EN.601.482 : Machine Learning: Deep Learning

Notebook

James Guo

Spring 2025

Contents

Acknowledgements:

- This notebook records the course contents for `EN.601.482` Machine Learning: Deep Learninginstructed by *Dr. Eric Nalisnick* at *Johns Hopkins University* in the Spring 2025 semester.
- The notes is a summary of the lectures, and it might contain minor typos or errors. Please point out any notable error(s).

I Supervised Learning with Univariate Linear Models

I.1 Predictive Modeling

The **goal** is to have predictive modeling.
The **features** are $x \in \mathcal{X}$, which are the inputs while $\mathcal{X}$ is the space of all possible inputs.
The **labels** are $y \in \mathcal{Y}$ are the dependent variable, which is a response to the input feature.

We want to generate a function $f : \mathcal{X} \to \mathcal{Y}$ that maps x to y. This is called a **deep neural network**.

A question to propose is how do we know if the above f is a good fitting?

- We think about the **data generating process**. Our features are generated as $x \sim \mathbb{P}(\mathcal{X})$ and labels as $y \sim \mathbb{P}(\mathcal{Y} \mid \mathcal{X})$.
- However, these labels are **unknown**. We want our **predictive model** to match these distributions, *i.e.*, $f(x) \sim \mathbb{P}(y \mid x)$.
- *Note:* Here, $\mathbb{P}$ is the distribution we don't know but exists. The label could be discrete categories or real numbers.

In our structure, we will have:

- **Training data**, $\mathcal{D} := \{(x_n, y_n)\}_{n=1}^{N}$, where $x_n \sim \mathbb{P}(\mathcal{X})$ and $y_n \sim \mathbb{P}(\mathcal{Y} \mid x_n)$, *i.e.*, the feature data is consisted of N pairs.

Here, we think about $x, y \in \mathbb{R}$, we can define a **univariate linear regression model** by $f(x; w) = w \cdot x$.

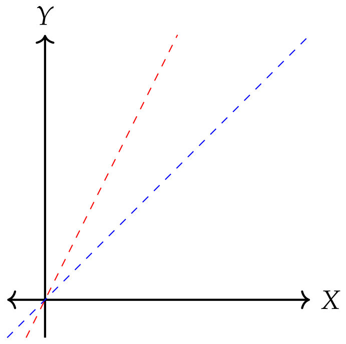


Figure I.1. Linear Model of $L(\mathbb{R}, \mathbb{R})$.

Here, w is determined by the data.

To evaluate, we shall think about the loss function.

- A predominant loss function is the **squared loss** function, that is:

$$\ell(w; \mathcal{D}) = \frac{1}{N} \sum_{n=1}^{N} \ell(w; x_n, y_n) = \frac{1}{N} \sum_{n=1}^{N} \left(f(x_n; w) - y_n\right)^2.$$

  - *Note:* This loss function assumes that we are trying to minimize the deviations without consideration of the set.

To **optimize a loss function**, we want to find w^* as:

$$w^* = \arg\min_w \ell(w;\mathcal{D}) = \arg\min_w = \frac{1}{N}\sum_{n=1}^{N}(\underbrace{w \cdot x}_{f(x_n;w)} - y_n)^2.$$

To minimize, the minimum is when the derivative is zero.

1. Take the derivatives:
$$\begin{aligned}\frac{d}{dw}\ell(w;\mathcal{D}) &= \frac{d}{dw}\left[\frac{1}{N}\sum_{n=1}^{N}(w \cdot x_n - y_n)^2\right] \\ &= \frac{2}{N}\left[\left(\sum_{n=1}^{N} w x_n^2\right) - \left(\sum_{n=1}^{N} y_n x_n\right)\right]\end{aligned}$$
2. Then, we set the derivative to 0 and solve for w, that is:
$$\begin{aligned}0 &= \frac{d}{dw}\ell(w;\mathcal{D}) = \frac{2}{N}\left[\left(\sum_{n=1}^{N} w x_n^2\right) - \left(\sum_{n=1}^{N} y_n x_n\right)\right], \\ w^* &= \frac{\sum_{n=1}^{N} y_n \cdot x_n}{\sum_{n=1}^{N} x_n^2}.\end{aligned}$$

Now, we have trained our first model.

Consider the **vectorized version**, we shall utilize the Graphics processing unit (GPU), that is more efficient over linear algebra, were we consider:

$$\begin{aligned}\mathrm{x} &= \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^\intercal, \\ \mathrm{y} &= \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix}^\intercal,\end{aligned}$$

so that we have:

$$w^* = \frac{\langle \mathrm{x}, \mathrm{y}\rangle}{\|\mathrm{x}\|^2} = \texttt{np.dot(x,y) / np.linalg.norm(x) ** 2}.$$

I.2 Maximum Likelihood Estimation

Statistical Divergences is like a loss function but applied to probability distributions. One of the most common one is the Kullback-Leibler divergence (KLD):

$$\mathbb{KLD}[p(z) \;||\; q(z)] := \mathbb{E}_{p(z)}\left[\log\frac{p(z)}{q(z)}\right] = \int_z p(z)\left(\log\frac{p(z)}{q(z)}\right) dz \text{ where } z \text{ is the random variable in interest.}$$

- *Note*: $\mathbb{KLD}[p(z) \;||\; q(z)]$ is not necessarily equal to $\mathbb{KLD}[q(z) \;||\; p(z)]$, it is **not** commutative.

Here, we consider the predictive models as distributions, that is:

$$\mathbb{P}\big(y;\theta = f(x)\big).$$

- *E.g.*, we consider the normal distribution $\mathcal{N}(y;\mu,\sigma^2)$, what we are claiming is that for each y, our prediction falls as a normal distribution around it.

Then, we think about the **maximum likelihood estimation** (MLE), that is:

$$\begin{aligned}\mathbb{KLD}\big[\mathbb{P}(y \mid x) \mid\mid \mathbb{P}(y,\theta = f(z))\big] &= \mathbb{E}_{\mathbb{P}(y|x)}\left[\log \frac{\mathbb{P}(y \mid x)}{\mathbb{P}\big(y;\theta = f(x)\big)}\right] \\ &= \underbrace{\mathbb{E}_{\mathbb{P}(y|x)}\left[\mathbb{P}(y \mid x)\right]}_{-\mathbb{H}[\mathbb{P}(y|x)]} - \mathbb{E}_{\mathbb{P}(y|x)}\left[\mathbb{P}\big(y;\theta = f(x)\big)\right] \\ &= \mathbb{E}_{\mathbb{P}(y|x)}\left[-\log \mathbb{P}\big(y;\theta = f(x)\big)\right] - \mathbb{H}[\mathbb{P}(y \mid x)].\end{aligned}$$

Here, we have $\mathbb{H}$ as the entropy of the true distribution.

Here, we consider the **optimization** as:

$$w^* = \arg\min_w \mathbb{E}_{\mathbb{P}(x)}\mathbb{E}_{\mathbb{P}(y|x)}\left[-\log \mathbb{P}\big(y;\theta = f(x)\big)\right] - \underbrace{\mathbb{H}[\mathbb{P}(y \mid x)]}_{\text{constant}}.$$

Again, the expectations are still not in closed form, so we need to consult with the **Monte Carlo Approximation**, that is:

$$\mathbb{E}_{\mathbb{P}(x)}[\phi(x)] \approx \frac{1}{S}\sum_{s=1}^{S}\phi(x_s), \text{ where } x_s \sim \mathbb{P}(x) \text{ where } S \text{ is the number of samples.}$$

Then, we can consider the final form of the approximation as:

$$\begin{aligned}w^* &= \arg\min_w \mathbb{E}_{\mathbb{P}(x)}\mathbb{E}_{\mathbb{P}(y|x)}\left[-\log \mathbb{P}\big(y;\theta = f(x)\big)\right] - \underbrace{\mathbb{H}[\mathbb{P}(y \mid x)]}_{\text{constant}} \\ &\approx \arg\min \frac{1}{N}\sum_{n=1}^{N} -\log \mathbb{P}\big(y_n; f(x_n;w)\big).\end{aligned}$$

Now, the obstacle is that the model would not be good when N is small, *i.e.*, with limited amount of data, we cannot have a perfect approximation.

In such case, we may derive the squared error as maximum likelihood:

$$\begin{aligned}w^* &= \arg\min \mathbb{E}_{\mathbb{P}(x)}\mathbb{KLD}[\mathbb{P}(y \mid x) \mid\mid p(y; f(x;w))] \\ &= \arg\min_w \mathbb{E}_{\mathbb{P}(x)}\mathbb{E}_{\mathbb{P}(y|x)}[-\log p(y; f(x;w))] \\ &\approx \frac{1}{N}\sum_{n=1}^{N} -\log p(y_n, f(x_n;w)). \qquad \text{(eq1)}\end{aligned}$$

Here, we assume that this is a normal distribution with mean being the function, so:

$$\begin{aligned}
\text{eq1} &= \arg\min \frac{1}{N}\sum_{n=1}^{N} -\log \mathcal{N}(y_n; \mu = f(x_n; w), \sigma^2) \\
&= \arg\min \frac{1}{N}\sum_{n=1}^{N} -\log\left[\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\frac{(y_n - f(x_n; w))^2}{\sigma^2}\right)\right] \\
&= \arg\min \frac{1}{N}\sum_{n=1}^{N} \log(\sigma\sqrt{2\pi}) + \frac{1}{2}\frac{(y_n - f(x_n; w))^2}{\sigma^2} \\
&= \arg\min \frac{1}{N}\frac{1}{2\sigma^2}\sum_{n=1}^{N} \big(y_n - f(x_n; w)\big)^2.
\end{aligned}$$

I.3 Generalized Linear Model

Then, we shall think about the generalized linear models.

- Check the support of $y \in Y$, for example, is it real, binary, counts, or etc.
- Choose the distribution $p(y; \theta)$ that agrees with the support.
- Parametrize our model, with $\mathbb{E}_p[y \mid x] = g^{-1}\big(f(x; w)\big) = g^{-1}(w \cdot x)$, where g^{-1} is the inverse link function.

An example is with binary data, say $y \in \{0, 1\}$.

- We pick the Bernoulli distribution with $p(y; \pi) = \pi^y (1 - \pi)^{1-y}$.
- Note that $\mathbb{E}[y \mid x] = \pi = g^{-1}(f(x; w)) = g^{-1}(x \cdot w)$, where $w \in \mathbb{R}$, we need $g^{-1} : \mathbb{R} \to (0, 1)$, say the logistic function (or `sigmoid`):
$$g^{-1}(z) = \frac{1}{1 + \exp(-z)}.$$
This is the CDF for logistic.

