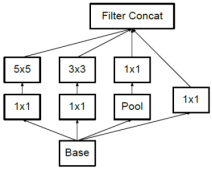
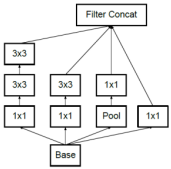
1 Modern CNN

You have been introduced to the original inception module.

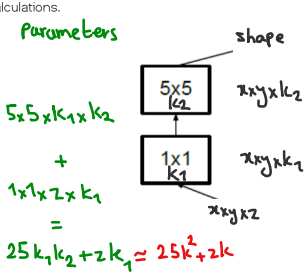


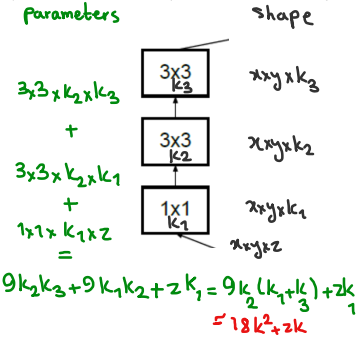
1.1 (5 Points)

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



The only difference between this model and the original one is in the left branch of architecture. So we should just compare these models by the parameter counts in the left branch. We assume that that the base is in shape x*y*z and we use k_i filters at each convolutional layer. For the simplicity we don't consider bias parameter in the following calculations.



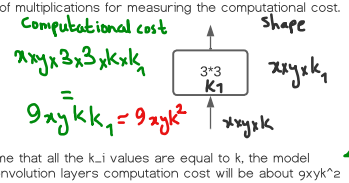


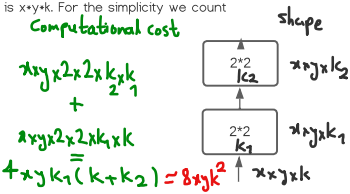
If we assume that all the k_i values are the same, the original model has 25k^2 + zk parameters and this model has 18k^2 + zk parameters so this model is more parameter efficient.

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.)

we assume that the shape of the input to the 3*3 convolution layer is x*y*k. For the simplicity we count the number of multiplications for measuring the computational cost.

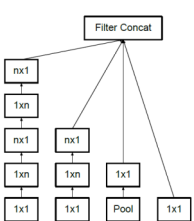




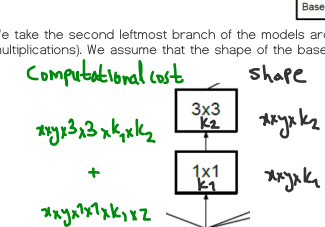
If we assume that all the k_i values are equal to k, the model with 3*3 convolution layers computation cost will be about 9xyk^2 whereas the model with of 3*3 convolutions replaced by two 2*2 convolutions computation cost will be about 8xyk^2. So we can save more computation by factorizing 3*3 convolution layers into two 2*2 convolutions.

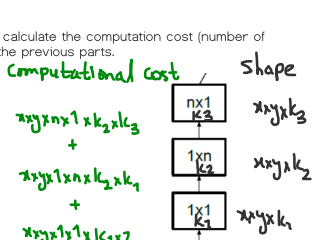
1.3 (10 Points)

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



We take the second leftmost branch of the models architecture and we calculate the computation cost (number of multiplications). We assume that the shape of the base is x*y*z just like the previous parts.

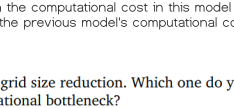




As it can be seen same as the previous parts if we assume that all the k_i values are equal to k we can easily understand that this asymetric convolution inception model saves more computation in comparison with the original model.
There is much more enhancement in the computational cost in this model than there is in the previous one because its computational cost is 6xyk^2 where the previous model's computational cost was 8xyk^2.

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?



The left method seems to save more computational cost because the inception layers first two dimensions are 17*17 and so when computing the cost we would have the summation of some terms of 17*17. For example in the previous question you can see that there is this xy term in the computational cost of the models. However int the right method first two dimensions of the input layers, is 34*34 and in compare to the left method this could lead to much more computational cost.
Also it can be derived from the figure that the max pooling layer uses a 3*3 filter with stride 2 so the computational cost of the max pooling layer in the left method is 3*3*17*17*320 and in the right method is 3*3*17*17*640. Maxpooling layer seems to have less computational cost in the right method but the since the inception cost would possibly be much higher in the right method we choose the left method as the one with less computational cost.

