Deep Learning Teaching Kit
Lab 1, Sample Solution 2

1 Backpropogation

1.1 Logistic Regression

Sigmoid Function:

$$P(y=1|x_{in}) = x_{out} = \sigma(x_{in}) = \frac{1}{1+e^{-x_{in}}}$$

, where x_{out} and x_{in} are scalars.

Derivative of Sigmoid Function:

$$\begin{aligned}
\frac{\partial x_{out}}{\partial x_{in}} &= \frac{0 - e^{-x_{in}}(-1)}{(1+e^{-x_{in}})^2} \\
&= \frac{1+e^{-x_{in}}-1}{(1+e^{-x_{in}})^2} \\
&= \frac{1+e^{-x_{in}}}{(1+e^{-x_{in}})^2} - \frac{1}{(1+e^{-x_{in}})^2} \\
&= \frac{1}{1+e^{-x_{in}}} - \frac{1}{(1+e^{-x_{in}})^2} \\
&= \frac{1}{1+e^{-x_{in}}}\left(1 - \frac{1}{1+e^{-x_{in}}}\right) \\
&= \sigma(x_{in})(1-\sigma(x_{in})) \\
&= x_{out}(1-x_{out})
\end{aligned}$$

Obtain $\frac{\partial E}{\partial x_{in}}$ using the chain rule:

$$\begin{aligned}
\frac{\partial E}{\partial x_{in}} &= \frac{\partial E}{\partial x_{out}}\frac{\partial x_{out}}{\partial x_{in}} \\
&= \frac{\partial E}{\partial x_{out}} x_{out}(1-x_{out})
\end{aligned}$$

1.2 Multinomial Logistic Regression

Softmax Function:

$$P(y = i|X_{in}) = (X_{out})_i = \frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}}$$

, where X_{out} and X_{in} are vectors.

Derivative of Softmax Function:

If $i = j$:

$$\begin{aligned}
\frac{\partial(x_{out})_i}{\partial(x_{in})_j} &= \frac{-\beta e^{-\beta(X_{in})_i}\sum_k e^{-\beta(X_{in})_k} - e^{-\beta(X_{in})_i}e^{-\beta(X_{in})_i}(-\beta)}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2} \\
&= \frac{\beta e^{-2\beta(X_{in})_i} - \beta e^{-\beta(X_{in})_i}\sum_k e^{-\beta(X_{in})_k}}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2} \\
&= \frac{\beta e^{-\beta(X_{in})_i}}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2} - \frac{\beta e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}} \\
&= -\beta\left[\frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}}\left(1 - \frac{e^{-\beta(X_{in})_i}}{\sum_k e^{-\beta(X_{in})_k}}\right)\right] \\
&= -\beta\,(X_{out})_i\,(1 - (X_{out})_i)
\end{aligned}$$

If $i \neq j$:

$$\begin{aligned}
\frac{\partial(x_{out})_i}{\partial(x_{in})_j} &= \frac{0 - e^{-\beta(X_{in})_i}e^{-\beta(X_{in})_j}(-\beta)}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2} \\
&= -\beta\left[-\frac{e^{-\beta(X_{in})_i}e^{-\beta(X_{in})_j}}{\left(\sum_k e^{-\beta(X_{in})_k}\right)^2}\right] \\
&= -\beta\ (-(X_{out})_i(X_{out})_j) \\
&= -\beta\,(X_{out})_i\,(0 - (X_{out})_j)
\end{aligned}$$

Thus,

$$\frac{\partial(x_{out})_i}{\partial(x_{in})_j} = -\beta\,(X_{out})_i\,(\delta_{i,j} - (X_{out})_j)$$

, where $\delta_{i,j}$ is the Kronecker delta.

2 MNIST Digit Recognition

2.1 Variations on the Original Architecture

We explored with various variations in model architecture, optimization methods, activation function, regularization, pooling procedure, and data augmentation. These variants were motivated by general trends in deep learning methdology, and specific models/papers which achieved top results on computer vision

tasks. We discuss each class of modifications and describe the intuition behind the approach:

- **Momentum for SGD ([5] & [7])**: For all of the models we ran, we utilized SGD with a momentum rate of 0.9. Momentum is widely used throughout the deep learning literature, and [7] point out that momentum is effective when network weights are well-initialized. Momentum augments the SGD update rule by adding an additional term where the update step at time t is $-(gradient_t + M * gradient_{t-1})$. This has the effect of taking a larger step if the previous update was pointing in a similar direction.

- **Dropout ([6])**: Dropout is a regularization method where nodes are randomly dropped from the network (output from that node set to 0) with a pre-specified probability. Dropout is utilized to avoid neurons from co-adapting to inputs coming from neurons in the previous layers, thus avoiding the weights of neurons from becoming too dependent on the weights of other neurons. Intuitively, the entire network can be viewed as an ensemble of all the randomly generated sub-networks.

