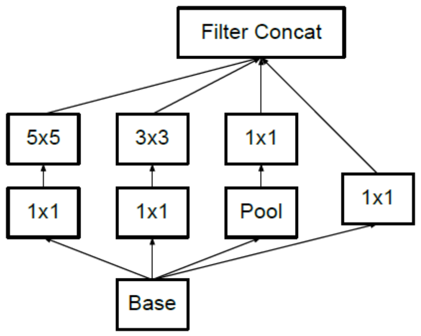
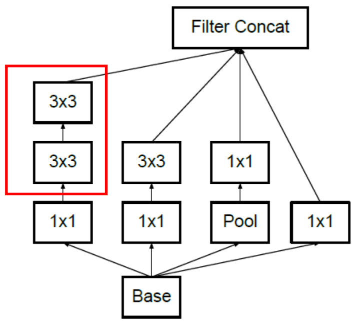
1 Modern CNN

You have been introduced to the original inception module.



1.1 (5 Points)

Show that the following module is more parameter efficient than the original one.



ANS:

- Looking at the modules we can realize that the difference is using two 3×3 conv layers instead of the 5×5 conv layer, as we have seen in the class n 3×3 conv layers have the same receptive field as a $(3 + 2n) \times (3 + 2n)$ conv layer so here by using two 3×3 conv layer our receptive field has no difference from a 5×5 conv layer but there is difference in the number of parameters! A 5×5 conv layer needs 25 parameters (not including bias terms) but two 3×3 conv layers need $2 \times 9 = 18$ parameters (not including bias terms) so by this we reduced number of parameters to $\frac{18}{25}$, it means 28% less parameters while getting the same result!

1.2 (5 Points)

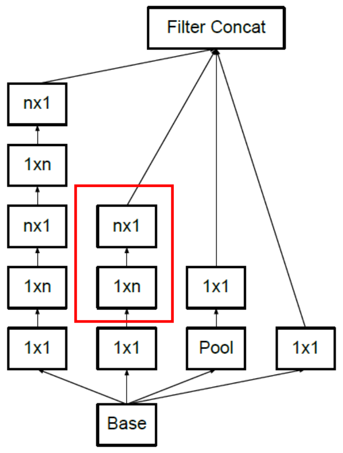
Show that you can save more computation by factorizing 3 × 3 convolutions into two 2 × 2 convolutions. (One branch is enough.)

ANS:

- If we assume that the spatial area and the depth of the output remains the same after factorization then for an imaginary input $x_i \times y_i \times d_i$, output size after propriate zero padding will be $x_o \times y_o \times d_o, : x_i = x_o, y_i = y_o, d_i = d_o$ and the number of computations will be:
  For 3×3 convolution layer (for k filter layers):
  $\#\text{c3} = 3 \times 3 \times x_o \times y_o \times d_o \times k$
  For two 2×2 convolution layer (for k filter layers):
  $\#\text{2c2} = 2(2 \times 2 \times x_o \times y_o \times d_o \times k)$
  So, number of computations is reduced to $\frac{\#2c2}{\#c3} = \frac{2(2\times2\times x_o\times y_o\times d_o\times k)}{3\times3\times x_o\times y_o\times d_o\times k} = \frac{8}{9}$, which means 11% less computations.

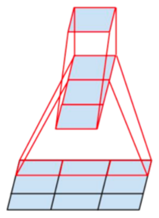
1.3 (10 Points)

Show that using asymmetric convolutions as follows saves computation. (One branch is enough.) How do you compare this enhancement with the previous one?



ANS:

- Let’s focus on the red box, the receptive field of the 2 layers is equal to the receptive field of one $n \times n$ convolutional layer (for better understanding look at image below),



so again, we saw that the receptive field is the same now let's compare number of computations:

For $n \times n$ convolution layer (for k filter layers):

$$\#\text{cnn} = n \times n \times x_o \times y_o \times d_o \times k$$

For $1 \times n$ *and* $n \times 1$ convolution layer (for k filter layers):

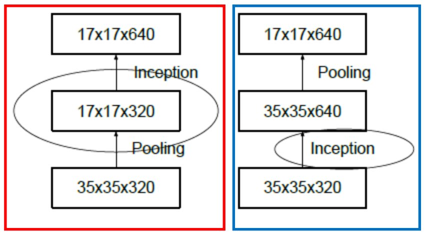
$$\#2\text{cn}1 = 1 \times n \times x_o \times y_o \times d_o \times k + n \times 1 \times x_o \times y_o \times d_o \times k = 2(n \times x_o \times y_o \times d_o \times k)$$

