NPFL114, Lecture 04

Convolutional Networks

Milan Straka

Dropout Implementation

```
def dropout(input, rate=0.5, training=False):
    def do_inference():
        return tf.identity(input)

    def do_train():
        random_noise = tf.random_uniform(tf.shape(input))
        mask = tf.cast(tf.less(random_noise, rate),
                       tf.float32)
        return x * mask / (1 - rate)

    if training == True:
        return do_train()
    if training == False:
        return do_inference()
    return tf.cond(training, do_train, do_inference)
```

Dropout Effect

Dropout Effect

Convergence

The training process might or might not converge. Even if it does, it might converge slowly or quickly.

There are *many* factors influencing convergence and its speed, we now discuss three of them.

Convergence – Saturating Non-linearities

Reference: Image from https://isaacchanghau.github.io/images/deeplearning/activationfunction/tanh.png.

Convergence – Parameter Initialization

Neural networks usually need random initialization to *break symmetry*.

- Biases are usually initialized to a constant value, usually 0.

Convergence – Parameter Initialization

Neural networks usually need random initialization to *break symmetry*.

- Biases are usually initialized to a constant value, usually 0.
- Weights are usually initialized to small random values, either with uniform or normal distribution.
  - The scale matters for deep networks!
  - Originally, people used $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ distribution.

Convergence – Parameter Initialization

Neural networks usually need random initialization to *break symmetry*.

- Biases are usually initialized to a constant value, usually 0.
- Weights are usually initialized to small random values, either with uniform or normal distribution.
  - The scale matters for deep networks!
  - Originally, people used $U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ distribution.
  - Xavier Glorot and Yoshua Bengio, 2010: *Understanding the difficulty of training deep feedforward neural networks*.

    The authors theoretically and experimentally show that a suitable way to initialize a $\mathrm{R}^{n\times m}$ matrix is

$$U\left[-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right].$$

Convergence – Parameter Initialization

Convergence – Parameter Initialization

Convergence – Gradient Clipping

Convergence – Gradient Clipping

Using a given maximum norm, we may clip the gradient.

$$g \leftarrow \begin{cases} g & \text{if } ||g|| \le c \\ c\frac{g}{||g||} & \text{if } ||g|| > c \end{cases}$$

The clipping can be per weight, per matrix or for the gradient as a whole.

Going Deeper

Convolutional Networks

Consider data with some structure (temporal data, speech, images, …).

Unlike densely connected layers, we might want:

- Sparse (local) interactions
- Parameter sharing (equal response everywhere)
- Shift invariance

Convolutional Networks

Reference: Image from https://i.stack.imgur.com/YDusp.png.

Convolution Operation

For a functions x and w, *convolution* $x * w$ is defined as

$$(x * w)(t) = \int x(a)w(t-a)\,\mathrm{d}a.$$

For vectors, we have

$$(\boldsymbol{x} * \boldsymbol{w})_t = \sum_i x_i w_{t-i}.$$

Convolution operation can be generalized to two dimensions by

$$(\boldsymbol{I} * \boldsymbol{K})_{i,j} = \sum_{m,n} \boldsymbol{I}_{m,n} \boldsymbol{K}_{i-m,j-n}.$$

Closely related is *cross-corellation*, where K is flipped:

$$S_{i,j} = \sum_{m,n} \boldsymbol{I}_{i+m,j+n} \boldsymbol{K}_{m,n}.$$

Convolution

Convolution Operation

The K is usually called a *kernel* or a *filter*, and we generally apply several of them at the same time.

Consider an input image with C channels. The convolution operation with F filters of width W, height H and stride S produces an output with F channels kernels of total size $W \times H \times C \times F$ and is computed as

$$(\boldsymbol{I} \star \boldsymbol{K})_{i,j,k} = \sum_{m,n,o} \boldsymbol{I}_{i \cdot S+m, j \cdot S+n, o} \boldsymbol{K}_{m,n,o,k}.$$

Convolution Operation

The K is usually called a *kernel* or a *filter*, and we generally apply several of them at the same time.

Consider an input image with C channels. The convolution operation with F filters of width W, height H and stride S produces an output with F channels kernels of total size $W \times H \times C \times F$ and is computed as

$$(\boldsymbol{I} \star \boldsymbol{K})_{i,j,k} = \sum_{m,n,o} \boldsymbol{I}_{i \cdot S+m, j \cdot S+n, o} \boldsymbol{K}_{m,n,o,k}.$$

There are multiple padding schemes, most common are:

- valid: we only use valid pixels, which causes the result to me smaller
- same: we pad original image with zero pixels so that the result is exactly the size of the input

Convolution Operation

There are two prevalent image formats:

- NHWC or channels_last: The dimensions of the 4-dimensional image tensor are batch, height, width, and channels.

  The original TensorFlow format, faster on CPU.

Convolution Operation

There are two prevalent image formats:

- NHWC or channels_last: The dimensions of the 4-dimensional image tensor are batch, height, width, and channels.

  The original TensorFlow format, faster on CPU.

