# Il Porcodio: Deep Learning Cheatsheet

Kotatsu, the Bringer of Jewishness

## 1 Matrix based notation

The activation $z_j^l$ of the $j$-th neuron of the $l$-th layer is

$$z_j^l = \sigma\left(\sum_k w_{jk}^l z_k^{l-1} + b_j^l\right)$$

Now take $\mathbf{W}^l$ as the matrix

$$\begin{bmatrix} w_{00}^l & w_{01}^l & w_{02}^l & \cdots \\ w_{10}^l & w_{11}^l & w_{12}^l & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}.$$

in matrix notation we write then

$$\mathbf{z}^l = \sigma(\mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}^l)$$

and we define

$$\mathbf{a}^l := \mathbf{W}^l \mathbf{z}^{l-1} + \mathbf{b}$$

so that $\mathbf{z}^l = \sigma(\mathbf{a}^\mathbf{l})$.
Hadamard product: stupid retarded product of matrices:

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \times 2 \\ 3 \times 4 \end{bmatrix}.$$

## 2 Cost function

It must be:

- Expressed as mean of the single inputs;
- It must be a function of the outputs of the network.

Example: quadratic cost function

$$C = \frac{1}{2n} \left\| \mathbf{y}(x) - \mathbf{z}^L(x) \right\|^2.$$

## 3 The Four Fundamental Equations

Define $\delta_j^l$ as the error at level $l$ of neuron $j$:

$$\delta_j^l = \frac{\partial C}{\partial a_j^l}.$$

### 3.1 BP1

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} \cdot \sigma'(a_j^L)$$
$$\Downarrow$$
$$\boldsymbol{\delta}^L = \nabla_z C \odot \sigma'(\mathbf{a}^L).$$

### 3.2 BP2

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(a_j^l)$$
$$\Downarrow$$
$$\boldsymbol{\delta}^l = ((\mathbf{W}^{l+1})^\intercal \boldsymbol{\delta}^{l+1}) \odot \sigma'(\mathbf{a}^l).$$

### 3.3 BP3

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

### 3.4 BP4

$$\frac{\partial C}{\partial w_{jk}^l} = z_k^{l-1} \delta_j^l \qquad \frac{\partial C}{\partial w} = z_{\text{in}} \delta_{\text{out}}.$$

**Proof 3.1: BP1**

Show that $\delta_j^L := \frac{\partial C}{\partial a_j^L} = \frac{\partial C}{\partial z_j^L} \sigma'(a_j^L)$. Use the chain rule:

$$\begin{aligned} \delta_j^L &= \frac{\partial C}{\partial a_j^L} \\ &= \sum_k \frac{\partial C}{\partial z_k^L} \frac{\partial z_k^L}{\partial a_j^L} \\ &= \frac{\partial C}{\partial z_j^L} \frac{\partial \overbrace{z_j^L}^{\sigma(a_j^L)}}{\partial a_j^L} \\ &= \frac{\partial C}{\partial z_j^L} \sigma'(a_j^L). \end{aligned}$$

**Proof 3.2: BP2**

Here we must show that

$$\begin{aligned} \delta_j^l := \frac{\partial C}{\partial a_j^l} &= \left[ (\mathbf{W}^{l+1})^\intercal \boldsymbol{\delta}^{l+1} \odot \sigma'(\mathbf{a}^l) \right]_j \\ &= \sum_k w_{kj}^{l+1} \delta_k^{l+1} \cdot \sigma(a_j^l). \end{aligned}$$

Start from the fact that we can think of $C$ as a function of $a_k^{l+1}$ so we can use the chain rule:

$$\begin{aligned} \delta_j^l := \frac{\partial C}{\partial a_j^l} &= \sum_k \frac{\partial C}{\underbrace{\partial a_k^{l+1}}_{\text{by def. } \delta_k^{l+1}}} \frac{\partial a_k^{l+1}}{a_j^l} \\ &= \sum_k \delta_k^{l+1} \frac{\partial a_k^{l+1}}{a_j^l}. \end{aligned}$$