Also the left method creates a representational bottleneck because it reduces the dimensions in the middle layer and some how loses some data but by using inception in the next layer it increases the dimensions. But in the right method the dimensions are first increased and then decreased, so the left method seems to be a better representation of bottleneck.

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)

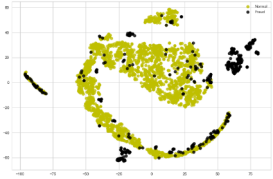
We have trained the network for 100 epochs but the loss function value does not decrease. What is the main reason?

One first simple reason for why the loss function doesn't decrease is that maybe the learning rate is too high and the loss function gets stuck in a loop or even gets too far from local minima.
Another reason could be because of bad initialization of the weights which led the loss function to get stuck in a loop or even increase.

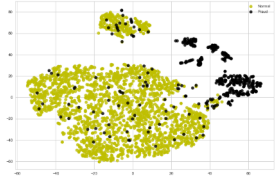
2.2 (10 Points)

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?

This 0.1 MSE loss is the loss of the encoder in the ability of encoding the inputs into the latent space so then it could be decoded to its original input. What we wanted to do was to catogrize the samples into cancer or normal using this encoder. The benefit of using such an auto encoder is that we can use the latent space and possibly classify the samples by using a simpler model like logistic regression or support vector classifier on the latent space. So in order to evaluate the models performance we should first encode all the test samples using the autoencoder and then use for example logistic regression and evaluate the accuracy of it's classification. In another word, we should check that if the encoded cancer and normal samples are easily seperable or not. For a better insight, you can see the plots of tsne dimension reduction of the transaction data in the task of finding fraud from transactions before and after using the autoencoder. This example is inspired from ml-classifying-data-using-an-auto-encoder.



Before encoding, the data is not easily seperable



After using an autoencoder and encoding the samples into latent space the fraud samples are even linearly seperable.

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.

By using a varational auto encoder we can solve the problem. In a varational auto encoder the data is encoded to a distribution on latent space and it prevents the model to be sensitive to the changing the input. The loss of a VAE is as follows:

$$Loss = \|x - \hat{x}\|^2 + KL\left[N(\mu_x, \sigma_x), N(0,1)\right]$$

A criterion for the difference between the two distributions

In varational auto encoder, the near points in latent space tend to have alike outputs and the outputs are meaningful.

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?

One important property of gene expression data in predicting a disease is that this data space has a low dimensional support. Meaning that the actual meaningful part of this data can be represented in a few dimensions or in another words in a few genes. In fact cancer samples could be detected by just observing the expression of some important genes and so this autoencoder could perfectly classify the samples using the latent space.

3 Generative adversarial networks (Optional)

In this question, we will workout backpropagation with Generative Adversarial Networks (GANs). Recall that a GAN consists of a Generator and a Discriminator playing a game. The Generator takes as input a random sample from some noise distribution (e.g., Gaussian), and its goal is to produce something from a target distribution (which we observe via samples from this distribution). The Discriminator takes as input a batch consisting of a mix of samples from the true dataset and the Generator's output, and its goal is to correctly classify whether its input comes from the true dataset or the Generator.
Definitions:

- $X^1, ..., X^n$ is a minibatch of n samples from the target data generating distribution For this question, we suppose that each X^i is a k dimensional vector. For example, we, we might be interested in generating a synthetic dataset of customer feature vectors in a credit scoring application
- $Z^1, ..., Z^n$ is a minibatch of n samples from some predetermined noise distribution Note that in general these minibatch sizes may be different.
  - The generator $g(.;\theta_g) : Z \to X$ is a neural network
  - The discriminator $d(.;\theta_d) : X \to (0, 1)$ is a neural network

The log likelihood of the output produced by the discriminator is:

$$L(\theta_d, \theta_g) = \sum_{i=1}^{n} \log d(X^i; \theta_d) + \log(1 - D(G(Z^i))) \tag{1}$$

The training of such a GAN system proceeds as follows; given the generator's parameters, the discriminator is optimized to maximize the above likelihood. Then, given the discriminator's parameters, the generator is optimized to minimize the above likelihood. This process is iteratively repeated. Once training completes, we only require the generator to generate samples from our distribution of interest; we sample a point from our noise distribution and map it to a sample using our generator.
The Discriminator (The generator architecture is defined analogously)

- Consider the discriminator to be a network with layers indexed by $1, 2, .., L_d$ for a total of L_d layers.
- Let the discriminator's weight matrix for layer l be W_d^l; lets assume there are no biases for simplicity
- Let the activations produced by a layer l be given by A_d^l, and the pre activation values by Z_d^l
- Let $g_d^l(.)$ be the activation function at layer l

The goal of the discriminator is to maximize the above likelihood function.