The number of computations is reduced to $\frac{\#2Cn1}{\#Cnn} = \frac{2(n\times x_o\times y_o\times d_o\times k)}{n\times n\times x_o\times y_o\times d_o\times k} = \frac{2}{n}$

- comparing this enhancement with the previous one:
  in Question 1.2 we saw that using two 2×2 conv layers instead of one 3×3 reduces the computational cost by 11% now let's see how much one 3×1 and one 1×3 conv layer helps us with the number of computations:
  $$\frac{\#2Cn1}{\#Cnn} = \frac{2(n \times x_o \times y_o \times d_o \times k)}{n \times n \times x_o \times y_o \times d_o \times k} = \frac{2}{n}$$
  $$n = 3: \frac{\#2C31}{\#C33} = \frac{2}{3}$$
  Number of computations is reduced to $\frac{2}{3}$ which means 33% less computations, comparing it with the previous enhancement we are saving more computations so it's a better approach!

1.4 (10 Points)

Consider the following methods for grid size reduction. Which one do you prefer in terms of computational cost? Which one creates a representational bottleneck?



ANS:

- considering computational cost, it's better to do the pooling first (red box) so the grid size will approximately decrease to $\frac{1}{4}$ (if the pooling is 2×2, which it usually is) and this will lead to decrease computational cost to $\frac{1}{4}$ as number of computational operations has direct relation with grid-size, but it's creating a **bottle neck representation**
- so, to avoid representational bottleneck it's better to use the inception module first but considering computation cost it's better to do the pooling first.

2 Autoencoders

1000 gene expression profile samples are given to us. These samples contain normal and cancer cases. The objective is to design a model which can distinguish cancer samples from normal ones. To do so, we have split the data into 800 train and 200 test samples. Each gene expression profile is a 20,000-dimension vector of numbers between 0 and 15. We have designed a 7-layer Autoencoder. The first and last layers have 2000 neurons, the second and 6th layers have 200 neurons and the fourth layer has only 20 neurons. Consider that all of the layers are fully-connected layers and the activation function for all of the neurons is sigmoid. The loss function has been chosen to be MSE.

2.1 (5 Points)

We have trained the network for 100 epochs but the loss function value does not decrease. What is the main reason?

ANS:

- there can be several reasons for that, one reason can be that the model is not complex enough to encode the data properly, or the optimizer is stuck in a local minima so it's not updating weights and the loss is not decreasing, which can be solved using SGD to avoid local minimas.

2.2 (10 Points)

After solving the problem, we again train the network for 100 epochs. We observe that MSE is less than 0.1 for test samples. Can we thereby conclude that the network is performing well? If not, what should we check?

ANS:

- No, we can't conclude that, it's better to also check other metrics such as accuracy, precision, recall, etc. another way to check is to visually compare the input and output.

2.3 (10 Points)

After the training process, we notice that with every small change in input values, a lot of neurons from the latent layer will change. What should we do in order to solve this issue? Provide a formula if possible.

ANS:

- we have to reduce the sensitivity of the Autoencoder, one way is to add regularization terms which result in weight decay, sparsity and also smaller derivatives, this regularization term is added to the former defined Loss function:

Loss function with regularization term: $\mathcal{L}(x, \hat{x}) + \Omega(x)$, $\mathcal{L}(x, \hat{x})$ can be MSE, MAE, Cross Entropy, … while $\Omega(x)$ can be L1, L2, KL-divergence, …

In order to solve the problem stated in the question we want the autoencoder to be robust to small changes in the input, so the L2 regularization term is added to Loss function:

Loss function with regularization term: $\mathcal{L}(x, \hat{x}) + \Omega(x)$, $\mathcal{L}(x, \hat{x})$ is MSE and $\Omega(x)$ is L2

$$\mathcal{L}(x,\hat{x}) + \Omega(x) = \frac{1}{2n}\sum(x_i - \hat{x}_i)^2 + \lambda\sum\|\nabla_x h_i\|^2$$

In which h_i represents the latent space, So its penalizing latent space rate of change with respect to the input.

2.4 (10 Points)

Finally, we succeed to train the network in a way that with low test error we can reconstruct input samples from the 20 latent neurons. What property of our input data has helped us in this success?