But we know that

$$\begin{aligned} a_k^{l+1} &= \sum_j w_{kj}^{l+1} z_j^l + b_k^{l+1} \\ &= \sum_j w_{kj}^{l+1} \sigma(a_j^l) + b_k^{l+1} \end{aligned}$$

so we have that

$$\frac{\partial a_k^{l+1}}{\partial a_j^l} = w_{kj}^{l+1} \sigma'(a_j^l).$$

So putting all together we get

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(a_j^l).$$

**Proof 3.3:** BP3

We must show that $\frac{\partial C}{\partial b_j^l} = d_j^l$. Think of $C$ as a function of $a_j^l$ and use chain rule:

$$\begin{aligned}\frac{\partial C}{\partial b_j^l} &= \sum_k \frac{\partial C}{\partial a_k^l}\frac{\partial a_k^l}{b_j^l} \\ &= \underbrace{\frac{\partial C}{\partial a_j^l}}_{=\delta_j^l}\underbrace{\frac{\partial a_j^l}{b_j}}_{=1}.\end{aligned}$$

**Proof 3.4:** BP4

We must show that $\frac{\partial C}{\partial w_{jk}^l} = z_k^{l-1}\delta_j^l$. Use the chain rule:

$$\begin{aligned}\frac{\partial C}{\partial w_{jk}^l} &= \sum_i \frac{\partial C}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_{jk}^l} \\ &= \frac{\partial C}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{jk}^l} \\ &= \delta_j^l \frac{\partial a_j^l}{\partial w_{jk}^l}\end{aligned}$$

but we know that

$$\begin{aligned}\frac{\partial a_j^l}{\partial w_{jk}^l} &= \frac{\partial}{\partial w_{jk}^l}\left(\sum_k w_{jk}^l z_k^{l-1} + b_j^l\right) \\ &= z_k^{l-1}.\end{aligned}$$

So

$$\frac{\partial C}{\partial w_{jk}^l} = z_k^{l-1}\delta_j^l.$$

## 4 Improving learning

Cross-entropy cost function:

$$C = -\frac{1}{n}\sum_x \left[y \ln z + (1-y)\ln(1-z)\right].$$

This yields:

$$\begin{aligned}\frac{\partial C}{\partial w_j} &= \frac{1}{n}\sum_x x_j(\sigma(a) - y) \\ \frac{\partial C}{\partial b} &= \frac{1}{n}\sum_x (\sigma(a) - y).\end{aligned}$$

We can generalize for multi-layer networks:

$$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln z_j^L + (1-y_j)\ln(1-z_j^L)\right].$$

Soft max activation with log-likelihood cost function:

$$\begin{aligned}z_j^L &= \frac{e^{a_j^L}}{\sum_k e^{a_k^L}} \\ C &= -\ln z_y^L\end{aligned}$$

## 5 Convergence

Consider the quadratic approximation of the error function around the minimum point $\mathbf{w}^\star$:

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \nabla(w)E(\mathbf{w}^\star)^\intercal(\mathbf{w}-\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^\intercal \mathbf{H}(\mathbf{w}-\mathbf{w}^\star)$$

but since $\nabla E = 0$ we get

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^\intercal \mathbf{H}(\mathbf{w}-\mathbf{w}^\star).$$