3.1 (10 Points)

Write down $\nabla_{\theta_d} L(X; \theta_d, \theta_g)$ in terms of $\nabla_{\theta_d} D(.)$

$$\nabla_{\theta_d} L(X; \theta_d, \theta_g) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta_d} \left(\log(D(X^i, \theta_d)) + \nabla_{\theta_d} \log(1 - D(G(z^i))) \right)$$

According to chain rule

$$= \frac{1}{n} \sum_{i=1}^{n} \frac{\nabla_{\theta_d} D(X^i, \theta_d)}{D(X^i, \theta_d)} - \frac{\nabla_{\theta_d} D(G(z^i))}{1 - D(G(z^i))}$$

3.2 (15 Points)

Write down

$$\partial L(\theta_d, \theta_g)/\partial z_d^{L_d} \tag{2}$$

taking help from your answer in the previous subpart. Remember that the activation function in the last layer of the discriminator is a sigmoid function as the output of the discriminator is a probability.

First note that $\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \times (1 - \sigma(x))$

Using the previous part's answer we write the $\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}}$:

$$\frac{\partial L(\theta_d, \theta_g)}{\partial z_d^{L_d}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\sigma(z_d^{L_d}(X_i)) * (1 - \sigma(z_d^{L_d}(X_i)))}{D(X^i)} -$$

$$\frac{\sigma(z_d^{L_d}(G(z_i))) * (1 - \sigma(z_d^{L_d}(G(z_i))))}{1 - D(G(z^i))}$$

3.3 (10 Points)

What is mode collapse in GANs? How can it be avoided?

As we know GANs consist of two parts, discriminator and generator. For a GAN to be trained successfully there should be two conditions holding:

1. The generator should reliably generate data and fool the discriminator.
2. The generator should generate data samples that are diverse as the distribution of the real word data when dealing with multi modal data.

Model collapse happens when the generator fails to achieve the second goal and for example all of the generated samples are very similar or even identical. In this case the generator may win by creating one realistic data sample that always fools the discriminator, but the second goal doesn't hold any more and this would be a problem especially in a multi modal data.

There are three solutions to solve this problem:

1. Using Wasserstein loss: Formulates the GAN loss functions to more directly represent minimizing the distance between two probability distributions.
2. Unrolling: Updating the generator's loss function to backpropagate through k steps of gradient updates for the discriminator. This lets the generator see k steps into the future which hopefully encourages the generator to produce more diverse and realistic samples.
3. Packing: Modifying the discriminator to make decisions based on multiple samples all of the same class (either real or artificial). When the discriminator looks at a pack of samples at once, it has a better chance of identifying an un-diverse pack as artificial.

4 Sequence-to-Sequence Models Comprehension

Answer the following questions

4.1 (15 Points)

How can vanishing and exploding gradients be controlled in RNN?

The exploding problem could be solved by gradient clipping because in this approach the gradients are ensured to have their maximum norm bound by a predefined hyperparameter and they are scaled if their norm is too big.

But for vanishing gradients issue in RNN the suggested solution is to change the structure of RNN and use LSTMs instead because in LSTM there is an uniterrupted gradient flow in each cell architecture similar to ResNet.

4.2 (5 Points)

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

Their advantage is that the information about future is considered when predicting each y. bi-directional LSTM is useful for modeling dependencies between words and phrases in both directions and each cell has information about both past and present.

4.3 (10 Points)

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

The reason that we use encoder-decoder is that by using this architecture we can save the input sentence into an encoded vector that preserves the meaning of it using encoder and then we build the output translated sentence using decoder. The problem with normal RNN architecture is that in those kinds of models for the task of translation the translation would be performed word by word without caring about the global meaning of the sentence. Encoder-decoder architecture has some problems too. One of its most important problems is that when the length of the input sequence gets large the model captures the essential information roughly. To solve this problem an approach called teacher forcing is used. Another problem of encoder-decoder architecture is that at the early stages of training the predictions of the model are very bad.