- NCHW or channels_first: The dimensions of the 4-dimensional image tensor are batch, channel, height, and width.

  Usual GPU format (used by CUDA and nearly all frameworks); on TensorFlow, not all CPU kernels are available with this layout.

Pooling

Pooling is an operation similar to convolution, but we perform a fixed operation instead of multiplying by a kernel.

- Max pooling: minor translation invariance
- Average pooling

High-level CNN Architecture

We repeatedly use the following block:

1. Convolution operation
2. Non-linear activation (usually ReLU)
3. Pooling

Reference: Image from https://cdn-images-1.medium.com/max/1200/0*QyXSpqpm1wc_Dt6V. .

AlexNet – 2012 (16.4% error)

AlexNet – 2012 (16.4% error)

Training details:

- 2 GPUs for 5-6 days
- SGD with batch size 128, momentum 0.9, weight decay 0.0005
- initial learning rate 0.01, manually divided by 10 when validation error rate stopped improving
- dropout with rate 0.5 on fully-connected layers
- data augmentation using translations and horizontal reflections (choosing random 224×224 patches from 256×256 images)
  - during inference, 10 patches are used (four corner patches and a center patch, as well as their reflections)

AlexNet – ReLU vs tanh

LeNet – 1998

Achieved 0.8% test error on MNIST.

Similarities in V1 and CNNs

The primary visual cortex recognizes Gabor functions.

Similarities in V1 and CNNs

Similar functions are recognized in the first layer of a CNN.

CNNs as Regularizers – Deep Prior

Reference: Figure 1 of paper "Deep Prior", https://arxiv.org/abs/1712.05016

CNNs as Regularizers – Deep Prior

Reference: Figure 7 of paper "Deep Prior", https://arxiv.org/abs/1712.05016

CNNs as Regularizers – Deep Prior

Reference: Figure 5 of supplementary materials of paper "Deep Prior", https://arxiv.org/abs/1712.05016

CNNs as Regularizers – Deep Prior

Reference: Figure 8 of paper "Deep Prior", https://arxiv.org/abs/1712.05016

Deep Prior paper website with supplementary material

VGG – 2014 (6.8% error)

Inception (GoogLeNet) – 2014 (6.7% error)

Inception (GoogLeNet) – 2014 (6.7% error)

Inception (GoogLeNet) – 2014 (6.7% error)

Inception (GoogLeNet) – 2014 (6.7% error)

aux. classifiers, 0.3 weight

Batch Normalization

Internal covariate shift refers to the change in the distributions of hidden node activations due to the updates of network parameters during training.

Let $\boldsymbol{x} = (x_1, \ldots, x_d)$ be d-dimensional input. We would like to normalize each dimension as

$$\hat{x}_i = \frac{x_i - \mathrm{E}[x_i]}{\sqrt{\mathrm{Var}[x_i]}}.$$

Furthermore, it may be advantageous to learn suitable scale γ_i and shift β_i to produce normalized value

$$y_i = \gamma_i \hat{x}_i + \beta_i.$$

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$, $\varepsilon \in \mathbb{R}$

Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\boldsymbol{\sigma}^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$, $\varepsilon \in \mathbb{R}$

Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\boldsymbol{\sigma}^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \boldsymbol{\gamma} \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch normalization is commonly added just before a nonlinearity. Therefore, we replace $f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ by $f(BN(\boldsymbol{W}\boldsymbol{x}))$.

Batch Normalization

Consider a mini-batch of m examples $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$.

Batch normalizing transform of the mini-batch is the following transformation.

Inputs: Mini-batch $(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)})$, $\varepsilon \in \mathbb{R}$

Outputs: Normalized batch $(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(m)})$

- $\boldsymbol{\mu} \leftarrow \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}^{(i)}$
- $\boldsymbol{\sigma}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{x}^{(i)} - \mu)^2$
- $\hat{\boldsymbol{x}}^{(i)} \leftarrow (\boldsymbol{x}^{(i)} - \boldsymbol{\mu}) / \sqrt{\sigma^2 + \varepsilon}$
- $\boldsymbol{y}^{(i)} \leftarrow \gamma \hat{\boldsymbol{x}}^{(i)} + \boldsymbol{\beta}$

Batch normalization is commonly added just before a nonlinearity. Therefore, we replace $f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$ by $f(BN(\boldsymbol{W}\boldsymbol{x}))$.

During inference, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are fixed. They are either precomputed after training on the whole training data, or an exponential moving average is updated during training.

Inception with BatchNorm (4.8% error)

Inception v2 and v3 – 2015 (3.6% error)

Reference: Figure 1 of paper "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567.

Reference: Figure 3 of paper "Rethinking the Inception Architecture for Computer Vision", https://arxiv.org/abs/1512.00567.

Inception v2 and v3 – 2015 (3.6% error)

Inception v2 and v3 – 2015 (3.6% error)

Inception v2 and v3 – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

ResNet – 2015 (3.6% error)

Reference: Figure 1 of paper "Visualizing the Loss Landscape of Neural Nets", https://arxiv.org/abs/1712.09913.