1 The Loss

1.1 In general

Mean square error (mean across the batch):

$$\mathrm{MSE} = \frac{1}{n}\sum\left(\mathrm{o}_i - \mathrm{ground\_truth}_i\right)^2$$

Negative log likelihood loss, negative log of probability of the data (when reduction = sum) according to the model ϕ:

$$\mathrm{NLLLoss} = -\log(P_\phi(\mathrm{ground\_truth}))$$

whe reduction = mean:

$$\mathrm{NLLLoss} = -\frac{1}{n}\log(P_\phi(\mathrm{ground\_truth}))$$

Softmax can turn vector of real numbers into probability distributuion:

$$\mathrm{SOFTMAX}(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{m}) = \left(\frac{e^{x_1}}{\sum_i e^{x_i}}, \ldots, \frac{e^{x_m}}{\sum_i e^{x_i}}\right)$$

but the default implementation may be numerically unstable. To alleviate this consider

$$z = \max_i x_i$$

$$\mathrm{SOFTMAX}(\mathrm{x}_1 - \mathrm{z}, \ldots, \mathrm{x}_\mathrm{m} - \mathrm{z}) =$$

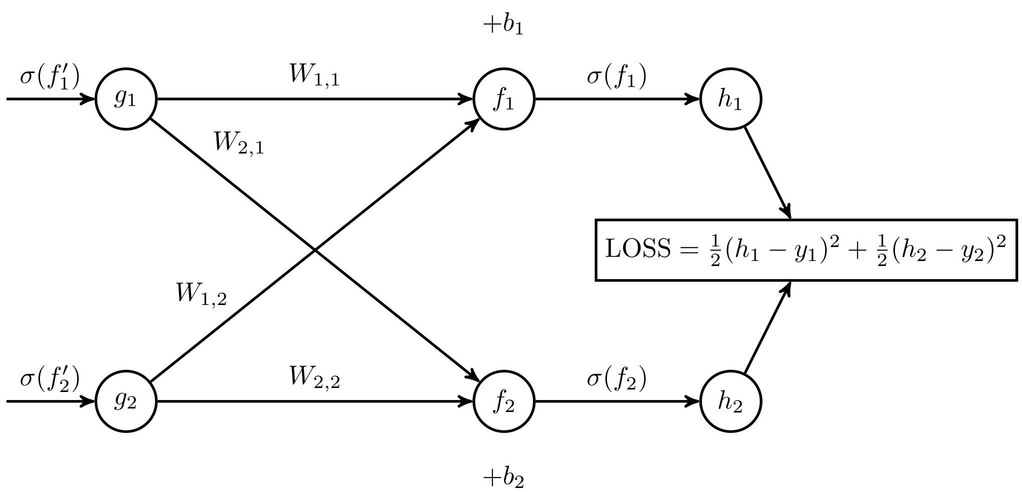
$$\left(\frac{e^{x_1 - z}}{\sum_i e^{x_i - z}}, \ldots, \frac{e^{x_m - z}}{\sum_i e^{x_i - z}}\right) =$$

$$\left(\frac{e^{-z}}{e^{-z}}\frac{e^{x_1}}{\sum_i e^{x_i}}, \ldots, \frac{e^{-z}}{e^{-z}}\frac{e^{x_m}}{\sum_i e^{x_i}}\right) =$$

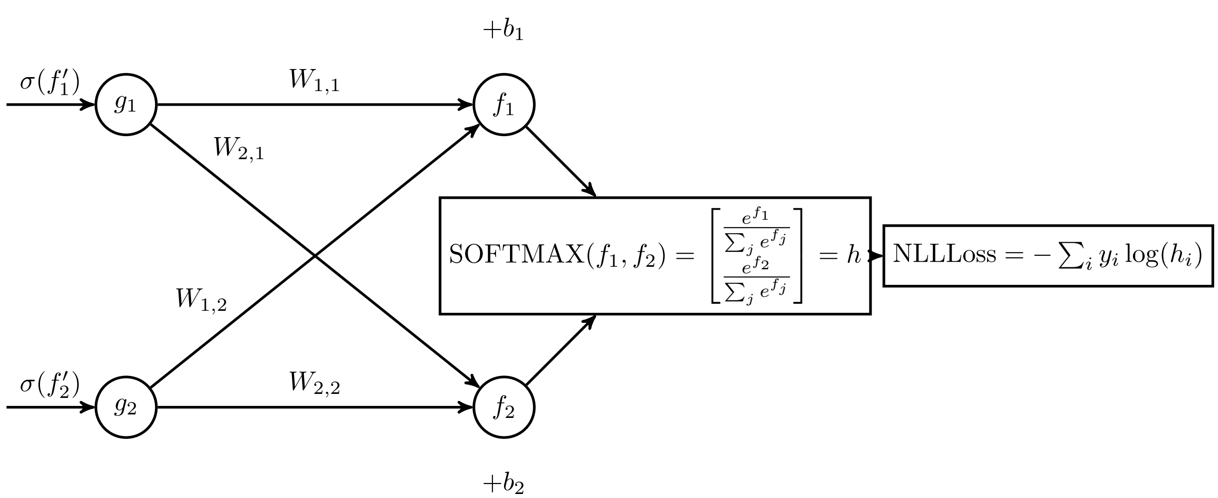
$$\mathrm{SOFTMAX}(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{m})$$

