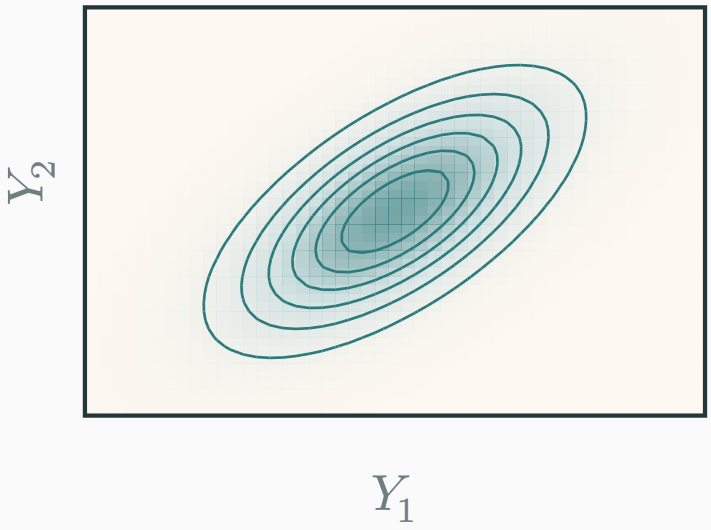
The definition

$$\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{Y}],$$

$$\Sigma = \mathrm{Cov}(\boldsymbol{Y}) = \mathbb{E}\big[(\boldsymbol{Y} - \boldsymbol{\mu})(\boldsymbol{Y} - \boldsymbol{\mu})^T\big].$$

$$\Sigma = \begin{pmatrix} \mathrm{Var}(Y_1) & \mathrm{Cov}(Y_1, Y_2) \\ \mathrm{Cov}(Y_1, Y_2) & \mathrm{Var}(Y_2) \end{pmatrix}.$$

In one dimension, variance sets the width. In several dimensions, covariance sets the widths and tilt of the Gaussian cloud.



Equal-density contours form a tilted ellipse around $\boldsymbol{\mu}$.

Read the entries

Diagonal: each output's variance.
Off-diagonal: how the two outputs move together.
The repeated off-diagonal entry makes $\Sigma = \Sigma^T$.

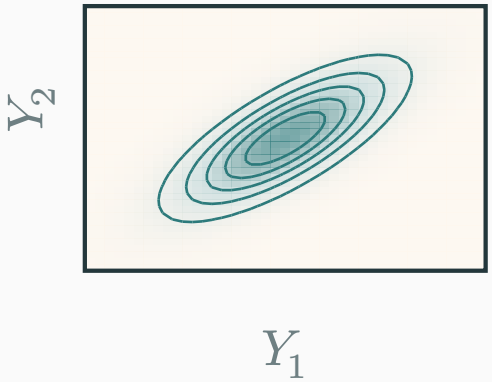
Covariance shows how two outputs move together

V VISUAL

$$\mathrm{Cov}(Y_1, Y_2) = \mathbb{E}[(Y_1 - \mu_1)(Y_2 - \mu_2)].$$

Each observation contributes the product of its two deviations from the means.

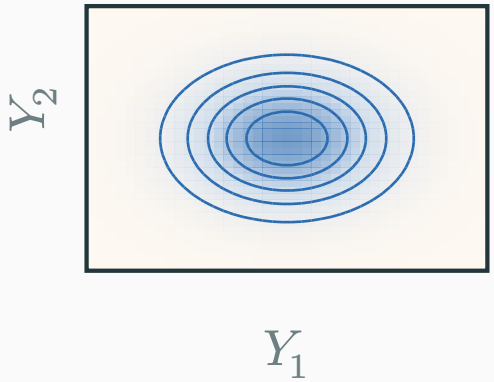
Positive covariance



$\rho = 0.75$

Same-sign deviations reinforce.

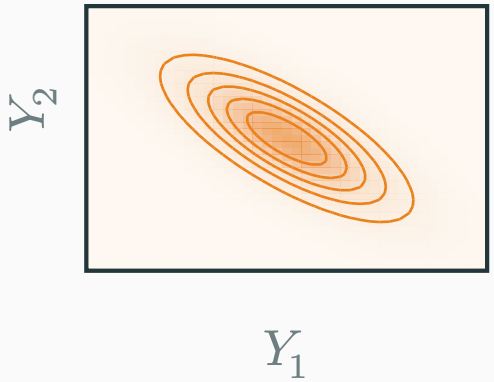
Zero covariance



$\rho = 0$

Positive and negative products cancel.

Negative covariance



$\rho = -0.75$

Opposite-sign deviations dominate.