Again, we want to have:

$$\begin{aligned}
w^* &= \arg\min_w \mathbb{E}_{\mathbb{P}(x)} \mathbb{KLD}\left[\mathbb{P}(y \mid x) \mid\mid \mathcal{B}(y, \pi = g^{-1}(f(x; w)))\right] \\
&= \arg\min_w \frac{1}{N}\sum_{n=1}^{N} -\log \mathcal{B}(y_n, \pi_n = \texttt{logistic}(f(x_n; w))) \\
&= \arg\min_w \frac{1}{N}\sum_{n=1}^{N} -\log\left[\pi_n^{y_n}(1 - \pi_n)^{1-y_n}\right] \\
&= \arg\min_w \frac{1}{N}\sum_{n=1}^{N} -y_n \log \pi_n - (1 - y_n)\log(1 - \pi_n) \\
&= \arg\min_w \frac{1}{N}\sum_{n=1}^{N} -y_n \log(\texttt{logistic}(x_n, w)) - (1 - y_n)\log(1 - \texttt{logistic}(x_n, w)).
\end{aligned}$$

(called binary entropy loss)

As always, we take the derivative as:

$$\begin{aligned}
\partial_w \ell(w;\mathcal{D}) &= \partial_w \left[\frac{1}{N}\sum_{n=1}^{N} -y_n \log(\texttt{logistic}(x_n,w)) - (1-y_n)\log(1-\texttt{logistic}(x_n,w))\right] \\
&= \frac{1}{N}\sum_{n=1}^{N} -y_n \partial_w \log(\texttt{logistic}(x_n,w)) - (1-y_n)\partial_w \log(1-\texttt{logistic}(x_n,w)) \\
&= \frac{1}{N}\sum_{n=1}^{N} [-y_n(1-\texttt{logistic}(x_n;w)) + (1-y_n)\texttt{logistic}(x_n;w)]x_n \\
&= \frac{1}{N}\sum_{n=1}^{N} \big(\texttt{logistic}(x_n \cdot w) - y_n\big)\cdot x_n.
\end{aligned}$$

I.4 Gradient Descent

Given some function we wish to minimize, say $\phi(z)$, a strategy we can do is:

- pick an initial value z_0,
- iterate the equation $z_{t+1} = z_t - \alpha \cdot \frac{d}{dz_t}\phi(z_t)$, where $\alpha \in \mathbb{R}^+$ is the step size.
- stop when $\left|\frac{d}{dz}\phi(z)\right| \leq \varepsilon$ (like 1×10^{-4}) or maximum number of iterations are reached.

Note that the step size α is important, the convergence gets slower (or even fails to converge) when the step size is too small or too large.

I.5 The Perceptron (1957)

Consider $\hat{y} = \psi(w \cdot x + b)$, where $w, b \in \mathbb{R}$ and $x \in \mathbb{R}$ is the feature. We have:

$$\psi(z) = -1 + 2 \cdot \mathbb{1}[z > 0].$$

In particular, we have the indicator function, *i.e.*:

- If $x \cdot w > -b$, then $\hat{y} = +1$, and
- if $x \cdot w < -b$, then $\hat{y} = -1$.

```
**Inputs:** $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, $w_0$, and $b_0$.
**for** $t \leftarrow 1, 2, \cdots, T$ **do**
    Pick random pair from $\mathcal{D}$, say $(x_r, y_r)$.
    Compute the prediction $\widehat{y_r} = \psi(w_0 \cdot x_r + b_0)$, and checked on correctness.
    **if** the prediction is not correct, *i.e.*, $\widehat{y_r} \neq y_r$ **then**
        Update the parameters: $w_{t+1} \leftarrow w_t + y_r \cot x_r$, $b_{t+1} = b_t + y_r$.
    **end if**
**end for**
```

We may also modify the logistic regression, with one data point per update, assume $\alpha = 1$ and assume $\texttt{logistic}(w \cdot x) \in \{0, 1\}$.
There, the update is:

$$w_t = w_{t-1} + \big(y - \texttt{logistic}(x \cdot w_{t-1})\big) \cdot x.$$

II Multiple Features Supervised Learning Model

II.1 Feature Expansions

Consider multiple features of $(x_1, \cdots, x_D) \in \mathbb{R}^D$ and $(w_1, \cdots, w_D) \in \mathbb{R}^D$, we have:

$$\mathbb{E}_p[\mathrm{y} \mid \mathbf{x}] := g^{-1}(\mathbf{w}^\intercal \mathbf{x}).$$

An example is to add a bias/offset parameter, we consider:

$$\tilde{w} = \begin{bmatrix} w_0 \\ w \end{bmatrix} \text{ and } \tilde{x} = \begin{bmatrix} 1 \\ x \end{bmatrix}$$

Now, as we consider polynomial regression, we take scalar feature x and replicate it by taking higher and higher powers of x. Consider $x \in \mathbb{R}$, we now have:

$$\tilde{\mathbf{x}} = \begin{bmatrix} x^0 & x^1 & \cdots & x^k \end{bmatrix}^\intercal,$$

and when we consider this, we have:

$$\mathbb{E}[y \mid \tilde{x}] = g^{-1}(\mathbf{w}^\intercal \tilde{\mathbf{x}}) = g^{-1}\left(\sum_{k=0}^{K} w_k \cdot x^k\right).$$

For the steepest decent with multi-variate derivatives (say $f : \mathbb{R}^D \to \mathbb{R}$), it needs gradient operator as:

$$\nabla_x f(\mathbf{x}) = \left[\frac{df}{d\mathbf{x}}\right]^\intercal = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} & \cdots & \frac{\partial f(x)}{\partial x_D} \end{bmatrix}$$

Then, the gradient descend it now:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \left[\nabla_{\mathbf{w}_k} \ell(\mathbf{w}_t; \mathcal{D})\right]^\intercal.$$

An example could be with the logistic regression, that is:

$$\nabla_w \ell(w; \mathcal{D}) = \nabla_w \left[\frac{1}{N} \sum_n -y_n \log\left[s(w^\intercal x_n)\right] - (1 - y_n) \log\left[1 - s(w^\intercal x_n)\right]\right] = \frac{1}{N} \sum_{n=1}^{N} \left[s(w^\intercal x_n) - y_n\right] \cdot x + n^\intercal.$$

II.2 GLMs with Multiple Features and Output Dimensions

Consider that:

$$\mathbb{E} = [\mathbf{y} \mid \mathbf{x}] := g^{-1}(W^\intercal \mathbf{x}),$$

where we have $\mathbf{x} = (x_1, \cdots, x_D) \in \mathbb{R}^D$, $w = \begin{bmatrix} \mathbf{w}_1 & \cdots & \mathbf{w}_k \end{bmatrix} \in \mathbb{R}^{D \times k}$.
One example is with the real-valued regression problem, say $\mathbf{y} \in \mathbb{R}^k$, and with $\mathbb{E}[\mathbf{y} \mid \mathbf{x}] = \mathbf{w}^\intercal \mathbf{x}$, under the

assumption $p(\mathbf{y}, \mu = \mathbf{w}^\intercal \mathbf{x}, \Sigma = \sigma^2 \operatorname{Id})$, we have:

$$\ell(\mathbf{w}; \mathcal{D}) = \frac{1}{N} \sum_n \sum_{k=1}^{K} (y_{n,k} - w_k^\intercal x_n)^2.$$

II.3 Categorical Regression, *aka*, Multi-Class Classification

Labels are with one-hot encoding, e.g. $\mathbf{y}_n = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^\intercal$.
The distribution is:

$$P(\mathbf{y}_n; \boldsymbol{\pi}_n) = \texttt{categorical}(\mathbf{y}, \boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{y_{n,k}} = \pi_k.$$

In particular, we have $\boldsymbol{\pi} \in [0,1]^K$.

For the GLM for categorical regression, we have:

$$\mathbb{E}[\mathbf{y} \mid \mathbf{x}] = \boldsymbol{\pi} = g^{-1}(\mathbf{w}^\intercal \mathbf{x})/$$

Such inverse link function is called the soft-max function:

$$\texttt{softmax}(z) := \frac{\exp(z_k)}{\sum_{j=1}^{k} \exp(z_j)}.$$

This is really a `soft argmax` function, as it keeps the argument related to link.
Consider the categorical cross-entropy loss as:

$$\begin{aligned} W^* &= \arg\min_W \mathbb{E}_{\mathbb{P}(x)}[\mathbb{KLD}[\mathbb{P}(\mathrm{y} \mid \mathrm{x}) \mid\mid \texttt{categorical}(\mathbf{y}, \boldsymbol{\pi} = \texttt{softmax}(W^\intercal \mathbf{x}))]] \\ &\approx \arg\min_W \frac{1}{N} \sum_{n=1}^{N} -\log \left[\prod_{k=1}^{K} \pi_{n,k}^{y_{n,k}}\right] \\ &= \cdots \qquad \text{(intermediate steps ommitted due to lengthiness)} \\ &= \arg\min_W \frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{K} -y_{n,k} \cdot \left[\mathbf{w}_k^\intercal \mathbf{x}_n - \log\left(\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n)\right)\right]. \end{aligned}$$

To find a solution, we use the gradient descent method, in various cases.

- Case 1: Let $y_{n,k} = 1$, we have:

$$\begin{aligned} \nabla_{\mathbf{W}_k} \ell(\mathcal{D}; \mathbf{W}) &= \frac{1}{N} \sum_{n=1}^{N} \nabla_{\mathbf{W}_k} \sum_{k=1}^{K} -y_{n,k} \cdot \left[\mathbf{w}_k^\intercal \mathbf{x}_n - \log\left(\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n)\right)\right] \\ &= \frac{1}{N} \sum_{n=1}^{N} -\nabla_{\mathbf{W}_k} \left[\mathbf{w}_k^\intercal \mathbf{x}_n - \log\left(\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n)\right)\right] \\ &= \frac{1}{N} \sum_{n=1}^{N} -\left[\mathbf{x}_n^\intercal - \frac{\exp(\mathbf{w}_k^\intercal \mathbf{x}_n)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n)} \cdot \mathbf{x}_n^\intercal\right] = \frac{1}{N} \sum_{n=1}^{N} (\pi_{n,k} - 1) \cdot \mathbf{x}_n^\intercal. \end{aligned}$$

  Note that since we have chosen a good link function, it is giving a clear derivative.

- Another case we concern is when $y_{n,i} = 0$, we have:

$$\begin{aligned}\nabla_{\mathbf{W}_k} \ell(\mathcal{D}; \mathbf{W}) &= \frac{1}{N} \sum_{n=1}^{N} \nabla_{\mathbf{W}_k} \sum_{k=1}^{K} -y_{n,k} \cdot \left[\mathbf{w}_k^\intercal \mathbf{x}_n - \log \left(\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n) \right) \right] \\ &= \frac{1}{N} \sum_{n=1}^{N} -\nabla_{\mathbf{W}_k} \left[\mathbf{w}_k^\intercal \mathbf{x}_n - \log \left(\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n) \right) \right] \\ &= \frac{1}{N} \sum_{n=1}^{N} - \left[-\frac{\exp(\mathbf{w}_k^\intercal \mathbf{x}_n)}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\intercal \mathbf{x}_n)} \cdot \mathbf{x}_n^\intercal \right] = \frac{1}{N} \sum_{n=1}^{N} \pi_{n,k} \cdot \mathbf{x}_n^\intercal.\end{aligned}$$

Note that when implementing this, we do not have to split the case since the both cases can be $\pi_{n,k} - y_{n,i}$, and $y_{n,i}$ is the indicator variable.

For the output $\widehat{\pi} = \texttt{softmax}(\mathbf{w}^\intercal \mathbf{x})$, we may consider the classifier as a vector valued so that we take the entry with highest classification value.

II.4 Model Evaluation

Recall that the goal is having $\mathbb{P}(y \mid x) \approx p\big(y, f(x)\big)$ when we only have the data $\mathcal{D}\{(x_n, y_n)\}_{n=1}^{N}$ and $(x_n, y_n) \in \mathbb{P}(x; y)$.

Our idea is to have:

- Train on $\mathcal{D} \to w^*$, and
- Evaluate $\ell(w^*, \mathcal{D}) = \delta$.

The problem is that it has seen all the points from $\mathcal{D}$. The solution turns out to be:

- Split the data with training (learn the task, $\sim$ 70%), validation (select the best model, $\sim$ 20%), and test data (how good is this model truly, $\sim$ 10%).

A method is called the k-fold cross validation. In this case, we have the data set split into training data and testing set. For the training data, we can do a k-fold, in which each training data contain a certain part as the validation set.

