Deep Learning: Loss Function and Regularization

Alvaro Soto

Computer Science Department, PUC

- What is a "loss function"? what is its role?
- What is the most typical classification loss?

Machine learning problem

$$f^* = \arg\min_{f \in \mathcal{H}} \mathcal{L}(f(x)) = \arg\min_{f \in \mathcal{H}} \int_{x_i \in T} \mathcal{L}(f(x_i)) d_T$$

We usually approximate this using a training set Tr:

$$f^* \approx f^*_{Tr} = \arg\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{x_i \in Tr}^{N} \mathcal{L}(f(x_i))$$

$\mathcal{H}$: hypothesis space.

$\mathcal{L}$: loss function

f^*: optimal hypothesis in $\mathcal{H}$ under $\mathcal{L}$.

Generic machine learning loss

$$f^* \approx f^*_{Tr} = \underset{f \in \mathcal{H}}{\arg\min} \frac{1}{N} \sum_{x_i \in Tr}^{N} \mathcal{L}(f(x_i))$$

Supervised learning

$$f^*_{Tr} = \underset{f \in \mathcal{H}}{\arg\min} \tfrac{1}{N} \sum_{x_i, y_i \in Tr}^{N} \mathcal{L}(f(x_i), y_i)$$

$$f^*_{Tr} = \arg\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{x_i, y_i \in Tr}^{N} \mathcal{L}(f(x_i), y_i)$$

Let's assume that f depends on parameters W, then we need to minimize the following loss:

$$L(W) = \frac{1}{N} \sum_{x_i, y_i \in Tr}^{N} L_i(f(x_i, W), y_i)$$

What is the loss function that we use in the AI class to optimize the parameters of a feedforward fully connected NN ?

$$L(W) = \frac{1}{N} \sum_{x_i, y_i \in Tr}^{N} ||\hat{y_i} - y_i||_2^2$$

We can pose this loss function as a Maximum Likelihood Estimation or MLE problem (MLE?). Actually, to keep minimizing we use the negative log-likelihood (NLL).

Considere the following conditional probability estimation problem:

$$L(W) = -\sum_{x_i, y_i \in Tr}^{N} \log p(y_i | x_i, W)$$

Let's assume a Gaussian deviation in the estimation $\hat{y_i} = f(x_i, W)$, with a fixed standard deviation σ:

$$L(W) = -\sum_{x_i, y_i \in Tr}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\hat{y_i} - y_i)^2}{2\sigma^2}} = -\sum_{x_i, y_i \in Tr}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} + \sum_{x_i, y_i \in Tr}^{N} \frac{(\hat{y_i} - y_i)^2}{2\sigma^2}$$

$$L(W) = \frac{N}{2} \log(2\pi) + N \log \sigma + \sum_{x_i, y_i \in Tr}^{N} \frac{(\hat{y_i} - y_i)^2}{2\sigma^2}$$

In this case, minimizing MSE is equivalent to minimizing NLL. Same loss value?

- Negative log-likelihood is a common loss function used in ML problems.

Loss Function

- Negative log-likelihood is a common loss function used in ML problems.
- In particular, it is well suited for cases where we need to fit continuous variables.

Loss Function

- Negative log-likelihood is a common loss function used in ML problems.
- In particular, it is well suited for cases where we need to fit continuous variables.
- This is because it assigns larger penalty to larger errors.

Loss Function

- Negative log-likelihood is a common loss function used in ML problems.
- In particular, it is well suited for cases where we need to fit continuous variables.
- This is because it assigns larger penalty to larger errors.
- However in the case of classification, a wrong class prediction does not depend on a larger difference to a target value.

- Negative log-likelihood is a common loss function used in ML problems.
- In particular, it is well suited for cases where we need to fit continuous variables.
- This is because it assigns larger penalty to larger errors.
- However in the case of classification, a wrong class prediction does not depend on a larger difference to a target value.
- Classification is usually implemented using a binary encoding (right/wrong prediction). The goal is not to be close, but to predict the right output class.