ANS:

- It happens when model can find a proper latent representation for the input data, meaning that the data was compressible.

4 Sequence-to-Sequence Models Comprehension

Answer the following questions

4.1 (15 Points)

How can vanishing and exploding gradients be controlled in RNN?

ANS:

- the vanishing and exploding gradient in RNN happen because we have to track the loss over the sequence and over the time, which means if the sequence is long or takes time to find the relations:

$$\frac{\partial L}{\partial W} \propto (\sum_{i=0}^{T} \left(\prod_{i=k=1}^{y} \frac{\partial h_i}{\partial h_{i-1}}\right)) \frac{\partial h_k}{\partial W}$$

One way to deal with this problem is not to back prop through the whole sequence and time, instead, do the forward and backward pass in a limited moving interval which is called "chunk", this method is known as "Truncated BPTT", the benefit of this method is that it considers only the near gradients so the $\prod_{i=k=1}^{y} \frac{\partial h_i}{\partial h_{i-1}}$ doesn't explode or vanish because it's multiplying less gradients.

https://www.codingninjas.com/codestudio/library/truncated-bptt

Some other approaches:

1- LSTM

4.2 (5 Points)

What are the advantages of bi-directional LSTM over simple LSTM?

ANS:

- In the simple LSTM the model only has information about the past, for example for predicting a word in middle of a sentence, LSTM can only predict using the words which came before the current word, but bi-directional LSTM can predict the word using its neighbors in both forward and backward direction, this feature helps bi-directional LSTM to predict words more precisely and as a result it has a better accuracy than the simple LSTM in different fields such as translation, handwritten recognition and etc.

4.3 (10 Points)

Why is encoder-decoder architecture used in Machine Translation? Explain problems of this architecture.

ANS:

- In machine translation tasks what matters is the meaning of the context, if the translation is done directly, it's a word by word translation which doesn't preserve the meaning, this problem can be solved using the encoder-decoder architecture, the encoder takes the input and compress it into the context vector, here the input is the word by word sentence but the vector contains it's meaning, now giving the vector to the decoder to turn the context vector to translated words so it's actually translating using the meaning!

in encoder-decoder the encoder compresses the input into a fixed length vector (context vector), now if the input size is very large comparing to the context vector it may result in loss of information, this problem can be solved using "Teacher Forcing" method.

4.4 (10 Points)

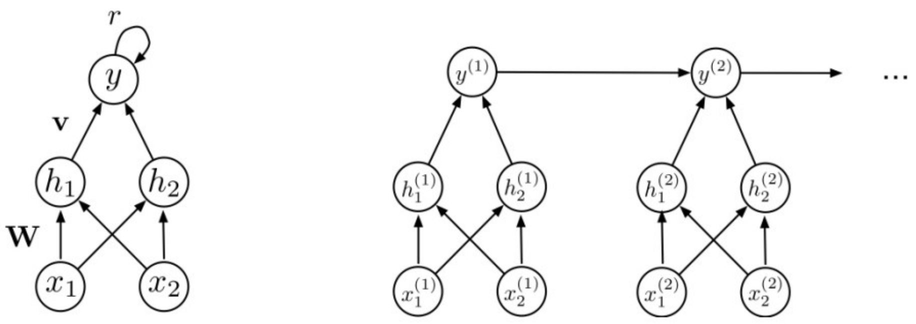
What is beam search algorithm? Explain this algorithm briefly.

ANS:

- Beam search is a searching algorithm like BFS, Greedy search, … but due to memory inefficiency of BFS and also inaccuracy of greedy search, beam search is mainly used in encoder-decoder modules, beam search just like BFS starts at the root and expands the successors, then sorts them (using max heap, time order: $O(\beta \log(\beta))$) by a heuristic defined formerly (for machine translation it can be probability of possible words), then in order to save time and memory it will just continue to expand first β(Beam Width) nodes (in greedy search the best node was expanded only which caused inaccuracy), this results in saving time and memory (no need to keep all nodes in the queue) but the algorithm may not find the optimal answer due to pruning in each step, so it's still more accurate than greedy search but not as accurate as BFS!

5 RNN Calculation 5.1 (20 points)

We want to process two binary input sequences with 0-1 entries and determine if they are equal. For notation, let $x_1 = x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(T)}$ be the first input sequence and $x_2 = x_2^{(1)}, x_2^{(2)}, \dots, x_2^{(T)}$ be the second. We Use RNN architecture shown in the Figure.