For example, we have the real-valued regression that:

$$\ell\big(w^*, \mathcal{D}_{\text{test}}\big) = \frac{1}{M} \sum_{n=1}^{M} (\underbrace{\mathbb{E}[y_n \mid x_n]}_{w^* x_n} - y_n)^2.$$

For the root mean square error, we have:

$$\text{RMSE}\big(w^*, \mathcal{D}_{\text{test}}\big) = \sqrt{\frac{1}{M} \sum_{n=1}^{M} (\underbrace{\mathbb{E}[y_n \mid x_n]}_{w^* x_n} - y_n)^2}.$$

The classification is a little bit different that is:

$$\operatorname{Acc}\big(\{\mathbb{E}[y_m \mid x_m]\}_{m=1}^{M}, \mathbf{y}\big) = \operatorname{Acc}\big(w^*, \mathcal{D}_{\text{test}}\big) = \frac{1}{M}\sum_{n=1}^{M} \mathbb{1}[y_n - \arg\max \pi_n].$$

With the **learning algorithm**, we train models from the training data, then with the **model parameters**, we use validation data to have the **validation metric**, and as long as it is good, the model could be **deployed to the world** with test data.

It is important to be careful of how many times we shall re-use validation sets. *Not to memorize the data set.*

The above is with **accuracy**, and we later consider **precision**.

We can consider a confusion matrix:

	Spam	Non-spam
Span	True positive	False negative
Non-spam	False positive	True negative

Table II.1. Columns are actual and rows are predicted.

This, of course can be extended to cases with more estimates.

The precision is:

$$\operatorname{Precision}(k; C) = \frac{TP}{TP + FP} = \frac{C_{k,k}}{\sum_{j=1}^{K} C_{j,k}}.$$

We consider the recall as:

$$\operatorname{Recall}(k; C) = \frac{TP}{TP + FN} = \frac{C_{k,k}}{\sum_{j=1}^{K} C_{k,j}}.$$

The Macro-Precisions and Macro-Recall are the average of each C.

F_1 score is the harmonic mean fo the precision and recall, so it is more sensitive to the smaller of the two values.

$$F_1(C) = \frac{2}{\frac{1}{\text{Macro-Precision}(C)} + \frac{1}{\text{Macro-Recall}(C)}}.$$

There are different sources of variations, like:

- number of data points in the training set.

II.5 Regularization

Now, the problem is considering **overfitting** and **underfitting**. We want a good fit.
Models have two important properties:

- **Bias**: The flexibility of the model class, the ability to represent true function (like degree 1 and 10).
- **Variance**: The variance is how much the model is fluctuating within the model itself.

Consider the small N, the high bias model is usually better, since the high variance models will overfit the training data. For a large N, consider the low bias model since it takes advantage of higher quality statistical signal.

Then, we need to combat with variance:

- **Regularization**: add a penalty term to encourage simple model.
  - L2 or Ridge penalty:
    $$\tilde{\ell}(w, \mathcal{D}) = \ell(w, \mathcal{D}) + \lambda \sum_{d=1}^{D} w_d^2.$$
    It depend on the choice if we penalize the offset term, like, is our data centered?
  - Consider the sum as a ℓ^p norm, we can have other penalties.
- **Ensemble Method**: aggregate the predictions of multiple models. The core idea is that if the variances in each model is uncorrelated, then the noise cancels out with just the signal.
  With the high flexibility models, we can try to reduce the variance by doing so.

III Feedforward Neural Network

III.1 Adaptive Features and Nonlinear Models

In the previous sense, we had:

$$\underbrace{\begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix}}_{x} \xrightarrow{g^{-1}(w^\intercal x)} \mathbb{E}[y \mid x].$$

which is really a shallow model, even if we had the polynomial basis, such as:

$$\underbrace{\begin{bmatrix} \bullet & \cdots & \bullet \\ \vdots & \ddots & \vdots \\ \bullet & \cdots & \bullet \end{bmatrix}}_{x} \xrightarrow{g^{-1}(w^\intercal \mathbf{x})} \mathbb{E}[y \mid x].$$

However, with a neural network, we have some intermediate components:

$$\underbrace{\begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix}}_{x} \longrightarrow \begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix} \longrightarrow \begin{bmatrix} \bullet \\ \vdots \\ \bullet \end{bmatrix} \xrightarrow{g^{-1}(w^\intercal \mathbf{x})} \mathbb{E}[y \mid x].$$

Here, we have:

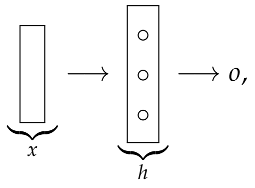
$$\mathbb{E}[y \mid x] = g^{-1}\big(w^\intercal \psi(x; u)\big),$$

where we have $\psi(\bullet; \bullet)$ as new feature representations, and u is the parameters that allow us to learn new representations.

We need some nonlinear functions, otherwise, it degenerates into the previous model (*Riesz Representation Theorem*), we want:

$$\mathbb{E}[y \mid x] = g^{-1}\big(w^\intercal \varphi(u^\intercal x)\big) \neq g^{-1}\big(\varphi(w^\intercal u^\intercal)x\big).$$

We want adaptive, non-linear feature basis, so we construct:



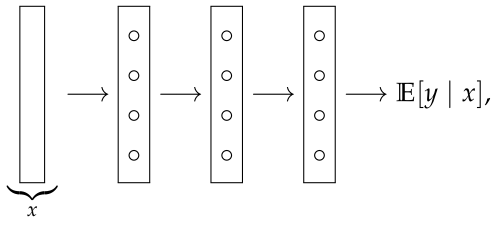
where we have h as a hidden layer composed of hidden units.

When we have $h = \varphi(u^\intercal x)$, we have u as the weights, φ as an activation function and $u^\intercal x$ is the pre-activation vector.

They are also called artificial neural networks or multilayer perceptron.

III.2 Deep Neural Network

To have multiple hidden layers, we can have deep neural networks.



where we have the operation as:

$$h_\ell = \varphi(w_\ell^\intercal h_{\ell-1}),$$

and the output of the L layers will be:

$$\mathbb{E}[y \mid x] = g^{-1}(w_L^\intercal h_L).$$

If we were to expand, we have:

$$\mathbb{E}[y \mid x] = g^{-1}(w_L^\intercal \varphi(w_{L-1}^\intercal \varphi(W_{L-2}^\intercal \varphi(\cdots)))).$$

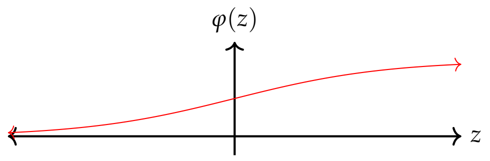
Also, note that we have to incorporate the offset parameter:

$$h_\ell = \begin{bmatrix} \varphi(w_{\ell-1}^\intercal h_{\ell-1}) & 1 \end{bmatrix}.$$

III.3 Activation Functions

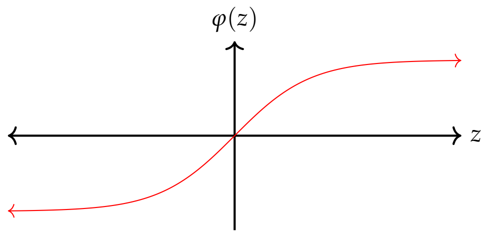
- Logistic/Sigmoid Function:

$$\varphi(z) = \frac{1}{1+e^{-z}}.$$



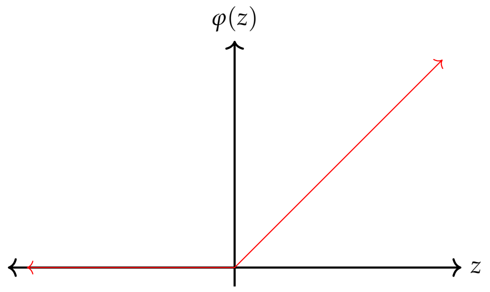
- Hyperbolic Tangent Function:

$$\varphi(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$



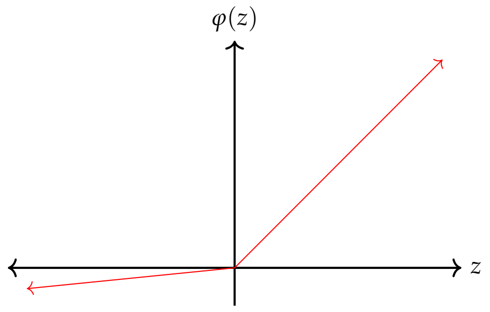
- Rectified Linear Unit (or ReLU):

$$\varphi(z) = \max(0, z).$$



- Leaky ReLU: Let $\varepsilon \in (0, 0.1)$ be fixed

$$\varphi(z) = \max(\varepsilon \cdot z, z).$$



Sometimes, we need to train with depth rather than width for restricted flexibility and training time.

III.4 Training a Neural Network

Now, we have the training as:

$$\mathbb{E}_{\mathbb{P}(x)}\mathbb{KLD}\left[\mathbb{P}(y \mid x) \;||\; p\big(y, f(x, w_0, \cdots, w_L)\big)\right].$$

While assuming real-valued regression, we have:

$$\approx \frac{1}{N}\sum_{n=1}^{N}\big(y_n - f(x_n, w_0, \cdots, w_L)\big)^2.$$

Thus, the loss function is:

$$\ell(w_0, \cdots, w_L, \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - w_L \cdot h_L)^2, \qquad (fcn.1)$$

whose partial derivative is:

$$\frac{d\ell}{dw_L} = \frac{1}{N}\sum_{n=1}^{N}2(y_n - w_L h_L)(-h_L).$$

Then, the next derivative is:

$$\frac{d\ell}{dw_{L-1}} = \frac{1}{N}2(y_n - w_L h_L)(-w_L)\frac{d\varphi}{d(w_{L-1}h_{L-1})}\cdot h_{L-1}.$$

Of course, we may expand (*fcn*.1) into more layers, namely:

$$\ell(w_0, \cdots, w_\ell, \cdots, w_L, \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\big(w_L^\intercal \varphi(w_{L-1}^\intercal h_{L-1}) - y_n\big)^2.$$

To apply gradient descent, we use:

$$w_\ell^{t+1} = w_\ell^t - \alpha\nabla_{w_\ell^t}\ell\ell(w_0, \cdots, w_\ell, \cdots, w_L, \mathcal{D}),$$

where w_i^t is the all parameters in the neural network at iteration (time) t, and the step size α.

Here, we give an example of scalar neural network, say:

$$\begin{aligned}
&\mathbb{E}[y \mid x] = w_2 \cdot h_2,\\
&h_2 = \text{logistic}(w_1 \cdot h_1),\\
&h_1 = \text{logistic}(w_0 \cdot x),
\end{aligned}$$

where we have $w_2, w_1, w_0, y_1, x \in \mathbb{R}$.
The derivative w.r.t. w_2 is:

$$\frac{d}{dw_2}\ell(w_2, w_1, w_0; \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\frac{d\ell_n}{d\mathbb{E}_n}\frac{d\mathbb{E}_n}{dw_2} = \frac{1}{N}\sum_{n=1}^{N}2\big((w_2 \cdot h_{n,2}) - y_n\big)\cdot h_{n,2}.$$

The derivative w.r.t. w_1 is:

$$\frac{d}{dw_1}\ell(w_2, w_1, w_0; \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\frac{d\ell_n}{d\mathbb{E}_n}\frac{d\mathbb{E}_n}{dh_{n,2}}\frac{dh_{n,2}}{d(w_1 \cdot h)}\frac{d(w_1 \cdot h)}{dw_1} = \frac{1}{N}\sum_{n=1}^{N}2\big((w_2 \cdot h_{n,2}) - y_n\big)\cdot w_2 \cdot h_{n,2}(1 - h_{n,2})\cdot h_{n,1}.$$

The derivative w.r.t. w_0 is then:

$$\begin{aligned}\frac{d}{dw_0}\ell(w_2, w_1, w_0; \mathcal{D}) &= \frac{1}{N}\sum_{n=1}^{N}\frac{d\ell_n}{d\mathbb{E}_n}\frac{d\mathbb{E}_n}{dh_{n,2}}\frac{dh_{n,2}}{d(w_1\cdot h)}\frac{d(w_1\cdot h)}{dh_{n,1}}\frac{dh_{n,1}}{d(w_0\cdot h)}\frac{d(w_0\cdot h)}{dw_0}\\ &= \frac{1}{N}\sum_{n=1}^{N}2\big((w_2\cdot h_{n,2})-y_n\big)\cdot w_2\cdot h_{n,2}(1-h_{n,2})\cdot w_1 h_{n,1}(1-h_{n,1})x.\end{aligned}$$

Diligent readers should already notice that the general structure of the derivatives are basically the same.