- **Rectified Linear Unit Activation ([3])**: The sigmoid and tanh activation function suffer from the vanishing gradient problem, where the derivative of the loss function w.r.t. to weights in the early layers being multiplied by a value smaller than 1 many times. This is due to the fact that the derivative of these activation w.r.t to their input is bounded between 0 and 1. Additionally, both sigmoid and tanh suffer from saturation, where a neuron is stuck at either 0 or 1, and since the derivative of the output from this activation is arbitrarily close to zero, the weights of the neuron can't be adjusted. The ReLu ($f(x) = max(0, x)$) activation function combats both of these issues because the output of the activation is not bounded from above, and the derivative at positive values is 1 and therefore gradients do not vanish as severely.

- **Max-pooling ([1])** : According to Andrej Karpathy's CS231 course at Stanford, pooling's "function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.". He goes on to explain that "In addition to max pooling, the pooling units can also perform other functions, such as average pooling or even L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.". Therefore, we also experimented with utilizing L2-pooling as well as max-pooling.

- **Overlapping Stride ([4])**: In the famous Krizhevsky, Hinton, Sutskever 2012 ImageNet paper, they state that "We generally observe during training that models with overlapping pooling find it slightly more difficult to overfit". By having multiple pooling strides "consider" the same pixel, the information from that pixel is spread across multiple neurons and acts as a form of regularization, where no single neuron will dominate the prediction.

- **Data augmentation ([8])**: We followed the data augmentation procedure from the "DropConnect Paper" [8]. This enabled us to multiply the number of data points by randomly selecting a contiguous 28x28 square, randomly rotating the image between [-20,20] degrees, and padding the image back to 32x32. Additionally, this could make the model generalize better, making it invariant to the digit being shifted within the 32x32 image or being written on a slightly tilted axis.

- **Multi-column Architecture ([2])**: The "Multi-Column" paper trains 5 independent CNN's with each receiving a particular permutation of the original input image. Due to training time considerations, we attempted a 2-column and 5-column architecture, but instead of averaging the prediction across the columns, we concatenate the output of each column and feed it into a multi-column-softmax layer. This can be interpreted as a ensemble method which encourages different columns to specialize in learning different feature extractors.

2.2 Experimental Setting

All of our experiments were run using:

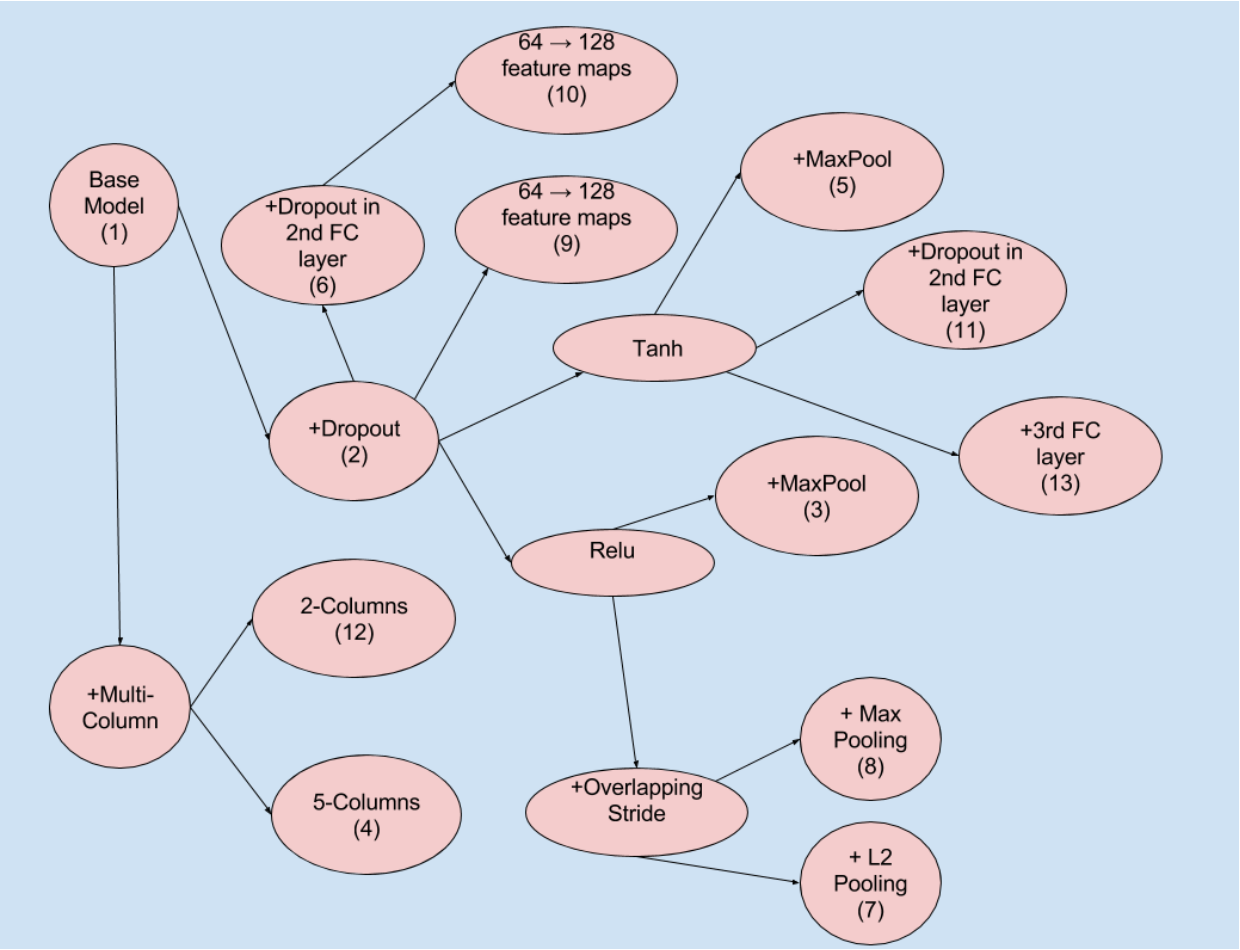
- Torch7
- With a max number of epochs set to 30
- Patience parameter for early stopping set to 10 epochs
- SGD learning rate set to 0.001
- Momentum parameter set to 0.9
- 50,000 images in training set, and 10,000 in validation (not accounting for data augmentation procedures)