Loss Function

- We demonstrated that MSE is equivalent to minimizing log-likelihood under a Gaussian deviation in the prediction.

Loss Function

- We demonstrated that MSE is equivalent to minimizing log-likelihood under a Gaussian deviation in the prediction.
- In deep learning, we will typically solve a multiclass classification problem using a sigmoidal or softmax output (logistic output).

Loss Function

- We demonstrated that MSE is equivalent to minimizing log-likelihood under a Gaussian deviation in the prediction.
- In deep learning, we will typically solve a multiclass classification problem using a sigmoidal or softmax output (logistic output).
- We will see that in this case, using cross-entropy as a loss function allows us to maximize the likelihood of a multinomial distribution.

- We demonstrated that MSE is equivalent to minimizing log-likelihood under a Gaussian deviation in the prediction.
- In deep learning, we will typically solve a multiclass classification problem using a sigmoidal or softmax output (logistic output).
- We will see that in this case, using cross-entropy as a loss function allows us to maximize the likelihood of a multinomial distribution.
- Cross-Entropy?.

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:
  - $I(x)$: information of a message x.

Loss Function

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:
  - $I(x)$: information of a message x.
  - $H(p(x))$: entropy of a probability distribution p(x).

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:
  - $I(x)$: information of a message x.
  - $H(p(x))$: entropy of a probability distribution p(x).
- In information theory, the concept of information aims to describe the *surprise* of an event. Low probability events carry more information. Specifically, $I(x) \propto \frac{1}{p(x)}$.

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:
  - $I(x)$: information of a message x.
  - $H(p(x))$: entropy of a probability distribution p(x).
- In information theory, the concept of information aims to describe the *surprise* of an event. Low probability events carry more information. Specifically, $I(x) \propto \frac{1}{p(x)}$.
- Then, Entropy H measures the average (expected) information per symbol x in a message: $H(x) = -E_{x \sim p} \log p(x)$, where symbols x are generated by distribution $p(x)$.

Cross-entropy is defined in the context of information theory as the overhead of transmiting a signal defined by a pdf $p(x)$ using an encoding that considers a pdf $q(x)$.

- To understand this, we should first briefly review two relevant concepts from information theory:
  - $I(x)$: information of a message x.
  - $H(p(x))$: entropy of a probability distribution p(x).
- In information theory, the concept of information aims to describe the *surprise* of an event. Low probability events carry more information. Specifically, $I(x) \propto \frac{1}{p(x)}$.
- Then, Entropy H measures the average (expected) information per symbol x in a message: $H(x) = -E_{x \sim p} \log p(x)$, where symbols x are generated by distribution $p(x)$.
- Using entropy as a reference, it is possible to obtain optimal codifications to store and/or transmit information (Shannon magic!).

Loss Function

- So, we have $H(x) = -E_{x \sim p} \log p(x)$. This can help us to build optimal encodings for sequences of symbols x generated according to $p(x)$.

[1] While information measures entropy using bits, i.e. $\log_2$, in the context of machine learning one typically uses $\log_e$ (nats).

- So, we have $H(x) = -E_{x\sim p} \log p(x)$. This can help us to build optimal encodings for sequences of symbols x generated according to $p(x)$.
- However, sometimes we do not know $p(x)$. In these cases, we can use an auxiliar pdf $q(x)$ to encode a message leading to a suboptimal encoding. How suboptimal?

[1]While information measures entropy using bits, i.e. $\log_2$, in the context of machine learning one typically uses $\log_e$ (nats).

- So, we have $H(x) = -E_{x \sim p} \log p(x)$. This can help us to build optimal encodings for sequences of symbols x generated according to $p(x)$.
- However, sometimes we do not know $p(x)$. In these cases, we can use an auxiliar pdf $q(x)$ to encode a message leading to a suboptimal encoding. How suboptimal?
- In the context of information theory, cross-entropy allows us to quantify this inneficiency.