Here, we have **forward propagation**, as we forwardly spread the knowledge to the output.

III.5 Back Propagation

Consider the derivative as:

$$\frac{d}{dw_\ell}\ell(w_0,\cdots,w_L,\mathcal{D}) = \frac{d\ell}{d\mathbb{E}}\frac{d\mathbb{E}}{dh_L}\frac{dh_{L-1}}{dh_{L-2}}\cdots\frac{dg_{\ell+2}}{dh_{\ell+1}}\frac{dh_{\ell+1}}{dw_\ell}.$$

This represents the error signals go backwards through the network.

This is the issue with the ordering of the models, the forward computes from the right, and the backward computes from the left.

Another issue is with **exploding ot banishing gradients**. With the long series of multiplication, product from the chain rule leads to the total derivative going to $\pm\infty$ or 0, just from a few unstable terms.

- An example of vanishing could also due to the activation saturation, recall the logistic $\varphi(z)$, we have:

$$\varphi'(z) = \varphi(z)\big(1-\varphi(z)\big).$$

- Also, with `ReLU`, when initializing the weights, it should have been positive, or those units will be evaluated to 0 and considered the dead units, pulling down the learning rate.

III.6 Vectorized Implementation of Deep Neural Network

Consider that:

$$X = \begin{bmatrix} x_1^\intercal \\ \vdots \\ x_N^\intercal \end{bmatrix} \text{ and } Y = \begin{bmatrix} y_1^\intercal \\ \vdots \\ y_N^\intercal \end{bmatrix},$$

where X is the feature matrix of size $N\times X_c$ and Y is the label matrix of size $N\times K$, we have:

$$H_\ell = \varphi(H_{\ell-1}W_{\ell-1}),$$

where H_ℓ is of size $N\times D_\ell$, $H_{\ell-1}$ has size $N\times D_{\ell-1}$ and $W_{\ell-1}$ is $D_{\ell-1}\times D_\ell$.

Here, we consider the loss as:

$$\ell(W_0,\cdots,W_L,X,Y) = -\frac{1}{N}\sum Y\odot\log\texttt{softmax}(H_L W_L),$$

where the output is $N \times K$ and $\odot$ denotes the element-wise product.

Here, the general form of the propagation is:

$$\nabla_{W_\ell} \ell(W_0, \cdots, W_L, X, Y) = \frac{1}{N}(\mathbb{E}[Y \mid X] - Y) W_L^\intercal \odot \varphi'(A_L) W_{L-1}^\intercal \odot \varphi'(A_{L-1}) \cdots,$$

so we can think about it as:

$$\nabla_{W_\ell} \ell(W_0, \cdots, W_L, X, Y) = \frac{1}{N}\left[(E[Y \mid X] - Y) \prod_{j=L}^{\ell+1} W_j^\intercal \odot \varphi'(A_j)\right]^\intercal H_\ell.$$

We might need to transpose again to reach to the same dimensions with the matrix.

III.7 Skip Connections

We consider the hidden units as:

$$h_\ell = \varphi(W_{\ell-1}^\intercal h_{\ell-1}) + h_{\ell-1},$$

and the residual connection is:

$$h_\ell - h_{\ell-1} = \varphi(W_{\ell-1}^\intercal h_{\ell-1}).$$

Consider the version with dimension change, we have:

$$h_\ell = \varphi(W_{\ell-1}^\intercal h_{\ell-1}) + U^\intercal h_{\ell-1},$$

where we have $U^\intercal$ of $D_\ell \times D_{\ell-1}$ as the parameter matrix to train.

Now, we consider the example of scalar neural network with skip connection as:

$$\mathbb{E}[y \mid x] = w_2 h_2, \qquad h_2 = \varphi(w_1 h_1) + h_1, \qquad \text{and } h_1 = \varphi(w_0 x),$$

for $x, y \in \mathbb{R}$, and we consider the derivatives again:

$$\frac{d\ell}{dw_2} = \frac{1}{N}\sum_{n=1}^{N} 2(\mathbb{E}[y_n \mid x_n] - y_n) h_{n,2},$$
$$\frac{d\ell}{dw_1} = \frac{1}{N}\sum_{n=1}^{N} 2(\mathbb{E}[y_n \mid x_n] - y_n) w_2 \cdot$$

However, this changes for w_0 case:

$$\frac{d\ell}{dw_0} = \frac{1}{N}\sum_{n=1}^{N} 2(\mathbb{E}[y_n \mid x_n] - y_n) w_2 \left(h_{n,2}(1 - h_{n,2}) w_1 + 1\right) h_{n,1}(1 - h_{n,1}) x.$$

Note that the additional 1 prevents the gradient to be zero even it was very small, so it makes drastic changes.

III.8 Weight Initialization

Recall the `logistic` and `ReLU` function, we want to have the initial weights to be at a nice region of input, otherwise it could screw up.

We can consider the Xavier initialization function, consider for `logistic` and tanh activation:

$$W_l \sim \texttt{Uniform}\left(\frac{-\sqrt{6}}{\sqrt{D_l + D_{l+1}}}, \frac{\sqrt{6}}{\sqrt{D_l + D_{l+1}}}\right)$$

This is symmetric about zero and the as the dimension of the layers increases, it is more focused onto zero.

In terms of the `ReLU`, he uses:

$$W_l \sim \mathcal{N}\left(0, \frac{2}{D_l}\right).$$

III.9 Batch and Layer Normalization

We want to stay in a good region during the training. We consider the Batch layer to "normalize" a region correspondingly:

$$\hat{a_d} = \frac{a_d - \hat{\mu_d}}{\sqrt{\hat{\sigma_d}^2 + \epsilon}}, \text{ where } H_l W_l = A_{l+1},$$

where we may grab a column from the hidden layer.

Then consider:

$$H_{\ell+1} = \varphi(\hat{A}_{l+1} \odot \gamma + \beta),$$

which is with the batchnorm parameters to be trained on.

This prevents the runaway gradients, to make sure the goodness of the initializations were kept.

Consider the **normalization transformation**, so we have:

$$\widetilde{a_{n,d}} = \beta_\alpha + \gamma_\alpha \frac{a_{n,d} - \mathbb{E}[a_{\cdot,d}]}{\sqrt{\text{Var}(a_{\cdot,d}) + \epsilon}},$$

where $\beta_d \in \mathbb{R}$ and $\gamma_d \in \mathbb{R}$, with $d \in [1, D]$ and $\epsilon > 0$ is a small positive constant for numerical stability.

LayerNorm is with respect the layers:

$$\overline{a_{n,d}} = \beta_\alpha + \gamma_\alpha \frac{a_{n,d} - \mathbb{E}[a_{n,\cdot}]}{\sqrt{\text{Var}(a_{n,\cdot}) + \epsilon}}.$$

The selection of Batch Norm and Layer Norm may be dependent on the dimension of the equation.

- BN is with fixed β and γ, and use train-time statistics, and
- LN is with fixed β and γ, and use to compute $\mathbb{E}[a]$ and $\text{Var}[a]$ for the test points.

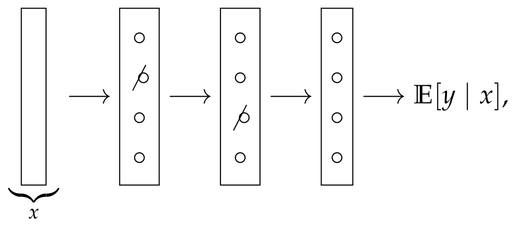
III.10 Capacity Control for Deep Neural Network

Similar to regularization, we have **weight decay**, namely:

$$\tilde{\ell}(w_0, \cdots, w_L; \mathcal{D}, \lambda) = \ell(w_0, \cdots, w_L; \mathcal{D}) + \lambda \sum_{\ell=0}^{L} \|W_\ell\|_2^2.$$

Similar to the ensemble method, we train multiple models and combine them via average/voting.
It is noted that such training is expensive, to train 3 models, right? So we want **cheap ensembles via dropout**.

Here, we ignore some neurons (hidden units) as zeros by random.



There were element-wise multiplication with binary vectors:

$$\mathbf{h}_1 \odot \mathbf{b},$$

which is like to randomly hide certain units.

IV Stochastic, Adaptive Optimizers

IV.1 Stochastic Gradient Descent

The main idea is:

$$\begin{aligned}\nabla_{\mathbf{W}_l^t}\ell(\mathbf{W}_0^t,\cdots,\mathbf{W}_l^t,\cdots,\mathbf{w}_L^t;\mathcal{D}) &\approx \nabla_{\mathbf{W}_l^t}\ell(\mathbf{W}_0^t,\cdots,\mathbf{W}_l^t,\cdots,\mathbf{w}_L^t;\mathcal{B})\\ &= \frac{1}{B}\sum_{b=1}^{B}\nabla_{\mathbf{W}_l^t}\ell(\mathbf{W}_0^t,\cdots,\mathbf{W}_l^t,\cdots,\mathbf{w}_L^t;(\mathrm{x}_b,y_b)).\end{aligned}$$

For example, we could have a lot of data, but a small proportion of it will give me a good idea on what us a smaller subset. There are also randomness in these sets.

The update is:

$$w_{t+1} = w_t - \alpha\cdot\nabla_{w_t}\ell(w_t;\mathcal{B}).$$

Might need to handle between large and small sizes.

Now, we can use **SGD with momentums**, consider:

$$\mathbf{V}_l^t = \beta\cdot\mathbf{V}_l^{t+1} = \mathbf{V}_{l-1}^{t-1} + \nabla_{\mathbf{W}_l}\ell(\mathbf{W}_l^t\cdot\mathbf{W}_L^t,\mathcal{B}),$$

and the update is:

$$w_l^{t+1} = w_l^t - \alpha\cdot V_l^t, \qquad v^{t=0} = 0,$$

and α.

IV.2 Adaptive Moment Estimation (ADAM)

We let:

$$\begin{aligned}v_l^t &= \beta_1 v_l^{t-1} + (1-\beta_1)\nabla_{W_l^t}\ell\\ s_l^t &= \beta_2 s_l^{t-1} + (1-\beta_2)(\nabla_{W_l^t}\ell)^2.\end{aligned}$$

Then, the final update is:

$$\hat{v}_l^t = \frac{v_l^t}{1-\beta_1^t}, \hat{S}_l{}^t = \frac{S_l^t}{1-\beta_2^t}$$

so:

$$w_l^{t+1} = w_l^t - \frac{\alpha}{\sqrt{\hat{s}_l^t}+\epsilon}\odot\hat{v}_l^t.$$

For instance, give some optimization surface and trajectory, the batch one for SVG will be jumping around. Likewise, for ADAM, the β close to 0 will be jumping around and β close to 1 will be more stable.

IV.3 Convolutional Neural Network

A basic example is the image classification problem:

- With an input image, we want to collapse it as a vector, called **vectorize**.
- The MNIST image is the visualization of the handwritten monocolors of 0 to 9.

Note that when we vectorize the image, the issue is that the identification could not necessarily be in the middle.
The example is with identifying if there is a **deer** in the image, it might not uniformly be on the center. We ant:

- Shift invariance or transitional invariance.