Since $\{\mathbf{u}_i\}_i$ is a orthonormal basis we can write any vector as a linear combination of $\mathbf{u}_i$ vectors, which allows us to write:

$$E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2.$$

**Proof 5.1: Taylor's shit**

We need to show that $E(\mathbf{w}) = E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i\alpha_i^2$. We know that:

$$\begin{aligned}E(\mathbf{w}) &= E(\mathbf{w}^\star) + \frac{1}{2}(\mathbf{w}-\mathbf{w}^\star)^\intercal \mathbf{H}(\mathbf{w}-\mathbf{w}^\star) \\ &= E(\mathbf{w}^\star) + \frac{1}{2}\left(\sum_i \alpha_i \mathbf{u}_i\right)^\intercal \mathbf{H}\left(\sum_i \alpha_i \mathbf{u}_i\right) \\ &= E(\mathbf{w}^\star) + \frac{1}{2}\left(\sum_i \alpha_i \mathbf{u}_i\right)^\intercal \left(\sum_i \alpha_i \mathbf{H}\mathbf{u}_i\right) \\ &= E(\mathbf{w}^\star) + \frac{1}{2}\left(\sum_i \alpha_i \mathbf{u}_i\right)^\intercal \left(\sum_i \alpha_i \lambda_i \mathbf{u}_i\right) \\ &= E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i\alpha_i^2.\end{aligned}$$

And this implies:

**Proof 5.2: Gradient's shit**

Show that $\nabla E = \sum_i \alpha_i\lambda_i\mathbf{u}_i$.

$$\begin{aligned}\nabla E(\mathbf{w}) &= \nabla\left(E(\mathbf{w}^\star) + \frac{1}{2}\sum_i \lambda_i\alpha_i^2\right) \\ &= \frac{1}{2}\sum_i \lambda_i 2\alpha_i \nabla\alpha_i.\end{aligned}$$

To compute $\nabla\alpha_i$ we use the fact that $\mathbf{w}-\mathbf{w}^\star = \sum_j \alpha_j\mathbf{u}_j$:

$$\begin{aligned}\mathbf{u}_i^\intercal(\mathbf{w}-\mathbf{w}^\star) &= \mathbf{u}_i^\intercal\left(\sum_j \alpha_j\mathbf{u}_j\right) \\ \mathbf{u}_i^\intercal(\mathbf{w}-\mathbf{w}^\star) &= \alpha_i \\ \sum_j w_j u_{i_j} - \sum_j w_j^\star u_{i_j} &= \alpha_i\end{aligned}$$

so

$$\begin{aligned}\frac{\partial}{\partial w_k}\left(\sum_j w_j u_{i_j} - \sum_j w_j^\star u_{i_j}\right) &= u_{i_k} \\ &= \frac{\partial \alpha_i}{\partial w_k} \implies \nabla\alpha_i = \mathbf{u}_i.\end{aligned}$$

We have that

- $\delta\mathbf{w} = \sum_i \delta\alpha_i\mathbf{u}_i$
- $\nabla E = \sum_i \alpha_i\lambda_i\mathbf{u}_i$

- $\Delta \mathbf{u} = -\eta \nabla E$
- $\Delta \alpha = -\eta \lambda_i \alpha_i \implies \alpha_i^{\text{new}} = (1-\eta\lambda_i)\alpha_i^{\text{old}}$.

This means that if $|1-\eta\lambda_i| < 1$ then $\alpha_i$ decreases for each of the $T$ steps.

- Fastest convergence: $\eta = \dfrac{1}{\lambda_{\max}}$ (convergence rate 0, minimum reached in one step).
- Direction of slowest convergence: $\lambda_{\min}$ where the rate is $1 - \dfrac{\lambda_{\min}}{\lambda_{\max}}$.
- Condition number of hessian matrix: $\dfrac{\lambda_{\max}}{\lambda_{\min}}$. The bigger it is, the slower the convergence.