[1] While information measures entropy using bits, i.e. $\log_2$, in the context of machine learning one typically uses $\log_e$ (nats).

- So, we have $H(x) = -E_{x \sim p} \log p(x)$. This can help us to build optimal encodings for sequences of symbols x generated according to $p(x)$.
- However, sometimes we do not know $p(x)$. In these cases, we can use an auxiliar pdf $q(x)$ to encode a message leading to a suboptimal encoding. How suboptimal?
- In the context of information theory, cross-entropy allows us to quantify this inneficiency.
- Specifically, cross-entropy [1] is defined as:

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

[1] While information measures entropy using bits, i.e. $\log_2$, in the context of machine learning one typically uses $\log_e$ (nats).

- So, we have $H(x) = -E_{x \sim p} \log p(x)$. This can help us to build optimal encodings for sequences of symbols x generated according to $p(x)$.
- However, sometimes we do not know $p(x)$. In these cases, we can use an auxiliar pdf $q(x)$ to encode a message leading to a suboptimal encoding. How suboptimal?
- In the context of information theory, cross-entropy allows us to quantify this inneficiency.
- Specifically, cross-entropy [1] is defined as:

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- I.e. cross-entropy considers that symbols are generated by $p(x)$ (expected value is over $p(x)$), but the corresponding information content is given by $q(x)$ (argument in the log is given by $q(x)$).

[1] While information measures entropy using bits, i.e. $\log_2$, in the context of machine learning one typically uses $\log_e$ (nats).

Loss Function

$$H(p(x), q(x)) = -E_{x\sim p} \log q(x)$$

- As a relevant consequence cross-entropy can be used as a proxy to quantify the distance between distributions $p(x)$ and $q(x)$.

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- As a relevant consequence cross-entropy can be used as a proxy to quantify the distance between distributions $p(x)$ and $q(x)$.
- Actually, cross-entropy is closely related to KL-divergence, a common function used to score distance between distributions:

$$KL(p(x), q(x)) = H(p(x), q(x)) - H(p(x))$$

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- As a relevant consequence cross-entropy can be used as a proxy to quantify the distance between distributions $p(x)$ and $q(x)$.
- Actually, cross-entropy is closely related to KL-divergence, a common function used to score distance between distributions:

$$KL(p(x), q(x)) = H(p(x), q(x)) - H(p(x))$$

- Then, optimizing cross-entropy is equivalent to optimizing KL-divergence, why?.

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- As a relevant consequence cross-entropy can be used as a proxy to quantify the distance between distributions $p(x)$ and $q(x)$.
- Actually, cross-entropy is closely related to KL-divergence, a common function used to score distance between distributions:

$$KL(p(x), q(x)) = H(p(x), q(x)) - H(p(x))$$

- Then, optimizing cross-entropy is equivalent to optimizing KL-divergence, why?.
- In the context of machine learning, our model outputs prediction $q(x)$ as an estimation of $p(x)$.

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- As a relevant consequence cross-entropy can be used as a proxy to quantify the distance between distributions $p(x)$ and $q(x)$.
- Actually, cross-entropy is closely related to KL-divergence, a common function used to score distance between distributions:

$$KL(p(x), q(x)) = H(p(x), q(x)) - H(p(x))$$

- Then, optimizing cross-entropy is equivalent to optimizing KL-divergence, why?.
- In the context of machine learning, our model outputs prediction $q(x)$ as an estimation of $p(x)$.
- Therefore, we can use cross-entropy as a loss function.

But, how can we calculate the cross-entropy?

$$H(p(x), q(x)) = -E_{x\sim p} \log q(x)$$

- We know $q(x)$, our model estimation, but we do not know $p(x)$, what we really want to estimate.

Loss Function

But, how can we calculate the cross-entropy?

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- We know $q(x)$, our model estimation, but we do not know $p(x)$, what we really want to estimate.
- The good news is that while we do not know $p(x)$, we do have samples from it. Samples?