We want to break this constraints.

- A solution could be to have multiple neural networks with proportions of the neural network.
- Then, we have the feed forward layer that could filter our some information.
- There could be some particular weight matrix to convolute about the image matrix.

Then, we consider the output layer as the last layer of the last hidden layer. Is there such object in the image?
We then can flatten the output since it contains the information of the spacial information.
Consider a $32 \times 32 \times 3$ image, the filter can be $5 \times 5 \times 3$, where the depth has to be the same, and we can convolve (slide) over all spatial location.

The stride effect is that:

$$\frac{N - F}{\text{stride}} + 1.$$

We many change the stride or manually changing the border by adding zeros.

For example, with $32 \times 32 \times 3$ and 10 5×5 filters with stride 1 and pad 2, the output volume size is:

$$\frac{32 + 2 \times 2 - 5}{1} + 1 = 32,$$

so the result is $32 \times 32 \times 10$.
The total number of parameters for each filter is $5 \times 5 \times 3 + 1 = 76$ parameters, so in total 760 parameters.

The convolution network preserves **spacial locality**, and have **coordinate invariant**. The process is to have a weight matrix and scan over the original matrix.

Implementation-wise, the matrices are typically square.

There were **pooling** strategies to get some representative data.
For a max pooling, it is pulling out the maximum in the filters and the strides, and it gives some activation information.

The application of CNN beyond images is with 1-D convolutions good for sentence streams of data.

- Such as input wave front for sound recognition.

- Graphs of molecules, such as a ways of writing methanol (C_2H_5OH).
- Can be generalized to other transformations beyond translations, such as astronomical objects.

V Recurrent Neural Network (RNN)

V.1 Time Series Data

Time series data is dependent on time, where we have:

$$\mathbf{x}_n = \begin{pmatrix} \mathrm{x}_{n,1} & \cdots & \mathrm{x}_{n,T} \end{pmatrix} \text{ and } \mathbf{y}_n = \begin{pmatrix} \mathrm{y}_{n,1} & \cdots & \mathrm{y}_{n,J} \end{pmatrix}.$$

For example, we have sentiment analysis, say:

$$x_n = [\text{“This”}, \text{“jacket”}, \text{“is”}, \cdots] \text{ and } y_n \in \{0, 1\}.$$

It can also be used in **forecasting**, such as considering:

$$x_n = \text{observable conditions and } t_{n,t} = \$ \text{ at time } t.$$

Also with **translation**, we consider:

$$\mathrm{x}_n = [\text{”I”}, \text{”have”}, \text{”3”}, \text{”dogs”}] \text{ and } \mathrm{y}_n = [\text{”Ik”}, \text{”heb”}, \text{”drie”}, \text{”honden”}] \text{ or } \tilde{\mathrm{y}}_n = [\text{”Tengo”}, \text{”tres”}, \text{”perros”}]$$

Note that the number of words does not necessarily corresponds.
For instance, the one-hot-encoding corresponds to the word.

Here are two important features of **time series** data:

- Information is encoded by time.
- Dynamic in size/length.

V.2 Recurrent Neural Network

Consider a feed-forward, we have:

$$\begin{tikzcd} {[\mathrm{x}_t]} \arrow[r, "w_0"] & {[h_t]} \arrow[r] \arrow[loop below, "w_1"] & \mathbb{E}[\mathrm{y} \mid \mathrm{x}] \end{tikzcd}$$

The output is:

$$\mathbb{E}[\mathrm{y}_n \mid \mathrm{x}_n] = g^{-1}(\mathbf{W}_2^\intercal h_T).$$

Instead, we want to have:

$$h_t = \phi(\mathbf{w}_0^\intercal x_{n,t} + \mathbf{w}_1^\intercal h_{t-1}),$$

with h_t as the time index, ϕ as the activation function, and h_{t-1} is the previous hidden state.

All the layers share the same parameters w_1 and w_0.

$$\begin{tikzcd}
{[h_0]} \arrow[r, "\mathbf{w}_1\cdot(-)"] & {[h_1]} \arrow[r, "\mathbf{w}_1\cdot(-)"] & {[h_2]} \arrow[r, "\mathbf{w}_1\cdot(-)"] & \cdots \arrow[r, "\mathbf{w}_1\cdot(-)"] & {[h_T]} \arrow[r, "\mathbf{w}_1\cdot(-)"] & \mathbb{E}[\mathrm{y} \mid \mathrm{x}] \\
0 \arrow[u, "\mathbf{w}_0\cdot(-)"] & x_1 \arrow[u, "\mathbf{w}_0\cdot(-)"] & x_2 \arrow[u, "\mathbf{w}_0\cdot(-)"] & & x_T \arrow[u, "\mathbf{w}_0\cdot(-)"] &
\end{tikzcd}$$

This seems nice, but it is hard to **backpropogate through time**.

Here, we consider the loss function as:

$$\ell(w_0, w_1, w_2; x_1, \cdots, x_T, y) = \left(\mathbb{E}[\mathrm{y} \mid x_{1:t}] - y\right)^2.$$

The derivative with respect to w_2 is:

$$\frac{d}{dw_2}\left(\mathbb{E}[\mathrm{y} \mid x_{1:t}] - y\right)^2 = 2(w_2 \cdot h_T - y) \cdot h_t.$$

When we think about the derivative with respect to w_0, we have:

$$\frac{d\ell}{dw_0} = \frac{d\ell}{dh_T}\frac{dh_T}{da_T}\left(\frac{da_T}{dw_0} + \frac{da_T}{dh_{T-1}}\frac{dh_{T-1}}{da_{T-1}}\left(\frac{da_{T-1}}{dw_0} + \frac{da_{T-1}}{dh_{T-2}}\frac{dh_{T-2}}{da_{T-2}}\left(\cdots\right)\right)\right)$$

Hence, for the derivative w.r.t. w_1, we have:

$$\frac{d\ell}{dw_1} = \frac{d\ell}{dh_T}\frac{dh_T}{da_T}\left(\frac{da_T}{dw_1} + \frac{da_T}{dh_{T-1}}\frac{dh_{T-1}}{da_{T-1}}\left(\frac{da_{T-1}}{dw_1} + \frac{da_{T-1}}{dh_{T-2}}\frac{dh_{T-2}}{da_{T-2}}\left(\cdots\right)\right)\right)$$

Note that for w.r.t. w_1, we have:

$$\frac{da_t}{dw_1} = h_{t-1} \text{ and } \frac{da_t}{dh_{t-1}} = w_1,$$

whereas for w.r.t. w_0, we have:

$$\frac{da_t}{dw_1} = x_t \text{ and } \frac{da_t}{dh_{t-1}} = w_1,$$

Note that the x_t is fixed for w_0, but it will be more computationally challenged for w_1, as h_{t-1} requires more computation.

- This is too nasty, so people might have clipping to end at some amount of time, or using random time-steps to compute.

For a simple RNN, we have:

$$\begin{aligned}
&\mathbb{E}[y_0 \mid x_{n,1:T}] = g^{-1}(w_2^\intercal h_T), \\
&h_t = \phi(W_1^\intercal h_{t-1} + W_0^\intercal x_{n,t}), \qquad h_0 = \mathbf{0}.
\end{aligned}$$

There is an extra 1 to make the bias implicitly. Consider training RNN with squared loss for $T = 4$:

$$\begin{aligned}
\frac{d\ell}{dw_1} &= \frac{1}{N}\sum_{n=1}^{N} \underbrace{2(\mathbb{E}[y_n \mid x_{n,1:4}] - y_n)}_{\frac{d\ell_n}{d\mathbb{E}}} \cdot \underbrace{w_2}_{\frac{d\mathbb{E}}{dh_T}} \cdot \underbrace{\phi'(a_4)}_{\frac{dh_T}{da_T}} \left(h_3 + \phi(a_3)\left(h_2 + \phi'(a_2)(h_1 + \phi'(a_1)\cdot h_0)\right)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N} \frac{d\ell_n}{d\mathbb{E}_n} w_2 \cdot \left(\phi'(a_4)h_3 + \phi'(a_4)\phi'(a_3)h_2 + \phi'(a_4)\phi'(a_3)\phi'(a_2)h_1 + \phi'(a_4)\phi'(a_3)\phi'(a_2)\phi'(a_1)h_0\right).
\end{aligned}$$

Eventually we have:

$$\frac{d\ell}{dw_1} = \frac{1}{N}\sum_{n=1}^{N}\frac{d\ell_n}{d\mathbb{E}_n}\cdot w_2 \cdot \sum_{j=T}^{1}\left(\prod_{i=T}^{j}\phi'(a_i)\right)h_{j-1}.$$

Note that for the logistic activation function, this will become smaller and smaller as it is the product of elements in $[0, 1]$.
For the ReLU, if it is in the far right, it might explode and might just be the sum of the hidden states.
There are two common coping mechanisms:

- **Back propagate through time** (BPTT) **Truncation**: stop the problem up to some time, so assuming less times were observed.
- **Randomized Truncations**: there will be two types of truncations:
  - hard truncation: pick a value of T randomly.
  - soft truncation: introduce a random variable $X \subset \{0, 1\}$ multiplied in the sum so that T is more likely to be 0 when in the deeper (earlier) layers.

One problem is that it has too much "equality" among all time steps. Consider the dog ("el perro"), but the "the" does not matter as much as "dog" in English.

V.3 Long Short-Term Memory (LSTM) Architecture

The idea is to have hidden states replaced by *memory cells*.

Here, memory cell has an "internal state" that is protected from gradient explosion or vanishing.

Memory cell has dedicated "gates" for:

- **input gate**: how much x_t should influence internal state?

  $$\iota_t = \vartheta(\mathbf{W}_0^{i\,\top}\mathrm{x}_t + \mathbf{W}_1^{i\,\top}h_{t-1}),$$

  where ϑ is the logistic function.
- **forget gate**: should internal state be flushed?

  $$f_t = \vartheta(\mathbf{W}_0^{f\,\top}\mathrm{x}_t + \mathbf{W}_1^{f\,\top}h_{t-1}).$$
- **output gate**: how much should internal state affect the current output?

  $$\mathcal{O}_t = \vartheta(\mathbf{W}_0^{o\,\top}\mathrm{x}_t + \mathbf{W}_1^{o\,\top}h_{t-1}).$$

Also, we define the **input node** as:

$$\tilde{c}_t = \tanh(\mathbf{W}_0^{c\,\top}x_t + \mathbf{w}_1^{c\,\top}h_{t-1}),$$

and the internal state as:

$$c_t = f_t \odot c_{t-1} + \iota_t \odot \tilde{c}_t.$$

Also, the hidden state:

$$h_t = \mathcal{O}_t \odot \tanh(c_t).$$

$$\begin{tikzcd}
\text{internal state } c_{t-1} \arrow[rr] & & \odot \arrow[rr] & & \oplus \arrow[rrr] \arrow[dr, bend left] & & & c_t \\
& & & & \odot \arrow[u] & \tanh \arrow[dr] & & \\
& & f_t \arrow[uu] & \iota_t \arrow[ur] & \tilde{c}_t \arrow[u] & \mathcal{O}_t \arrow[r] & \odot \arrow[dr] & \\
\text{hidden state } h_{t-1} \arrow[r] & \amalg \arrow[ur] \arrow[urr, bend right] \arrow[urrr, bend right] \arrow[urrrr, bend right] & & & & & & h_t \\
& \text{input } x_t \arrow[u] & & & & & &
\end{tikzcd}$$

Figure V.1. LSTM Memory Cell.