# 6 Momentum

Normally we have

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} + \Delta \mathbf{w}^{(\tau-1)}$$

but adding the momentum we get

$$\Delta \mathbf{w}^{(\tau-1)} = -\eta \nabla E(\mathbf{w}^{(\tau-1)}) + \mu \Delta \mathbf{w}^{(\tau-2)}$$

so

$$\Delta \mathbf{w} = -\frac{\eta}{1-\mu} \nabla E.$$

# 7 Learning rate scheduling

- Linear: $\eta^{(\tau)} = (1 - \frac{\tau}{K})\eta_0 + (\frac{\tau}{K}\eta_K)$.
- Power law: $\eta^{(\tau)} = \eta_0 (1 + \frac{\tau}{s})^c$.
- Exponential decay: $\eta^{(\tau)} = \eta_0 c^{\frac{\tau}{s}}$

# 8 Normalization

## 8.1 Data normalization

$$\tilde{x}_{ni} = \frac{x_{ni} - \mu_i}{\sigma_i}$$

for each dimension $i$.

## 8.2 Batch normalization

$$\mu_i = \frac{1}{K}\sum_{n=1}^{K} a_{ni}$$
$$\sigma_i^2 = \frac{1}{K}\sum_{n=1}^{K} (a_{ni} - \mu_i)^2$$
$$\hat{a}_{ni} = \frac{a_{ni} - \mu_i}{\sqrt{\sigma_i^2 + \delta}}$$

After training we use a moving average of the mean and variance.

$$\begin{aligned} \overline{\mu}_i^{(\tau)} &= \alpha \overline{\mu}_i^{(\tau-1)} + (1-\alpha)\mu_i \\ \overline{\sigma}_i^{(\tau)} &= \alpha \overline{\sigma}_i^{(\tau-1)} + (1-\alpha)\sigma_i \end{aligned} \qquad 0 \leqslant \alpha \leqslant 1$$

# 9 CNNs

The cross-entropy between teo discrete distributions $p$ and $q$ measures how much $q$ differs from $p$.

$$H(p,q) = -\sum_v p(v) \cdot \log(q(v)).$$

CNNs employ the cross-entropy loss:

$$-\sum_{i=1}^{S} y_i \cdot \log(p_i).$$

# 10 Autoencoders

Remember how PCA works:

$$f(\mathbf{x}) = \arg\min_{\mathbf{h}} ||\mathbf{x} - g(\mathbf{h})||_2$$

Where $g(\mathbf{h}) = \mathbf{D}\mathbf{h}$. So we are interested in measuring the loss of the reconstruction

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))).$$

## 10.1 Sparse autoencoders

Here the loss function has a penalty on $\mathbf{h}$:

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h}).$$

Consider the distribution

$$p_{\text{model}}(\mathbf{h}, \mathbf{x}) = p_{\text{model}}(\mathbf{h}) p_{\text{model}}(\mathbf{x}|\mathbf{h})$$

and marginalizing

$$p_{\text{model}}(\mathbf{x}) = \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x})$$
$$\Downarrow$$
$$\log p_{\text{model}}(\mathbf{x}) = \log \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x})$$

So, given a $\widetilde{\mathbf{h}}$ generated by the encoder we have

$$\begin{aligned} \log p_{\text{model}}(\mathbf{x}) &= \log \sum_{\mathbf{h}} p_{\text{model}}(\mathbf{h}, \mathbf{x}) \\ &\approx \log p(\widetilde{\mathbf{h}}, \mathbf{x}) = \log p_{\text{model}}(\widetilde{\mathbf{h}}) + \log p_{\text{model}}(\mathbf{x}|\widetilde{\mathbf{h}}). \end{aligned}$$

If we set $\Omega(\mathbf{h}) = \lambda \sum_i |h_i|$ ($L^1$ norm of $\mathbf{h}$) then minimizing the sparsity terms is equal to maximizing the log likelihood of $p(\mathbf{h})$ assuming a Laplace prior over each component independently.