$\rho = \frac{\mathrm{Cov}(Y_1, Y_2)}{\mathrm{sd}(Y_1)\mathrm{sd}(Y_2)} \in [-1, 1]$ is unitless. Covariance carries multiplied units—for example, cm·kg. For a jointly Gaussian pair, $\rho = 0$ also means independence.

PSD vs PD: can a combination have zero variance?

V VISUAL

Choose weights $\boldsymbol{w} = (w_1, w_2)$ and combine the outputs:

$$S = w_1 Y_1 + w_2 Y_2, \quad \mathrm{Var}(S) = \boldsymbol{w}^T \Sigma \boldsymbol{w}.$$

$\boldsymbol{w}$ is simply the pair of weights; S is one weighted sum of the two outputs.

Positive semidefinite (PSD)

Every choice of weights gives $\mathrm{Var}(S) \geq 0$; a nonzero choice may still give 0.

$$\Sigma = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \boldsymbol{w} = (1, -1)$$

$$S = Y_1 - Y_2, \quad \mathrm{Var}(S) = 0.$$

The difference is constant: the two outputs are perfectly locked together.

Positive definite (PD)

Every nonzero choice of weights gives $\mathrm{Var}(S) > 0$; zero is not allowed.

$$\Sigma = I_2$$

$$\mathrm{Var}(S) = w_1^2 + w_2^2 > 0 \textbf{ if } \boldsymbol{w} \neq \mathbf{0}.$$

No nontrivial weighted combination is perfectly fixed.

Why does $\Sigma = LL^T$ guarantee positive variance?

Q QUESTION

For a 2-D target, the network emits three unrestricted numbers:

$$\boldsymbol{q} = (q_{11}, q_{21}, q_{22}) \in \mathbb{R}^3.$$

$$L(\boldsymbol{q}) = \begin{pmatrix} \mathrm{softplus}(q_{11}) + \varepsilon & 0 \\ q_{21} & \mathrm{softplus}(q_{22}) + \varepsilon \end{pmatrix}$$

$$\Sigma = LL^T.$$

Read the construction

$\boldsymbol{q}$**:** raw network outputs.

$\boldsymbol{L}$**:** lower-triangular factor.

Softplus: makes the two diagonal entries positive.

$\boldsymbol{\varepsilon > 0}$**:** a small numerical floor.

Pick any nonzero weights $\boldsymbol{w}$. Positive diagonal entries make L invertible, so $L^T\boldsymbol{w} \neq \mathbf{0}$ and $\boldsymbol{w}^T\Sigma\boldsymbol{w} = \left\|L^T\boldsymbol{w}\right\|^2 > 0$. Thus every nontrivial weighted combination has positive variance, so Σ is PD.

In PyTorch, pass the factor directly

V VISUAL

```
mu, q = model(x)                    # [B, 2], [B, 3]

L = q.new_zeros(q.shape[:-1] + (2, 2))
L[..., 0, 0] = F.softplus(q[..., 0]) + 1e-4
L[..., 1, 0] = q[..., 1]
L[..., 1, 1] = F.softplus(q[..., 2]) + 1e-4

dist = MultivariateNormal(mu, scale_tril=L)
loss = -dist.log_prob(y).mean()
```

PyTorch `MultivariateNormal` documentation

`scale_tril=L` passes the factor directly; PyTorch need not form or invert $\Sigma = LL^T$.

How gradients reach q

Autograd applies the chain rule. For the first diagonal entry,

$$\frac{\partial \ell}{\partial q_{11}} = \frac{\partial \ell}{\partial L_{11}} \operatorname{sigmoid}(q_{11}).$$

This uses $\operatorname{softplus}'(q) = \operatorname{sigmoid}(q)$; the same rule holds for q_{22}.

Since $L_{21} = q_{21}$, its gradient passes through unchanged.

Unlike ReLU, softplus has no exactly flat negative branch, although its slope can become very small for a very negative input.

One recipe: raw outputs in, valid parameters out

V VISUAL

The network may output any real numbers. A fixed transformation gives the model only legal parameters.

Model needs	Raw output	Transformation
Positive scale	$u \in \mathbb{R}$	$\sigma = \mathrm{softplus}(u) + \varepsilon$
Probability	$z \in \mathbb{R}$	$p = \mathrm{sigmoid}(z)$
Probability vector	$\boldsymbol{z} \in \mathbb{R}^K$	$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{z})$
PD covariance	raw lower-triangle entries	positive-diagonal L; $\Sigma = LL^T$

Main idea: let the network learn unrestricted coordinates, and enforce the model's domain in the transformation.