4.4 (10 Points)

What is beam search algorithm? Explain this algorithm briefly.

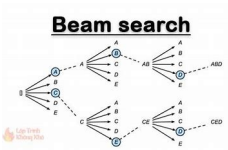
Beam search is a type of search algorithm we use for finding the inference of y_t.

$$\max P(y_t, y_{t-1}, \cdots, y_1 \mid input) = \max \prod_{i=1}^{t} P(y_i \mid y_{i-1}, \cdots, y_1, input)$$

We would like the argmax of the written probability to find the combination of ys which has the largest probability of happening.
The usual approach to find this probability is to use greedy search and at each step select the word with the highest probability.

$$\max \prod_{i=1}^{t} P(y_i \mid y_{i-1}, \cdots, y_1, input) \simeq \prod_{i=1}^{t} \max P(y_i \mid y_{i-1}, \cdots, y_1, input)$$

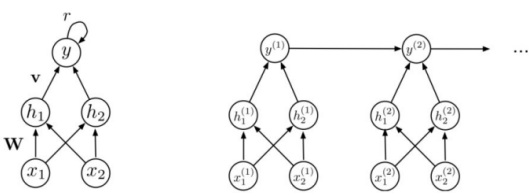
But this search algorithm doesn't find the best combination neccesarily. Another search algorithm is Brute force search that computes the probability for all the available combinations which demands a lot of computation. However there is an approach which is something between the brute force and greedy search algorithm and finds the combination better than greedy approach and demands less computation than brute force. In this algorithm we keep a fixed number(k) of candidates having the most probability instead of one in greedy search and we try to expand these k candidates into their y at the next t and again we just select the best k candidates of the resultant candidates. The following figure shows this algorithm steps for k=2.



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 $x_1 = x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(T)}$ be the first input sequence and $x_2 = x_2^{(1)}, x_2^{(2)}, \ldots, x_2^{(T)}$ be the second. We use the 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, C_0 are scalars.

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

$$y^{(t)} = \begin{cases} g(v^T h^{(t)} + ry^{(t-1)} + C) & \text{if t} > 1 \\ g(v^T h^{(t)} + C_0)) & \text{if t} = 1 \end{cases}$$

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

Find W, b, v, r, C, C_0 such that at each step $y^{(t)}$ indicates whether all inputs have matched up to the current time or not.

We start with $t=1$: $y^{(1)} = g(v^T h^{(1)} + C_0) = g(v^T g(Wx^{(1)} + b) + C_0)$

$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$ $\quad W = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix}$

$\Rightarrow v^T g(b) + C_0 > 1$

Lets guess g in each case

(*) $g(b) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow v_1 + v_2 + C_0 > 1$

$v^T g(\begin{bmatrix} w_2 \\ w_4 \end{bmatrix} + b) + C_0 \leq 1 \quad g(\begin{bmatrix} w_2 \\ w_4 \end{bmatrix} + b) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \Rightarrow v_1 + C_0 \leq 1$

$v^T g(\begin{bmatrix} w_1 \\ w_3 \end{bmatrix} + b) + C_0 \leq 1 \quad g(\begin{bmatrix} w_1 \\ w_3 \end{bmatrix} + b) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \Rightarrow v_2 + C_0 \leq 1$

$v^T g(\begin{bmatrix} w_1 + w_2 \\ w_3 + w_4 \end{bmatrix} + b) + C_0 > 1 \quad g(\begin{bmatrix} w_1 + w_2 \\ w_3 + w_4 \end{bmatrix} + b) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \Rightarrow v_1 + v_2 + C_0 \geq 1$

input	$y^{(1)}$	desired output
$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	$g(v^T g(b) + C_0)$	1
$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$	$g(v^T g(\begin{bmatrix} w_2 \\ w_4 \end{bmatrix} + b) + C_0)$	0
$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$g(v^T g(\begin{bmatrix} w_1 \\ w_3 \end{bmatrix} + b) + C_0)$	0
$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$g(v^T g(\begin{bmatrix} w_1 + w_2 \\ w_3 + w_4 \end{bmatrix} + b) + C_0)$	1