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$
$$\Downarrow$$
$$-\log p_{\text{model}}(\mathbf{h}) = \sum_i \left( \lambda |h_u| - \log \frac{1}{2} \right) = \Omega(\mathbf{h}) + \text{const}$$

## 10.2 Denoising autoencoders

They minimize

$$\mathcal{L}(\mathbf{x}, g(f(\widetilde{\mathbf{x}})))$$

## 10.3 Contractive autoencoders

They minimize

$$\mathcal{L}(\mathbf{x}, g(f(\mathbf{x}))) + \Omega(\mathbf{h})$$

with

$$\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i ||\nabla_{\mathbf{x}} h_i||^2.$$

## 11 Transformers

Consider the attention to embedding $\mathbf{y}_n$ as

$$a_{nm} = \frac{\exp(\mathbf{x}_n^\mathsf{T}\mathbf{x}_m)}{\sum_{m'=1}^{N}\exp(\mathbf{x}_n^\mathsf{T}\mathbf{x}_{m'})}.$$

Therefore we can express our new embeddings $\mathbf{Y}$ as

$$\begin{aligned}\mathbf{Y} &= \mathrm{SoftMax}\left[\mathbf{X}\mathbf{X}^\mathsf{T}\right]\mathbf{X}\\ &= \mathrm{SoftMax}\left[\mathbf{Q}\mathbf{K}^\mathsf{T}\right]\mathbf{V}.\end{aligned}$$

Where queries, keys and values are trainable.

$$\begin{aligned}\mathbf{Q} &= \mathbf{X}\mathbf{W}^{(q)}\\ \mathbf{K} &= \mathbf{X}\mathbf{W}^{(k)}\\ \mathbf{V} &= \mathbf{X}\mathbf{W}^{(v)}.\end{aligned}$$

Then the embeddings get scaled by the dimensionality of key vectors

$$\mathbf{Y} = \mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{SoftMax}\left[\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{D_k}}\right]\mathbf{V}.$$

In a multi-head scenario where $\mathbf{H}_h = \mathrm{Attention}(\mathbf{Q}_h,\mathbf{K}_h,\mathbf{V}_h)$ we have

$$\overbrace{\mathbf{Y}(\mathbf{X})}^{N\times D} = \overbrace{\mathrm{Concat}[\mathbf{H}_1,\dots,\mathbf{H}_H]}^{N\times HD_v}\overbrace{\mathbf{W}^{(o)}}^{HD_v\times D}.$$

To improve learning it is possible to add a residual connection

$$\mathbf{Z} = \mathrm{LayerNorm}[\mathbf{Y}(\mathbf{X})+\mathbf{X}] \qquad \mathbf{Z} = [\mathbf{Y}(\mathrm{LayerNorm}(\mathbf{X}))+\mathbf{X}]$$

and then passing through a MLP with ReLU activation

$$\widetilde{\mathbf{X}} = \mathrm{LayerNorm}[\mathrm{MLP}(\mathbf{Z})+\mathbf{Z}] \qquad \widetilde{\mathbf{X}} = \mathrm{MLP}[\mathrm{LayerNorm}(\mathbf{Z})]+\mathbf{Z}.$$

### 11.1 Positional encoding

We concatenate input $\mathbf{x}$ to positional encoding $\mathbf{r}$ obtaining the representation $\mathbf{x}|\mathbf{r}$. We can apply a linear transformation $\mathbf{w}_\mathbf{x}|\mathbf{w}_\mathbf{r}$:

$$\begin{bmatrix}\mathbf{w}_\mathbf{x} & \mathbf{w}_\mathbf{r}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{r}\end{bmatrix} = \mathbf{w}_\mathbf{x}\mathbf{x}+\mathbf{w}_\mathbf{r}\mathbf{r} = \mathbf{w}(\mathbf{x}+\mathbf{r}).$$

Encoding must be:

- unique for each position;
- bounded;
- generalizable to sequences of arbitrary length;
- capable of expressing relative positions.