We have the following equations in which W is a 2 × 2 matrix, b is a two-dimensional bias vector, v is a two-dimensional weight vector and r, C, C0 are scalars.

$$h^{(t)} = g\left(Wx^{(t)} + b\right)$$

$$y^{(t)} = \begin{cases} g\left(v^T h^{(t)} + r y^{(t-1)} + C\right), & t > 1 \\ g\left(v^T h^{(t)} + C_0\right), & t = 1 \end{cases}$$

$$g(z) = \begin{cases} 1, & z > 1 \\ 0, & z \leq 1 \end{cases}$$

Find W, b, v, r, C, C0 such that at each step y$^{(t)}$ indicates whether all inputs have matched up to the current time or not.

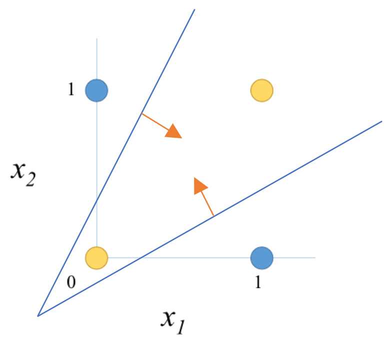
ANS:

- if T=1:

$x_1^{(1)}$	$x_2^{(1)}$	$y^{(1)}$
1	1	1
1	0	0
0	1	0
0	0	1

it's exactly the XNOR table so we need to implement XNOR, we know that XNOR is not a linearly separable function so its solved using MLP with one hidden layer just like our RNN structure:

separating class 1 and class 2 using 2 lines:



As XNOR gate is pretty famous we know that the following weights and biases work for the above classification:

$$W = \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}, b = \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}, v = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

But for the C_0 we have to find the boundaries:

$$y^{(1)} = g\left(v^T h^{(1)} + C_0\right), h^{(1)} = g\left(Wx^{(1)} + b\right)$$

In order to find the boundaries, we try all possible cases:

$$h^{(1)} = g\left(\begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix} x^{(1)} + \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}\right) = \begin{pmatrix} g(-x_1 - x_2 + \frac{3}{2}) \\ g(x_1 + x_2 - \frac{1}{2}) \end{pmatrix}$$

$$y^{(1)} = g\left(\begin{pmatrix} 1 \\ 1 \end{pmatrix}^T \begin{pmatrix} g\left(-x_1 - x_2 + \frac{3}{2}\right) \\ g\left(x_1 + x_2 - \frac{1}{2}\right) \end{pmatrix} + C_0\right) = g\left(g\left(-x_1 - x_2 + \frac{3}{2}\right) + g\left(x_1 + x_2 - \frac{1}{2}\right) + C_0\right)$$

Case x_1, x_2:

- 0,0: $y^{(1)} = g\left(g\left(-0 - 0 + \frac{3}{2}\right) + g\left(0 + 0 - \frac{1}{2}\right) + C_0\right) = g(1 + 0 + C_0) = 1 \rightarrow 1 + C_0 > 1$
- 1,1: $y^{(1)} = g\left(g\left(-1 - 1 + \frac{3}{2}\right) + g\left(1 + 1 - \frac{1}{2}\right) + C_0\right) = g(0 + 1 + C_0) = 1 \rightarrow 1 + C_0 > 1$
- 1,0: $y^{(1)} = g\left(g\left(-1 - 0 + \frac{3}{2}\right) + g\left(1 + 0 - \frac{1}{2}\right) + C_0\right) = g(0 + 0 + C_0) = 0 \rightarrow C_0 \leq 1$
- 0,1: $y^{(1)} = g\left(g\left(-0 - 1 + \frac{3}{2}\right) + g\left(0 + 1 - \frac{1}{2}\right) + C_0\right) = g(0 + 0 + C_0) = 0 \rightarrow C_0 \leq 1$

$$0 < C_0 \leq 1 \xrightarrow{we\ assume} C_0 = \frac{1}{2}$$

To find other variables (r, C) we have to solve the problem for T>1:

- If T>1:

$x_1^{(T)}$	$x_2^{(T)}$	$y^{(T-1)}$	$y^{(T)}$
1	1	1	1
1	0	1	0
0	1	1	0
0	0	1	1
1	1	0	0
1	0	0	0
0	1	0	0
0	0	0	0

$$y^{(T)} = y^{(T-1)} AND\ (x_1^{(T)}\ XNOR\ x_2^{(T)})$$

$$y^{(T)} = g\left(v^T h^{(T)} + r y^{(T-1)} + C\right) = g\left(g\left(-x_1^{(T)} - x_2^{(T)} + \frac{3}{2}\right) + g\left(x_1^{(T)} + x_2^{(T)} - \frac{1}{2}\right) + r y^{(T-1)} + C\right)$$

In the previous section we saw that if $x_1^{(T)} = x_2^{(T)}$ then $v^T h^{(T)} = 1$, so for the ease of computation the cases will be like:

Case $x_1^{(T)} = x_2^{(T)}, y^{(T-1)}$:

- True, 1: $y^{(T)} = g(1 + r \times 1 + C) = 1 \rightarrow 1 + r + C > 1$
- True, 0: $y^{(T)} = g(1 + r \times 0 + C) = 0 \rightarrow 1 + C \leq 1$
- False, 1: $y^{(T)} = g(0 + r \times 1 + C) = 0 \rightarrow r + C \leq 1$
- False, 0: $y^{(T)} = g(0 + r \times 0 + C) = 0 \rightarrow C \leq 1$

$$C \leq 0\ \&\ 0 < r + C \leq 1 \xrightarrow{we\ assume} C = -\frac{1}{2}, r = 1$$

The final parameters for our RNN are:

$$\boldsymbol{W} = \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} \frac{3}{2} \\ -\frac{1}{2} \end{pmatrix}, \quad \boldsymbol{v} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \boldsymbol{C_0} = \frac{1}{2}, \quad \boldsymbol{C} = -\frac{1}{2}, \quad \boldsymbol{r = 1}$$

6 Word Embedding

6.1 (20 points)

By using the following figure (next page), explain skip-gram. Your explanation must include the probability the model wants to estimate, the concept of the context window, the output of the model, and the loss function used in the training.

ANS:

- skip-gram is a prediction-based approach, instead of counting words which usually come together it predicts possible neighbors(outputs) of a given center word(input), in order to predict the neighbors skip-gram maximizes the following probability: $P_1 = p(W_1,\ W_2, \dots,\ W_C|\ W_{center};\theta)$, $W_1,\ W_2, \dots,\ W_C$ are the context words and θ is the concatenated weight matrixes W and W'.

- Concept of Context Window: in skip-gram model we are trying to maximize P_1 but if the given corpus contains lots of words (L words) it would be so hard to calculate P_1 and it would be unnecessary too, because words which are far away from the center word doesn't relate to it as much the near words do, so in order to reduce the complexity of P_1 we use the concept 'Context-window' which determines the number of neighbors of the center word we are going to include in $P_1$and it usually ranges from 1 to 10.

- Looking at the figure (next page) the output of the model is C (number of neighbors we want to predict, defined by Context Window) one-hot encoded vectors (y_i) which determine $W_1,\ W_2, \dots,\ W_C$.

- The loss function:

In skip-gram we are trying to maximize P_1 but in order to make it a minimization problem we can minimize $-P_1$:

$$P_1 = p(W_1,\quad W_2, \dots,\quad W_C|\ W_{center};\theta),$$

$$goal: argmax\ p(W_1,\quad W_2, \dots,\quad W_C|\ W_{center};\theta),$$

$$argmax\ F(x) = argmax \ln(F(x)) \rightarrow goal: argmax \ln\ (p(W_1,\quad W_2, \dots,\quad W_C|\ W_{center};\theta)),$$

$$= argmax\big(\ln\big(MLE(p(W_1,\quad W_2, \dots,\quad W_C|\ W_{center};\theta))\big)\big) = \ln\Big(\prod p(w_i|w_{center};\theta\Big)$$

$$= \sum \ln\ (p(w_i|w_{center};\theta) = \sum \ln\big(softmax(z_i)\big) = \sum \ln\left(\frac{e^{z_i}}{\sum e^{z_j}}\right) = \sum\Big(\ln(e^{z_i}) - \ln\Big(\sum e^{z_j}\Big)\Big) =$$

$$\sum(z_i - \ln\Big(\sum e^{z_j}\Big)),$$

$$z_i = W_i'^T(W^T x),$$

$$\sum W_i'^T(W^T x) - \ln\ (\sum e^{W_i'^T(W^T x)}),$$

$$loss = -goal = [-\sum W_i'^T(W^T x)] + [C \ln\Big(\sum e^{W_i'^T(W^T x)}\Big)]$$

6.2 (20 points)

Suppose that the context window (C) is equal to 1. (For example, for each center word, we just consider its left neighbor) We also denote the loss function as

$$L(x, \hat{y}, W, W^{'})$$

In which $\hat{y}$ is the one-hot encoded vector of the word in the context window and x is the one-hot encoded vector of the center word.