V.4 Gated Recurrent Units (GRUs)

There will just be two gates:

- **Reset** gates:
$$r_t = \vartheta(\mathbf{W}_0^{r\intercal} x_t + \mathbf{W}_1^{r\intercal} h_{t-1}),$$
- **Update** gates:
$$u_t = \vartheta(\mathbf{W}_0^{u\intercal} x_t + \mathbf{W}_1^{u\intercal} h_{t-1}),$$

Here, GRU also computes a candidate hidden state:

$$\tilde{h}_t = \tanh(\mathbf{W}_0^{h\intercal} x_t + \mathbf{W}_1^{h\intercal}(r_t \odot h_{t-1})),$$

it then computes the final hidden state:

$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tilde{h}_t.$$

$$\begin{tikzcd}
\text{hidden state } h_{t-1} \arrow[rrr] \arrow[ddr] \arrow[drr] & & & \odot \arrow[rr] & & \oplus \arrow[r] & h_t \\
& & \odot \arrow[dd] & r_t \arrow[l] & u_t \arrow[ul] \arrow[r] & \odot \arrow[u] & \\
& \amalg \arrow[urr, bend right] \arrow[urrr, bend right] & & & & \tanh \arrow[u] \arrow[r, dashed] & \tilde{h}_t \\
& & \amalg \arrow[urrr, bend right] & & & & \\
& & \text{input } x_t \arrow[u] \arrow[uul] & & & &
\end{tikzcd}$$

Figure V.2. GRU Memeory Cell.

Now, we have the three basic architectures:

- **Simple RNN** having $h_t = \phi(\mathbf{W}_0^\intercal x_t + \mathbf{W}_1^\intercal h_{t-1})$.
- LSTM and GRM, which had gates to have different weights given to the structures.

There are four patterns for employing RNNs:

- **Many-to-one Setting**, where we have:

$$\begin{tikzcd}
 & & & & \mathrm{y} \\
\square \arrow[r] & \square \arrow[r] & \square \arrow[r] & \cdots \arrow[r] & \square \arrow[u] \\
\mathrm{x}_1 \arrow[u] & \mathrm{x}_2 \arrow[u] & \mathrm{x}_3 \arrow[u] & & \mathrm{x}_T \arrow[u]
\end{tikzcd}$$

  This is major used for the classification (categorizing the documents/events) and or positive/negative point of view.
  For image generation, the `GenAI` gets a series of text input (*e.g.* a frog fighting a duck) and give a image output.
  The ultimate goal is to approximate $p(\mathrm{y} \mid \mathrm{x}_1, \cdots, \mathrm{x}_\mathrm{T})$, where we have:

$$\mathbb{E}[\mathrm{y} \mid \mathrm{x}_1, \cdots, \mathrm{x}_\mathrm{T}] = g^{-1}(\mathbf{W}_{\mathrm{out}}^\intercal h_T),$$

  and the recurrence relationship as:

$$h_t = f(h_{t-1}, \mathrm{x}_t).$$

- **One-to-many Setting**, where we have:

$$\begin{tikzcd}
\mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 & & \mathrm{y}_T \\
\square \arrow[u] \arrow[r] & \square \arrow[u] \arrow[r] & \square \arrow[u] \arrow[r] & \cdots \arrow[r] & \square \arrow[u] \\
\mathrm{x} \arrow[u] & & & &
\end{tikzcd}$$

  Here, we think about the conditional text generation, such as having a trained model, and have an input of topic to generate a paragraph.
  Here, we want to consider that:

$$\begin{aligned}
&p(\mathrm{y}_1, \cdots, \mathrm{y}_T \mid \mathrm{x}) \\
&\qquad := \prod_{t=1}^{T} p(\mathrm{y}_t \mid \mathrm{x}) && \text{option \#1,} \\
&\qquad := \prod_{t=1}^{T} p(\mathrm{y}_t \mid \mathrm{y}_{t-1}, \cdots, \mathrm{y}_1, \mathrm{x}) && \text{option \#2.}
\end{aligned}$$

  The second was the auto-regression model, where we utilize the regressions from the previous models. In particular, the chain rule:

$$p(A, B, C) = p(C)p(C \mid B)p(A \mid B, C).$$

Note that this depend on if the model depends on **Markov property**, of if the current steps were impacted on the previous steps. *For the stock market example, if the fluctuation is random, use the first model. For the language generation or other related states, then we shall use the second model.*

- LLMs like ChatGPT is using the second model for the generation, as the texts has inter-word interactions.

In particular, for the second model, we have:

$$p(\mathrm{y}_1, \cdots, \mathrm{y}_T \mid \mathrm{x}) = \prod_{t=1}^{T} p(\mathrm{y}_t \mid \mathrm{y}_{t-1}, \cdots, \mathrm{y}_1, \mathrm{x}),$$

and so the expectation value is:

$$\mathbb{E}[\mathrm{y}_t \mid \mathrm{y}_{t-1}, \cdots, \mathrm{y}_1, \mathrm{x}] = g^{-1}(\mathbf{W}_{\text{out}}^{\intercal} h_t), \text{ and}$$
$$h_t = f(h_{t-1}, \mathrm{y}_{t-1}, \mathrm{x}).$$

Alternatively, x could just define h_0.

- **Many-to-many, Aligned Setting**, which is represented as:

$$\begin{tikzcd}
\mathrm{y}_1 & \mathrm{y}_2 & \mathrm{y}_3 & & \mathrm{y}_T \\
\square \arrow[u] \arrow[r] & \square \arrow[u] \arrow[r] & \square \arrow[u] \arrow[r] & \cdots \arrow[r] & \square \arrow[u] \\
\mathrm{x}_1 \arrow[u] & \mathrm{x}_2 \arrow[u] & \mathrm{x}_3 \arrow[u] & & \mathrm{x}_T \arrow[u]
\end{tikzcd}$$

This could be applied to *speech tagging*, and it is more restricted when there is a one-to-one correspondence between the speeches.
Here, the representing equation is:

$$p(\mathrm{y}_1, \cdots, \mathrm{y}_\mathrm{T} \mid \mathrm{x}_1, \cdots, \mathrm{x}_T) = \prod_{t=1}^{T} p(\mathrm{y}_t \mid \mathrm{y}_{[}t-1], \cdots, \mathrm{y}_1, \mathrm{x}_t, \cdots, \mathrm{x}_t).$$

In particular, the expectation is:

$$\mathbb{E}[\mathrm{y}_t \mid \mathrm{y}_{t-1}, \cdots, \mathrm{y}_1, \mathrm{x}_t, \mathrm{x}_1] = g^{-1}(\mathbf{W}_{\text{out}}^{\intercal} h_t).$$

The recurrence relationship becomes:

$$h_t = f(h_{t-1}, \mathrm{y}_{t-1}, \mathrm{x}_t).$$

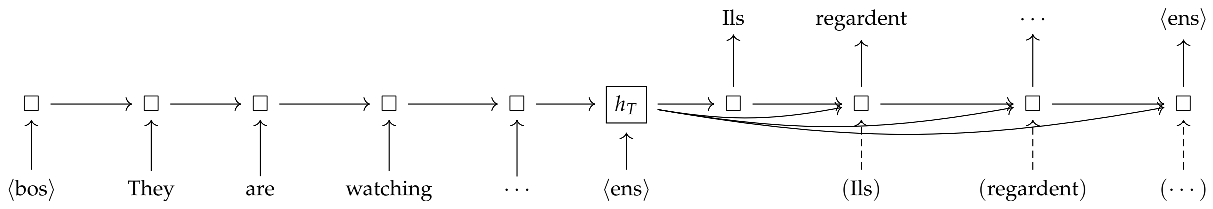
- **Many-to-many, not Aligned Setting**, which is represented as:

$$\begin{tikzcd}
 & & & \mathrm{y}_1 & \mathrm{y}_2 & & \mathrm{y}_{T'} \\
\square \arrow[r] & \square \arrow[r] & \cdots \arrow[r] & \square \arrow[u] \arrow[r] & \square \arrow[u] \arrow[r] & \cdots \arrow[r] & \square \arrow[u] \\
\mathrm{x}_1 \arrow[u] & \mathrm{x}_2 \arrow[u] & & \mathrm{x}_T \arrow[u] & & &
\end{tikzcd}$$

This can be the pattern for translation. The model would be:

$$p(\mathrm{y}_1, \cdots, \mathrm{y}_{T'} \mid \mathrm{x}_1, \cdots, \mathrm{x}_T) = \prod_{t'=1}^{T'} p(\mathrm{y}_{t'} \mid \mathrm{y}_1, \cdots, \mathrm{y}_{t'-1}, \mathrm{x}_1, \cdots, \mathrm{x}_T).$$

There will be a **encoder-decoder architecture**, with the example as follows from English to French.



Here, the encoder parameter is:

$$h_t = f(h_{t-1}, \mathrm{x}_t),$$

which f could be any of the three architectures. For the decoder parameter, we have:

$$s_{t'} = \psi(s_{t'-1}, y_{t-1}, h_T),$$

and the ψ could be any of the three architectures.
So, the expectation of the random variable is:

$$\mathbb{E}[\mathrm{y}_{t'} \mid \mathrm{y}_{t'-1}, \cdots, \mathrm{y}_1, \mathrm{x}_1, \cdots, \mathrm{x}_T] = g^{-1}(\mathbf{W}_{\text{out}}^{\intercal} s_t).$$

At the same time, this could be applied as to chat-bots in terms of applications.

Then, we think about **decoding/sampling sequences at test-time**.

- **Greedy search**:

$$\begin{aligned}\hat{y}_{t'} &= \arg\max_{\mathrm{y}_{t'}} \texttt{Categorical}\big(\mathrm{y}_{t'}, \texttt{softmax}(\mathbf{W}_{\text{out}}^{\intercal} s_{t'})\big) \\ &= \arg\max_{k\in[1,M]} \texttt{softmax}(\mathbf{W}_{\text{out}}^{\intercal} s_{t'}).\end{aligned}$$

  Note that the `Categorical` is the discrete distance over French words. *Note that this is okay*, but not optimal.

- **Exhaustive search**:

$$(\hat{y}_1, \cdots, \hat{y}_{T'}) = \arg\max_{\hat{y}_1, \cdots, \hat{y}_{T'}} p(\mathrm{y}_1, \cdots, \mathrm{y}_n \mid \mathrm{x}_1, \cdots, \mathrm{x}_T),$$

  and note that this will have the computational cost as $|V|^{T'}$.