But, how can we calculate the cross-entropy?

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- We know $q(x)$, our model estimation, but we do not know $p(x)$, what we really want to estimate.
- The good news is that while we do not know $p(x)$, we do have samples from it. Samples?
- Training set corresponds to samples from $p(x)$, the true underlying distribution.

But, how can we calculate the cross-entropy?

$$H(p(x), q(x)) = -E_{x \sim p} \log q(x)$$

- We know $q(x)$, our model estimation, but we do not know $p(x)$, what we really want to estimate.
- The good news is that while we do not know $p(x)$, we do have samples from it. Samples?
- Training set corresponds to samples from $p(x)$, the true underlying distribution.
- We can use these samples to calculate the cross-entropy. Let's see an example.

What is the cross-entropy for a binary classification problem?

- Let's consider that the model's output $\hat{y} = q(x)$, corresponds to the estimation that the probability of target output is $y = 1$.
- Then the probability of $y = 0$ is given by $1 - q(x)$.
- Using the training set, the cross-entropy is given by:

$$
\begin{aligned}
H(p(x), q(x)) &= -E_{x \sim p} \log q(x) \\
H(p(x), q(x)) &= \sum_{x_i, y_i \in Tr}^{N} [y_i \log q(x_i) + (1 - y_i) \log(1 - q(x_i))]
\end{aligned}
$$

- Notice that here we assume that the samples in Tr are coming from $p(y|x)$. So, when we minimize $H(p(x), q(x))$, we look for function $q(y|x)$ that resembles $p(y|x)$ (why?).

- Cross-entropy is more suitable than MSE for classification problems. Logistic output heavily penalizes cases where you are predicting the wrong output class (you're either right or wrong, unlike real-valued predictions where the goal is to be close).

- Cross-entropy is more suitable than MSE for classification problems. Logistic output heavily penalizes cases where you are predicting the wrong output class (you're either right or wrong, unlike real-valued predictions where the goal is to be close).
- It is possible to demonstrate that cross-entropy in combination with a softmax activation function lead to maximizing the likelihood of a multinomial distribution.

Loss Function

- Cross-entropy is more suitable than MSE for classification problems. Logistic output heavily penalizes cases where you are predicting the wrong output class (you're either right or wrong, unlike real-valued predictions where the goal is to be close).
- It is possible to demonstrate that cross-entropy in combination with a softmax activation function lead to maximizing the likelihood of a multinomial distribution.
- Furthermore, in combination with soft-max outputs, cross-entropy loss is easier to optimize than squared loss leading to faster training as well as improved generalization.

- Cross-entropy is more suitable than MSE for classification problems. Logistic output heavily penalizes cases where you are predicting the wrong output class (you're either right or wrong, unlike real-valued predictions where the goal is to be close).
- It is possible to demonstrate that cross-entropy in combination with a softmax activation function lead to maximizing the likelihood of a multinomial distribution.
- Furthermore, in combination with soft-max outputs, cross-entropy loss is easier to optimize than squared loss leading to faster training as well as improved generalization.
- In the context of deep learning, cross-entropy is the favorite loss function for classification problems.

Loss Function and Regularization

Regularization

- Regularization is the action of constraining the hypothesis space to foster *"good"* solutions. Actually, the training data itself is the main source to constraint or "regularize" the solution.
- The most basic form of regularization is given by limiting the capacity of the model (limiting capacity?).
- Actually, deep learning models are usually BIG size, so we need to find alternatives to constraint (regularize) the hypothesis space, so we can avoid overfitting, local minima, and other evil problems.

- The most common form of regularization is to add a norm penalty $\Omega(W)$ over the model parameters (weights):

$$L(W) = \mathcal{L}(f(x), y; W) + \alpha \, \Omega(W)$$

- α is a hyperparameter that weights the relative contribution of the norm penalty Ω.
- Why adding a norm penalty is a convenient regularization?