2.2.1 Results

We attempted many different combinations of the variations we discussed above. The architecture for each architecture and the error on the test set are presented in the table below. The figure below tracks the path we took through the "model-space".

Model	Activation	Nb Maps	Filter	Dropout	Pooling	Stride	Data Aug	Error
1	*tanh*	64, 64, 128	5, 5	no	L2	no	no	0.46
2	*tanh*	64, 64, 128	5, 5	1 in FC	L2	no	4 times	**0.35**
3	*relu*	64, 64, 128	5, 5	1 in FC	Max	no	no	0.56
4	*relu*	5 Col, 64, 64, 128	5, 5	1 in FC	Max	no	no	0.56
5	*relu*	64, 64, 128	5, 5	1 in FC	L2	no	4 times	0.42
6	*relu*	64, 64, 128	5, 5	2 in FC	L2	no	4 times	0.38
7	*relu*	64, 64, 128	5, 5	2 in FC	L2	yes	no	0.56
8	*relu*	64, 64, 128	5, 5	2 in FC	Max	yes	no	0.53
9	*tanh*	64, 128, 128	5, 5	1 in FC	L2	no	4 times	0.40
10	*tanh*	64, 128, 128	5, 5	CV&FC	L2	no	4 times	0.40
11	*tanh*	64, 64, 128	5, 5	2 in FC	L2	no	4 times	0.41
12	*tanh*	64, 64, 256	5, 5	2 in FC	L2	no	4 times	0.41
13	*tanh*	2 Col, 64, 64, 256	5, 5	1 in FC	L2	no	4 times	0.37

Col= Multi-col convNet, FC = Fully Connected Layers, CV = Conv Layers



Training We found that the combination of ReLu with MaxPooling trained much faster than other activation-pooling settings. SGD with batch size of 1 converged to better minima, which supports the theory that the randomness in online-SGD helps escape local minima.
The Best Model: We first trained 27 epochs (terminated by early stopping). Since the validation accuracy curve was still moving upward, we ran additional 10 epochs. Each epoch took approximately 30 minutes. The total training time was approximately 19 hours. We conclude that since the base model already performed very well, there was not much room for improvement. Data augmentation, which resulted in a four times larger training set, made the most significant difference in terms of accuracy.
Interpretation We hypothesize that many of the other variations we tried did not bear fruit because the dataset is relatively small, and the networks we ran were not particularly deep. MaxPooling, overlapping stride, and dropout are all regularization methods, and since the network was not so big and we lim-ited the number of training epochs to 30, these method were not as effective as they would have been had we trained larger models for more epochs and been more prone to overfit. Similarly, ReLu activation did not play a big effect, per-haps because the network was not too deep and vanishing gradients were not a prevalent issue.

References

[1] Cs231n course at stanford.

[2] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[3] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[7] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[8] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.