1.2 In our case

Simplified setup with MSE and batch size = 1. Old setup:



New setup with softmax:



Where $y_i \in \{0, 1\}$ one-hot encodes the desired output.

1.3 Gradient of NLLLoss with softmax

$$\text{NLLLoss} = -\sum_i y_i \log(h_i) = \sum_i y_i \log\left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)$$

$$\frac{\partial \text{NLLLoss}}{\partial f_k} = -\sum_i y_i \frac{\partial \left(\log\left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)\right)}{\partial f_k}$$

$$\frac{\partial \left(\log\left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)\right)}{\partial f_k} = \frac{1}{\frac{e^{f_i}}{\sum_j e^{f_j}}} \frac{\partial \left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)}{\partial f_k}$$

$$\frac{\partial \left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)}{\partial f_k} = \begin{cases} \frac{e^{f_i}}{\sum_j e^{f_j}} - \left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)^2 & \text{if } i = k \\ 0 - \frac{e^{f_i} e^{f_k}}{\left(\sum_j e^{f_j}\right)^2} & \text{otherwise} \end{cases}$$

$$\frac{1}{\frac{e^{f_i}}{\sum_j e^{f_j}}} \frac{\partial \left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right)}{\partial f_k} = \begin{cases} 1 - \left(\frac{e^{f_i}}{\sum_j e^{f_j}}\right) & \text{if } i = k \\ 0 - \frac{e^{f_k}}{\left(\sum_j e^{f_j}\right)} & \text{otherwise} \end{cases}$$

As y is a vector that one-hot encodes the answer so we can write:

$$h - y$$

2 Adam vs AdamW

Adam optimizer in Pytorch implements:

- Weight decay
  We want to penalize large weights. During the lecture we have observed that large values can be used to almost exactly compute functions from $\{0,1\}^n$ to $\{0,1\}$. The resulting intuition was that we want to penalize large weights so that it would be harder to overfit.
  Parameter: λ.
  If you are dealing with SGD just add $\frac{1}{2}\lambda\phi^2$ to the loss, or multiply the weights by $(1-\lambda)$.
- Momentum
  We want to speed up model on hills.
- Uses second moment to deal with noisy gradients

Now we will discuss the implementation in Adam/AdamW (Pytorch 2.1).:
Let M_ϕ - our model with parameters ϕ.

$$\text{grads} = \nabla_\phi(\text{LOSS}(M_\phi, x, y))$$
$$\text{grads} = \text{grads} + \cancel{\lambda\phi}$$

$$\text{momentum} = \beta_1 \text{momentum} + (1-\beta_1)\text{grads}$$
$$\text{var} = \beta_2 \text{var} + (1-\beta_2)\text{grads}^2$$

$$\text{m} = \frac{\text{momentum}}{1-\beta_1^{\text{step}}}$$
$$\text{v} = \frac{\text{var}}{1-\beta_2^{\text{step}}}$$

$$\phi = \phi - \text{lr}\left(\frac{\text{m}}{\sqrt{\text{v}}+\epsilon} + \underline{\lambda\phi}\right)$$

We usually do not decay biases!

3 Dropout

Briefly speaking drop parts of the activations randomly with probability p.
Note that we need to correct for this during the training (multiply by $(\frac{1}{1-p})$).

Example:

$$\begin{bmatrix} \sigma(f_1) \\ \sigma(f_2) \\ \sigma(f_3) \end{bmatrix} \rightarrow \begin{bmatrix} \frac{1}{1-p}\sigma(f_1) \\ 0 \\ \frac{1}{1-p}\sigma(f_3) \end{bmatrix}$$