Using the previous part, calculate $\frac{\partial L}{\partial w_k}$ in which w_k is the kth row of matrix W. For simplification purposes, you can assume that $\frac{\partial Softmax(y)}{\partial y}$ is given as $S'(y)$. However, we highly recommend a complete calculation.

ANS:

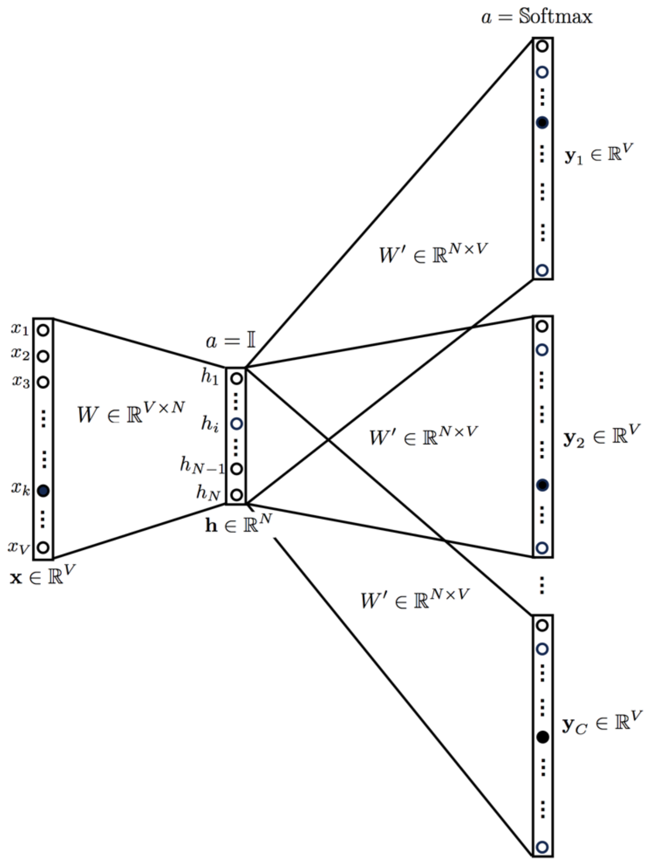
- in order to calculate L:

$$W^T x = h, \qquad W'^T h = z \rightarrow loss = \left[-\sum z_i\right] + \left[C \ln\left(\sum e^{z_i}\right)\right],$$

$$\frac{\partial loss}{\partial w_{k,j}} = \frac{\partial loss}{\partial z_i} \times \frac{\partial z_i}{\partial w_{k,j}} \rightarrow loss = -\sum \ln\left(softmax(z_i)\right) \rightarrow \frac{\partial loss}{\partial z_i} = \frac{\partial - \sum \ln\left(softmax(z_i)\right)}{\partial z_i}$$
$$= \frac{-1}{\sum softmax(z_i)} \times S'(z_i),$$

$$\frac{\partial z_i}{\partial w_{k,j}} = \frac{\partial W'^T W^T x}{\partial w_{k,j}} = \sum W'_{j,k}\, x_k,$$

$$\frac{\partial loss}{\partial w_{k,j}} = \frac{\partial loss}{\partial z_i} \times \frac{\partial z_i}{\partial w_{k,j}} \rightarrow \frac{1}{-\sum softmax(z_i)} \times S'(z_i) \times \sum W'_{j,k}\, x_k$$


$a = \mathbb{S}\text{oftmax}$
$\mathbf{y}_1 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$a = \mathbb{I}$
x_1
x_2
x_3
x_k
x_V
$W \in \mathbb{R}^{V \times N}$
h_1
h_i
h_{N-1}
h_N
$\mathbf{h} \in \mathbb{R}^N$
$\mathbf{x} \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_2 \in \mathbb{R}^V$
$W' \in \mathbb{R}^{N \times V}$
$\mathbf{y}_C \in \mathbb{R}^V$