- **Beam Search**:
  - Step 1, we greedily select the top M words, where M is the beam size.
  - Step 2, we perform greedy selection, thereafter for M independent sequences.
  - Step 3, we pick sequence with the highest probability.

$$\begin{tikzcd}
& \mathrm{y}_1 & & & \\
& \vdots & & & \\
& \mathrm{y}_3 \arrow[r] & \{\langle \mathrm{bos} \rangle, \mathrm{y}_3\} \arrow[r, "\text{highest probability}"] & \mathrm{y}_{84} \arrow[r] & \{\langle \mathrm{bos} \rangle, \mathrm{y}_3, \mathrm{y}_{84}\} \\
\langle \mathrm{bos} \rangle \arrow[uuur] \arrow[ur] \arrow[dr] \arrow[dddr] & \vdots & & & \\
& \mathrm{y}_{100} \arrow[r] & \{\langle \mathrm{bos} \rangle, \mathrm{y}_{100}\} \arrow[r, "\text{highest probability}"] & \mathrm{y}_{34} \arrow[r] & \{\langle \mathrm{bos} \rangle, \mathrm{y}_{100}, \mathrm{y}_{34}\} \\
& \vdots & & & \\
& \mathrm{y}_M & & &
\end{tikzcd}$$

V.5 Attention Models

An issue with the encoder/decoder model is that is has some **information bottleneck** when transformed between the encoder and decoder step. Here are some solutions:

- **The Attention Mechanism**, which allows the decoder to "attend" to different parts of the input sequence $\mathbf{x}_1, \cdots, \mathbf{x}_T$.
  We let $\mathcal{D} = \{(K_1, V_1), \cdots, (K_m, V_m)\} \subset \mathbb{R}^{D_k} \times \mathbb{R}^{D_v}$ be a "database" of m key-value pairs.
  Here, we de note a query as $q \in \mathbb{R}^{rk}$, and the attention over $\mathcal{D}$ is defined as:
  $$\texttt{Attention}(q, \mathcal{D}) := \sum_{m=1}^{M} \alpha(q, K_m) \cdot V_m,$$
  where $\sum_m \alpha(q, K_m) = 1$, and $0 \leq \alpha(q, K_m) \leq 1$.
  To compute the weights α, we use the `softmax` function to parameterized, as:
  $$\alpha(q, K_m) = \texttt{softmax}(q; K^\intercal q) = \frac{\exp\left\{K_m^\intercal q/\sqrt{D_k}\right\}}{\sum_{j=1}^{M} \exp\left\{K_j^\intercal q/\sqrt{D_k}\right\}}.$$
  The lookup process is by the *inner produce step*, but we have the $\sqrt{D_k}$ to normalize the result.
  - The question comes, why would the inner product account for the difference?
    We note that:
    $$(k-q)^2 = k^2 - 2kq + q^2,$$
    which would account for the difference.
- **Batched Computation**, in this model, we consider the attention as:
  $$\texttt{Attention}(Q; \mathcal{D}) = \texttt{softmax}\left(\frac{QK^\intercal}{\sqrt{D}}\right) V,$$

where $Q \in \mathbb{R}^{N \times D_k}$, so we have the first inner product as $N \times M$ dimensions, $V \in \mathbb{R}^{M \times D_v}$, and the `softmax` function is implemented row-wise.
The output would be $N \times D_v$ dimension.

Consider the sequence with attention:

- Let $\mathbf{c}$ be the context variable produced by the encoder.
- In the original formulation ($\sim$2014), we have:

$$\mathbf{c} := \mathbf{h}_T.$$

The problem is that $\mathbf{h}_T$ is to static, and does not allow dynamic context for $\mathbf{c}$.

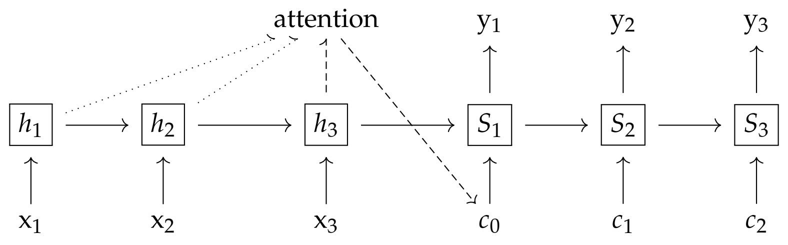
To do so, we consider the decoder with attention. Consider:

$$\mathbf{c}_{t'} = \sum_{t=1}^{T} \alpha(q = S_{t'-1}, K_t = \mathbf{h}_t)\mathbf{h}_t,$$

with:

$$S_{t'} := g(\mathrm{y}_{t'-1}, \mathbf{c}_{t'}, S_{t'-1}),$$

where g is the function that could be LSTM, GRU, or etc.



Then, we consider the **multi-head attention**. Here, we want to define multiple attention mechanisms, just like *ensemble method*.

- For example, when reading an article, we might need attention of different perspective (such as theme, style, and rhetorics).

The problem is that we do not want to define I-independent attentions models, but rather define I headers, $h_1, \cdots, h_i, \cdots, h_I$, where:

$$h_i = \texttt{Attention}(\mathbf{W}_1^a, \mathbf{W}_i^k K, \mathbf{W}_1^v v),$$

where $\mathbf{W}_i$'s are matrices of trainable parameters.
Given the computations of each h_i's, we will result in:

$$\texttt{Multi-Head Attention}(h_1, \cdots, h_I) = \mathbf{W}_O \cdot \begin{pmatrix} h_1, \\ \vdots \\ h_I \end{pmatrix},$$

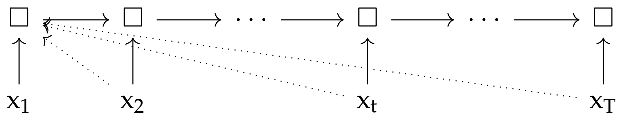
in which $\mathbf{W}_0 \in \mathbb{R}^{O \times (I \times D_v)}$, and the matrix is concatenation of the I headers.

The other is **self attention**. Given a sequence $(x_1, \cdots, x_T)$, the self attention model is:

$$\texttt{Self-Attention}(q = x_t, \mathcal{D} = \{(x_t, x_t)\}_{t=1}^{T}).$$

This works like the clustering of the data, as it is looking for similarity of the data.

Here, the main point is to compute the parallel:



V.6 Transformer Architecture

This is in correspondence to the issue with the RNN that the computation cannot be done in parallel.

- We have **unaligned**, sequence-to-sequence data, and we want to create the result.
- The data are $\tau_x = \{\tau_{x,1}, \cdots, \tau_{x,t}\}$, for example, an English sentence of length T, and $\tau_y = \{\tau_{y,1}, \cdots, \tau_{v,T'}\}$, for example, a French sentence length T', in which $\tau_{x,t}, \tau_{y,t'} \in \mathbb{N}^{\geq 0}$.

Again, we want to have the loss function:

$$\begin{aligned}
&\mathbb{KLD}\left[p^*(\tau_{y,1}, \cdots, \tau_{v,T'} \mid \tau_{x,1}, \cdots, \tau_{x,t}) \mid\mid p(\tau_{y,1}, \cdots, \tau_{v,T'} \mid \tau_{x,1}, \cdots, \tau_{x,t})\right] \\
\approx& \frac{1}{N} \sum_{n=1}^{N} -\log p(\tau_{y,n,1}, \cdots, \tau_{y,n,T'} \mid \tau_{x,n,1}, \cdots, \tau_{x,n,t}) \\
=& \frac{1}{N} \sum_{n=1}^{N} -\log \left(\prod_{t'=1}^{T'} p(\tau_{y,n,t'} \mid \tau_{x,n,1}, \cdots, \tau_{x,n,t}, \tau_{y,n,1}, \cdots, \tau_{y,n,t'-1}) \right) \\
=& \frac{1}{N} \sum_{n=1}^{N} \sum_{t'=1}^{T'} -\log \texttt{Categorical}(\tau_{y,n,t'} \mid \tau_{x,n,1}, \cdots, \tau_{x,n,t}, \tau_{y,n,1}, \cdots, \tau_{y,n,t'-1}) \\
=& \frac{1}{N} \sum_{n=1}^{N} \sum_{t'=1}^{T'} -\log \texttt{softmax}_{\tau_{y,n,t'}}\left(f(\tau_{y,n,t'} \mid \tau_{x,n,1}, \cdots, \tau_{x,n,t}, \tau_{y,n,1}, \cdots, \tau_{y,n,t'-1})\right)
\end{aligned}$$

Consider the embedding that:

$$E_x \in \mathbb{R}^{V \times D_x},$$

with each row corresponding to a vector representation of each word, that:

$$\texttt{one-hot}(\tau_{x,t})^{\intercal} E_x = E_x[\tau_{x,t} :]$$

- The first step is the input encoding that is having $x \in \mathbb{R}^{T \times D_x}$,
- Then, for the positional encoding, we have:

$$\tilde{x} = x + P, \text{ where } P \in [-1, 1]^{T \times D_x} \text{ that is positional encoding,}$$

and the odd and even cases are:

$$P_{t,2d} = \sin\left(\frac{t}{10000^{2d/D_x}}\right) \qquad P_{t,2d+1} = \cos\left(\frac{t}{10000^{2d/D_x}}\right).$$

- For the multihead self-attention model, we have:

$$A_e = \texttt{softmax}\left(\frac{(\tilde{x}W_{q,s})(\tilde{x}W_{q,s})^\intercal}{\sqrt{D_{e,k}}}\right)(\tilde{x}W_{w,e}).$$

- Then, we have a skip connection and layer norm for the add & norm:

$$\texttt{LayerNorm}(\tilde{x} + \texttt{MHSA}(\tilde{x})) =: \tilde{x_1}.$$

- We continue with positionwise feed-forward NN:

$$\texttt{PWFF}(\tilde{x_1}) = \begin{pmatrix} \texttt{ReLU}(\tilde{x_{1,1}}, W_1)W_2 \\ \vdots \\ \texttt{ReLU}(\tilde{x_{1,T}}, W_1)W_2 \end{pmatrix} \in \mathbb{R}^{T\times D_x}.$$

- Eventually, we have another add & norm layer, that is:

$$\texttt{LayerNorm}(\tilde{x} + \texttt{PWFF}(\tilde{x_1})) =: \tilde{x_2}.$$

Note that the **encoder** repeat the steps from `MHSA` till the last add & norm for all L layers.

The thing is that you can change dimensions for the `MHSA` and `PWFF` parameters, but the dimension of the input/output must be the same due to the skip-connection layers.

For the encoder, we have:

$$\tilde{x}_{L_{\text{enc}},z} = \texttt{Encoder}(\tau_{x,n,1}, \tau_{x,n,2}, \cdots, \tau_{x,n,T}),$$

where L_{enc} is the number of layers and z is the index of sub-layer.

For the decoder, we have:

$$\texttt{Decoder}(\tilde{x}_{L_{\text{enc}},z}, \tau_{x,n,1}, \tau_{x,n,2}, \cdots, \tau_{x,n,T}).$$

Here, we can consider the masking as to hide from the decoder all target elements at the current time step t' and beyond.

$$\tau_y = \{\underbrace{\tau_{y,1}, \cdots, \tau_{y,t'-1}}_{\text{revealed}}, \underbrace{\tau_{y,t'}, \cdots, \tau_{y,T}}_{\text{hidden}}\}.$$

Consider the matrix:

$$\tilde{Y}_{t'-1} = Y_{t'-1} + P_y,$$

where all the rows below $t'-1$ are encoded with non-information token `<PAD>`. This is the step called **masked multi-head attention**.

Then, the most step is the Encoder-decoder attention, where we have:

$$A_e = \texttt{softmax}\left(\frac{(\tilde{Y}_{t',1}W_{q,e})(\tilde{X}_{L_{\text{enc}},2}W_{e,v})^\intercal}{\sqrt{D_{e,k}}}\right)(\tilde{X}_{L_{\text{enc}},2}W_{e,v}).$$

Again, we have the skip connection and the layer norm as:

$$\tilde{Y}_{t',2} = \texttt{LayerNorm}\big(\tilde{Y}_{t',1} + \texttt{MHA}(Q = \tilde{Y}_{t'}, K = \tilde{X}_{L_{\text{enc}},2}, V = \tilde{X}_{L_{\text{enc}},2})\big).$$

Still, we have the positionwise feedforward, we have:

$$\tilde{Y}_{t',3} = \texttt{LayerNorm}\big(\tilde{Y}_{t',2} + \texttt{PWFF}(\tilde{Y}_{t',2})\big).$$

Then, we can think of the model training loss of the unaligned sequence-to-sequence loss as:

$$\ell(\theta; \{\tau_{x,n}, \tau_{y,n}\}_{n=1}^N) = \frac{1}{N}\sum_{n=1}^N \sum_{t'=1}^T -\log \texttt{softmax}_{\tau_{y,n,t'}}\big(\texttt{Transformer}(\tau_{y,n,t'-1}, \cdots, \tau_{y,n,T})\big).$$

VI Unsupervised Learning and Deep Generative Model

VI.1 Unsupervised Learning with Neural Networks

Generally, we consider the $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ to learn the $x_n \mapsto y_n$ as supervised learning.
For unsupervised learning, we consider $\{x_n\}_{n=1}^N$, and we want to learn $x_n \mapsto z_n$, which is a hidden structure or latent variable.