Regularization

- L^2 is the most common norm penalty over the model parameters, a.k.a., weight decay:

$$\Omega(W) = \frac{1}{2}||W||_2^2 = \frac{1}{2}W^T\ W$$

- Then the loss is given by:

$$L(W) = \mathcal{L}(f(x), y; W) + \frac{\alpha}{2}\ W^T\ W$$

- Then the gradient respect to weight w is given by:

$$\nabla_w L(W) = \nabla_w \mathcal{L}(f(x), y; W) + \alpha\ w$$

- Weight update:

$$w^{new} = w^{old} - \eta\,(\nabla_w \mathcal{L}(f(x), y; W) + \alpha\ w)$$

$$w^{new} = (1 - \eta\alpha)w^{old} - \eta(\nabla_w \mathcal{L}(f(x), y; W)$$

- By adding the penalty, weights shrink by a constant factor at each update.

Why a bias towards small weights is a good idea?

Why a bias towards small weights is a good idea?

- Large weights are usually not a good idea, they are sensitive to small noise in the input data. They amplify noise so the output changes a lot with small variations in the input.

Why a bias towards small weights is a good idea?

- Large weights are usually not a good idea, they are sensitive to small noise in the input data. They amplify noise so the output changes a lot with small variations in the input.
- By distributing the model on a large set of small parameter values, we foster a model with better generalization, avoiding to overfit by memorizing relevante correlations between parameters and inputs.

Why a bias towards small weights is a good idea?

- Large weights are usually not a good idea, they are sensitive to small noise in the input data. They amplify noise so the output changes a lot with small variations in the input.
- By distributing the model on a large set of small parameter values, we foster a model with better generalization, avoiding to overfit by memorizing relevante correlations between parameters and inputs.
- In deep learning one usually adds a weight regularization at each or selected layer.

Why a bias towards small weights is a good idea?

- Large weights are usually not a good idea, they are sensitive to small noise in the input data. They amplify noise so the output changes a lot with small variations in the input.
- By distributing the model on a large set of small parameter values, we foster a model with better generalization, avoiding to overfit by memorizing relevante correlations between parameters and inputs.
- In deep learning one usually adds a weight regularization at each or selected layer.

Regularization

Why a bias towards small weights is a good idea?

- Large weights are usually not a good idea, they are sensitive to small noise in the input data. They amplify noise so the output changes a lot with small variations in the input.
- By distributing the model on a large set of small parameter values, we foster a model with better generalization, avoiding to overfit by memorizing relevante correlations between parameters and inputs.
- In deep learning one usually adds a weight regularization at each or selected layer.

Further regularizations

- Several alternative norm-penalty function can be used, ex. L_1.
- H. Lobel et al., *CompactNets: Compact Hierarchical Compositional Networks for Visual Recognition*.

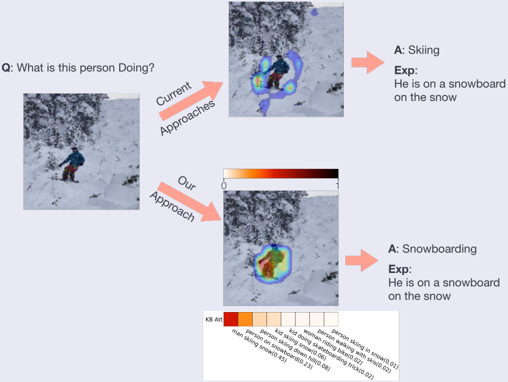
Auxiliary Functions as a Form of Regularization (one of my favorite)

Auxiliary functions in action recognition

J. Ji et al., *End-to-End Joint Semantic Segmentation of Actors and Actions in Video*, ECCV 2018.

Auxiliary Functions as a Form of Regularization (one of my favorite)

Constraining the internal representation of the model



B. Zhang et al., *Interpretable Visual Question Answering by Visual Grounding from Attention Supervision Mining*, WACV 2018.

F. Riquelme et al., *Explaining VQA predictions using visual grounding and a knowledge base* , ICCV 2019.