$\Rightarrow 1 + C_0 \leq v_1 + v_2 + 2C_0 \leq 2 \Rightarrow C_0 < 1$

lets assume $C_0 = 0$ $\Rightarrow$ $v_1, v_2 \leq 1$, $v_1 + v_2 > 1$

guess v_1, v_2 $\Rightarrow$ $v_1 = v_2 = 1$

(*) in order to the guesses we made for g to be true we should have:

$b_1, b_2 > 1 \quad b_1 + w_2 > 1 \quad b_2 + w_4 \leq 1 \quad b_1 + w_1 \leq 1 \quad b_2 + w_3 > 1 \quad w_1 + w_2 + b_1 > 1 \quad w_3 + w_4 + b_2 > 1$

again we make some raw guesses for b W based on the above inequalities,

$b = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad W = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$

Now we check the RNN for $t > 1$

$y^t = g(v^T g(Wx^t + b) + ry^{t-1} + C)$

So we have $C_0 = 0$, $b = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$, $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $W = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$

if $y^{t-1} = 0$ we would like $y^t = 0$ even if $x_1^t = x_2^t$

So if $y^{t-1} = 0 \Rightarrow y^t = g(v^T g(Wx^t + b) + C)$ (this could be maximumly 2)

$\Rightarrow C = -1$

if $y^{t-1} = 1$ then it is similar to $t=0$ where $C_0 = 0$

$\Rightarrow ry^{t-1} + C = 0 \Rightarrow r - 1 = 0 \Rightarrow r = 1$

Answer: $C_0 = 0$, $C = -1$, $r = 1$

$W = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$, $b = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$, $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

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.

Skip-gram is a nlp task in which given a center word we would like to find its context words or in another words, we would like the most probably surrounding words of this center word. As it can be understood from the figure on next page, this task is done in 6 steps:
input: center word x
desired output: a list of words (context words) surrounding this center word with the highest probability (y1, ..., yc)
Here we imagine that the context window's length is m and we are considering 2m surrounding words of the center word.(Context window is the number of words to be predicted which can occur in the range of the given word)
W: |V|*N

W': N*|V|

1- We generate the one hot input vector x
2- We encode this vector using W matrix into a n dimensional vector by computing h=W^T . x
3- Since there is just one embedded input vector we don't need to average different vectors as opposed to CBOW.
4- We then Generate 2m score vectors, u_{c-m}, . . . , u_{c-1}, u_{c+1}, . . . , u_{c+m} using u=W'^T . h
5- Now, to find the probability of each vector we'll use the softmax function
6- The word with the highest probability is the result and if the predicted word for a given context position is wrong then we'll use backpropagation to modify our weight vectors W and W'

Probability the model wants to estimate: $P(w_{c-m}, w_{c-m+1}, \dots, w_{c+m-1}, w_{c+m} \mid w_c)$

$$\text{Loss} = -\log P(w_{c-m}, \dots, w_{c+m} \mid w_c) = -\log\left(\prod_{\substack{j=0 \\ j\neq m}}^{2m} P(w_{c-m+j} \mid w_c)\right) = -\log\left(\prod_{\substack{j=0 \\ j\neq m}}^{2m} P(u_{c-m+j} \mid h)\right)$$

$$= -\log \prod_{\substack{j=0 \\ j\neq m}}^{2m} \frac{\exp(u_{c-m+j}^T h)}{\sum_{k=1}^{|V|} \exp(u_k^T h)} = -\sum_{\substack{j=0 \\ j\neq m}}^{2m} u_{c-m+j}^T h + 2m \log\left(\sum_{k=1}^{|V|} \exp(u_k^T h)\right)$$

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.

$$L(x, \hat{y}, W, W') = -\log \prod_{\substack{j=0 \\ j\neq m}}^{2m} P(w_{c-m+j} \mid w_c) = -\log \prod_{\substack{j=0 \\ j\neq m}}^{2m} P(u_{c-m+j} \mid h) = -\log \prod_{j=0}^{2m} \frac{\exp(\hat{y}_j^T w'^T h)}{\sum_{k=0}^{|V|} \exp(w_k'^T h)}$$

w_j': means the jth column of W'
w_i: means the ith row of W

$2m=1 \Rightarrow$ Loss $= -\hat{y}^T w'^T h + \log\left(\sum_{k=0}^{|V|} \exp(w_k'^T h)\right) \Rightarrow L(x, \hat{y}, W, W') = -\hat{y}^T w'^T w^T x + \log\left(\sum_{k=0}^{|V|} \exp(w_k'^T w^T x)\right)$