An example could be the **clustering problem**, we consider having data $\{x_n\}$ and trying to cluster them into pieces z_i's.

First, we need some unsupervised loss function:

$$\ell(\theta; \mathcal{D} = \{x_n\}_{n=1}^N = \frac{1}{N}\sum_{n=1}^N \ell(x_n),$$

and with autoencoding, we have:

$$\frac{1}{N}\sum_{n=1}^N \ell(x_n, g(h_n)) = \frac{1}{N}\sum_{n=1}^N (x_n - \hat{x}_n)^2.$$

We consider the transformation:

$$\begin{tikzcd} \hat{x}_n \\ h_n \arrow[u, "g"] \\ x_n \arrow[u, "f"'] \end{tikzcd}$$

Here, we want to f as the encoder and g as the decoder.

Note that we want:

$$\dim(h_n) < \dim(x_n).$$

When f and g are invertible, then we basically have a **PCA** (Principle component analysis).

- The `Autoencoders` is the NNs trained to predict their own input. The purpose is to find a new compressed feature space:

  $$[x] \longrightarrow [h_1] \longrightarrow \cdots \longrightarrow [h_L] \longrightarrow \mathbb{E}[x \mid x]$$

  Note that there will be some information bottleneck in the $h.$ levels to constraint information flow to the output.

Here, we have:

$$\mathbb{E}[x \mid x] = g^{-1}(\mathbf{W}_L^\intercal \mathbf{h}_L),$$

which is the same as the traditional hidden layer. For $x \in \mathbb{R}$, we have:

$$-\log p(x) = -\log \mathcal{N}(x, \boldsymbol{\mu} = \mathbf{W}_L^\intercal \mathbf{h}_L, \sigma_0^2 \operatorname{Id}).$$

For $x \in [0,1]^p$, we have:

$$-\log p(x) = -\log \prod_{d=1}^{D} \texttt{Bernoulli}(x_d, \texttt{logistic}(\mathbf{W}_{L,d}^\intercal \mathbf{h}_L)).$$

Then, we have the KLD as:

$$\mathbb{KLD}[\mathbb{P}(x) \mid\mid p(x)] \approx \frac{1}{N}\sum_{n=1}^{N} -\log p(x_n) = \frac{1}{N}\sum_{n=1}^{N} -\log\left[\prod_{d=1}^{D} \texttt{Bernoulli}(x_d, \texttt{logistic}(\mathbf{W}_{L,d}^\intercal \mathbf{h}_L))\right].$$

As a historical note, AEs were predominantly used to build deep NNs by first doing supervised learning.

Then, there are also variational autoencoders, which we have:

$$[x] \longrightarrow [z] \longrightarrow \mathbb{E}[x \mid z]$$

The goal is to build a generative model of the data distribution $\mathbb{P}(x)$, where we have:

$$z \sim p(z) \text{ and } x \sim p(x \mid z).$$

To train a VAE, we do:

$$\begin{aligned}
\mathbb{KLD}[\mathbb{P}(x) \mid\mid p(x)] &\approx \frac{1}{N}\sum_{n=1}^{N} -\log p(x_n) \\
&= -\frac{1}{N}\sum_{n=1}^{N} -\log\left[\int_{z_n} p(x_n \mid z_n) p(z_n) dz\right] \\
&= \frac{1}{N}\sum_{n=1}^{N} -\log\left[\int_{z_n} \frac{q(z_n; \psi(x_n))}{q(z_n; \psi(x_n))} \cdot p(x_n \mid z_n) p(z_n) dz_n\right] \\
&\leq \frac{1}{N}\sum_{n=1}^{N} \int_{z_n} q(z_n; \psi(x_n)) \left[-\log p(x_n \mid z_n) - \log p(z_n) + \log q(z_n; \psi_n)\right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{q(z_n)}\left[-\log p(x_n \mid z_n)\right] + \mathbb{E}_{q(z_n)}\left[\log q(z_n;\psi(x_n)) - \log p(z_n)\right] \\
&= \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{q(z_n)}[-\log p(x_n \mid z_n)] + \mathbb{KLD}[q(z_n;\psi(x_n)) \mid\mid p(z_n)] \\
&= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{S}\sum_{s=1}^{S} -\log p(x_n \mid \hat{z}_{n,s})\right) + \mathbb{KLD}[q(z_n;\psi(x_n)) \mid\mid p(z_n)].
\end{aligned}
$$

Hence, we have the structure as:

$$\begin{tikzcd}
{[x_n]} \arrow[r, dash] & \bullet \arrow[r] \arrow[d, dash] & q(z_n \mid x_n) \arrow[r] & \hat{z}_{n,s} \arrow[r, dash] & \bullet \arrow[r] \arrow[d, dash] & \mathbb{E}[x \mid z] \\
& \text{Encoder} & & & \text{Decoder} &
\end{tikzcd}$$

The goal is to have sample $\hat{z} \sim \mathcal{N}(\mu, \sigma^2)$, and for the implementation in local-scale form:

$$\mathcal{E} \sim \mathcal{N}(0,1), \qquad \hat{z} = \mu + \sigma \cdot \mathcal{E}, \qquad \hat{z} \sim \mathcal{N}(\mu, \sigma^2).$$

Here, we have the $\hat{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}$:

$$\begin{tikzcd}
& & \mu \arrow[dr] & & & \\
{[x_n]} \arrow[r, dash] & \texttt{enc} \arrow[ur] \arrow[dr] & & {[\hat{z}]} \arrow[r, dash] & \texttt{dec} \arrow[r] & \mathbb{E}[x \mid z] \\
& & \sigma \arrow[ur] & & & \\
& & & \mathcal{E} \sim \mathcal{N}(0,1) \arrow[uu] & &
\end{tikzcd}$$

In general, we can use the reparametrization trick that:

$$\ell = \frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{S}\sum_{s=1}^{S} -\log p\left(x_n \mid \hat{z}_n = r\big(\psi(x_n); \hat{\varepsilon}_{n,s}\big)\right)\right) + \mathbb{KLD}[q(z_n;\psi(x_n)) \mid\mid p(z_n)].$$

After training, the sample need data via:

$$\hat{z} \sim p(\hat{z}) \qquad \hat{x} \sim p(x, \hat{z}).$$

VI.2 Deep Generative Model

Models based on NNs that can generate data via a probabilistic formalism, and VAEs are just one example. There are other models:

- Generative Adversarial Network (GAN)
- Normalizing Flows
- Diffusion Models.

In these models, there will be "noise" and there are neural-net transformation.

A deep VAE would have multiple stochastic latent variables $z_1, \cdots, z_L$, where we draw samples of x via $z_L \sim p(z_L)$, $z_{L-1} \sim p(z_{L-1} \mid z_L)$. In general, we have:

$$z_l \sim p(z_l \mid z_{l-1}) \text{ and } x \sim p(x \mid z_1).$$

VI.3 Diffusion Model

In a graph, the model is:

$$\begin{tikzcd}
 & \mu_L \arrow[dr, dash] & & \mu_{L-1} \arrow[dr, dash] & & & & \\
{[x]} \arrow[ur] \arrow[dr] & & z_L \arrow[ur] \arrow[dr] & & z_{L-1} \arrow[r] & \cdots \arrow[r] & z_1 \arrow[r] & \mathbb{E}[x \mid z_1, \cdots, z_L] \\
 & \sigma_L \arrow[ur, dash] & & \sigma_{L-1} \arrow[ur, dash] & & & &
\end{tikzcd}$$

Here, we consider the de-noising of diffusion models, we have:

- Motivation to single stoch layer VAE are hard, and not powerful enough, and multi stoch layers are hard to train.
- The core idea is to use Neural networks only for the decoder patron, use diffusion processes as encoder.

Consider an input image X_0, we want to have noise $X_T \sim \mathcal{N}(0, \mathrm{Id})$.

- The distribution would just be:

$$\begin{tikzcd}
X_0 \arrow[r] & X_1 \arrow[r] \arrow[d, dashed, dash] & X_2 \arrow[r] \arrow[d, dashed, dash] & \cdots \arrow[r] & X_T \sim \mathcal{N}(0, \mathrm{Id}) \\
 & q(X_1 \mid X_0) & q(X_2 \mid X_1) & &
\end{tikzcd}$$

Here, let $X_0 \in \mathbb{R}$ be the data and T be the maximum number of time steps. The model is:

$$p(X_0) = \int_{X_1, \cdots, X_T} p(X_0, X_1, \cdots, X_T) dX_1 \cdots X_T.$$

Here, they all have the same dimensionally $x_t \in \mathbb{R}^{D_1 \times D_2}$, and X_i's are the latent variables akin to z in a VAE.

Hence, for the forward process, we have:

$$q(X_1, \cdots, X_T) = \prod_{t=1}^{T} q(X_t \mid X_{t-1}),$$

and we have the distribution as:

$$q(X_t \mid X_{t-1}) = \mathcal{N}(X_t; \mu = \sqrt{1 - \beta_t} X_t, \Sigma_t = B_t \, \mathrm{Id}).$$

Hence, we have $\{\beta_1, \cdots, \beta_T\} \in (0,1)^T$ is the *variance schedule* that determines how quickly we fo from data to noise.

Ho et al. set $\beta_1 = 10^{-4}$, $\beta_T = 0.02$, and the rests are set with linear interpolation.

Due to the diffusion sets being normal distributions, we can directly compute $q(X_t \mid X_0)$ for arbitrary t:

$$q(X_t \mid X_0) = \mathcal{N}(\mu_t = \sqrt{\bar{\alpha}_t}X_0, \Sigma_t - (1-\bar{\alpha}_t)\,\mathrm{Id}), \qquad \text{where } \bar{\alpha}_t = \prod_{t'=1}^{t}(1-\beta_{t'}).$$

Then, we consider the **reverse process** from the noise to data. We need to learn a "de-noiser" that inverts the diffusion process:

$$p_\theta(X_{t-1} \mid X_t) := \mathcal{N}\big(X_{t-1}, \mu_\theta(X_t,t), \Sigma_t(X_t,t)\big).$$

In general, these can be neural nets that take in X_t and t and produce μ_t, Σ_t.

To **train the denoiser**, we have:

$$\begin{aligned}
\mathbb{KLD}[\mathbb{P}(X_0) \mid\mid p_0(X_0)] &\approx \frac{1}{N}\sum_{n=1}^{N} -\log p_\theta(X_n) \\
&= \frac{1}{N}\sum_{n=1}^{N} -\log \int_{X_1,\cdots,X_T} p_\theta(X_0,\cdots,X_T)dX \\
&= \frac{1}{N}\sum_{n=1}^{N} -\log \int_{X_1,\cdots,X_T} p_\theta(X_t)\prod_{t=1}^{T} p_\theta(X_{t-1} \mid X_t)dX \\
&= \frac{1}{N}\sum_{n=1}^{N} -\log \int_{X_1,\cdots,X_T} \frac{q(X_1,\cdots,X_T \mid X_0)}{q(X_1,\cdots,X_T \mid X_0)} p_\theta(X_t)\prod_{t=1}^{T} p_\theta(X_{t-1} \mid X_t)dX \\
&\leq \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{q(X_1,\cdot,X_T)}\left[-\log \frac{p_\theta(X_t)\prod_{t=1}^{T} p_\theta(X_{t-1} \mid X_t)}{\prod_{t=1}^{T} q(X_1,\cdots,X_T \mid X_0)}\right] \\
&= \mathbb{E}_{q(X_1,\cdot,X_T \mid X_0)}\left[-\log p(X_T) - \sum_{t=1}^{T}\log\frac{p_\theta(X_{t=1} \mid X_t)}{q(X_t \mid X_{t-1})}\right].
\end{aligned}$$