Sinusoidal positional encoding:

$$\mathbf{r}_n = \begin{bmatrix}\sin(w_1\cdot n)\\ \cos(w_1\cdot n)\\ \sin(w_2\cdot n)\\ \cos(w_2\cdot n)\\ \vdots\\ \sin(w_{\frac{D}{2}}\cdot n)\\ \cos(w_{\frac{D}{2}}\cdot n)\end{bmatrix}, \qquad w_i = \frac{1}{10000^{\frac{2i}{D}}}.$$

This is good because

$$\mathbf{r}_n^\mathsf{T}\mathbf{r}_m = \sum_{i=1}^{\frac{D}{2}}\cos(w_i\cdot(n-m)).$$

The encoding of $n+m$ can always be expressed as a linear combination of the encodings of $n$ and $m$ and it is always possible to find a matrix $\mathbf{M}$ that depends only on $k$ such that $\mathbf{r}_{n+k} = \mathbf{M}\mathbf{r}_n$.

$$\begin{bmatrix}v_1 & v_2\\ v_3 & v_4\end{bmatrix}\cdot\begin{bmatrix}\sin(w_i\cdot n)\\ \cos(w_i\cdot n)\end{bmatrix} = \begin{bmatrix}\sin(w_i\cdot(n+k))\\ \cos(w_i\cdot(n+k)).\end{bmatrix}$$

**Proof 11.1: Matrix shit**

We have

$$\begin{bmatrix}v_1 & v_2\\ v_3 & v_4\end{bmatrix}\cdot\begin{bmatrix}\sin(w_i\cdot n)\\ \cos(w_i\cdot n)\end{bmatrix} = \begin{bmatrix}v_1\sin(w_i\cdot n) & v_2\cos(w_i\cdot n)\\ v_3\sin(w_i\cdot n) & v_4\cos(w_i\cdot n)\end{bmatrix}$$

but

$$\begin{bmatrix}\sin(w_i\cdot(n+k))\\ \cos(w_i\cdot(n+k))\end{bmatrix} = \begin{bmatrix}\sin(w_i\cdot n)\cos(w_i\cdot k)+\cos(w_i\cdot n)\sin(w_i\cdot k)\\ \cos(w_i\cdot n)\cos(w_i\cdot k)-\sin(w_i\cdot n)\sin(w_i\cdot k)\end{bmatrix}$$

so

$$\begin{aligned}&\begin{bmatrix}v_1\sin(w_i\cdot n)+v_2\cos(w_i\cdot n)\\ v_3\sin(w_i\cdot n)+v_4\cos(w_i\cdot n)\end{bmatrix}\\ =&\begin{bmatrix}\sin(w_i\cdot n)\cos(w_i\cdot k)+\cos(w_i\cdot n)\sin(w_i\cdot k)\\ \cos(w_i\cdot n)\cos(w_i\cdot k)-\sin(w_i\cdot n)\sin(w_i\cdot k)\end{bmatrix}\end{aligned}$$

and this means

$$\begin{bmatrix}v_1 & v_2\\ v_3 & v_4\end{bmatrix} = \begin{bmatrix}\cos(w_i\cdot k) & \sin(w_i\cdot k)\\ -\sin(w_i\cdot k) & \cos(w_i\cdot k)\end{bmatrix}$$

### 11.2 GPTs

The goal is to use transformers to build an autoregressive model of the form

$$p(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n|\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_{n-1}).$$

Here the attention weights are computed using $\mathbf{Q}\mathbf{K}^\mathsf{T}$ as before, but we can set the attention weights to zero for all future tokens and computing $(\mathbf{Q}\mathbf{K})^\mathsf{T}_{nm}$ as the attention weights between tokens $n$ and $m$ multiplied by a mask matrix $\mathbf{M}$ that has $-\infty$ in the upper triangular part.

$$\mathbf{Y} = \mathrm{SoftMax}\left[\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{D_k}}\circ\mathbf{M}\right]\mathbf{V}.$$

Temperature scaling:

$$y_i = \frac{\exp\left(\frac{a_i}{T}\right)}{\sum_j\exp\left(\frac{a_j}{T}\right)}$$