$h = W^T x$

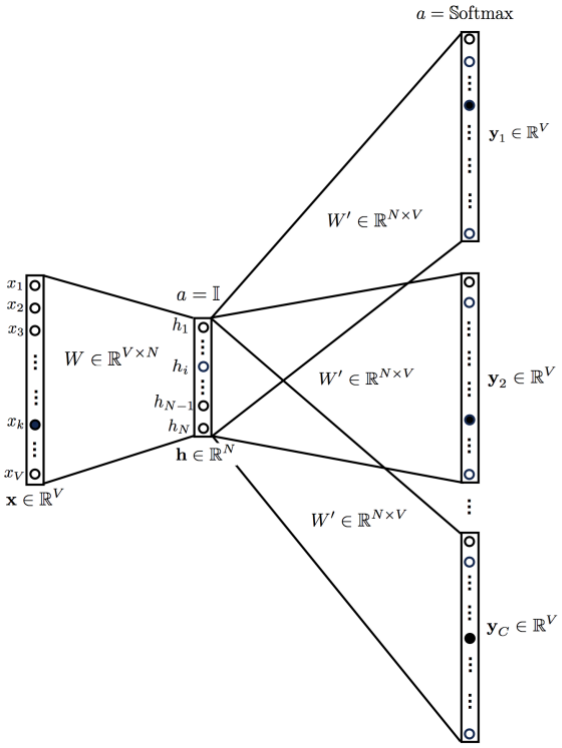
$$\hat{y} = \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_{|V|} \end{bmatrix} \quad W'^T = \begin{bmatrix} w_1'^T \\ w_2'^T \\ \vdots \\ w_{|V|}'^T \end{bmatrix} \quad W^T = \begin{bmatrix} | & | & & | \\ w_1 & w_2 & \cdots & w_{|V|} \\ | & | & & | \end{bmatrix}, x = \begin{bmatrix} x_1 \\ \vdots \\ x_{|V|} \end{bmatrix} \Rightarrow W^T x = x_1 w_1 + \cdots + x_{|V|} w_{|V|} \Rightarrow W'^T W^T x = \begin{bmatrix} w_1'^T(x_1 w_1 + \cdots + x_{|V|} w_{|V|}) \\ \vdots \\ w_{|V|}'^T(x_1 w_1 + \cdots + x_{|V|} w_{|V|}) \end{bmatrix}$$

$$\Rightarrow \hat{y}^T W'^T W^T x = \hat{y}_1 w_1'^T(x_1 w_1 + \cdots + x_{|V|} w_{|V|}) + \cdots + \hat{y}_{|V|} w_{|V|}'^T(x_1 w_1 + \cdots + x_{|V|} w_{|V|}) \Rightarrow \frac{\partial \hat{y}^T W'^T W^T x}{\partial w_{ik}} = -x_k(\hat{y}_1 w_1' + \cdots \hat{y}_{|V|} w_{|V|}') = -x_k w_i' \hat{y}$$

$$\frac{\partial \log\left(\sum_{i=0}^{|V|} \exp(w_i'^T w^T x)\right)}{\partial w_{ik}} = \frac{1}{\sum_{i=0}^{|V|} \exp(w_i'^T w^T x)} \times \sum_{i=0}^{|V|} \exp(w_i'^T w^T x) x_k w_i'$$

$$\Rightarrow \frac{\partial L}{\partial w_{ik}} = -x_k w_i' \hat{y} + x_k \sum_{i=0}^{|V|} \frac{\exp(w_i'^T w^T x) w_i'}{\sum_{i=0}^{|V|} \exp(w_i'^T w^T x)}$$

where w_i' is the ith column of W'
Note that the $\frac{\partial L}{\partial w_{ik}}$ we computed is now a $\frac{\partial L}{\partial w_k}$ column vector so we can transpose it to be in the same dimension as w_k


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